Bases de dades

CFGS.INF.M02/0.12

Desenvolupament d'aplicacions multiplataforma
Desenvolupament d'aplicacions web

Generalitat de Catalunya
Departament d'Educació

ioc
institut obert
de catalunya

Aquesta col·lecció ha estat dissenyada i coordinada des de l'Institut Obert de Catalunya.

Coordinació de continguts
Verònica Mascarós Álvarez

Redacció de continguts
Isidre Guixà Miranda
Carlos Manuel Martí Hernández
Cristina Obiols Llopart
Joan Anton Pérez Braña
Aina del Tio Esteve

Adaptació de continguts
Isidre Guixà Miranda
Verònica Mascarós Álvarez
Cristina Obiols Llopart

Imatge de coberta
Creative Tools

Primera edició: febrer 2013

Dipòsit legal: DL B 12715-2016

Introducció

Avui en dia la interacció amb les bases de dades és una cosa tan freqüent que moltes vegades ni tan sols ens adonem d'aquest fet. No cal ser un professional informàtic per fer-ne un ús quotidià. Senzillament, qualsevol persona que treu diners d'un caixer automàtic, que paga les seves compres amb targetes de crèdit o de dèbit, que emplena un formulari per Internet per sol·licitar una visita amb el seu metge de capçalera, etc. està utilitzant bases de dades. Per tant, tot tècnic superior en informàtica ha de tenir prou coneixements de l'anàlisi i el disseny de bases de dades.

Aquest mòdul té com a finalitat inicial aproximar-nos conceptualment al món de les dades i al de la seva representació informàtica per excel·lència, les bases de dades. Veurem l'evolució de les bases de dades i com s'ha passat de representar les dades com una sèrie de taules amb columnes i atributs en el model relacional, a la seva representació mitjançant objectes, permetent així una major integració amb les aplicacions.

La contínua millora de capacitats tecnològiques en els últims anys ha permès construir aplicacions extremadament complexes en els sistemes d'informació amb bases de dades. Per crear aquests nous sistemes (basats en objectes complexos amb interrelacions també complexes) els dissenyadors i administradors de bases de dades, així com els programadors d'aplicacions, han hagut d'incorporar noves tecnologies orientades a objectes, fent ressorgir una metodologia pròpia, les bases de dades objecte-relacionals.

Al començament de la unitat "Introducció a les bases de dades", s'examinen els tres àmbits que cal diferenciar per tal de treballar correctament amb les dades: el món real, el món conceptual i el món de les representacions. També s'hi estudia el programari específic que gestiona les bases de dades, els sistemes gestors de bases de dades. D'altra banda, es fa un cop d'ull als diversos models de bases de dades que hi ha hagut al llarg de la història així com els diversos models més utilitzats en l'actualitat.

L'altra gran finalitat d'aquest mòdul és aprendre a utilitzar un conjunt d'eines conceptuals per descriure les dades, les seves interrelacions, el seu significat, i les limitacions necessàries per tal de garantir-ne la coherència.

En la unitat "Model Entitat-Relació", s'estudia el model de dades més àmpliament utilitzat, el model Entitat-Relació (ER). Probablement, el seu èxit ve del fet que la seva notació consisteix en una sèrie de diagrames molt senzills i entenedors. Així, doncs, s'hi estudien les estructures bàsiques del model ER (sobretot entitats, atributs i interrelacions), i les anomenades *extensions del model ER*, que comprenen algunes estructures més complexes (generalitzacions, especialitzacions i entitats associatives). A més s'examina en què consisteix el disseny de bases de dades, les fases en què es desglossa, i les decisions que cal prendre durant les diferents etapes del disseny conceptual. També s'hi donen pautes per identificar els diferents conceptes del model ER en l'àmbit de situacions concretes en què

ens podem trobar en el món real. Aquesta identificació és clau per al disseny i posterior explotació d'una base de dades.

La unitat "Model relacional i normalització", a banda d'introduir-nos els conceptes bàsics del model relacional, permet aprendre les tècniques de traducció del model ER al model relacional. Aquest pas és importantíssim i permet implementar en sistemes gestors de bases de dades relacionals els dissenys prèviament ideats. La part dedicada a la normalització ens ofereix procediments per a obtenir bases de dades íntegres i sense redundàncies havent partit de bases de dades inicialment no òptimes i/o anòmales.

Els SGBD ofereixen eines per a la gestió de les BD. El llenguatge SQL n'és una de les més importants. L'SQL facilita tant l'obtenció de la informació que hi ha emmagatzemada en una BD com la introducció, modificació, eliminació i estructuració d'aquesta. En la unitat didàctica "Llenguatge SQL.Consultes" ens introduirem en la possibilitat d'obtenir la informació mitjançant les consultes que anomenem *simples*. Però també aprofitarem la potència del llenguatge SQL per obtenir la informació ordenada, agrupada, filtrada, combinada, etc.

En la unitat "Llenguatge SQL per a la manipulació i definició de les dades. Control de transaccions i concurrència ", aprendrem a definir, modificar i gestionar les dades de les BD mitjançant les sentències SQL de definició i manipulació de dades. També coneixerem el concepte de transacció i aprendrem a dissenyar-ne i a implementar aquest mecanisme tan important a l'hora de garantir la consistència de les dades.

La unitat formativa "Llenguatges SQL: DCL i extensió procedimental ", està composta per la unitat "Gestió d'usuaris", en què es fa resó de la importància de la seguretat en un sistema d'informació i s'aprèn com gestionar els privilegis que tindran els usuaris i grups d'usuaris sobre els elements que conformen una base de dades; i per la unitat "Programació de bases de dades", en què s'ensenya a programar usant la extensió procedimental del llenguatge SQL PL/pgSQL.

La unitat formativa "Bases de dades objecte-relacionals" està composta per la unitat que porta el mateix nom. En l'apartat "Bases de dades objecte-relacionals" es fa ressò de com les bases de dades objecte-relacionals agrupen certes particularitats de les bases de dades relacionals i les orientades a objectes, i s'aprofundeix en aquestes característiques. En l'apartat "Objectes i col·leccions" s'ensenya a distingir amb quins elements treballen aquests tipus de bases de dades enfront de les bases purament relacionals, així com el seu funcionament bàsic. En l'apartat "DDL i DML de les bases de dades objecte-relacional" es veuran tant el llenguatge de definició de dades (DDL) com el llenguatge de manipulació de dades (DML) aplicats a les bases objecte-relacionals; el primer permetrà a l'usuari definir tant les estructures de dades com les funcions o procediments que permetran consultar-les, i el segon permetrà dur a terme tasques de consulta o manipulació de dades.

Els continguts d'aquest mòdul tenen una orientació bàsicament professionalitzadora, es proposa una formació pràctica i aplicable amb l'objectiu que l'alumnat aprengui a saber fer. Un cop llegits detingudament els continguts de cada apartat, cal realitzar les activitats i els exercicis d'autoavaluació per tal de posar en pràctica i comprovar l'assoliment dels coneixements adquirits. I, sobretot en les unitats referides al llenguatge SQL i als SGBDR concrets, és molt important la pràctica en els propis ordinadors dels exemples que es descriuen en el material en pdf.

Resultats d'aprenentatge

En finalitzar aquest mòdul l'alumne/a:

Introducció a les bases de dades

1. Reconeix els elements de les bases de dades analitzant les seves funcions i valorant la utilitat dels sistemes gestors.
2. Dissenya models lògics normalitzats interpretant diagrames entitat/relació.

Llenguatges SQL: DML i DDL

1. Consulta i modifica la informació emmagatzemada en una base de dades emprant assistents, eines gràfiques i el llenguatge de manipulació de dades.
2. Realitza el disseny físic de bases de dades utilitzant assistents, eines gràfiques i el llenguatge de definició de dades.

Llenguatges SQL: DCL i extensió procedimental

1. Implanta mètodes de control d'accés utilitzant assistents, eines gràfiques i comandes del llenguatge del sistema gestor de bases de dades corporatiu.
2. Desenvolupa procediments emmagatzemats avaluant i utilitzant les sentències del llenguatge incorporat en el sistema gestor de bases de dades corporatiu.

Bases de dades objecte-relacionals

1. Gestiona la informació emmagatzemada en bases de dades objecte-relacionals, avaluant i utilitzant les possibilitats que proporciona el sistema gestor.

Continguts

Introducció a les bases de dades

Unitat 1

Introducció a les bases de dades

1. Introducció a les bases de dades i als sistemes gestors de bases de dades
2. Models de bases de dades

Unitat 2

Model Entitat-Relació

1. Conceptes del model Entitat-Relació. Entitats. Relacions
2. Diagrames Entitat-Relació
3. Annex: Decisions de disseny

Unitat 3

Model relacional i normalització

1. Model relacional
2. Normalització

Llenguatges SQL: DML i DDL

Unitat 4

Llenguatge SQL. Consultes

1. Consultes de selecció simples
2. Consultes de selecció complexes
3. Annex 1. MySQL

Unitat 5

Llenguatge SQL per a la manipulació i definició de les dades. Control de transaccions i concurrència

1. Instruccions per a la manipulació de dades
2. DDL
3. Control de transaccions i concurrències

Llenguatges SQL: DCL i extensió procedimental

Unitat 6

Gestió d'usuaris

1. Gestió d'usuaris i privilegis
2. Vistes i regles

Unitat 7

Programació de bases de dades

1. El dialecte SQL de PostgreSQL
2. PL/PgSQL: extensió procedimental del llenguatge SQL
3. Cursors i control d'errors
4. Disparadors

Bases de dades objecte-relacionals

Unitat 8

Bases de dades objecte-relacionals

1. Bases de dades d'objectes relacionals
2. Objectes i col·leccions
3. DDL i DML de les bases de dades objecte-relacionals

Introducció a les bases de dades

Carlos Manuel Martí Hernández

Índex

Introducció

Avui en dia la interacció amb les bases de dades és una cosa tan freqüent que moltes vegades ni tan sols ens adonem d'aquest fet. Senzillament, quan traiem diners d'un caixer automàtic estem utilitzant bases de dades. Per tant, tot tècnic superior en informàtica ha de tenir molt clars una sèrie de conceptes al voltant de les dades i de la seva representació informàtica.

Aquesta unitat didàctica té com a objectiu principal el d'aproximar-nos conceptualment al món de les dades i al de les seva representació informàtica per excel·lència, les bases de dades.

Al llarg de l'apartat "Introducció a les bases de dades i als sistemes gestors de bases de dades", examinarem els tres àmbits que hem de ser capaços de diferenciar per tal de treballar correctament amb les dades: el món real, el món conceptual i el món de les representacions.

Per món real, s'entén la part de la realitat (tingui un caire tangible o no) que en un moment determinat ens interessa informatitzar perquè hem rebut un encàrrec en aquest sentit. Mitjançant una sèrie de processos que impliquen, en primer lloc, l'observació de la realitat i, a continuació, un conjunt d'abstraccions de les informacions considerades rellevants, es construeix un model que conceptualitza els aspectes de la realitat amb els quals volem treballar. Finalment, cal implementar alguna representació informàtica concreta dels conceptes abstrets durant la fase anterior, per tal de poder-hi treballar fent servir les tecnologies que ens ofereixen les bases de dades i, també, els seus sistemes gestors.

També coneixerem les diferències que comporta treballar amb bases de dades en contraposició a treballar utilitzant altres mitjans, com ara els fitxers. I abordarem el programari que gestiona i controla les bases de dades, és a dir, els sistemes gestors. Farem un repàs d'algunes de les orientacions més importants que han tingut, que tenen i que (potser) tindran, i dels objectius generals que persegueixen. I, finalment, ens introduirem en algunes nocions bàsiques relatives a l'arquitectura dels SGBD.

Al llarg de l'apartat "Models de bases de dades", es plantegen diferents models de bases de dades (jeràrquic, en xarxa, relacional, orientat a objectes, etc.) que han tingut, tenen i tindran, més o menys implantació al llarg del temps, segons l'evolució tant de les necessitats tècniques com dels mercats. I també és fa una mirada en les bases de dades distribuïdes, en la seva arquitectura i el disseny d'aquests sistemes d'emmagatzematge d'informació.

Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Reconeix els elements de les bases de dades analitzant les seves funcions i valorant la utilitat dels sistemes gestors.
    - Identifica els diferents elements, objectes i estructures d'emmagatzematge físic disponibles en un SGBD corporatiu i relacionar-lo amb els elements de l'esquema físic de la base de dades.
    - Identifica els diferents sistemes lògics d'emmagatzematge i les seves característiques.
    - Identifica els diferents tipus de bases de dades en funció de la ubicació de la informació.
    - Identifica un sistema gestor de bases de dades: funcions, components, objectius, tipus de llenguatge de bases de dades i diferents usuaris de la base de dades.
    - Identifica l'estructura d'un diccionari de dades.
    - Diferencia entre el nivell intern, el nivell conceptual i el nivell físic d'una base de dades.
    - Diferencia entre els diferents models de bases de dades.
    - Identifica les bases de dades distribuïdes: utilitat, diferències, avantatges i inconvenients, distribució de les dades, arquitectura, seguretat i recuperació.
    - Identifica el disseny d'una base de dades distribuïda.
    - Identifica les bases de dades centralitzades i les bases de dades distribuïdes: utilitat, diferències, avantatges i inconvenients.
    - Diferencia entre les diferents tècniques de fragmentació en un model distribuït.
    - Identifica les tècniques de distribució de dades.

1. Introducció a les bases de dades i als sistemes gestors de bases de dades

Les dades que s'utilitzen de manera informatitzada s'emmagatzemen, habitualment, en bases de dades (BD).

Per tal d'estar en condicions d'abordar l'estudi de les BD i del programari que serveix per gestionar-les, és a dir, els sistemes gestors de bases de dades (SGBD), és imprescindible entendre prèviament uns quants conceptes teòrics fonamentals, referents a les dades i a la seva representació. A banda, doncs, dels conceptes relatius a les dades i a les bases de dades, caldrà conèixer quines representacions de la informació s'utilitzen habitualment així com l'evolució del programari d'aquest àmbit.

1.1 Les dades i les bases de dades

Per a tot informàtic que hagi de treballar amb bases de dades (BD), és imprescindible saber distingir tres àmbits ben diferenciats, però que al mateix temps estan fortament interrelacionats, els quals fan referència, respectivament, a la realitat, a la seva conceptualització, i a la seva representació informàtica ulterior. Així, doncs, considerem els tres "mons" següents:

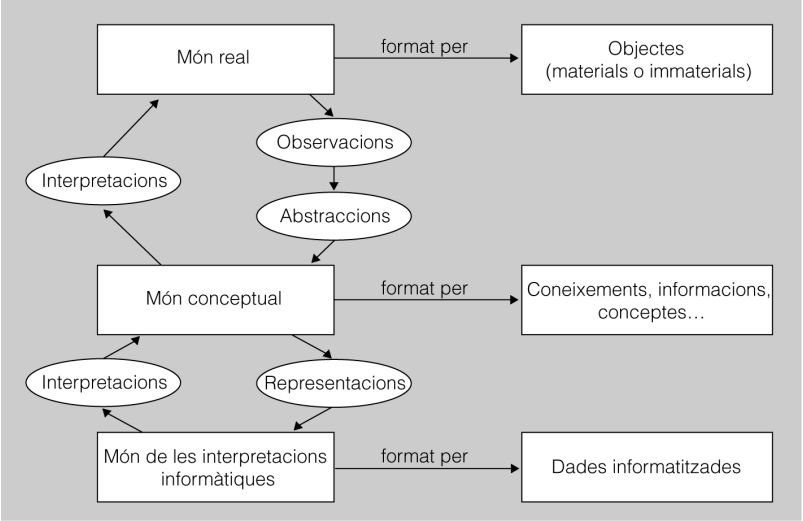
- **El món real**. Està constituït pels objectes (materials o no) de la realitat que ens interessen i amb els quals haurem de treballar.

- **El món conceptual**. És el conjunt de coneixements o informacions obtinguts gràcies a l'observació de la part del món real que ens interessa. Un mateix món real pot donar lloc a diferents mons conceptuals, en funció de la manera de percebre la realitat, o els interessos de l'observador d'aquesta.

- **El món de les representacions**. Està format per les representacions informàtiques, o dades, del món conceptual, necessàries per poder treballar.

En la figura 1.1 es representen, de manera esquematitzada, els tres mons que els informàtics han de considerar, i les interrelacions que mantenen.

FIGURA 1.1. Els tres móns



1.1.1 Les dades i la seva representació

Un cop estructurats, els conceptes entorn de la realitat passen a ser veritables informacions, amb les quals els humans ens podem comunicar i començar a treballar.

Però encara cal donar un altre pas que ens permeti representar aquestes informacions, de tal manera que les puguem tractar informàticament mitjançant BD i aplicacions, i aprofitem així tot el potencial de les noves tecnologies.

Les **dades** són representacions informàtiques de la informació disponible, relativa als objectes del món real del nostre interès.
El **món de les representacions** està format per les dades informatitzades amb les quals treballem.

Ara bé, la conversió de les concepcions en dades no és automàtica, ni de molt. Requereix passar per dues fases successives de disseny, en què es prenen decisions que poden derivar en resultats dispars. Aquestes dues fases de disseny són les següents:

1. **Fase de disseny lògic**. Es treballa amb el model abstracte de dades obtingut al final de l'etapa de disseny conceptual, per tal de traduir-ho al model de dades utilitzat pel SGBD amb el qual es vol implementar i mantenir la futura BD.

2. **Fase de disseny físic**. Es poden fer certes modificacions sobre l'esquema lògic obtingut en la fase de disseny anterior, per tal d'incrementar l'eficiència en algunes de les operacions que s'hagin de fer amb les dades.

Per tant, cal ser conscients que, en un mateix conjunt de coneixements entorn d'una mateixa realitat, aquests es poden representar de maneres diferents a causa, per exemple, dels factors següents:

- Les decisions de disseny preses (tant a nivell conceptual, com a nivell lògic i físic).
- La tecnologia emprada (fitxers, BD relacionals, BD distribuïdes, etc.).

La possibilitat que hi hagi aquestes diferències no implica que tots els resultats es puguin considerar equivalents, sense més ni més, ja que, normalment, les representacions diferents donen lloc a nivells d'eficiència també diferents. Aquest fet pot tenir conseqüències importants, ja que la responsabilitat de tot informàtic inclou garantir la correcció de les representacions, però també l'eficiència de les implementacions.

1.1.2 Entitats, atributs i valors

Tres elements caracteritzen fonamentalment les informacions:

1. Les **entitats** són els objectes del món real que conceptualitzem. Són identificables, és a dir, distingibles els uns dels altres. I ens interessen algunes (com a mínim una) de les seves propietats.
2. Els **atributs** són les propietats de les entitats que ens interessen.
3. Els **valors** són els continguts concrets dels atributs, les determinacions concretes que assoleixen.

En principi, els atributs haurien d'emmagatzemar un sol valor en cada instant. D'aquesta manera, els nostres models seran, de bon principi, compatibles amb el model lògic de dades més àmpliament utilitzat: el **model relacional**.

Atributs multivaluats

Permeten emmagatzemar més d'un valor simultàniament. El seu ús no és gaire recomanable perquè són incompatibles amb el model relacional i amb la immensa majoria dels SGBD que hi ha en el mercat.

Exemple d'entitat, atributs i valors

Considerarem que una pel·lícula concreta és una entitat, perquè és un objecte del món real, que hem conceptualitzat dins d'una categoria (la dels films cinematogràfics), i que al mateix temps és distingible d'altres entitats de la mateixa categoria (és a dir, d'altres films).

D'aquesta pel·lícula ens interessaran alguns aspectes, que anomenarem *atributs*, com per exemple, el títol, el director i l'any de producció.

Finalment, aquests atributs adoptaran uns valors concrets com ara, i respectivament, *Paths of glory*, Stanley Kubrick i 1957.

Si només coneixem dos d'aquests tres elements, no disposarem d'una veritable informació, ja que es produirà alguna de les mancances següents:

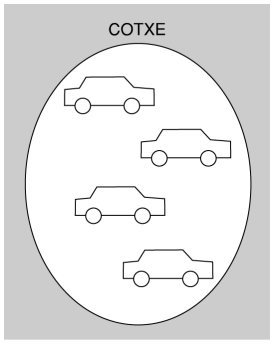
- Desconeixerem l'entitat (l'objecte) a què va associat l'atribut i el valor respectiu, i per tant no servirà de gaire conèixer els altres aspectes.

- Desconeixerem quin atribut (propietat) de l'entitat adopta el valor obtingut, la qual cosa pot donar lloc a equívocs.
- Sabrem que l'entitat té una certa propietat, però en desconeixerem el valor, i per tant aquest coneixement difícilment ens resultarà útil.

1.1.3 Entitats tipus i entitats instància

El terme *entitat* fa referència a algun objecte concret del món real, conceptualitzat, i del qual ens interessen algunes característiques. Però aquest terme pot tenir dos significats diferents, segons a què faci referència. Per tal de distingir-los, es pot afegir un adjectiu al substantiu esmentat i obtenir, així, els dos sintagmes següents:

Entitat tipus. Es tracta d'un tipus genèric d'entitat o, si es prefereix, d'una abstracció, que fa referència a una classe de coses com, per exemple, els cotxes, en general.



Entitats instància

Entitat instància. Es refereix a la conceptualització d'un objecte concret del món real, com ara un cotxe concret, distingible dels altres objectes del mateix tipus, gràcies a alguna propietat (com podria ser el valor de l'atribut Matricula).

En terminologia de teoria de conjunts, podríem dir que una entitat tipus és un conjunt, i que cada entitat instància és un element del conjunt, tal com es reflecteix en la figura adjacent.

1.1.4 Tipus de dada i domini dels atributs

Anomenem **domini** tot el conjunt de valors que un atribut determinat pot prendre vàlidament.

El concepte domini no es correspon amb el de *tipus de dada*, utilitzat tant en l'àmbit de les BD com també en el de programació.

Un **tipus de dada** defineix un conjunt de valors amb unes característiques comunes que els fan compatibles, per la qual cosa també defineix una sèrie d'operacions admissibles sobre aquests valors.

Exemple de tipus de dada

Podem considerar els nombres enters com un tipus de dada (diferent d'altres tipus, com per exemple els nombres reals, els caràcters, etc.), sobre el qual es poden definir certes operacions, com la suma, la resta, la multiplicació o la divisió entera (però no la divisió exacta, que només és possible entre els nombres reals).

Així, doncs, ambdós conceptes (domini i tipus de dada) s'assemblen, ja que tots dos limiten els valors acceptables. Allò que els diferencia és que un domini no estableix per si mateix cap conjunt d'operacions, mentre que un tipus de dada, per definició, si que ho fa.

Una altra diferència és que, en la pràctica, un domini és un subconjunt de valors possibles d'un tipus de dada. En alguns casos, pot interessar delimitar el rang de valors admesos per un tipus de dada determinat. Això es fa establint un domini.

Exemple de domini

Imaginem que, en l'àmbit d'uns estudis determinats, s'exigeix un mínim d'assistència a classes presencials per tal d'aconseguir el títol corresponent, independentment de les qualificacions obtingudes.

Imaginem que s'admet, durant tot el curs acadèmic, un màxim de vint faltes injustificades. Doncs bé, hi podria haver un atribut de l'entitat ALUMNE, anomenat, per exemple, NombreFaltes, que recollís aquesta circumstància.

Aquest atribut podria emmagatzemar dades de tipus enter. I també se'n podria limitar el domini de 0 (per indicar que no hi ha hagut cap inassistència) a vint faltes injustificades, ja que en arribar a aquest límit es produiria l'expulsió de l'alumne.

1.1.5 Valor nul dels atributs

De vegades, el valor d'un atribut és desconegut o, fins i tot, no existeix. Per representar aquesta circumstància, l'atribut en qüestió haurà d'admetre el valor nul.

L'expressió **valor nul** indica que no hi ha cap valor associat a un atribut determinat d'una entitat instància concreta.

Perquè un atribut admeti el valor nul, s'ha d'especificar aquesta possibilitat a l'hora de definir-ne el domini.

Exemple de valor nul

Considerem ara que l'entitat ALUMNE disposa d'un atribut anomenat Telefon, per emmagatzemar un número telefònic de contacte de cadascuna de les persones matriculades.

Si l'atribut Telefon no admetés valors nuls, el sistema no permetria que es matriculessin persones que no disposessin de telèfon.

En canvi, si s'ha admès aquesta possibilitat en definir el domini de l'atribut que ara ens ocupa, el sistema acceptarà la matrícula de les persones que no disposin de telèfon, o bé de les que senzillament no se'l saben de memòria i que, per tant, no el poden indicar en el moment de formalitzar la matrícula, de manera que la seva incorporació en la BD queda pendent.

No s'ha de confondre valor nul amb el zero, si estem tractant amb valors numèrics, o amb l'espai en blanc, si estem treballant amb caràcters. Tant l'un com l'altre són valors amb un significat propi. En canvi, el valor nul implica l'absència total de valor.

1.1.6 Atributs identificadors i claus

Un **atribut identificador** és el que permet distingir inequívocament cada entitat instància de la resta, pel fet que el seu valor és únic, i no es repeteix en diferents entitats instància.

Els atributs d'una entitat seran identificadors, o no, en funció de l'objecte del món real que l'entitat vulgui modelitzar.

Exemple d'atribut identificador

L'atribut DNI pot servir molt bé per identificar les instàncies d'una entitat que modelitzi els alumnes d'un centre docent, ja que cada alumne tindrà un DNI diferent.

Però si l'entitat amb què treballem vol modelitzar les qualificacions finals dels alumnes, el DNI per si sol no permetrà identificar les diferents entitats instància, ja que a cada alumne correspondrà una nota final per a cada assignatura en la qual s'hagi matriculat.

De vegades, un sol atribut no és suficient per identificar inequívocament les diferents instàncies d'una entitat. Aleshores, cal recórrer a la combinació dels valors de dos o més atributs de la mateixa entitat.

Tot atribut o conjunt d'atributs que permeten identificar inequívocament les instàncies d'una entitat s'anomenen **claus**.

Tot atribut identificador és, al mateix temps, una clau. Però els atributs que formen part d'un conjunt de més d'un atribut que actua com a clau de l'entitat no són identificadors, ja que, per si mateixos, no són capaços d'identificar les entitats instància.

Exemple de clau formada per més d'un atribut

Per tal de diferenciar les instàncies d'una entitat que vol reflectir les notes finals dels alumnes en cada assignatura en què s'hagin matriculat, cal combinar els valors de dos atributs: un que designi l'alumne de què es tracti (típicament, el DNI), i un altre que indiqui l'assignatura a la qual correspon la nota (que podria ser una cosa com ara CodiAssignatura).

Això és així perquè un alumne podrà estar matriculat en diferents assignatures, per la qual cosa el seu DNI es repetirà en diferents instàncies. I, al mateix temps, diversos d'alumnes podran estar matriculats d'una mateixa assignatura i, per tant, els valors de l'atribut CodiAssignatura també es podran repetir.

En canvi, cada alumne tindrà una instància per a cada assignatura en què s'hagi matriculat, fins que l'aprovi, per tal de reflectir la nota final. Modelitzada l'entitat d'aquesta manera, la combinació de valors de DNI i CodiAssignatura no es repetirà mai, i el conjunt format per aquests dos atributs podrà servir com a clau.

Per definició, ni els atributs identificadors ni els que formen part d'una clau poden admetre mai el valor nul, perquè aleshores no servirien per distingir les entitats instància sense valor en un dels tipus d'atribut esmentat de la resta.

1.1.7 El món de les representacions

Ja sabem que les dades són informacions representades informàticament. Per tant, també podríem anomenar *món de les dades* el món de les representacions.

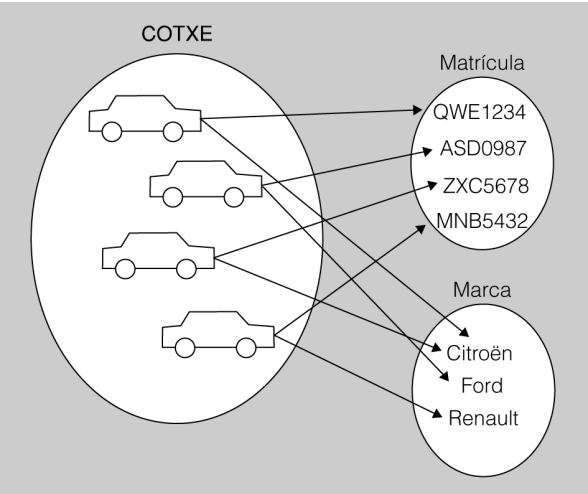
La representació informàtica més freqüent en l'àmbit de les BD és la representació tabular, la qual s'implementa habitualment en fitxers que s'estructuren en registres i camps.

En el fons, les BD només són conjunts de fitxers interrelacionats (o, si es prefereix, que emmagatzemen dades que estan interrelacionades). Però no serveix de res emmagatzemar dades si, posteriorment, no hi accedim. Hi ha diferents tipus d'accés a les dades que convé examinar: seqüencial, directe, per valor i per posició.

1.1.8 Les representacions tabulars i la seva implementació: els fitxers

Les informacions són conceptualitzacions obtingudes a partir de l'observació del món real. Ara bé, les informacions no són gaire còmodes per treballar.

FIGURA 1.2. Exemple de representació no informatitzada



En la figura 1.2 podem veure una representació gràfica, no informatitzada, de l'entitat COTXES, que consta de dos atributs: Matrícula i Marca. Evidentment, si augmentés el nombre d'entitats instància, o bé el nombre d'atributs a considerar, aquest tipus de representació no serviria per a res.

És necessari representar les informacions per facilitar les tasques a fer amb elles, com ara les consultes, els processaments, les transmissions, etc.

La representació més freqüent en l'àmbit informàtic de les BD és l'anomenada **representació tabular** (o, el que és el mateix, *en forma de taula*).

Cada taula representa una entitat genèrica, i està estructurada en files (agrupacions horitzontals de cel·les) i columnes (agrupacions verticals de cel·les):

- Cada fila representa una entitat instància.
- Cada columna representa un atribut.
- Cada cel·la (és a dir, cada intersecció d'una fila i d'una columna) emmagatzema el valor que tingui l'atribut de l'entitat instància de què es tracti.

Fitxer

El terme fitxer té altres accepcions, com ara en l'àmbit dels sistemes operatius, que no tenen gaire a veure amb el concepte que hem exposat aquí.

La implementació informàtica de les representacions tabulars es materialitza mitjançant els anomenats **fitxers de dades**. S'entén per fitxer la implementació informàtica d'una taula, amb les dades estructurades en registres i camps.

Els fitxers s'implementen seguin aquestes consideracions:

- La implementació de cada entitat instància s'anomena **registre**, i equival a una fila de la representació tabular.
- La implementació de cada atribut s'anomena **camp**, i equival a una columna de la representació tabular.
- Cada intersecció d'un registre i d'un camp emmagatzema el **valor** que tingui el camp del registre de què es tracti.

Els fitxers s'han d'emmagatzemar en algun dispositiu de memòria externa de l'ordinador, típicament un disc dur, per tal de conservar les dades permanentment. L'emmagatzemament en la memòria interna no satisfà aquest objectiu perquè és volàtil.

En la taula 1.1, podem veure una representació tabular de l'entitat COTXES, que només consta de dos camps: Matrícula i Marca. Si augmentés el nombre de registres a emmagatzemar només caldria afegir més files. I si haguéssim de considerar més camps, només caldria afegir més columnes.

Però en cap cas no es comprometria la complexitat de l'estructura tabular, que seria, essencialment, la mateixa.

Taula 1.1. Exemple de representació tabular

Cotxes	
Matrícula	**Marca**
QWE1234	Citroën
ASD0987	Ford
ZXC5678	Citroën
MNB5432	Renault

1.1.9 Les BD

Normalment, quan hàgim de representar informàticament certes informacions (corresponents, doncs, al món conceptual), no ens trobarem amb una sola entitat tipus, sinó amb unes quantes.

Intuïtivament, podem pressuposar que si partim d'un nombre concret d'entitats tipus necessitarem, com a mínim, el mateix nombre de taules per representar-les (i probablement algunes més). Ara bé, aquestes taules, o fitxers, no seran objectes inconnexos, sinó que hauran d'estar interrelacionats.

Les **interrelacions** són informacions que permeten associar les entitats entre elles.

Les interrelacions entre els registres de dues (o més) taules es fan mitjançant camps del mateix tipus de dada que emmagatzemin els mateixos valors.

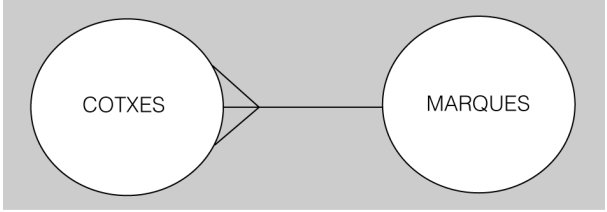
Exemple d'entitats interrelacionades

Imaginem ara que construïm una petita BD. Només hi ha dues entitats tipus del nostre interès: COTXES i MARQUES.

També tenim una altra informació complementària, sobre la interrelació entre ambdues entitats: cada cotxe serà d'una marca concreta, però hi podrà haver molts cotxes de cada marca.

En la figura 1.3, es mostra una representació de les dues entitats (COTXES i MARQUES) interrelacionades.

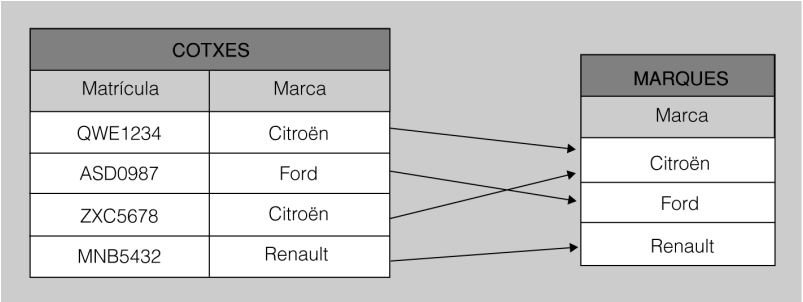
Figura 1.3



L'entitat MARQUES només té l'atribut Marca, i l'entitat COTXES només té l'atribut Matrícula. El fitxer per representar COTXES haurà d'afegir un camp addicional, per tal de permetre la interrelació amb el fitxer que representa l'entitat MARQUES.

La figura 1.4 mostra la interrelació entre els dos fitxers corresponents a l'entitat COTXES i a l'entitat MARQUES.

Figura 1.4. Exemple de fitxers interrelacionats



La interrelació entre fitxers implica que els canvis de valor dels camps que serveixen per interrelacionar-los (o, fins i tot, la seva supressió) han de quedar reflectits en tots els fitxers implicats, per tal de mantenir la coherència de les dades. Per tant, els programes que treballen amb fitxers de dades interrelacionats sempre tindran un plus de complexitat, derivat de l'exigència que acabem de comentar.

Una **BD** consisteix en un conjunt de fitxers de dades interrelacionats.

Un sistema gestor de bases de dades (SGBD) és un tipus de programari especialitzat en gestionar i administrar bases de dades.

En aquest sentit, els **SGBD** s'han anat desenvolupant tenint com a un dels objectius principals facilitar la programació amb accés a dades persistents, i per gestionar l'accés simultani a les dades per part de diferents usuaris.

1.1.10 El nivell lògic i el nivell físic

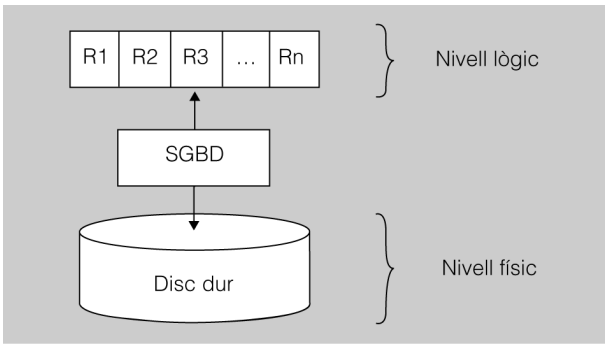
L'organització de les dades, i el seu enregistrament i accés, es poden considerar des de dos punts de vista, més o menys propers a la implementació física de l'enregistrament de les dades:

- **Nivell lògic**. Permet treballar amb les dades de manera més senzilla, independentment de la implementació física concreta, que no cal conèixer. És la manera de treballar més productiva i, per tant, la més recomanable, sempre que les circumstàncies no ens obliguin a fer optimitzacions a nivells més baixos.

- **Nivell físic**. Implica un coneixement a baix nivell de la implementació física de l'organització de les dades i el seu accés.

En la figura 1.5, es mostra la doble perspectiva apuntada, la lògica i la física. En el disc dur es desen les dades, organitzades de determinada manera a nivell físic. L'SGBD ens permet accedir a les dades emmagatzemades en el disc dur considerant només aspectes de nivell lògic, com ara seqüències de registres.

FIGURA 1.5. Nivell lògic i nivell físic



Exemples de treball a nivell lògic i a nivell físic

Nivell lògic: es treballa tenint en compte, fonamentalment, les taules, amb els camps i registres corresponents, i les seves interrelacions.

Nivell físic: es treballa considerant altres factors a més baix nivell, com ara l'encadenament dels registres físics, la compressió de dades, les tipologies d'índexs, etc.

1.2 Conceptes de fitxers i bases de dades

Les BD són conjunts estructurats de dades organitzades en entitats, les quals estan interrelacionades.

És important conèixer el concepte de BD i el seu origen. També és interessant establir una comparativa d'avantatges i d'inconvenients segons si s'utilitzen fitxers tradicionals o BD.

Hi ha diferents perspectives des de les quals es pot observar una BD, i diferents tasques i mètodes de treball sobre una BD, que tot informàtic ha de conèixer.

En el mercat hi ha diferents models de BD: el jeràrquic, en xarxa, relacional, orientat a objectes, etc. És interessant, doncs, conèixer les característiques de cadascun dels models.

1.2.1 Concepte i origen de les BD

Les BD no existeixen per casualitat. Van aparèixer per donar resposta a una sèrie de necessitats. La millor manera d'entendre tant aquestes circumstàncies com el concepte de BD que van originar és fer una petita aproximació històrica a la seva evolució.

Les aplicacions informàtiques dels anys seixanta del segle XX tenien, per regla general, les característiques següents:

- Normalment, consistien en processos per lots (en anglès, *batch processing*), amb les particularitats següents:
  - Un lot és un conjunt finit de feines que es volen tractar com un tot.
  - El seu processament, una vegada engegat, no necessita cap interacció amb l'usuari.
  - Normalment, aquest tipus d'execució es realitza en tasques repetitives sobre gran volums d'informació.

- Realitzaven tasques molt específiques, relacionades amb molt poques entitats tipus, com per exemple l'emissió de factures, la confecció de nòmines de personal, etc.
- Normalment, els programes treballaven de manera seqüencial sobre un sol fitxer mestre, que estava emmagatzemat en una cinta magnètica (encara no, en un disc dur), i generaven un altre fitxer com a resultat.
- Quan es detectava la necessitat d'implementar una nova aplicació que utilitzés parcialment les dades contingudes en un fitxer i, a més, algunes altres de noves, es dissenyava un nou fitxer amb tots els camps necessaris, i s'omplia amb les dades corresponents:
  - Les dades que ja eren en l'antic fitxer es podien copiar en el nou, justament, mitjançant un altre processament per lots (*batch*).

- D'aquesta manera s'aconseguia que el nou programa no hagués de treballar amb molts fitxers, la qual cosa en simplificava el codi, d'una banda, i d'una altra n'optimitzava el temps d'execució.
- Com a contrapartida, aquesta manera de treballar comportava la redundància d'algunes dades, que eren repetides en diferents fitxers. Aquest fet dificultava el manteniment de la coherència d'aquestes dades.

Posteriorment, l'evolució tecnològica va fer possible la implantació progressiva de tres nous elements:

- Els terminals. Dispositius de maquinari per introduir o mostrar dades de les computadores.
- Els discos durs. Dispositius d'emmagatzemament d'alt rendiment, que no estaven limitats *de facto* a l'accés seqüencial.
- Les xarxes de comunicació.

A partir d'aquestes innovacions, els programes informàtics van haver d'implementar la possibilitat de realitzar consultes i actualitzacions de les mateixes dades, simultàniament, per part de diferents usuaris.

Al mateix temps, es va anar produint un fenomen que consistia en la integració de les diferents aplicacions informàtiques que utilitzava cada organització (per exemple, gestions d'estocs, facturacions, proveïdors...). Aquesta tendència requeria les accions següents:

- Interrelacionar els fitxers de les antigues aplicacions.
- Eliminar la redundància, és a dir, la repetició innecessària de dades, que a més de resultar ineficient posa en risc la seva coherència.

Tant la interrelació dels fitxers com el fet que cada vegada hi accedien simultàniament més usuaris exigien unes estructures físiques que proporcionessin accessos raonablement ràpids, com ara índexs.

Al començament dels anys setanta, aquests conjunts de fitxers interrelacionats, compartits per diferents processos i usuaris simultàniament, i amb estructures complexes, van rebre inicialment el nom de *data bases*, o DB (*bases de dades*, o BD, en català).

I al final dels anys setanta van anar sortint al mercat programaris encara més sofisticats, que permetien gestionar més fàcilment les relacions entre fitxers, i ja estaven en condicions de garantir l'actualització simultània de dades per part de diferents usuaris, etc. Aquests programaris es van anomenar **sistemes gestors de bases de dades**, o **SGBD** (*data base management systems*, o **DBMS**, en anglès).

Amb aquesta perspectiva històrica, doncs, podem donar una definició de BD més completa.

> Una **BD** és la representació informàtica dels conjunts d'entitats instància corresponents a diferents entitats tipus i de les relacions entre aquestes. Aquest conjunt estructurat de dades ha de poder ser utilitzat de manera compartida i simultània per una pluralitat d'usuaris de diferents tipus.

1.2.2 Fitxers i BD

Els fitxers tradicionals (i els programes necessaris per treballar-hi) s'han trobat amb serioses dificultats per satisfer les creixents necessitats dels usuaris en pràcticament tots els àmbits.

Per aquesta raó, les BD s'han anat implantant com a mecanisme per excel·lència d'emmagatzematge, processament i obtenció d'informació, tot desplaçant progressivament els fitxers de la seva posició preeminent anterior. La taula 1.2 conté una breu descripció de les principals diferències entre els sistemes basats en fitxers tradicionals i les BD.

TAULA 1.2. Fitxers i BD

	Fitxers	**Bases de dades**
Entitats tipus	Les entitats instància d'un fitxer pertanyen a una sola entitat tipus.	Les BD contenen entitats instància d'infinitat d'entitats tipus interrelacionades.
Interrelacions	El sistema no interrelaciona fitxers.	El sistema té previstes eines per interrelacionar fitxers.
Redundàncies	És necessari crear fitxers a mida de cada aplicació, amb totes les dades necessàries, encara que estiguin repetides en altres fitxers.	Tècnicament, totes les aplicacions poden treballar amb la mateixa BD, la qual cosa evita la redundància de dades i els riscos que comporta.
Inconsistències	És possible que els valors d'unes mateixes dades en diferents fitxers no coincideixin, si els programadors no les han actualitzat degudament.	Si les interrelacions estan ben dissenyades, les dades només han d'estar emmagatzemades en la BD un sol cop. Per tant, no hi ha risc d'inconsistències.
Obtenció de dades	Si no hi ha una aplicació que obtingui les dades que volem, o bé s'ha de fer un programa a mida, o bé s'ha d'aprofitar la sortida d'un programa amb objectius similars, i fer els càlculs necessaris manualment.	Permeten obtenir qualsevol conjunt de dades, segons les necessitats, dels del seu propi entorn de treball, sense haver d'escriure, compilar i executar cap nou programa d'aplicació contra la BD.
Aïllament de dades	Les dades estan disperses i aïllades en diferents arxius, la qual cosa dificulta el desenvolupament de les aplicacions.	Totes les dades són en la mateixa BD, interconnectades, la qual cosa en facilita l'obtenció.
Integritat de dades	Els programes han d'implementar totes les restriccions sobre les dades, afegint el codi font corresponent. El manteniment és complicat quan la informació es conté en diferents fitxers utilitzats per diferents aplicacions.	La BD s'encarrega directament d'implementar les restriccions sobre les dades. Els programes no han d'incorporar codi font addicional per garantir-les.
Atomicitat	Alguns conjunts d'operacions sobre les dades s'han d'executar de manera indivisible (o tots o cap), independentment de les fallades que el sistema pugui presentar (com ara per un tall de subministrament elèctric). Però això és molt difícil de garantir amb un sistema d'informació basat en fitxers.	Les BD incorporen la tècnica de les transaccions per tal de garantir fàcilment l'execució atòmica d'una pluralitat de processos sobre les dades.
Accés concurrent	L'actualització simultània de dades d'un mateix fitxer per part de diferents usuaris o aplicacions en pot provocar fàcilment la inconsistència.	Amb la tècnica del bloqueig, les BD garanteixen automàticament la consistència de les dades, malgrat que més d'un usuari o més d'una aplicació les vulguin actualitzar simultàniament.
Seguretat	Habitualment, cada fitxer serveix per a un sol usuari o una sola aplicació (sobretot simultàniament), i ofereix una visió única del món real. Però no sempre tots els usuaris que utilitzen un fitxer haurien de tenir accés a totes les dades que conté.	Una BD pot ser compartida per molts usuaris de diferents tipus (fins i tot, simultàniament), els quals poden tenir diferents visions (vistes) del món real, en funció del seu perfil i dels permisos que s'hagin de concedir en cada cas.

Evidentment, les prestacions de les BD són molt superiors a les que proporcionen els sistemes de fitxers. Però això no vol dir que en alguns casos no sigui millor

utilitzar fitxers, com ara quan el volum de les dades a contenir és molt petit, o quan només hem de treballar amb una entitat instància i, per tant, no cal considerar interrelacions.

Algunes utilitzacions possibles dels fitxers en l'actualitat són les següents:

- Fitxers de configuració d'aplicacions.
- Fitxers de configuració de sistemes.
- Fitxers de registres d'esdeveniments (*logs*).

En casos com aquests, no compensaria carregar innecessàriament el sistema amb una BD (i amb el sistema gestor corresponent), ja que no s'aprofitarien els avantatges de les BD però, en canvi, empitjoraria el rendiment del sistema.

1.2.3 L'accés a les dades: tipologies

No serveix de res estructurar dades i emmagatzemar-les si després no s'hi pot d'accedir, per consultar-les, modificar-les o transmetre-les.

En general, hi ha dues maneres bàsiques d'accedir a les dades:

- L'**accés seqüencial** a un registre determinat, que implica l'accés previ a tots els registres anteriors.
- L'**accés directe** a un registre concret, que implica l'obtenció directa del registre.

A més, hi ha una altra classificació habitual de tipologies d'accessos:

- L'**accés per valor**, que permet obtenir el registre en funció del valor d'algun (o alguns) dels seus camps, sense considerar la posició que ocupa el registre.
- L'**accés per posició**, que obre l'accés a un registre que ocupa una posició determinada, sense considerar el contingut del registre.

Combinant les dues dicotomies anteriors, resulten quatre mètodes d'accés a les dades, tal com es mostra en la taula 1.3, que s'ajusten més a la realitat.

Taula 1.3. Mètodes d'accés a les dades

	P - per posició	**V - per valor**
S - seqüencial	SP	SV
D - directe	DP	DV

Examinem, doncs, les quatre tipologies d'accés a dades més freqüents:

- **SP (accés seqüencial per posició)**. Després d'haver accedit a un registre que es troba en una posició determinada, s'accedeix al registre que ocupa la posició immediatament posterior.
- **DP (accés directe per posició)**. S'obté directament un registre pel fet d'ocupar una posició determinada.
- **SV (accés seqüencial per valor)**. Després d'haver accedit a un registre que té un valor concret, s'accedeix al registre que ocupa la posició immediatament posterior, segons l'ordenació establerta a partir d'un camp determinat (o més). L'ordre serà creixent o decreixent, si es tracta d'un camp numèric, o alfabètic ascendent o descendent, si es tracta d'un camp de caràcters.
- **DV (accés directe per valor)**. S'obté directament un registre pel fet de tenir un valor determinat en un dels seus atributs (o més).

Exemples de tipus d'accés a les dades

Imaginem que disposem un fitxer en què s'emmagatzema informació relativa als alumnes d'un centre docent: DNI, nom, cognoms, data de naixement, adreça, telèfon, etc. A continuació, es dóna un exemple de cadascun del quatre mètodes d'accés estudiats.

SP (accés seqüencial per posició): la llista de tots els alumnes, sense establir cap ordenació.

DP (accés directe per posició): en l'àmbit de la programació, el cas més típic és el de les cerques dicotòmiques en vectors ordenats; en l'àmbit de les BD, aquest tipus d'accés es produeix en utilitzar un índex de tipus *hashing*.

SV (accés seqüencial per valor): la llista de tots els alumnes, seguint un ordre alfabètic ascendent, en primer lloc dels cognoms i després del nom.

DV (accés directe per valor): obtenció de les dades emmagatzemades en un registre corresponent a un alumne concret, que es digui, per exemple, Pere García Manent (és a dir, que el camp Nom conté el valor "Pere", i el camp Cognoms conté el valor "García Manent").

1.2.4 Les diferents visions de les dades

Un dels principals objectius de les BD és proporcionar, als usuaris, una **visió abstracta de les dades**. Amb aquesta finalitat, el sistema amaga als usuaris certs detalls relatius a l'emmagatzemament i manteniment de les dades, per facilitar-los la feina, d'una banda, però també per garantir certs aspectes en matèria de seguretat.

Perquè les BD resultin útils, han de ser mínimament eficients a l'hora de recuperar les dades. Per aquest motiu, els sistemes de BD tenen implementades, a baix nivell, estructures de dades bastant complexes.

S'utilitzen tres nivells d'abstracció -físic, lògic i de vistes- per tal d'amagar aquestes estructures complexes i simplificar, d'aquesta manera, la interacció dels usuaris amb el sistema.

1. **Nivell físic**: és el nivell d'abstracció més baix de tots els utilitzats.

    - Descriu com s'emmagatzemen realment les dades a baix nivell, especificant en detall les complexes estructures que es necessiten.
    - No és freqüent treballar a aquest nivell. Només es fa quan calen optimitzacions en l'estructuració de les dades a baix nivell.

2. **Nivell lògic**: és el nivell d'abstracció intermedi.

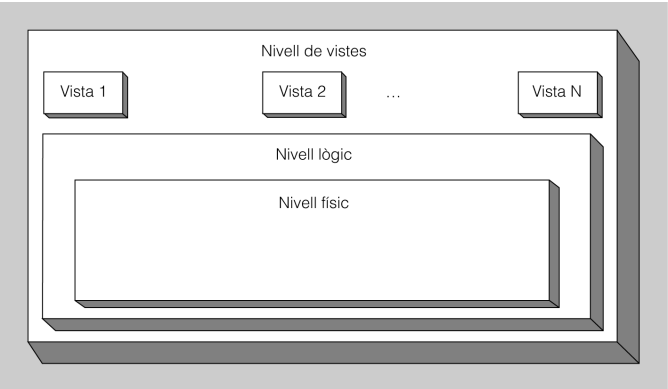
    - Descriu totes les dades emmagatzemades en la BD i les seves interrelacions, mitjançant un nombre no gaire elevat d'estructures força simples (típicament, taules).
    - La implementació d'aquestes estructures lògiques pot comportar la presència d'estructures molt més complexes a nivell físic. Però els usuaris del nivell lògic no s'han de preocupar d'aquesta complexitat. Ni tan sols necessiten conèixer-la.
    - Els administradors de BD treballen habitualment amb aquest nivell d'abstracció.

3. **Nivell de vistes**: és el nivell d'abstracció més alt.

    - La majoria dels usuaris no necessiten conèixer tota l'estructuració lògica de la BD amb què treballen. Tractant-se d'una BD gran, a més, conèixer tota la seva estructura pot comportar un esforç considerable.
    - D'altra banda, sovint, i per motius tant de seguretat com de privacitat, no resulta convenient que els usuaris tinguin accés a totes les dades, sinó solament a la part que estrictament necessiten per realitzar la seva feina.
    - Cada vista només descriu una part de la BD. L'establiment de vistes simplifica la interacció de l'usuari amb el sistema, el fa més segur, i proporciona més privacitat. Es poden establir diferents vistes, segons les necessitats, sobre la mateixa BD.

En la figura 1.6, es poden veure els diferents nivells d'abstracció utilitzats per facilitar la interacció dels usuaris amb les BD.

FIGURA 1.6. Nivells d'abstracció de dades



1.3 Els SGBD

Els **SGBD** són un tipus de programari que té com a finalitats la gestió i el control de les BD.

És interessant conèixer l'evolució d'aquest tipus de programari al llarg de la seva història, i els objectius que tots ells persegueixen amb més o menys encert. També cal destacar que hi ha nocions relatives tant a l'arquitectura dels SGBD com a les aplicacions que els fan servir. També s'ha de destacar que hi ha diferents tipologies d'usuaris i administradors de BD, i llenguatges que tots han d'utilitzar per comunicar-se amb els sistemes gestors.

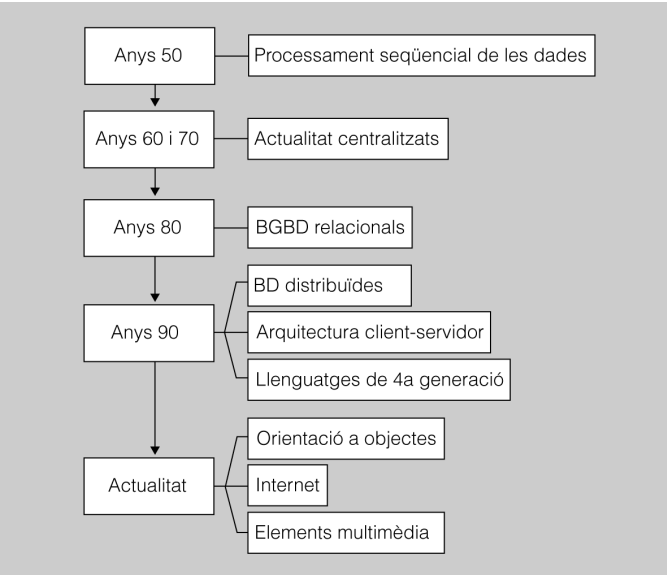
1.3.1 Evolució dels SGBD

Per tal d'entendre millor per què els SGBD són avui dia tal com els coneixem, convé repassar breument la seva història.

Igual que en altres àmbits de programari (com, per exemple, en el dels sistemes operatius), l'evolució dels SGBD ha estat, sovint, intrínsecament lligada a l'evolució del maquinari.

La figura 1.7 mostra un esquema de les etapes evolutives per les quals han passat els SGBD, en el qual se n'indiquen les principals característiques.

FIGURA 1.7. Evolució històrica dels SGBD



Anys cinquanta: processament seqüencial

Inicialment, l'únic maquinari disponible per emmagatzemar la informació consistia en paquets de cintes perforades i en cintes magnètiques. Aquests dispositius només es podien llegir de manera seqüencial i, per tant, el programari de l'època estava limitat per aquesta circumstància. Com que la grandària de les dades a processar era molt superior a la de la memòria principal de les computadores, els programes només podien realitzar processos per lots, de la manera següent:

- Obtenint les dades en un ordre determinat (des d'una o més cintes).
- Fent algun càlcul sobre les dades.
- Escrivint el resultat (en una nova cinta).

Exemple de processament seqüencial

Imaginem que una empresa necessita actualitzar els preus dels productes que ofereix: en primer lloc, caldria gravar els increments dels preus en targetes perforades.

A continuació, s'aniria llegint el paquet de les cintes perforades anteriors, sincronitzadament amb la cinta mestra que contingués totes les dades relatives als productes. Les targetes perforades haurien de respectar l'ordre dels registres del fitxer de productes contingut en la cinta.

Amb totes les dades relatives als productes contingudes en la cinta mestra, més els preus actualitzats en funció dels nous valors reflectits en les targetes perforades, es gravaria una nova cinta, que passaria a ser la nova cinta mestra dels productes de l'empresa.

Anys seixanta i setanta: sistemes centralitzats

Durant la major part de les dues dècades dels anys seixanta i setanta, els SGBD van tenir una estructura centralitzada, com corresponia als sistemes informàtics d'aleshores: un gran ordinador per a cada organització que se'l pogués costejar, i una xarxa de terminals no intel·ligents, sense capacitat pròpia per processar dades.

Inicialment, només es feien servir per gestionar processos per lots amb grans volums de dades. Posteriorment, amb l'aparició dels terminals, de vegades connectats mitjançant la línia telefònica, es van anar elaborant aplicacions transaccionals, per exemple per reservar i comprar bitllets en línies de transports, o per realitzar operacions financeres.

Els programes encara estaven molt lligats al nivell físic, i s'havien de modificar sempre que es feien canvis en el disseny de la BD, ja que aquests canvis implicaven, al seu torn, modificacions en l'estructura física de la BD. El personal que realitzava aquestes tasques havia d'estar altament qualificat.

Anys vuitanta: SGBD relacionals

Tot i que des del principi dels anys setanta ja s'havia definit el model relacional, i l'accés no procedimental a les dades organitzades seguint aquest model, no va ser fins als anys vuitanta quan van anar apareixent SGBD relacionals en el mercat.

La raó d'aquesta demora en l'ús dels sistemes relacionals va ser el pobre rendiment que oferien inicialment els productes relacionals en comparació amb les BD jeràrquiques i en xarxa. Però la innovació en el maquinari, primer amb els miniordinadors i posteriorment amb els microordinadors, va comportar un cert abaratiment de la informàtica i la seva extensió a moltes més organitzacions.

Fins aleshores, la feina dels programadors que treballaven amb BD prerelacionals havia estat massa feixuga, ja que, d'una banda, havien de codificar les seves consultes de manera procedimental, i d'una altra, havien d'estar pendents del seu rendiment i fer consideracions d'índole física a l'hora de codificar-les.

Però a causa de l'expansió de la informàtica que va tenir lloc durant la dècada que comentem, calia simplificar el desenvolupament de les aplicacions. Els SGBD ho van aconseguir, tot independitzant els programes dels aspectes físics de les dades.

SQL

El 1986, l'Institut Nacional Nordamericà de Normalització (American National Standards Institute, o ANSI, en anglès) va publicar les primeres normes que enunciaven la sintaxi i la semàntica de l'SQL.

A més, l'aparició del llenguatge de consulta estructurat (*structured query language*, o SQL, en anglès) i, sobretot, la seva estandardització a partir de l'any 1986 van facilitar enormement l'ús dels sistemes relacionals i, per tant, la seva implantació massiva.

Finalment, les BD relacionals van poder competir, fins i tot, en matèria de rendiment amb les jeràrquiques i amb les estructurades en xarxa, amb la qual cosa van acabar reemplaçant les seves competidores en la majoria dels casos.

Anys noranta: BD distribuïdes, arquitectures client/servidor, i llenguatges de quarta generació

Com ja sabem, els primers sistemes de BD eren centralitzats: totes les dades del sistema estaven emmagatzemades en un únic gran ordinador al qual es podia accedir des de diferents terminals. Però l'èxit gradual dels ordinadors personals (*personal computers*, o PC, en anglès), cada vegada més potents i amb preus més competitius, juntament amb el desenvolupament de les xarxes, va possibilitar la distribució d'una mateixa BD en diferents ordinadors (o nodes).

En funció del nombre de SGBD utilitzats, els sistemes distribuïts poden ser de dos tipus:

- **Homogenis**, si tots els nodes utilitzen el mateix SGBD. Les interaccions entre els diferents nodes són més senzilles. Però les actualitzacions del sistema gestor implicaran necessàriament a tots els nodes.
- **Heterogenis**, si cada node utilitza un SGBD diferent. Les interaccions entre els diferents nodes poden ser més complicades. Però hi haurà més flexibilitat a l'hora d'actualitzar el sistema gestor de cada node.

Els punts a favor dels sistemes distribuïts són fonamentalment dos:

- **Rendiment**. Si el sistema està ben dissenyat, la majoria de les operacions es realitzaran amb dades emmagatzemades localment. D'aquesta manera

les respostes seran més ràpides, disminuirà la despesa en comunicacions, i s'evitarà la dependència d'un node central col·lapsat.

- **Disponibilitat**. Els sistemes distribuïts són més resistents a les aturades que no pas els centralitzats. En un sistema centralitzat, l'aturada del node central atura tot el sistema. En canvi, en un sistema distribuït, si un dels nodes queda temporalment fora de servei per qualsevol eventualitat, la resta continuarà funcionant normalment, i podrà donar servei sempre que no es necessitin les dades emmagatzemades en el node aturat. Però, a més, segons quin esquema de disseny s'hagi seguit en fer la distribució, si les dades del node aturat estan duplicades en un altre, continuaran estant disponibles.

Però, evidentment, no tot són avantatges. Per exemple, en el cas dels sistemes heterogenis, sovint és necessari realitzar conversions de dades per permetre la comunicació dels diferent nodes entre ells. A més, en general, la comunicació és més complexa i, per tant, la quantitat d'errors i de problemes derivats d'aquest fet que s'han de controlar és molt més gran que en un sistema centralitzat.

Arquitectura client/servidor

Una arquitectura client/servidor es caracteritza pel fet de disposar d'un sistema en xarxa on hi ha uns ordinadors que actuen com a servidors de peticions i uns altres (els clients) que demanen serveis.

La tecnologia utilitzada habitualment en la distribució de BD és l'**arquitectura client/servidor** (coneguda també com a **arquitectura C/S**). Actualment, tots els SGBD comercials estan adaptats a aquesta realitat.

El funcionament dels sistemes basats en aquest tipus d'arquitectura és, esquemàticament, el següent: dos processos s'executen en un mateix sistema o en dos de diferents, de tal manera que un fa de client (o peticionari d'un servei), i l'altre de servidor (o proveïdor del servei demanat).

La classificació dels processos en les categories de client i de servidor és de tipus lògic (i no físic) i, per tant, cal tenir en compte alguns aspectes que deriven d'aquesta circumstància, com ara els següents:

- Un client pot demanar serveis a diversos servidors.
- Un servidor pot rebre peticions de molts clients.
- El client i el servidor poden residir en un mateix sistema.

4GL

Aquests són alguns entorns actuals de desenvolupament que utilitzen llenguatges de quarta generació: eDeveloper, de Magic Software, Oracle Developer, d'Oracle Corporation, SAS System, de SAS Institute Inc., etc.

Finalment, durant els anys noranta, la implantació arreu de les BD, fins i tot en petits sistemes personals, va motivar l'aparició dels anomenats **llenguatges de quarta generació** (*fourth generation languages*, o 4GL, en anglès), els quals es continuen utilitzant en l'actualitat.

Es tracta de llenguatges molt senzills, però al mateix temps molt potents, especialitzats en el desenvolupament d'aplicacions centrades en l'accés a BD. Ofereixen moltes facilitats per definir, generalment de manera visual, finestres des de les quals es poden consultar, introduir, modificar o esborrar dades, fins i tot en entorns client/servidor.

Tendències actuals: orientació a objectes, Internet, i elements multimèdia

Actualment, els SGBD relacionals acaparen el mercat. Han evolucionat de tal manera que ja no tenen competidors en rendiment, fiabilitat o seguretat. D'altra banda, ja no necessiten habitualment tasques de manteniment planificades que en comportin l'aturada periòdica. Per tant, la disponibilitat que ofereixen és molt elevada, ja que s'acosta a les vint-i-quatre hores de tots els dies de l'any.

Però, al mateix temps, tots estan immersos en un complex procés de transformació per tal d'adaptar-se a les innovacions tecnològiques de més èxit:

1. Les tecnologies multimèdia. Els tipus de dades que tradicionalment admetien els SGBD ara es veuen incrementats per altres de nous que permeten emmagatzemar imatges i sons.

> **Exemple d'incorporació de tecnologies multimèdia en un SGBD**
>
> D'aquesta manera, per exemple, la taula que emmagatzemi les dades personals dels empleats d'una empresa determinada podrà contenir la foto de cadascun, la qual cosa pot resultar especialment interessant si l'organització disposa d'un entorn virtual de treball (de tipus intranet, per exemple) en què els correus electrònics incorporin la foto del remitent, a fi de permetre la identificació visual.

2. L'orientació a objectes. Aquest paradigma de la programació ha acabat influint en l'orientació de molts SGBD, que segueixen alguna de les dues línies esbossades a continuació:

- SGBD relacionals amb objectes, els quals admeten tipus abstractes de dades (TAD), a més dels tipus tradicionals.
- SGBD orientats a objectes, que estructuren les dades en classes, les quals comprenen tant les dades (atributs) com les operacions sobre elles (mètodes). Les classes s'estructuren jeràrquicament, de tal manera que les de nivells inferiors (subclasses) hereten les propietats de les de nivells superiors (superclasses).

3. Internet. Avui en dia, la majoria de SGBD professionals incorporen els recursos necessaris per donar suport als servidors de pàgines web dinàmiques (és a dir, amb accés a les dades contingudes en un SGBD allotjat en el servidor web corresponent).

Una altra línia d'innovació que segueixen alguns SGBD és el treball amb els anomenats **magatzems de dades** (*data warehouse*, en anglès). Aquests magatzems consisteixen en rèpliques elaborades de les dades generades pel funcionament quotidià de l'organització o l'empresa de què es tracti, durant un cert període de temps per tal de realitzar anàlisis estratègiques d'índole financera, de mercats, etc.

El **llenguatge de marques extensible** (*extensible markup language*, o XML, en anglès) també influeix en el món dels SGBD, tot i que inicialment no es va concebre com una tecnologia per donar servei a les BD, sinó per estructurar documents molt grans. Però aquesta capacitat per emmagatzemar les dades de què es compon un document el fan susceptible de ser utilitzat, també, en l'àmbit de les BD.

Hi ha dues maneres d'incorporar la tecnologia XML en els SGBD:

- Els **SGBD relacionals amb suport per a XML**, que en el fons continuen essent relacionals i que, per tant, emmagatzemen tota la informació de manera tabular. La seva utilitat principal és que les dades que retornen les consultes sobre la BD poden estar expressades en format XML, si així es demana al sistema gestor.
- Els **sistemes gestors per a BD natives XML**, que no són en realitat relacionals, sinó més aviat jeràrquics, i que no solament estan en condicions d'oferir els resultats de les consultes en format XML, sinó que també emmagatzemen la informació en documents XML. El llenguatge estàndard de consultes per a aquest tipus de SGBD s'anomena XQuery.

XQuery

L'XQuery és un llenguatge de consultes dissenyat per consultar col·leccions de dades XML, creat pel World Wide Web Consortium (conegut abreujadament com a W3C). El W3C és un consorci internacional que produeix estàndards per a la World Wide Web (coneguda abreujadament com a WWW).

La utilització de dades estructurades mitjançant l'estàndard XML resulta especialment interessant en l'intercanvi d'informació entre sistemes basats en plataformes poc compatibles entre elles.

1.3.2 Objectius i funcionalitats dels SGBD

Tots els SGBD del mercat volen assolir una sèrie d'objectius i oferir una sèrie de funcionalitats, amb més o menys encert, que actualment es consideren indispensables per al bon funcionament de qualsevol sistema d'informació:

- Possibilitar les consultes no predefinides de qualsevol complexitat.
- Garantir la independència física i la independència lògica de les dades.
- Evitar o solucionar els problemes derivats de la redundància.
- Protegir la integritat de les dades.
- Permetre la concurrència d'usuaris.
- Contribuir a la seguretat de les dades.

Possibilitar les consultes no predefinides de qualsevol complexitat

Els usuaris autoritzats d'un SGBD han de poder plantejar directament al sistema qualsevol consulta sobre les dades emmagatzemades, de la complexitat que sigui necessària, tot respectant, això sí, les regles sintàctiques perquè la sentència sigui correcta.

A continuació, el SGBD ha de ser capaç de respondre ell mateix a la consulta formulada, sense que sigui necessari recórrer a cap aplicació externa.

Quan no existien ni les BD ni els sistemes gestors, per tal d'obtenir un subconjunt de les dades emmagatzemades en un fitxer, hi havia dues possibilitats:

- Llançar un llistat seqüencial de totes les dades i, a continuació, seleccionar manualment les que interessessin.
- Escriure el codi font d'un programa específic per resoldre la consulta en qüestió, compilar-lo i executar-lo.

Els SGBD actuals, en canvi, estan perfectament capacitats per interpretar directament les consultes, expressades habitualment com a sentències formulades en el llenguatge de consulta SQL.

Evidentment, això no vol dir que no es puguin continuar produint programes que incorporin consultes mitjançant les quals accedeixen a BD. De fet, aquesta és l'opció més encertada i còmoda si es tracta de consultes que s'han de repetir al llarg del temps.

Garantir la independència física i la independència lògica de les dades

Cal garantir la màxima independència física de les dades respecte als processos usuaris, en general (és a dir, tant pel que fa a les consultes interpretades pel SGBD com als programes externs que accedeixen a la BD), de tal manera que es puguin dur a terme tot tipus de canvis tecnològics d'índole física per millorar el rendiment (com ara afegir o treure un índex determinat), sense que això impliqui haver de modificar ni les consultes a la BD ni les aplicacions que hi accedeixen.

De manera similar, també és desitjable la independència lògica de les dades, la qual implica que les modificacions en la descripció lògica de la BD (per exemple, afegir un nou atribut o suprimir-ne un altre) no han d'impedir l'execució normal dels processos usuaris no afectats per aquelles.

I, pel que fa a la independència lògica de les dades, fins i tot pot interessar (i, de fet, aquesta és una opció freqüent) que convisquin diferents visions lògiques d'una mateixa BD, en funció de les característiques concretes dels diferents usuaris o grups d'usuaris.

Evitar o solucionar els problemes derivats de la redundància

Tradicionalment, la repetició de les dades s'ha considerat una cosa negativa, ja que comporta un cost d'emmagatzematge innecessari. Avui en dia, però, aquesta característica, tot i ser certa, gairebé no es té en compte, a causa de l'abaratiment dels discos durs i de l'augment de la seva capacitat i rendiment.

Però hi ha un altre aspecte a considerar que no ha perdut vigència, i és el fet que la repetició de les dades és perillosa, ja que quan s'actualitzen poden perdre la integritat. Quan es modifica el valor d'una dada que està repetida, s'han de modificar simultàniament els valors de les seves repeticions perquè es mantingui la coherència entre totes.

Són dades íntegres les que es mantenen senceres i correctes.

La **redundància** consisteix en la repetició indesitjada de les dades, que incrementa els riscos de pèrdua d'integritat d'aquestes quan s'actualitzen.

Malgrat tot, els SGBD han de permetre al dissenyador de BD la definició de dades repetides, ja que de vegades (sobretot en matèria de fiabilitat, de disponibilitat o de costos de comunicació) és útil mantenir certes rèpliques de les dades.

Ara bé, en tots aquests casos, l'objectiu de l'SGBD ha de ser garantir l'actualització correcta de totes les dades allà on estiguin duplicades, de manera automàtica (és a dir, sense que l'usuari del SGBD s'hagi d'encarregar de res).

Un altre tipus de duplicitat admissible és la que constitueixen les anomenades **dades derivades**. Es tracta de dades emmagatzemades en la BD, que en realitat són el resultat de càlculs fets amb altres dades també presents en la mateixa BD.

Les dades derivades també poden ser acceptables, tot i que comporten una repetició evident d'algunes dades, si permeten fer consultes de caire global molt ràpidament, sense haver d'accedir a tots els registres implicats.

Però també aquí, el SGBD s'ha d'encarregar d'actualitzar degudament les dades derivades en funció dels canvis soferts per les dades primitives de les quals depenen.

Protegir la integritat de les dades

A més de la redundància, hi ha molts altres motius que poden fer malbé la consistència de les dades, com ara els següents:

- Els errors humans.
- Les deficiències en la implementació dels algoritmes de les aplicacions.
- Les avaries dels suports físics d'emmagatzematge.
- Les transaccions incompletes com a conseqüència de les interrupcions del subministrament elèctric.

Els SGBD han de protegir la integritat de les dades en tots aquests casos. Per a això disposen, d'una banda, de les regles d'integritat, també anomenades *restriccions*, i d'una altra, dels sistemes de restauració basats en còpies de seguretat.

Mitjançant les **regles d'integritat**, el sistema valida automàticament certes condicions en produir-se una actualització de dades, i l'autoritza si les compleix, o denega el permís en cas contrari.

Exemple de restricció del model

Un SGBD relacional mai no acceptarà que una taula emmagatzemi registres amb una clau primària idèntica, perquè aleshores la clau no serviria per identificar inequívocament els registres entre ells.

Les regles d'integritat poden ser de dos tipus:

- **Restriccions del model**. Són regles inherents al model de dades que utilitza el SGBD (com ara el model relacional). El sistema les incorpora predefinides, i sempre s'acompleixen.
- **Restriccions de l'usuari**. Són regles definides pels usuaris (dissenyadors i administradors, fonamentalment) de la BD, que no incorpora *a priori* el SGBD, i que serveixen per modelitzar aspectes específics del món real.

Exemple de restricció de l'usuari

En una taula que emmagatzema els alumnes d'un centre docent, es vol evitar que hi hagin alumnes matriculats en cicles formatius de grau superior que fossin menors d'edat, ja que la normativa vigent no ho admet. Aleshores, cal establir una restricció en aquest sentit per calcular si la diferència entre la data del sistema en el moment de la matrícula i la data de naixement de cada alumne és igual o superior als 18 anys. En aquest cas, el sistema permetria la matrícula, i en cas contrari la prohibiria.

Els SGBD també proporcionen eines per realitzar periòdicament **còpies de seguretat de les dades** (o *backups*, en anglès) que permeten restaurar les dades malmeses i retornar-les a un estat consistent.

Ara bé, els valors restaurats es correspondran amb els que hi havia en el moment de realitzar la còpia de seguretat, abans de l'incident que origina la restauració, i per tant mai no estaran del tot actualitzats, encara que siguin correctes.

Permetre la concurrència d'usuaris

La concurrència d'usuaris en una BD consisteix en l'accés simultani a la BD per part de més d'un usuari.

Un objectiu fonamental de tot SGBD és possibilitar de manera eficient l'accés simultani a la BD per part de molts usuaris. A més, aquesta necessitat ja no està circumscrita només a grans companyies o a administracions públiques amb molts usuaris, sinó que cada cop és més freqüent, a causa de l'expansió d'Internet i l'èxit de les pàgines dinàmiques, allotjades en servidors web que han d'incorporar un SGBD.

Hem de considerar dues tipologies d'accessos concurrents, amb problemàtiques ben diferenciades:

- **Accessos de consulta de dades**. Poden provocar problemes de rendiment, derivats sobretot de les limitacions dels suports físics disponibles (per exemple, si la lentitud de gir del disc dur que conté la BD, o del moviment del braç que porta incorporat el capçal, no permeten atendre degudament totes les peticions d'accés que rep el sistema).

- **Accessos de modificació de dades**. Les peticions simultànies d'actualització d'unes mateixes dades poden originar problemes d'interferència que tinguin com a conseqüència l'obtenció de dades errònies i la pèrdua d'integritat de la BD.

Per tractar correctament els problemes derivats de la concurrència d'usuaris, els SGBD fan servir fonamentalment dues tècniques: les **transaccions** i els **bloquejos**.

Una **transacció** consisteix en un conjunt d'operacions simples que s'han d'executar com una unitat.

Les operacions incloses dins d'una transacció mai no s'han d'executar parcialment. Si per algun motiu no s'han pogut executar totes correctament, el SGBD ha de desfer automàticament els canvis produïts fins aleshores. D'aquesta manera, es podrà tornar a llançar l'execució de la mateixa transacció, sense haver de fer cap modificació en el codi de les diferents operacions que inclogui.

COMMIT i ROLLBACK

La instrucció COMMIT indica, al SGBD, que un conjunt d'operacions determinat s'ha d'executar de manera transaccional. L'operació ROLLBACK desfà els canvis produïts en cas que les operacions d'una transacció s'hagin executat parcialment.

Exemple de transacció

Imaginem que el conveni col·lectiu d'una empresa determina que els salaris dels treballadors s'han d'apujar el mes de gener un 3%. La millor opció consistirà a actualitzar la taula d'empleats i, més concretament, els valors del camp que recull els sous dels empleats, mitjançant una consulta d'actualització que modifiqui totes aquestes dades i les incrementi en un 3%.

Si volem garantir que l'actualització del salaris no quedi a mitges, haurem de fer que totes les instruccions que impliqui aquest procés es comportin de manera transaccional, és a dir, que s'executin totes o bé que no se n'executi cap.

Però també es pot donar la situació en què diferents transaccions vulguin accedir a la BD simultàniament. En aquests casos, encara que cada transacció, individualment considerada, sigui correcta, no es podria garantir la consistència de les dades si no fos per l'ús de la tècnica del bloqueig.

Un **bloqueig** consisteix a impedir l'accés a determinades dades durant el temps en què siguin utilitzades per una transacció. Així s'aconsegueix que les transaccions s'executin com si estiguessin aïllades, de tal manera que no es produeixen interferències entre elles.

Exemple de bloqueig

Imaginem que el departament de recursos humans de l'empresa de l'exemple anterior disposa d'una aplicació que li proporciona certes dades de caire estadístic sobre de les remuneracions dels empleats, com ara el salari mitjà.

Si es vol executar de manera concurrent la transacció descrita, que incrementa els sous en un 3%, i al mateix temps es llança l'execució de l'aplicació que calcula el salari mitjà de tots els empleats de l'empresa, el resultat obtingut per aquest programa probablement serà erroni.

Per tal de garantir la correcció del càlcul, s'haurà de bloquejar una de les dues transaccions mentre l'altra s'executa.

Si es bloqueja l'actualització de dades, el salari mitjà estarà calculat a partir dels sous antics (és a dir, abans de ser actualitzats).

En canvi, si es bloqueja el programari estadístic, en primer lloc s'actualitzaran totes les dades, i després es calcularan els resultats a partir dels nous valors del camp que emmagatzemi el salari.

Els bloquejos provoquen esperes i retencions, i per això les noves versions dels diferents SGBD del mercat s'esforcen a minimitzar aquests efectes negatius.

Contribuir a la seguretat de les dades

Exemple de dades confidencials

Resulta evident la necessitat de restringir l'accés als secrets militars o, fins i tot, comercials (com ara les dades comptables). Però també s'ha de respectar la privacitat, fins i tot per imperatiu legal, en altres vessants aparentment més modestos, però en realitat no menys importants, com són les dades personals.

L'expressió **seguretat de les dades** fa referència a la seva confidencialitat. Sovint, l'accés a les dades no ha de ser lliure o, com a mínim, no ho ha de ser totalment.
Els **SGBD** han de permetre definir autoritzacions d'accés a les BD, tot establint permisos diferents en funció de les característiques de l'usuari o del grup d'usuaris.

Actualment, els SGBD permeten definir autoritzacions a diferents nivells:

- Nivell global de tota la BD
- Nivell d'entitat
- Nivell d'atribut
- Nivell de tipus d'operació

Exemples de drets d'accés

Els usuaris del departament de comptabilitat potser no haurien de tenir accés a l'entitat que emmagatzema les dades personals dels empleats de l'empresa, a diferència de l'usuari que ostenta el càrrec de director general, que les podrà consultar per tal d'optimitzar la ubicació dels treballadors el l'organigrama en funció del respectiu perfil, o també dels usuaris del departament de recursos humans, que haurien de poder fins i tot modificar-les.

En general, els usuaris no haurien de tenir accés als atributs que emmagatzemen l'adreça particular dels empleats, a no ser que es tracti del personal de recepció, si resulta que és l'encarregat d'enviar-los certa correspondència a domicili (com ara les nòmines o els certificats de retencions d'IRPF), i per tant hauria, si més no, de poder consultar-los.

Aquests mecanismes de seguretat requereixen que cada usuari es pugui identificar. El més freqüent és utilitzar un nom d'usuari i una contrasenya associada per a cada usuari. Però també hi ha qui fa servir mecanismes addicionals, com ara targetes magnètiques o de reconeixement de la veu. Actualment, s'investiga en altres direccions com, per exemple, en la identificació personal mitjançant el reconeixement de les empremtes dactilars.

Un altre aspecte a tenir en compte en parlar de la seguretat de les dades és la seva encriptació. Molts SGBD ofereixen aquesta possibilitat, en alguna mesura.

Les **tècniques d'encriptació** permeten emmagatzemar la informació utilitzant codis secrets que no permeten accedir a les dades a persones no autoritzades i que, per tant, no disposen dels codis esmentats.

L'encriptació pot fer disminuir el rendiment en l'accés a les dades, ja que comporta la utilització d'algoritmes addicionals en les operacions de consulta. Per això se n'ha de dosificar l'ús. Ara bé, sempre que sigui possible, és convenient encriptar les contrasenyes.

1.3.3 Llenguatges de SGBD

La comunicació entre els SGBD i els seus usuaris s'ha de realitzar mitjançant algun tipus de llenguatge. Els llenguatges de BD es poden classificar en dues grans tipologies segons la finalitat:

1. **Llenguatges de definició de dades** (*data definition languages*, en anglès, o DDL). Estan especialitzats en la definició de l'estructura de les BD mitjançant l'especificació d'esquemes.
2. **Llenguatges de manipulació de dades** (*data management languages*, en anglès, o DML). Possibiliten la consulta, modificació i eliminació, de les dades emmagatzemades, i també la inserció de noves informacions. Podem considerar l'existència de dos subtipus, bàsicament:
    - **Procedimentals**. Requereixen especificar no solament les dades que es necessiten, sinó també els procediments que s'han de seguir per obtenir-les. S'utilitzaven de manera exclusiva abans de l'arribada del model relacional. Actualment, es continuen utilitzant, però només quan cal optimitzar algun procés que no rendeix prou, pel fet d'estar implementat de manera declarativa. Són més eficients que els declaratius, però més complicats, ja que exigeixen tenir certs coneixements sobre el funcionament intern del SGBD utilitzat.
    - **Declaratius**. Només requereixen especificar quines dades es necessiten, sense que calgui especificar com s'han d'obtenir. Són més senzills d'aprendre que els procedimentals, però també menys eficients.

El llenguatge més utilitzat per interaccionar amb els SGBD relacionals és l'**SQL.**

L'SQL engloba les dues tipologies de llenguatges de BD descrites. Les seves operacions es poden classificar, doncs, en un dels dos tipus esmentats (DDL i DML) amb finalitats pedagògiques, però en realitat totes formen part d'un únic llenguatge.

Exemples d'operacions DDL i DML del llenguatge SQL

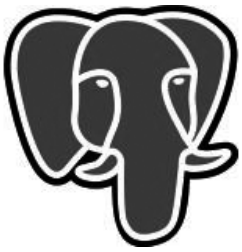
Com a instruccions de tipus DML, podem esmentar SELECT (per fer consultes), i també INSERT, UPDATE i DELETE (per realitzar el manteniment de les dades).

I com a instruccions de tipus DDL, podem considerar CREATE TABLE (que ens permet definir les taules, les seves columnes i les restriccions que calgui).

En relació al component DML de l'SQL, cal dir que és fonamentalment declaratiu, tot i que té possibilitats procedimentals, que es poden explotar en diferents SGBD:

PostgreSQL és un SGBD distribuït amb llicència BSD (Berkeley Software Distribution)

- **PL/SQL**, llenguatge procedimental per treballar amb els SGBD creats per Oracle.
- **PL/PgSQL**, similar al PL/SQL d'Oracle, però dissenyat per treballar amb PostgreSQL.



Logotip de PostgreSQL

Cal dir que, a més del respectiu llenguatge nadiu de BD (habitualment, SQL), els SGBD ofereixen, des de ja fa molt de temps, dues possibilitats més per incrementar la productivitat en el treball amb BD, que són els **llenguatges de quarta generació** i les interfícies visuals, sovint proporcionades dins de l'entorn d'una sola eina.

Sovint, l'accés a les BD també es fa des d'aplicacions externes al SGBD, escrites en llenguatges de programació (com per exemple C, Java, etc.), els quals normalment no incorporen instruccions pròpies que permetin la connexió amb BD.

Per respondre a aquesta necessitat, hi ha dues opcions:

1. Realitzar, dins del programa, crides a diferents funcions que són en llibreries que implementen estàndards de connectivitat de BD amb programes escrits en certs llenguatges, com ara els següents:
   - ODBC (*open data base connectivity*), sistema creat per Microsoft i compatible amb molts sistemes com, per exemple, Informix, MS Access, MySQL, Oracle, PostgreSQL, SQL Server, etc.
   - JDBC (*Java data base connectivity*), per realitzar operacions amb BD des d'aplicacions escrites en Java.
2. Hostatjar les sentències del llenguatge de BD que siguin necessàries, dins d'un programa amfitrió escrit en el llenguatge de programació utilitzat. És imprescindible que el compilador utilitzat accepti la introducció de sentències escrites en el llenguatge de BD utilitzat (que normalment serà l'SQL).

1.3.4 Usuaris i administradors

Les persones que treballen amb SGBD es poden classificar com a usuaris en sentit estricte, els quals simplement interactuen amb el sistema (tot i que de diferents

maneres i amb diferents finalitats), o bé com a administradors, si a més realitzen tasques de gestió i control.

Usuaris d'SGBD

Podem diferenciar tres categories d'usuaris de SGBD en funció de la manera en què interactuen amb el sistema: externs, sofisticats i programadors d'aplicacions.

1. Usuaris externs. Són usuaris no sofisticats, que no interactuen directament amb el sistema, sinó mitjançant alguna aplicació informàtica desenvolupada prèviament per altres persones amb aquesta finalitat.

Exemple d'usuari extern

Qualsevol persona assumeix aquest rol quan treu diners d'un caixer automàtic, ja que accedeix a la BD de l'entitat financera identificant-se mitjançant una targeta magnètica i un número d'identificació personal secret (*personal identification number*, en anglès, o PIN).

Una vegada autoritzada a entrar en el sistema, podrà realitzar diferents operacions de consulta o, fins i tot, d'actualització. En el cas plantejat, després de treure diners, el saldo del compte corrent associat a la targeta patirà el decrement corresponent.

2. Usuaris sofisticats. Interactuen directament amb el sistema, sense utilitzar les interfícies proporcionades per programes intermediaris. Formulen les consultes en un llenguatge de BD (normalment, SQL), des de dins de l'entorn que el SGBD posa a la seva disposició. Tradicionalment, aquest entorn ha estat una consola en què es podien escriure les consultes, però cada vegada són més freqüents entorns que permeten construir les consultes de mode visual, com autèntiques eines CASE.

3. Programadors d'aplicacions. Són professionals informàtics que creen els programes que accedeixen als SGBD i que, posteriorment, són utilitzats pels usuaris que hem anomenat *externs*. Aquestes aplicacions es poden desenvolupar mitjançant diferents llenguatges de programació i eines externes al SGBD. Però molts SGBD comercials també inclouen entorns propis de desenvolupament i llenguatges de quarta generació que faciliten enormement la generació de formularis i informes que permeten visualitzar i modificar les dades.

Eines CASE

Les eines CASE (acrònim de *computer aided software engineering*, o enginyeria del programari assistida per ordinador) són aplicacions informàtiques destinades a augmentar la productivitat en el desenvolupament de programari reduint el cost del desenvolupament en termes de temps i de diners.

Administradors d'SGBD

Els **administradors** són uns usuaris especials que realitzen tasques d'administració i control centralitzat de les dades, i gestionen els permisos d'accés concedits als diferents usuaris i grups d'usuaris, per tal de garantir el funcionament correcte de la BD.

Els administradors han d'actuar, evidentment, per solucionar les eventuals aturades del sistema, però la seva responsabilitat fonamental consisteix, justament, a evitar que es produeixin incidents.

La feina dels administradors no és fàcil, tot i que els SGBD incorporen cada vegada més eines per facilitar-la, i en la majoria dels casos amb interfície visual. Es tracta, per exemple, d'eines de monitoratge de rendiment, d'eines de monitoratge de seguretat, de verificadors de consistència entre índexs i dades, de gestors de còpies de seguretat, etc.

Una llista no exhaustiva de les tasques dels administradors podria ser la següent:

- Crear i administrar els esquemes de la BD.
- Administrar la seguretat: autoritzacions d'accés, restriccions, etc.
- Realitzar còpies de seguretat periòdiques.
- Controlar l'espai de disc disponible.
- Vigilar la integritat de les dades.
- Observar l'evolució del rendiment del sistema i determinar quins processos consumeixen més recursos.
- Assessorar els programadors i els usuaris sobre la utilització de la BD.
- Fer canvis en el disseny físic per millorar el rendiment.
- Resoldre emergències.

1.3.5 Components funcionals dels SGBD

El programari que conforma els SGBD es divideix en diferents **mòduls**, encarregats de les respectives funcionalitats que ha de garantir el sistema.

El components funcionals dels SGBD més importants són el gestor d'emmagatzemament i el processador de consultes.

Gestor d'emmagatzemament

Les BD corporatives tenen enormes requeriments d'espai d'emmagatzematge en disc. Les dades es transfereixen entre el disc en què estan emmagatzemades i la memòria principal de l'ordinador només quan és necessari. I, com que la transferència de dades cap al disc o des del disc és lenta en comparació amb la velocitat de la unitat central de processament, és molt important que el SGBD estructuri les dades de tal manera que se'n minimitzi la necessitat de transferència entre el disc i la memòria principal.

El gestor d'emmagatzemament proporciona la interfície entre les dades, considerades a baix nivell, i les consultes i els programes que accedeixen a la BD. Les dades s'emmagatzemen en disc fent servir el sistema d'arxius que proporciona

el sistema operatiu utilitzat. I el gestor tradueix les instruccions DML a ordres comprensibles pel sistemes d'arxius a baix nivell.

Els components del gestor d'emmagatzemament són els següents:

- **Gestor d'autoritzacions i d'integritat**. Comprova que se satisfacin tant les restriccions d'integritat com les autoritzacions dels usuaris per accedir a les dades.
- **Gestor de transaccions**. Assegura que la BD es mantingui en un estat de consistència malgrat les fallades del sistema, i també que les transaccions concurrents no s'interfereixin entre elles.
- **Gestor d'arxius**. Gestiona la reserva d'espai d'emmagatzemament en disc i les estructures de dades utilitzades per representar la informació emmagatzemada en disc.
- **Gestor de memòria intermèdia**. Transfereix les dades des del disc a la memòria principal, i decideix quines dades s'han de tractar en memòria cau. Permet al sistema tractar amb dades de grandària molt superior a la de la memòria principal.

D'altra banda, el gestor d'emmagatzemament utilitza certes estructures de dades que formen part de la implementació física del sistema:

- **Arxius de dades**. Emmagatzemen la BD pròpiament considerada.
- **Diccionari de dades**. Emmagatzema les metadades relatives a tota l'estructura de la BD.
- **Índexs**. Proporcionen un accés ràpid a certes dades en funció dels seus valors.

Processador de consultes

El processador de consultes ajuda el SGBD a simplificar l'accés a les dades. Les vistes a alt nivell contribueixen a assolir aquest objectiu, ja que eviten que els usuaris hagin de conèixer detalls de la implementació física del sistema. Però això no treu que el sistema no hagi de perseguir l'eficàcia en el processament de les consultes i de les actualitzacions de dades. De fet, el sistema ha de traduir les instruccions escrites, a nivell lògic, en un llenguatge no procedimental (típicament, SQL), a una seqüència d'operacions a nivell físic, amb uns mínims d'eficiència.

Els components del processador de consultes són els següents:

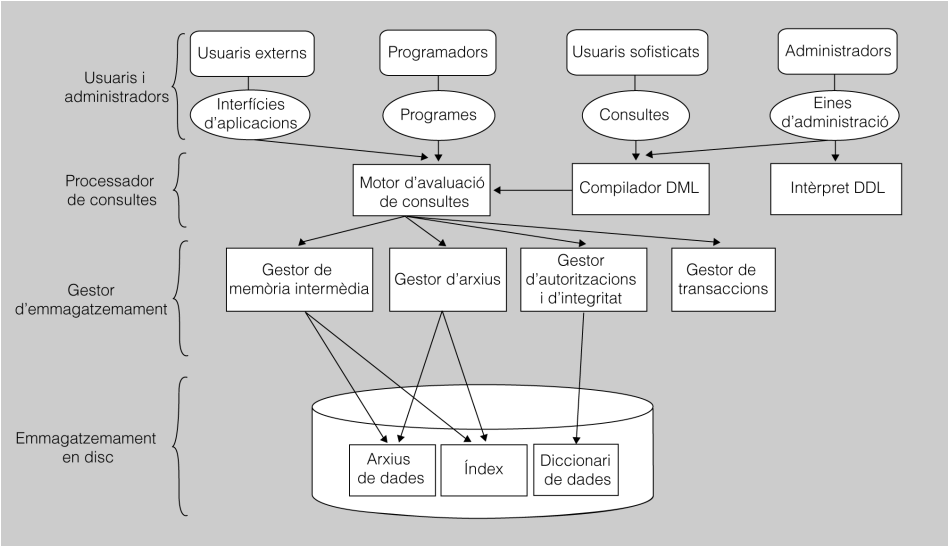
- **Intèrpret DDL**. Interpreta les instruccions de tipus DDL i registra les definicions en el diccionari de dades.
- **Compilador DML**. Tradueix les instruccions DML formulades en un llenguatge de consultes (normalment, SQL) a una sèrie d'instruccions a baix

nivell que pot interpretar el motor d'avaluació de consultes. En realitzar la traducció esmentada, un bon compilador DML també s'encarregarà de fer una optimització de consultes triant, entre totes les alternatives, la de menor cost.

- **Motor d'avaluació de consultes**. Executa les instruccions de baix nivell generades pel compilador DML.

La figura 1.8 mostra tots aquests components i les connexions entre ells.

FIGURA 1.8. Components funcionals dels SGBD



1.3.6 Diccionari de dades

Un diccionari de dades d'una base de dades és el conjunt de metadades que proporcionen informació sobre el contingut i l'organització de la base de dades.

> Un **diccionari de dades**, segons *IBM Dictionary of Computing* es pot definir com un "repositori centralitzat d'informació sobre dades com ara el significat, relacions amb altres dades, orígen, ús i format."

De vegades el concepte de diccionari de dades o metadades també és conegut amb el nom de catàleg del sistema o, també, com a repositori de metadades.

DBA

DBA és l'acrònim de *Data Base Administrator* o Administrador de Bases de Dades, que és la persona encarregada de gestionar les BD, dintre del SGBD.

Els diferents SGBD específics implementen de diverses formes el diccionari de dades, de forma que cada programari implementa amb un conjunt de taules i ofereix un conjunt de vistes a diferents tipus d'usuaris del sistema. Així doncs, normalment, un usuari amb privilegis d'administrador del sistema (DBA) pot visualitzar més informació i de tipus més específic que un usuari sense aquests privilegis.

Els elements que es troben habitualment a un diccionari de dades inclouen:

- Definicions de l'esquema de la base de dades.
- Descripcions detallades de taules i camps, així com de tots els objectes de la base de dades (vistes, clusters, índexos, sinònims, funcions i procediments, triggers, etc.).
- Restriccions d'integritat referencial.
- Informació de control d'accés, com ara noms d'usuari, rols, i privilegis.
- Paràmetres d'ubicació de l'emmagatzemament.
- Estadístiques d'ús de la base de dades.

Tot aquest conjunt d'informació, s'emmagatzema en centenars de taules d'informació. Tot i que hi ha organismes que intenten estandarditzar l'organització d'aquesta meta-informació, la realitat és que cada distribuïdor de programari específic (cada SGBD) ofereix la seva pròpia estructura.

Un administrador del sistema o DBA haurà de conèixer com accedir i consultar tota aquesta informació consultant la guia de referència del sistema amb què treballi.

El diccionari de dades en Oracle

En Oracle és l'usuari `SYS` el propietari del diccionari de dades i l'únic que pot fer actualitzacions sobre la informació que conté. Tot i que, alterar o manipular el diccionari de dades és una operació d'administració que cal fer amb moltes precaucions, doncs pot provocar danys permanents.

En Oracle es creen tres tipus de vistes diferents, que permeten consultar informació sobre diferents tipus de dades. Així les vistes poden tenir un d'aquests prefixos:

- `USER`: Per a visualitzar informació sobre els objectes propietat de l'usuari.
- `ALL`: Per a consultar informació sobre els objectes on té accés l'usuari.
- `DBA`: Que dóna accés a tots els objectes del sistema, per a poder ser controlats i gestionats.

Així doncs, per exemple, es poden consultar el conjunt d'objectes a què es té accés des d'un usuari donat d'Oracle amb la següent sentència:

```
SELECT owner, object_name, object_type FROM ALL_OBJECTS;
```

Diccionari de dades en MySQL

En MySql, en canvi, és la vista `INFORMATION_SCHEMA` qui proporciona informació sobre les bases de dades que s'emmagatzemen en el sistema.

Per a obtenir certa informació sobre una base de dades concreta anomenada `db_name` en un SGBD MySQL podríem executar una sentència com ara:

```
SELECT table_name, table_type, engine FROM information_schema.tables WHERE
table_schema = 'db_name';
```

2. Models de bases de dades

Les bases de dades (BD) representen informàticament la part del món real del nostre interès, que prèviament hem conceptualitzat, mitjançant uns processos d'observació i d'abstracció.

Per tant, podem afirmar que les BD són models de la realitat. Però no hem de confondre aquesta característica de les BD amb el que s'entén per *model de dades* (o *model de base de dades*). L'estructura concreta de cada BD està construïda a partir del model de dades respectiu, triat pel dissenyador en funció de les necessitats i de les eines disponibles.

Els **models de dades** són uns conjunts d'eines lògiques per descriure les dades, les seves interrelacions, el seu significat i les restriccions a aplicar per tal de garantir-ne la coherència.

Tots els models de BD, en general, proporcionen tres tipus d'eines:

- **Estructures de dades**. Elements amb els quals es construeixen les BD, com ara taules, arbres, etc.
- **Regles d'integritat**. Restriccions que les dades hauran de respectar, com per exemple tipus de dada, dominis, claus, etc.
- **Operacions a realitzar amb les dades**. Altes, baixes, modificacions i consultes, com a mínim.

2.1 Arquitectura dels SGBD

El 1975, el comitè ANSI/X3/SPARC va proposar una arquitectura per als SGBD estructurada en tres nivells d'abstracció (intern, conceptual i extern), que resulta molt útil per separar els programes d'aplicació de la BD considerada des d'un punt de vista físic.

SGBD

Acrònim de Sistema Gestor de Bases de Dades. És un programari especialitzat en la gestió de BD -bases de dades- (enteses, aquestes, com un conjunt estructurat d'informació).

D'altra banda, també resulta interessant examinar l'arquitectura dels SGBD des d'un punt de vista funcional, ja que coneixent els diferents components, i els fluxos de dades i de control podem entendre el funcionament dels SGBD, i estarem en millors condicions d'utilitzar-los d'una manera òptima.

2.1.1 Esquemes i nivells

Per gestionar les BD, els SGBD han de conèixer la seva estructura (és a dir, les entitats, els atributs i les interrelacions que conté, etc.). Els SGBD necessiten disposar d'una descripció de les BD que han de gestionar. Aquesta definició de l'estructura rep el nom d'esquema de la BD, i ha d'estar constantment a l'abast del SGBD perquè aquest pugui complir les seves funcions.

Comitè ANSIX3SPARC

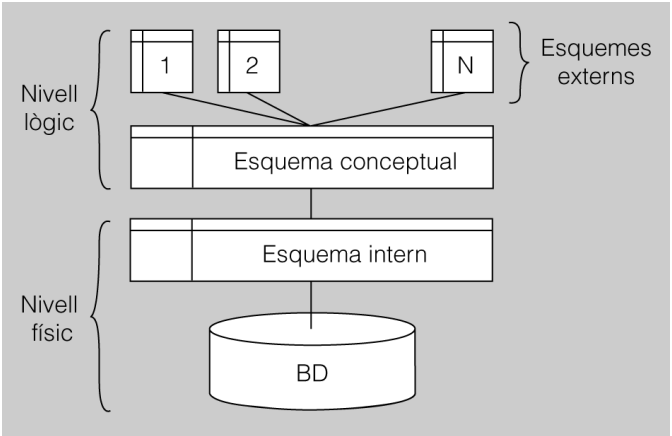
És un grup d'estudi de l'*Standard Planning and Requirements Committee* (SPARC) de l'ANSI (*American National Standards Institute*), dins del comitè X3, que s'ocupa d'ordinadors i d'informàtica.

D'acord amb l'estàndard ANSI/X3/SPARC, hi hauria d'haver tres nivells d'esquemes:

- En el **nivell extern** se situen les diferents visions lògiques que els processos usuaris (programes d'aplicació i usuaris directes) tenen de les parts de la BD que utilitzen. Aquestes visions s'anomenen *esquemes externs*.
- En el **nivell conceptual** hi ha una sola descripció lògica bàsica, única i global, que anomenem *esquema conceptual*, i que serveix de referència per a la resta d'esquemes.
- En el **nivell físic** hi ha una única descripció física, que anomenem *esquema intern*.

La figura 2.1 representa la relació, d'una banda, entre el nivell físic i l'esquema intern; i, d'una altra banda, entre el nivell lògic i l'esquema conceptual, i els diferents esquemes externs (o vistes).

FIGURA 2.1. Esquemes i nivells



En definir un esquema extern, només s'inclouran els atributs i entitats que interessin, els podrem reanomenar, podrem definir dades derivades, podrem definir una entitat de tal manera que les aplicacions que utilitzen aquest esquema extern creguin que en són dues, o podrem definir combinacions d'entitats perquè en semblin una de sola, etc.

En l'esquema conceptual, es descriuran les entitats tipus, els seus atributs, les interrelacions i també les restriccions o regles d'integritat.

L'esquema intern o físic contindrà la descripció de l'organització física de la BD: camins d'accés (índexs, *hashing*, apuntadors...), codificació de les dades, gestió de l'espai, mida de la pàgina, etc.

2.2 Els models de bases de dades més comuns

Els models de dades més utilitzats al llarg del temps han estat els següents, exposats en ordre d'aparició:

1. Jeràrquic
2. En xarxa
3. Relacional
4. Relacional amb objectes / orientat a objectes

Actualment, hi ha algunes noves tendències incipients, gràcies al fenomen Internet, tot i que la informació en les empreses segueix utilitzant els models clàssics de BD.

2.2.1 Model jeràrquic

Les BD jeràrquiques es van concebre al principi del anys seixanta, i encara s'utilitzen gràcies al bon rendiment i la millor estabilitat que proporcionen amb grans volums d'informació.

Les **BD jeràrquiques** emmagatzemen la informació en una estructura jeràrquica que podem imaginar amb una forma d'arbre invertit, on cada node pare pot tenir diferents fills. El node superior, que no té pare, es coneix com a *arrel*. I els nodes que no tenen fills s'anomenen *fulles*.

Les dades s'emmagatzemen en forma de registres. Cada registre té un tuple de camps. Un conjunt de registres amb els mateixos camps forma un fitxer.

Però les BD jeràrquiques no ofereixen una perspectiva lògica per sobre de la física. Les relacions entre les dades s'estableixen sempre a nivell físic, mitjançant adreçaments físics al mitjà d'emmagatzemament utilitzat (és a dir, indicant sectors i pistes, en el cas més habitual dels discos durs).

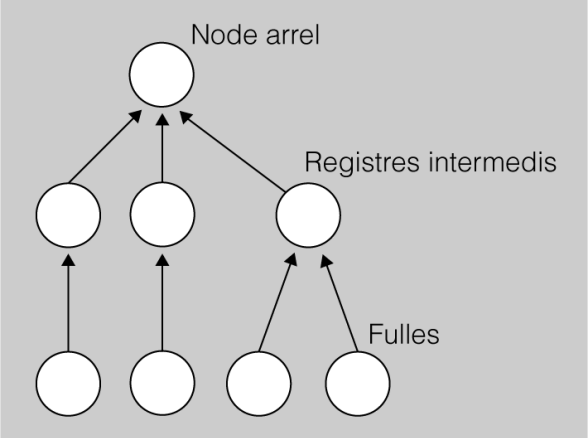
Així, doncs, les interrelacions s'estableixen mitjançant punters entre registres. Qualsevol registre conté l'adreça física del seu registre pare en el mitjà d'emmagatzemament utilitzat. Aquesta circumstància proporciona un rendiment molt bo: l'accés des d'un registre a un altre és pràcticament immediat.

Però, si bé el model jeràrquic optimitza les consultes de dades des dels nodes cap al node arrel, amb les consultes en sentit invers es produeix el fenomen contrari, ja que aleshores cal fer un recorregut seqüencial de tots els registres de la BD.

Altres limitacions d'aquest model consisteixen en la seva incapacitat per evitar la redundància (les repeticions indesitjades de les dades) o per garantir la integritat referencial (ja que els registres poden quedar "orfes" en esborrar-se el node pare respectiu). La responsabilitat per evitar aquests problemes queda a les mans de les aplicacions externes a la BD.

La figura 2.2 mostra l'estructura de nodes interrelacionats d'una BD jeràrquica.

Figura 2.2. Estructura d'una BD jeràrquica



Dos dels gestors de BD jeràrquiques que tenen més implantació són els següents:

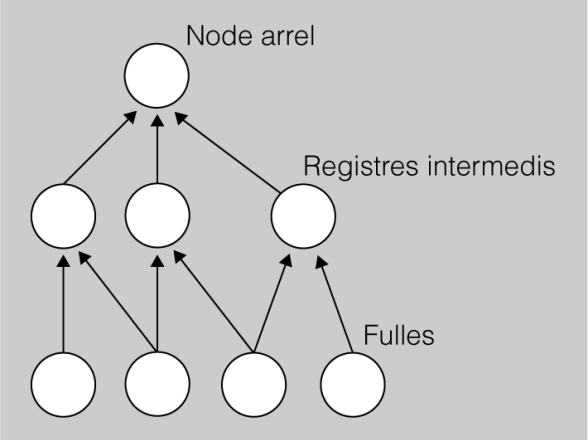
- IMS(*information management system*), de la multinacional nordamericana IBM.
- Adabas (*adaptable database system*), de l'empresa alemanya Software AG.

2.2.2 Model en xarxa

Al començament del anys setanta, en el mercat, van anar sorgint BD que seguien un **model en xarxa**, semblant al model jeràrquic, amb registres interrelacionats mitjançant una estructura en forma d'arbre invertit, però més flexible, ja que permetia que els nodes tinguessin més d'un sol pare.

La figura 2.3 mostra l'estructura de nodes interrelacionats d'una BD en xarxa.

Figura 2.3. Estructura d'una BD en xarxa



El model en xarxa va comportar una millora respecte al model jeràrquic, perquè permetia controlar de manera més eficient el problema de la redundància de dades.

Malgrat aquests avantatges, el model en xarxa no ha tingut tanta fortuna com el seu predecessor, a causa de la complexitat que comporta l'administració de les BD que l'adopten.

El consorci de la indústria de les tecnologies de la informació CODASYL (acrònim de *Conference on Data Systems Languages*) va proposar un estàndard que van seguir la majoria de fabricants.

Un dels gestors de BD en xarxa més coneguts, i que segueix l'estàndard CODASYL, és l'IDMS (*integrated database management system*), de Computer Associates.

2.2.3 Model relacional

El **model relacional** es basa en la lògica de predicats i en la teoria de conjunts (àrees de la lògica i de les matemàtiques). Actualment, és el sistema més àmpliament utilitzat per modelitzar dades.

Model relacional

Va ser proposat formalment per E. F. Codd, l'any 1970, en el seu treball *A Relational Model of Data for Large Shared Data Banks* ('Un model relacional de dades per a grans bancs de dades compartides').

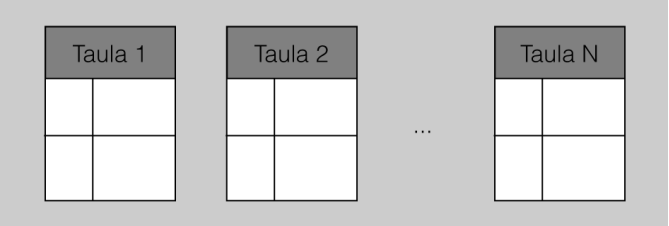
A partir dels anys vuitanta, es van començar a comercialitzar gran quantitat de BD que aplicaven aquest model.

Les dades s'estructuren en representacions tabulars, anomenades **taules**, que representen entitats tipus del món conceptual, i que estan formades per files i columnes. Les columnes formen els **camps**, que implementen els atributs, és a dir, les característiques que ens interessen de les entitats. I les files són els **registres**, que implementen les entitats instància, constituïdes pels conjunts dels valors que presenten els camps corresponents a cada instància.

En els models de dades jeràrquic i en xarxa les dades s'estructuraven gràcies a dos elements: els registres i les interrelacions. Però el model relacional només consta d'un element: les **relacions** o **taules**.

La figura 2.4 mostra l'estructura de taules corresponent a una BD relacional.

FIGURA 2.4. Estructura d'una BD relacional



Les interrelacions s'han d'implementar utilitzant les taules: quan és necessari s'han d'afegir, a les taules, un o més camps que actuïn com a el que anomenarem clau forana i que, per tant, "apuntin" al camp o camps referenciats d'una altra taula, els quals han de formar la seva clau primària. En coincidir els valors dels camps de la clau primària i de la clau forana, s'estableix la interrelació entre els registres.

El model relacional comporta certs avantatges respecte al model jeràrquic i al model en xarxa:

- Proporciona eines per evitar la duplicitat de registres, mitjançant claus primàries i foranes que permeten interrelacionar les taules.
- Vetlla per la integritat referencial: en eliminar-se un registre o en modificar-se el seu valor, o bé no permet fer-ho si hi ha registres interrelacionats en altres taules, o bé s'esborren o es modifiquen en cascada els registres interrelacionats, en funció de quina orientació hàgim seguit en administrar la BD.
- En no tenir importància la ubicació física de les dades, afavoreix la comprensibilitat. De fet, el model relacional, com a tal, es limita al nivell lògic, i deixa de banda el nivell físic. Per aquest motiu, es diu que el model relacional possibilita la independència física de les dades.

2.2.4 El paradigma de l'orientació a objectes

El **model de dades relacionals amb objectes** és una extensió del model relacional en sentit estricte.

TAD
Un tipus abstracte de dades (TAD) és un concepte que defineix les dades juntament amb les seves operacions associades.

Aquest nou model admet la possibilitat que els tipus de dades siguin, a més dels tradicionals, **tipus abstractes de dades** (**TAD**). Amb aquesta particularitat, s'acosten els sistemes de BD relacionals al paradigma de la programació orientada a objectes.

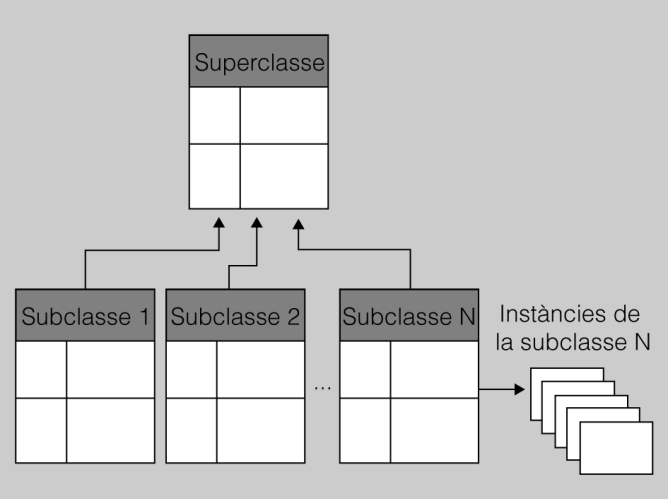
Els models estrictament orientats a objectes defineixen les BD en termes d'objectes, de les seves propietats i, el que és més innovador, de les seves operacions. Els objectes amb una mateixa estructura i comportament pertanyen a una classe, i les classes s'organitzen en jerarquies. Les operacions de cada classe s'especifiquen en termes de procediments predefinits, anomenats *mètodes*.

Oracle

Alguns SGBD presents en el mercat des de fa molt de temps, van estenent el model relacional per tal d'incorporar conceptes relatius a l'orientació a objectes. Aquest és el cas de la coneguda firma Oracle, la qual està treballant en aquesta línia des de la versió 8 del seu producte.

La figura 2.5 mostra l'estructura de classes, subclasses i instàncies, d'una BD orientada a objectes.

Figura 2.5. Estructura d'una BD orientada a objectes



Nous models de bases de dades

Actualment, hi ha organitzacions que gestionen grans quantitats de dades i que aquestes dades les han de consultar habitualment i els cal que aquestes consultes siguin ràpides. Típicament aquesta necessitat es va crear en empreses que necessitaven consultar dades per a la presa de decisions.

Es pot pensar, per exemple, en una multinacional que disposa de BD relacionals on emmagatzema totes les seves factures, totes les línies de factures, que s'han anat fent al llarg de 15 anys d'història. Disposar d'un resum global de l'evolució de la facturació dels darrers 10 anys, pot suposar, en aquesta BD relacional la consulta de milions de registres. Es pot intuir, doncs, que resultarà un procés costós obtenir aquesta informació.

El model relacional, doncs, moltes vegades no dóna una resposta prou eficient per a gestionar aquests tipus d'informació i per això van començar a aparèixer els sistemes `Datawarehouse`. Els `Datawarehouse` són magatzems de dades que integren eines per a extreure transformar i carregar informació anomenada d'intel·ligència empresarial així com informació de metadades.

Més enllà dels `Datawarehouse` existeixen les BD multidimensionals.

Les **BD multidimensionals** són BD dissenyades per a optimitzar el processament analític en línia, conegut com OLAP (*On-Line Analytical Processing*). La característica més destacable del processament OLAP és l'estructuració de les dades en els anomenats cubs OLAP (OLAP-cube, és el terme anglès amb el que es coneix aquest concepte).

Un cub OLAP és una estructura de dades que permet accessos ràpids a la informació i que s'organitza aquesta informació en diverses perspectives o dimensions.

Exemple de cub OLAP

Un exemple típic de cub OLAP es pot donar en una empresa que requereixi analitzar informació financera des de diferents perspectives:

- Per productes
- Per períodes de temps
- Per poblacions
- Per comparació amb el pressupost
- Etc.

Disposar de forma eficient de la informació organitzada des de totes aquestes perspectives es pot veure com a un hipercub d'informació, on cada dimensió del cub (que pot ser de dues, tres, quatre, etc. dimensions) correspon a una forma (perspectiva) d'accedir a aquestes dades.

Un cas particular de BD multidimensional són les **BD multivalor**. Les BD multivalor solen dissenyar BD on els atributs emmagatzemen llistes de valors, a diferència de les BD relacionals on els atributs són monovalor. Moltes vegades es coneix a les BD multivalor com a BD post-relacionals.

Les BD multivalor proporcionen llenguatges d'accés a les dades molt més semblants al llenguatge natural i, per tant, més senzills que SQL. La dificultat actual resideix en què cada implementació de BD multivalor disposa del seu propi llenguatge, no existint, en l'actualitat un estàndard comú.

2.2.5 Modelització de dades amb l'UML

Últimament, i per influència de les tècniques de desenvolupament de programari orientades a l'objecte, han aparegut alguns models semàntics com a noves extensions del model ER originari, entre les que destaca especialment l'UML (*unified modeling language*).

La notació del llenguatge UML és diagramàtica, com la notació del model ER. Els diagrames UML es poden utilitzar per expressar els dissenys conceptuals, tal com es fa amb els diagrames ER.

Però cal tenir en compte que el disseny de BD s'encarrega, fonamentalment, de la part estàtica dels sistemes d'informació (és a dir, la que no hauria de canviar al llarg del temps, com ara, justament, l'estructuració de les dades).

Classe

Una classe en orientació a objectes és una abstracció que agrupa un conjunt d'instàncies d'objectes. Per exemple, podem tenir la classe Cotxe que defineixi les característiques dels diferents objectes cotxe (instàncies).

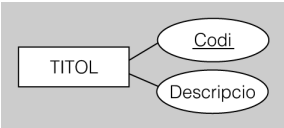
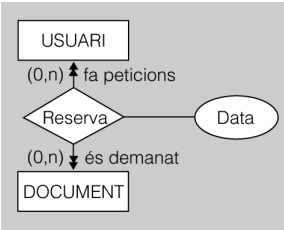
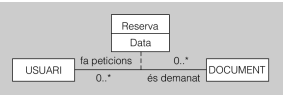
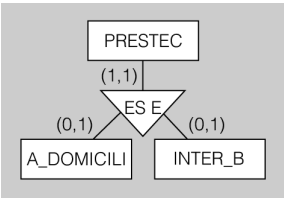
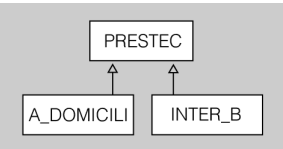
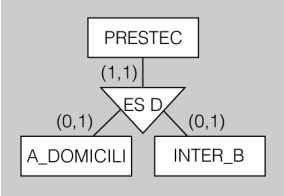
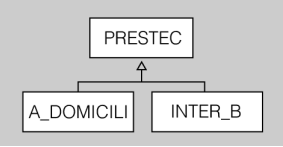
En l'UML, la part estàtica dels sistemes es representa mitjançant els anomenats **diagrames de classes** i, concretament, amb els components següents:

- Les classes i els atributs respectius (però no necessàriament, les operacions).
- Les relacions entre classes, com les associacions i les generalitzacions (però no tant, les dependències ni les realitzacions).

Per tant, aquí no considerarem ara altres aspectes de l'UML que s'ocupen de modelitzar més aviat la part dinàmica dels sistemes d'informació (és a dir, els aspectes que canvien amb el temps).

En la taula 2.1, podem veure les equivalències més importants entre la notació emprada en els diagrames ER, i l'emprada en els diagrames de classes UML.

Taula 2.1. Taula d'equivalències entre els diagrames ER i els diagrames de classe UML

	Diagrama ER	Diagrama de classe UML
Entitats: atributs		
Interrelacions: cardinalitats, rols i atributs		
Generalitzacions encavalcades		
Generalitzacions disjuntes		

Entitats i atributs

L'UML considera les entitats tipus com a classes, i les especifica amb requadres dividits verticalment en tres seccions:

- La superior, en què es col·loca el nom de la classe.
- La intermèdia, en què apareixen els atributs respectius.
- La inferior, en què han de constar els noms de les operacions de la classe (no hem mostrat aquest apartat perquè ara mateix només ens interessen els aspectes estàtics de les dades).

Les classes permeten especificar molts altres detalls, com ara el tipus de dada de cada atribut, la seva visibilitat, etc.

Interrelacions: cardinalitats, rols i atributs

L'UML considera les interrelacions com a associacions entre classes.
Les interrelacions binàries es representen en els diagrames de classe UML simplement amb una línia contínua que connecta les dues classes implicades.

Aquesta línia pot ser dirigida, és a dir, que pot acabar amb una punta de fletxa (no tancada) en un dels extrems (o en tots dos).

A més, les associacions, poden incloure una etiqueta amb el nom que tinguin assignat o, fins i tot, dues, si preferim especificar el rol que adopta cadascuna de les dues classes.

Si, en un diagrama ER, una interrelació té atributs propis, el diagrama de classes UML equivalent haurà d'incorporar un requadre addicional, dividit verticalment en dues seccions. En la secció superior haurà de constar el nom de l'associació, i en la inferior s'hauran d'incloure els atributs. Finalment, aquest requadre haurà d'anar unit amb una línia discontinua amb l'associació.

Les restriccions de cardinalitat s'especifiquen en els diagrames de classe UML de manera molt semblant a com es fa en els diagrames ER. També s'utilitza la forma `màx .. mín` que ja coneixem, per tal d'indicar el màxim i el mínim d'associacions en què pot participar cada instància de la classe. Ara bé, habitualment, la *n* es representa amb un asterisc (*), i la ubicació de les etiquetes que indiquen les restriccions de cardinalitat és exactament la inversa de la que correspon en un diagrama ER.

Cal parar atenció en el fet que les interrelacions n-àries d'ordre superior a 2 no es poden representar directament en els diagrames de classe UML.

Generalitzacions encavalcades i disjuntes

Els **diagrames de classe UML** poden representar explícitament generalitzacions i especialitzacions, i considerar-les genèricament a totes dues com a generalitzacions entre una superclasse i unes quantes subclasses.
Les generalitzacions es representen mitjançant línies contínues que parteixen de les subclasses i acaben en una punta de fletxa buida que apunta a la superclasse.

Si es tracta d'una generalització encavalcada (és a dir, quan les instàncies de la superclasse també poden ser instàncies, simultàniament, de més d'una subclasse), cal establir una línia i una punta de fletxa pròpia per a cada subclasse.

En canvi, quan es vol representar una generalització disjunta (és a dir, quan les instàncies de la superclasse només poden ser, també, instàncies d'una sola de les subclasses), les línies que parteixen de les subclasses han d'acabar confluint en una única punta de fletxa buida que apunti a la superclasse.

2.3 Bases de dades distribuïdes

BD

BD és l'acrònim de *base de dades*, entesa com a conjunt estructurat de dades emmagatzemades que permeten obtenir informació.

Un dels sectors informàtics on més s'està evolucionant darrerament, tot integrant el desenvolupament tecnològic amb la innovació metodològica, és el relatiu als sistemes distribuïts d'informació.

Amb aquesta darrera expressió volem fer referència a la utilització de dades emmagatzemades en diferents ubicacions, de vegades molt distants entre si, però que al mateix temps estan connectades, mitjançant una xarxa de comunicacions.

Aquesta tendència es fa palesa en l'ús habitual d'Internet per part de qualsevol de nosaltres, però també hi ha nombroses empreses i institucions que necessiten que els sistemes informàtics (i per tant, també les BD) s'adaptin cada cop més a la seva estructura geogràfica o funcional.

Un cas concret d'aquests sistemes el constitueixen les bases de dades distribuïdes. És important conèixer les diferents arquitectures aplicades als sistemes de bases de dades, en general. Podem distingir entre les arquitectures centralitzades (incloent-hi els sistemes client-servidor) i les descentralitzades, en què s'ha de tenir en compte el funcionament dels sistemes paral·lels.

També és important conèixer les diferents metodologies per distribuir BD, en funció dels objectius plantejats en la fase de disseny, i també de si l'estratègia emprada té un caràcter ascendent o descendent. S'han de tenir en compte, però, les dues conseqüències més problemàtiques de la distribució de BD: la duplicació i la fragmentació de les dades, on hem de diferenciar entre fragmentacions horitzontals, verticals i mixtes.

No hem d'oblidar tampoc les problemàtiques específiques que representen per a les BD distribuïdes tant les transaccions com la concurrència, ni els diferents protocols amb els quals es dóna resposta a aquestes eventualitats.

2.3.1 Arquitectures de sistemes de bases de dades: centralitzades, descentralitzades, client-servidor

L'arquitectura de tot sistema de BD està molt condicionada per les característiques del sistema informàtic sobre el qual s'executa, i en especial pels aspectes següents:

- La connexió en xarxa de diferents computadores.
- El processament paral·lel de consultes dins d'una mateixa computadora.
- La distribució de les dades en diferents computadores, fins i tot allunyades entre si.

Aquestes innovacions tecnològiques han permès, respectivament, el desenvolupament, a partir dels inicials sistemes totalment centralitzats en una sola computadora, de diferents arquitectures de sistemes de BD més evolucionades, i que permeten donar resposta a una gran varietat de necessitats dels usuaris i de les organitzacions en què aquests treballen, com ara:

- Sistemes client-servidor.
- Sistemes paral·lels.
- Sistemes distribuïts.

Arquitectures centralitzades i client-servidor

Inicialment els sistemes de BD eren de tipus estrictament centralitzat, en el sentit que s'executaven sobre un únic sistema informàtic, sense necessitat d'interaccionar amb cap altre.

Però l'abaratiment dels ordinadors personals i el desenvolupament vertiginós de les seves capacitats, juntament amb la implantació de les xarxes i d'Internet, ha fet evolucionar els sistemes centralitzats cap a arquitectures de tipus client-servidor.

Sistemes centralitzats en una sola computadora

Concurrència

Es parla de concurrència quan diversos processos s'executen paral·lelament, i, en aquest cas, fan ús de les mateixes dades.

En parlar de sistemes de BD centralitzats, es fa referència tant als petits sistemes monousuaris que s'executen en un únic ordinador personal, com als grans sistemes multiusuaris d'alt rendiment.

En els sistemes monousuaris, els sistemes de BD centralitzats són petits sistemes de BD pensats per a les tasques que pugui fer un sol usuari amb una estació de treball. Aquests sistemes no sempre ofereixen totes les possibilitats que sempre ha de garantir qualsevol sistema de BD multiusuari, per modest que sigui, com per exemple el control automàtic de la concurrència.

En els sistemes multiusuaris, en canvi, es tracta de grans sistemes que poden donar servei a un gran nombre d'usuaris. Aquests només disposen per interaccionar amb el sistema de terminals sense capacitat pròpia per emmagatzemar dades ni tampoc per processar consultes, ja que de la realització d'aquestes tasques s'encarrega l'únic sistema centralitzat existent.

Sistemes client-servidor

Els sistemes client-servidor tenen les seves funcionalitats repartides entre el sistema servidor central i múltiples sistemes clients que li envien peticions.

ODBC i JDBC

ODBC i *JDBC* són els acrònims d'*open database connectivity* i *Java database connectivity*, respectivament. Es tracta de dos dels protocols més utilitzats per a la interconnexió de les aplicacions de BD.

Gradualment, els antics terminals dels sistemes centralitzats han estat substituïts per ordinadors personals, igualment connectats als subsistents sistemes centrals. Com a conseqüència d'això, pràcticament tots els sistemes centralitzats actuen avui en dia com a sistemes servidors que satisfan les peticions que els envien els respectius sistemes clients.

Les arquitectures basades en servidors de dades s'han implantat especialment en les BD orientades a objectes.

Actualment, els estàndards ODBC i JDBC permeten que tot client que utilitzi qualsevol dels dos, es pugui connectar a qualsevol servidor que proporcioni la interfície respectiva.

Les arquitectures basades en servidors de consultes són les que tenen més implantació.

Podem distingir dues tipologies de sistemes servidors:

- **Servidors de dades**. S'utilitzen en xarxes d'àrea local en què s'arriba a una alta velocitat de connexió entre els clients i el servidor, sempre que les estacions de treball siguin comparables al servidor quant a la capacitat de processament. En entorns així definits, pot tenir sentit enviar les dades als clients, fer allà totes les tasques de processament d'aquestes dades, i finalment reenviar, si cal, els resultats al servidor.

- **Servidors de consultes**. Proporcionen una interfície, mitjançant la qual els clients els envien peticions per tal que resolguin consultes, i els retornin els resultats obtinguts. Així doncs, les transaccions s'executen sobre el servidor, però les dades resultants es visualitzen en el client del qual provenia la petició, si escau.

Arquitectures descentralitzades

Memòria principal vs. memòria persistent

Entenem per memòria principal la memòria volàtil, habitualment la RAM. Entenem per memòria persistent aquella que no és volàtil. Habitualment en forma de disc o altres dispositius d'emmagatzematge extern.

La descentralització de les arquitectures de BD pot consistir en el repartiment de la càrrega de feina entre diferents components físics del sistema (bàsicament pel que fa a processadors, memòria principal i memòria persistent) comunicats entre si mitjançant una xarxa d'interconnexió.

Però una arquitectura descentralitzada també pot consistir en la distribució de la mateixa BD en diferents computadores, seguint la metodologia més adient per assolir l'objectiu proposat.

Sistemes paral·lels

L'objectiu principal dels sistemes paral·lels consisteix a augmentar la velocitat de processament i d'E/S mitjançant la utilització en paral·lel d'UCP, memòria i discos durs.

E/S i I/O són els acrònims d'entrada i sortida, i de l'original en anglès input/output.

Els sistemes paral·lels són molt útils (o fins i tot són imprescindibles) en el treball quotidià amb BD molt grans (de l'ordre de terabytes), o que han de processar moltes transaccions (de l'ordre de milers per segon), ja que normalment els sistemes centralitzats i els sistemes client-servidor no tenen prou capacitat per donar resposta a aquest tipus de necessitats.

UCP i CPU són els acrònims d'unitat central de procés, i de l'original en anglèscentral processing unit.

Avui en dia molts ordinadors de gamma alta ofereixen un cert grau de paral·lelisme, atès que incorporen dos o quatre processadors. Però hi ha computadores paral·leles que suporten centenars de processadors i discos durs.

Ara bé, una de les característiques que permet avaluar la utilitat d'un sistema paral·lel de BD és la seva ampliabilitat, la qual ha de garantir el funcionament ulterior del sistema a una velocitat acceptable, encara que creixi la grandària de la BD o el nombre de transaccions.

Però la veritable ampliabilitat dels sistemes paral·lels ve donada per la comunicació dels seus components mitjançant alguna xarxa que faci possible connectar-los entre si. Les tres tipologies de xarxa més utilitzades són les següents:

En una malla, un component pot arribar a estar a 2(n - 1) nodes de distància d'altres components.

- **Bus**: es pot tractar d'una xarxa *ethernet* o una interconnexió paral·lela. En tot cas, aquesta estructura només és apta per al treball amb un petit nombre de processadors.
- **Malla**: cada component està connectat amb tots els nodes adjacents. (Si la malla és bidimensional els nodes adjacents seran 4, i si és tridimensional, 6).
- **Hipercub**: s'assigna a cada component un nombre binari exclusiu, de tal manera que dos components han de tenir una connexió directa entre si sempre que els nombres binaris respectius només difereixin en un sol

bit. Així doncs, cadascun dels n elements del sistema estarà directament connectat amb uns altres $\log(n)$ components.

Retard de les comunicacions en un hipercub

En un hipercub, un component pot estar, com a màxim, a $\log(n)$ nodes de distància d'altres components. El retard de les comunicacions en un hipercub és menor que en una malla.

D'altra banda, hi ha diferents models d'arquitectures paral·leles de BD. Alguns dels models més importants són:

- **Memòria compartida**. Tots els processadors comparteixen una memòria comuna, habitualment mitjançant un bus. Les arquitectures de memòria compartida han de dotar cada processador de molta memòria cau, per tal d'evitar els accessos a la memòria compartida sempre que això sigui possible. Ara bé, el límit raonable de processadors treballant en paral·lel ve donat, justament, pel cost que representa el manteniment de la coherència de la memòria cau.
- **Discos compartits**. Tots els processadors comparteixen un conjunt de discos comú, mitjançant una xarxa d'interconnexió. Aquest model permet el treball en paral·lel d'un nombre de processadors més gran que amb el model de memòria compartida, però com a contrapartida la comunicació entre ells és més lenta, ja que utilitzen una xarxa en lloc d'un bus.
- **Sense compartició**. Els processadors no comparteixen ni memòria ni discos. Aquest model té unes potencialitats d'ampliació encara més grans que el de compartició de discos, però en canvi els costos de comunicació són superiors als del model esmentat.
- **Jeràrquic**. Combinació de les característiques dels models anteriors. Aquesta arquitectura s'estructura en diferents nivells. Al nivell més alt, els nodes, connectats mitjançant una xarxa d'interconnexió, no comparteixen ni memòria ni discos. Cadascun d'aquests nodes pot ser, internament, un sistema de memòria compartida entre diferents processadors. O bé cadascun d'aquests nodes pot ser, internament, un sistema de discos compartits que inclogui, a un tercer nivell, un sistema de memòria compartida.

Sistemes distribuïts

Una **BD distribuïda** està formada per un conjunt de BD parcialment independents, emmagatzemades en diferents computadores, que comparteixen un esquema comú i que coordinen el processament de les transaccions que accedeixen a dades remotes.

Els processadors de les diferents computadores que formen un sistema distribuït es comuniquen entre si mitjançant una xarxa de comunicacions, però no comparteixen ni memòria ni discos. Cadascuna d'aquestes computadores constitueix, per tant, un **node del sistema distribuït**.

SGBD

SGBD és l'acrònim de *sistema gestor de bases de dades*. Es tracta d'un programari especialitzat en la gestió i l'emmagatzematge de BD.

Normalment, els nodes de les BD distribuïdes es troben en llocs geogràficament distants, s'administren parcialment de manera independent i tenen una interconnexió força lenta entre ells.

Normalment, tot SGBD actual és capaç de treballar amb un altre d'idèntic. Però això no sempre és tan fàcil entre SGBD de diferents fabricants. En funció d'aquesta eventualitat, es distingeix entre dos tipus de sistemes distribuïts de BD:

- **Sistemes homogenis**: són sistemes fortament acoblats, en què tots els nodes utilitzen el mateix SGBD o, en el pitjor dels casos, diferents SGBD del mateix fabricant.
- **Sistemes heterogenis**: els SGBD que utilitzen els nodes són diferents, i, per tant, solen ser més difícils d'acoblar.

L'acoblament és el grau d'interacció i dependència que tenen dues parts d'un sistema.

En tot cas, els usuaris dels sistemes distribuïts de BD no han de conèixer els detalls d'emmagatzemament de les dades que utilitzi, com ara la seva ubicació concreta o la seva organització. D'aquesta característica se'n diu**transparència**. Evidentment, els administradors de la BD sí que hauran de ser conscients d'aquests aspectes.

Avantatges i inconvenients de la distribució de BD

Hi ha bones raons per implementar BD distribuïdes, com ara la compartició de la informació, la disponibilitat de les dades o l'agilització del processament d'algunes consultes:

- **Compartició de la informació i autonomia local**. Un avantatge de compartir les dades mitjançant la distribució consisteix en el fet que des de cada node es pot controlar, fins a cert punt (de fet, fins allà on permeti l'administrador global de la BD), l'administració de les dades emmagatzemades localment. Aquesta característica es coneix com a *autonomia local*.
- **Fiabilitat i disponibilitat**. Si es produeix una fallada en algun node d'un sistema distribuït o en les comunicacions amb aquest, és possible que els altres nodes puguin continuar treballant, si les dades que conté el node caigut o incomunicat estan repetides en altres nodes del sistema.
- **Agilització del processament de consultes**. Quan una consulta necessita accedir a dades emmagatzemades en diferents nodes, pot ser possible dividir la consulta en diferents subconsultes que s'executin en els nodes respectius.

Però l'ús de BD distribuïdes també té els seus punts febles, com per exemple l'increment dels costos en desenvolupament de programari i l'augment de possibilitat d'errors, i el temps addicional a afegir al temps de processament:

- **Increment en els costos de desenvolupament de programari**. És més difícil estructurar un sistema distribuït de BD, i també són més complicades les aplicacions que han de treballar amb aquest sistema, que no pas si es tracta d'un sistema de BD centralitzat.
- **Més possibilitat d'errors**. Com que els diferents nodes del sistema distribuït operen en paral·lel, és més difícil garantir la correcció dels algorismes.

- **Temps extra que cal afegir al temps de processament**. L'intercanvi de missatges i els càlculs necessaris per garantir la integritat de les dades distribuïdes entre tots els nodes comporten un afegitó extra de temps, inexistent en els sistemes centralitzats.

2.3.2 Disseny de bases de dades distribuïdes. Estratègies. Metodologies

El disseny de la distribució d'una BD implica adoptar certes decisions. La primera té a veure amb la mateixa idoneïtat d'utilitzar una BD distribuïda i no pas una altra arquitectura. Aquesta decisió s'ha de fonamentar en el rendiment esperat de cadascuna de les arquitectures disponibles, aplicades al mateix conjunt de necessitats.

A continuació, en el cas d'optar per una BD distribuïda, cal prendre altres decisions no menys importats sobre quina és la millor manera d'ubicar en els diferents nodes del sistema tant les dades com les aplicacions que hagin d'accedir a aquestes dades.

La ubicació de les aplicacions habitualment no comporta grans problemes. Depèn de les funcionalitats que aquestes han d'oferir i dels llocs des dels quals s'utilitzaran majoritàriament o exclusivament. Però també s'ha de tenir molt en compte si hauran de treballar amb un sistema homogeni o heterogeni, i els SGBD utilitzats en cada cas.

Però la distribució de les dades és més crítica i ha de perseguir la consecució de certs objectius: potenciació del processament local, distribució ideal de la càrrega de feina i reducció dels costos d'emmagatzemament. A més, cal seguir una estratègia general de disseny ascendent o descendent, tot i que tots dos enfocaments no són mútuament excloents i es poden emprar en un mateix projecte en diferents etapes d'aquest. Finalment, i com a conseqüència de tot això, s'ha d'adoptar una metodologia concreta de distribució de dades, és a dir, multiplicació, divisió, etc.

Objectius de la distribució

Hi ha un cert consens en alguns dels objectius bàsics que ha de perseguir tot sistema distribuït de BD, com ara:

- **Potenciació del processament local**. En un sistema distribuït hi ha dos tipus de transaccions: les locals i les globals. Les primeres únicament necessiten accedir al mateix node del qual parteix la petició. En canvi, les segones necessiten accedir a dades ubicades en altres nodes, la qual cosa comporta un cost addicional, ja que cal utilitzar la xarxa que els comunica. Si les dades són distribuïdes acostant-les a les aplicacions que més les utilitzen, es maximitza el processament local.

- **Distribució ideal de la càrrega de feina**. La distribució de les dades també ha de tenir en compte les característiques de les diferents computadores ubicades a cada node i els usos més adients per a cadascuna d'elles. D'aquesta manera es potencia el paral·lelisme en l'execució de les aplicacions. Ara bé, cal advertir que la persecució d'aquest objectiu pot afectar negativament la potenciació del processament local.

- **Reducció dels costos d'emmagatzemament**. La repetició de les dades en diferents nodes d'un sistema distribuït pot contribuir a la disponibilitat d'aquestes, ja que si es produeix una fallada en un dels nodes, es podrà continuar treballant amb les duplicacions existents en un altre. Això comporta, entre altres coses, un increment dels costos d'emmagatzematge que ha de ser tingut en compte, encara que últimament resulta irrellevant si es compara amb els costos derivats en matèria d'UCP, E/S i transmissions per la xarxa.

Estratègies: dissenys ascendent i descendent

A l'hora de dissenyar una BD distribuïda podem optar, fonamentalment, per dues estratègies: disseny ascendent i disseny descendent.

El **disseny ascendent** és una estratègia que pot ser aplicada quan s'ha de dissenyar una nova BD a partir de petites BD preexistents que han de ser integrades en una de sola, però conservant en la mesura que es pugui la ubicació originària de les dades.

Bottom-up designa com s'anomena, en anglès, el disseny ascendent.

En el disseny ascendent de BD distribuïdes s'han de sintetitzar els esquemes lògics locals per arribar a construir l'esquema lògic global del sistema distribuït.

Els sistemes distribuïts resultants d'un disseny ascendent amb BD preexistents són amb certa freqüència heterogenis, llevat que el projecte prevengui la migració a un SGBD distribuït, comú a tots els nodes.

El **disseny descendent** de BD distribuïdes és l'estratègia més adient quan es tracta de dissenyar aplicacions i BD noves, o quan es pot prescindir de conservar les estructures de dades anteriors, en cas que n'hi hagi. Evidentment, en aquests casos en què el dissenyador té més capacitat decisòria, el més recomanable és optar per implantar un sistema homogeni.

Top-down designa com s'anomena, en anglès, el disseny descendent.

En el disseny descendent de BD distribuïdes s'ha de partir de l'anàlisi de requeriments inicials, per tal de definir en primer lloc el disseny conceptual i simultàniament el disseny de les vistes dels usuaris finals de la futura BD.

De fet, en l'àmbit de les BD distribuïdes, el disseny conceptual (que dóna lloc a entitats i a interrelacions entre elles) es pot interpretar com una integració de les diferents vistes dels usuaris.

Posteriorment, el disseny lògic inclourà totes les decisions en matèria de distribució de les dades. En funció de la metodologia de distribució adoptada, les relacions s'ubicaran senceres en diferents nodes del sistema, o bé es dividiran en fragments per ser distribuïts entre els nodes d'aquesta manera.

Finalment, en els nodes en què es consideri oportú, es podrà fer el disseny físic, a nivell local.

Metodologies de distribució

Hi ha diferents metodologies per orientar la distribució de les BD, cadascuna de les quals té els seus avantatges i els seus inconvenients:

- Multiplicació.
- Divisió.
- Distribució amb node principal.
- Distribució amb duplicacions en nodes seleccionats.

Multiplicació

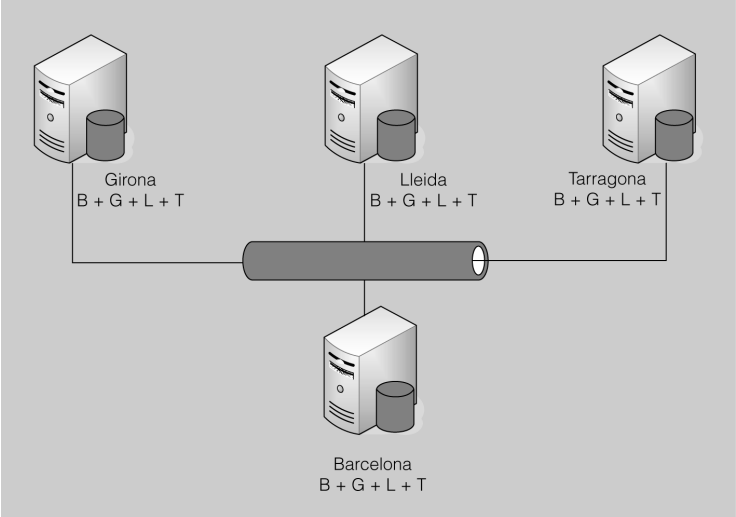
La multiplicació implica que la BD està replicada íntegrament a cada node del sistema. A la pràctica, aquest sistema és molt poc utilitzat.

L'avantatge principal de la multiplicació és evident, ja que les consultes es fan localment, sense haver d'accedir a la xarxa de comunicacions i, per tant, de manera molt ràpida. A més, en cas que caigui algun node, la resta pot continuar treballant.

Però no en falten de desavantatges, ja que les operacions d'actualització de dades s'han de fer en tots els nodes per tal de mantenir la coherència de la BD, la qual cosa implica un trànsit molt intens a la xarxa. I encara que actualment, en la majoria de casos, sigui un problema menor, cal dir que aquest model multiplica pel nombre total de nodes l'espai necessari per emmagatzemar la BD.

En la figura 2.6 es mostra un exemple de BD multiplicada, on cada node del sistema conté replicada completament la BD.

Figura 2.6. BD multiplicada, on es mostra la ubicació de les diferents parts (B, G, L i T) de les dades de la BD

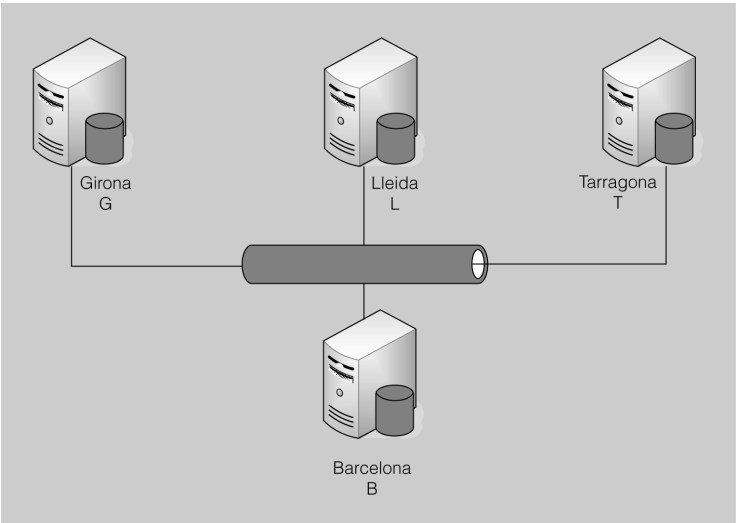


Divisió

Amb el mètode de la divisió, la BD està distribuïda de tal manera que no hi ha cap part que estigui replicada en més d'un node.

L'avantatge principal de la divisió és que les operacions d'actualització són molt senzilles, ja que no es requereix actualitzar de manera transaccional les mateixes dades en diferents nodes. A més, tampoc no es necessita més espai d'emmagatzemament del que es necessitaria si es tractés d'una BD centralitzada.

Figura 2.7. BD dividida, on es mostra la ubicació de les diferents parts (B, G, L i T) de les dades de la BD



Com a contrapartides, podem dir que les operacions de consulta són sempre molt costoses, ja que la majoria són globals, la qual cosa comporta un trànsit molt intens a la xarxa. A més, la caiguda de qualsevol node implica la impossibilitat de poder accedir a aquella part de la BD emmagatzemada en ell.

En la figura 2.7 tenim un exemple de BD dividida, on cada node conté només les dades que li són pròpies, sense que estigui duplicada cap part de la BD en més d'un node.

Distribució amb node principal

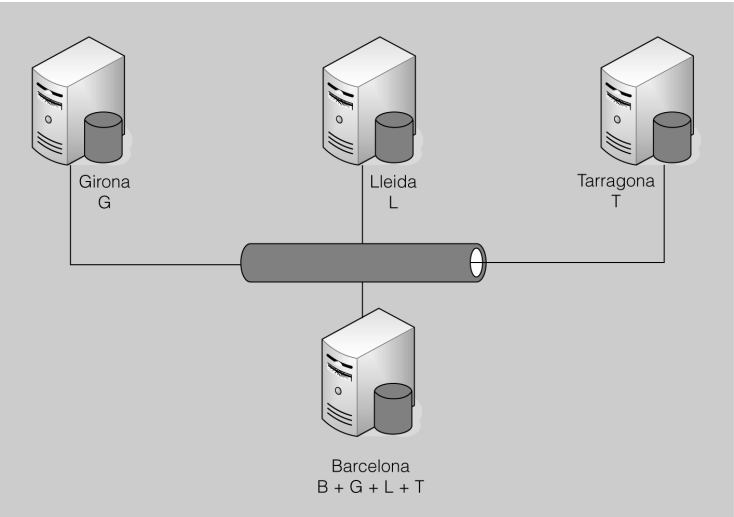
Quan s'utilitza el model de distribució amb node principal, un dels nodes, al qual es pot considerar principal, conté la BD sencera, i cadascun dels altres nodes conté replicada alguna part de la BD.

L'avantatge principal de la distribució amb node principal és que les dades s'acosten als nodes que més les utilitzen, la qual cosa potencia el processament local. A més, la disponibilitat és força bona, ja que totes les dades estan replicades com a mínim una vegada, pel fet d'estar emmagatzemades en algun dels nodes del sistema, a més d'estar-ho en el node principal.

Els desavantatges consisteixen fonamentalment en el fet que els costos de les operacions d'actualització i el d'emmagatzemament sempre seran més elevats (tot i que sense arribar als extrems de les BD multiplicades) que els produïts en sistemes centralitzats.

La figura 2.8 mostra un exemple de BD distribuïda amb un node principal (Barcelona) que conté tota la BD, mentre que la resta de nodes només contenen, duplicades, les dades que els són pròpies.

FIGURA 2.8. BD distribuïda, amb node principal on es mostra la ubicació de les diferents parts (B, G, L i T) de les dades de la BD



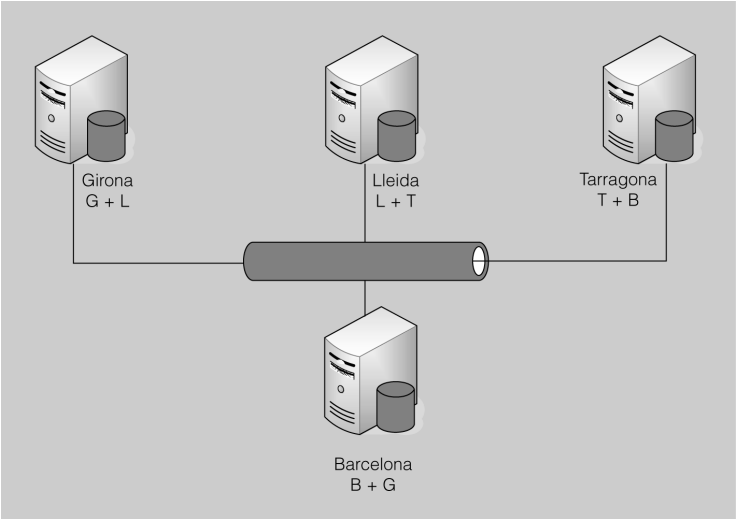
Distribució amb duplicacions en nodes seleccionats

Seguint la metodologia de distribució amb duplicacions en nodes seleccionats, cap node conté la BD completa, però cada node replica alguna part de la BD, de tal manera que tota la BD, globalment considerada, està duplicada.

La distribució amb duplicacions en nodes seleccionats es tracta d'una solució amb uns avantatges i uns inconvenients similars als del model basat en la distribució amb node principal. L'avantatge respecte a aquell és que, com que no hi ha un node principal que contingui tota la BD, la disponibilitat és un xic més elevada.

La figura 2.9 proporciona un possible exemple de BD distribuïda amb duplicacions en nodes seleccionats. Cada node conté les dades que li són pròpies, i a més, conté replicada una altra part de la BD. No hi ha un node principal que contingui tota la BD, però la duplicació d'aquesta és completa si considerem les dades contingudes en tots els nodes.

FIGURA 2.9. BD distribuïda, amb duplicacions en nodes seleccionats on es mostra la ubicació de les diferents parts (B, G, L i T) de les dades de la BD



2.3.3 Conseqüències de la distribució de les dades: duplicació, fragmentació

Relacions en una BD

En el cas més habitual de BD relacionals, basades en el model relacional, una relació és la unitat lògica d'organització de les dades, que es correspon a una taula de BD.

Les BD distribuïdes emmagatzemen les relacions seguint principalment un dels dos esquemes referits a continuació:

- **Duplicació**. El sistema conserva còpies idèntiques (com a mínim una) de cada relació en diferents nodes.
- **Fragmentació**. Les relacions es divideixen en diferents fragments i s'emmagatzemen en diferents nodes. La fragmentació es realitza seguint alguna de les metodologies disponibles (horitzontal, vertical, etc.).

La duplicació i la fragmentació es poden utilitzar de manera combinada, fragmentant les relacions i distribuint còpies de cadascun dels fragments resultants entre els diferents nodes del sistema.

Duplicació

La **duplicació** consisteix en l'emmagatzematge de relacions senceres, o de fragments d'aquestes, en diferents nodes de la xarxa.

La duplicació és un tipus d'esquema d'emmagatzematge de les BD distribuïdes que comporta tant avantatges com inconvenients.

D'una banda, la duplicació de les dades millora la seva disponibilitat respecte als sistemes centralitzats, ja que si falla un dels nodes que conté una relació o un fragment d'aquesta, es pot acudir (encara que només sigui temporalment) a un dels altres nodes que contingui les mateixes dades.

D'altra banda, la duplicació de les dades en diferents nodes incrementa el paral·lelisme, ja que fa augmentar les possibilitats de que les dades es trobin en el mateix node des del qual es llença la consulta.

Però, al mateix temps, la duplicació de dades implica un increment de la sobrecàrrega del sistema quan es produeixen actualitzacions de dades, ja que cal garantir la consistència de totes les rèpliques existents.

Fragmentació

La **fragmentació** consisteix a dividir les relacions en diferents fragments. Aquests fragments han de contenir tota la informació necessària per tal de reconstruir les relacions originàries corresponents, en cas necessari.

El problema fonamental de la fragmentació inherent a les BD distribuïdes consisteix a trobar la unitat ideal de distribució de les dades. Normalment les relacions no són la millor opció de distribució per moltes raons.

D'una banda, les vistes que proporcionen les aplicacions habitualment són subconjunts de relacions. Per tant, pot ser molt més convenient considerar aquests subconjunts de relacions com les unitats desitjables de distribució.

Però, a més, la descomposició d'una relació en diferents fragments, allotjats en diferents nodes del sistema, pot contribuir a millorar el rendiment del sistema, ja que permet l'execució concurrent de transaccions, i provoca, en molts casos, l'execució paral·lela de les consultes, quan s'han de dividir en diferents subconsultes per tal d'operar sobre els diferents fragments.

Fragmentació horitzontal

La **fragmentació horitzontal** consisteix a dividir els tuples d'una relació (és a dir, les files) en dos o més subconjunts en funció dels valors que aquelles tinguin en un o més atributs.
Aquests valors han de ser indicatius dels nodes que més consultes realitzaran sobre els respectius tuples, per tal d'acostar aquests als seus usuaris més habituals. Els tuples poden estar presents en més d'un fragment, però han d'estar com a mínim en un d'ells per tal que la fragmentació sigui correcta.

La taula 2.2 mostra una relació, anomenada PROVEIDOR, per tal d'exemplificar els mètodes principals de fragmentació.

Taula 2.2. Relació PROVEIDOR

PROVEIDOR				
NIF*	**Nom**	**Telefon**	**Adreça**	**Localitat**
33333333K	L'abastadora, SL	902456456	Pol. Ind. Polièdric, s/n	Lleida
44444444L	Proveïdora Ibèrica, SA	906789789	C/ del pi, 3	Lleida
55555555M	Supplies & Co. Ltd.	900123123	C/ del call, 4	Girona
66666666N	Assortiments de l'Onyar, SCP	908852852	Pg. De la ribera, s/n	Girona

*NIF és la clau principal de la taula proveïdor.

La taula 2.3 i taula 2.4 mostren dos possibles fragments horitzontals de la relació PROVEIDOR. Els tuples s'han dividit en funció de la localitat de cada proveïdor.

Taula 2.3. Primer fragment horitzontal de la relació PROVEIDOR

PROVEIDOR				
NIF*	**Nom**	**Telefon**	**Adreça**	**Localitat**
33333333K	L'abastadora, SL	902456456	Pol. Ind. Polièdric, s/n	Lleida
44444444L	Proveïdora Ibèrica, SA	906789789	C/ del pi, 3	Lleida

*NIF és la clau principal de la taula proveïdor.

Taula 2.4. Segon fragment horitzontal de la relació PROVEIDOR

PROVEIDOR				
NIF*	**Nom**	**Telefon**	**Adreça**	**Localitat**
55555555M	Supplies & Co. Ltd.	900123123	C/ del call, 4	Girona
66666666N	Assortiments de l'Onyar, SCP	908852852	Pg. De la ribera, s/n	Girona

*NIF és la clau principal de la taula proveïdor.

Fragmentació vertical

La fragmentació vertical consisteix a dividir els atributs de la relació (és a dir, les columnes) en diferents fragments. Els fragments resultants han de contenir els atributs que utilitzaran més freqüentment els usuaris del node on seran respectivament emmagatzemats.

A més dels atributs seleccionats, cada fragment haurà de contenir la clau primària de la relació, per tal de poder associar els tuples de tots els fragments pertanyents a una mateixa relació, que estiguin allotjats en els diferents servidors del sistema.

Els atributs poden estar presents en més d'un fragment, però han d'estar com a mínim en un d'ells per tal que la fragmentació sigui correcta.

La taula 2.5 i taula 2.6 mostren dos possibles fragments verticals de la relació PROVEIDOR. Els fragments resulten de seleccionar només els atributs de cada proveïdor que utilitzaran amb més freqüència els nodes que respectivament els emmagatzemin.

Taula 2.5. Primer fragment vertical de la relació PROVEIDOR

PROVEIDOR		
NIF*	**Nom**	**Telefon**
33333333K	L'abastadora, SL	902456456
44444444L	Proveïdora Ibèrica, SA	906789789
55555555M	Supplies & Co. Ltd.	900123123
66666666N	Assortiments de l'Onyar, SCP	908852852

*NIF és la clau principal de la taula proveïdor.

Taula 2.6. Segon fragment vertical de la relació PROVEIDOR

PROVEIDOR			
NIF*	**Nom**	**Adreça**	**Localitat**
33333333K	L'abastadora, SL	Pol. Ind. Polièdric, s/n	Lleida
44444444L	Proveïdora Ibèrica, SA	C/ del pi, 3	Lleida
55555555M	Supplies & Co. Ltd.	C/ del call, 4	Girona
66666666N	Assortiments de l'Onyar, SCP	Pg. De la ribera, s/n	Girona

*NIF és la clau principal de la taula proveïdor.

Fragmentacions mixtes

Les fragmentacions mixtes consisteixen a aplicar tant la fragmentació horitzontal com la vertical.

En funció de com es combinen les fragmentacions horitzontal i vertical, s'obtenen quatre tipologies mixtes:

1. **Fragmentació VH**. Es desenvolupa en primer lloc la fragmentació vertical, i a continuació l'horitzontal.
2. **Fragmentació HV**. S'aplica primer una divisió horitzontal i tot seguit es desenvolupa una altra de vertical sobre els fragments prèviament generats.
3. **Fragmentació semàntica**. La fragmentació de les relacions es fa alternant successivament fragmentacions horitzontals i verticals, però sempre tenint en compte el significat de les operacions més habituals que s'han de fer sobre les dades.
4. **Fragmentació simultània**. S'apliquen de manera simultània, i no pas seqüencial, la fragmentació horitzontal i la vertical, i la relació originària es transforma en una matriu, les cel·les de la qual són els fragments que s'han de distribuir. El nivell de fragmentació així obtingut normalment és molt elevat, la qual cosa no vol dir que sempre sigui més eficient.

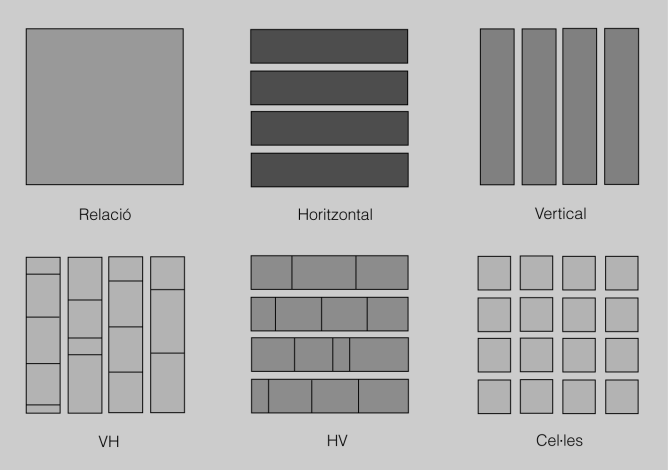
Grau de fragmentació

En fragmentar una BD ha de ser valorat el grau de fragmentació que assolirà aquesta BD, ja que aquest paràmetre influirà notablement en el rendiment del sistema a l'hora d'executar consultes.

El grau de fragmentació serà igual a zero, en absència de fragmentació, és a dir, quan es prenguin les relacions com a unitats de fragmentació. I el grau serà màxim quan cada tuple (en la fragmentació horitzontal) o cada atribut (en la fragmentació vertical) de cada relació constitueixin un fragment.
Davant d'aquest dos extrems, habitualment cal buscar solucions de compromís, tenint en compte la utilització que de la BD hagin de fer les aplicacions i els usuaris, des de tots els nodes de la xarxa.

La figura 2.10 mostra esquemàticament algunes possibilitats de fragmentació d'una relació.

Figura 2.10. Tipologies de fragmentació d'una relació



2.3.4 Transaccions i protocols de compromís

ACID properties és l'acrònim que s'obté amb la primera lletra de les quatre propietats de les transaccions en anglès: *atomicity, consistency, isolation, durability*.

Els SGBD en general, i els distribuïts en particular, han de garantir la integritat de les dades, mitjançant el compliment d'aquestes característiques:

- **Atomicitat**. O bé es fan correctament en la BD totes les operacions incloses en la transacció, o bé no se'n fa cap.
- **Consistència**. L'execució aïllada de la transacció (és a dir, sense la concurrència de cap altra transacció) conserva la consistència de la BD.
- **Aïllament**. Davant la concurrència de transaccions, el sistema garanteix que l'execució de cadascuna d'elles pressuposa, o bé que l'execució de les altres encara no ha començat, o bé que ja ha finalitzat completament.
- **Durabilitat**. Després de la finalització amb èxit d'una transacció, els canvis produïts en la BD romanen, fins i tot, en cas de fallades del sistema.

Les BD distribuïdes tenen els mateixos requisits transaccionals que les BD centralitzades. Però, evidentment, és més difícil garantir l'atomicitat i l'aïllament de les transaccions globals en un sistema distribuït que no pas les transaccions ocasionades en un altre de centralitzat.

Per tal de garantir la integritat de les dades en l'execució de transaccions, els SGBD distribuïts executen els anomenats protocols de compromís.

Compromís en dues fases

Els protocols de compromís serveixen per garantir que tots els nodes del sistema, implicats en l'execució d'una mateixa transacció, coincideixin en el resultat final obtingut.

El protocol de compromís més senzill, però que a la vegada és també un dels més utilitzats, s'executa en dues fases:

- **1a fase**. Un dels servidors dels nodes implicats en la transacció, que actua com a coordinador, pregunta als servidors dels altres nodes si estan en condicions de confirmar la transacció. Aleshores cadascun d'aquests servidors fa les comprovacions necessàries (per exemple, en matèria de restriccions d'integritat) per tal de donar una resposta. El servidor coordinador confirmarà la transacció si obté una resposta afirmativa per part de cadascun dels altres servidors implicats. En cas contrari, haurà de cancel·lar la transacció.
- **2a fase**. El servidor que actua com a coordinador envia un missatge a la resta de servidors implicats en la transacció comunicant-los si han de confirmar o cancel·lar la transacció.

Aquest protocol és molt sensible a la caiguda dels nodes implicats, a les fallades de la xarxa, o fins i tot, a la disminució de la seva velocitat, atès el nombre de missatges a intercanviar entre els nodes implicats en cada transacció.

Però el problema potencial més important d'aquest protocol és la possibilitat de bloqueig del sistema. Si el servidor que actua com a coordinador cau durant el procés, o s'interrompen les comunicacions amb ell, la transacció pot quedar activa en la resta de servidors, la qual cosa acabaria per produir un bloqueig del sistema, o d'alguns dels seus nodes, ja que aquests servidors no haurien de cancel·lar unilateralment l'operació, ja que no sabrien si la resta de servidors han confirmat ja els canvis i haurien d'esperar indefinidament fins al restabliment de les comunicacions amb el servidor coordinador de la transacció.

Compromís en tres fases

Per a moltes aplicacions el problema latent del bloqueig durant l'execució del protocol de compromís en dues fases no és acceptable, per la qual cosa s'han anat desenvolupat protocols alternatius, que si bé comporten certs avantatges, tampoc no estan exempts de problemes.

El protocol de compromís en tres fases evita alguns inconvenients del protocol de compromís en dues fases, del qual es pot considerar una extensió. Però, al mateix temps, comporta uns increments de complexitat i de sobrecàrrega tals, que no és gaire utilitzat.

Aquest protocol pressuposa que no pot tenir lloc una fragmentació de la xarxa, i que mai no fallaran més d'un nombre predeterminat de nodes. Aleshores s'evita la possibilitat de bloqueig, afegint-hi una tercera fase addicional, en la qual s'impliquen uns quants nodes addicionals al que actua com a coordinador en la presa de decisió del compromís.

Model Entitat-Relació

Carlos Manuel Martí Hernández

Índex

Introducció

Un model de dades consisteix en un conjunt d'eines conceptuals per descriure les dades, les seves interrelacions, el seu significat, i les limitacions necessàries per tal de garantir-ne la coherència.

En aquesta unitat estudiarem el model de dades més àmpliament utilitzat, el model Entitat-Relació (o, abreujadament, model ER). El model ER és un model de dades d'alt nivell. Es basa en una percepció del món real que es tradueix en una col·lecció d'objectes anomenats *entitats* (*entities*), i de relacions (*relationships*) entre aquelles.

El seu èxit és degut, probablement, al fet que la seva notació es basa en una sèrie de diagrames molt senzills i entenedors. Per aquest motiu, actualment, la majoria d'eines de disseny de bases de dades (BD) fan servir els conceptes del model ER.

En l'apartat "Conceptes del model entitat-relació. Entitats. Relacions", s'estudien aquestes estructures bàsiques, les quals es corresponen, fonamentalment, amb les proposades en la formulació original del model. La utilització d'aquests elements més simples (sobretot entitats, atributs i interrelacions) pot resultar especialment útil en la comunicació entre els dissenyadors de BD i els usuaris.

En l'apartat "Diagrames entitat-relació", s'examina en què consisteix el disseny de bases de dades, les fases en què es desglossa, i les decisions que cal prendre durant les diferents etapes del disseny conceptual. També s'examinen les anomenades *extensions del model ER*, que comprenen algunes estructures més complexes (generalitzacions, especialitzacions i entitats associatives), les quals ens permetran modelitzar qualsevol situació del món real que ens interessi.

Al llarg de tota la unitat, es recomana analitzar amb deteniment els exemples que s'hi exposen, ja que permetran comprendre millor els conceptes teòrics que il·lustren.

Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Dissenya models lògics normalitzats interpretant diagrames entitat/relació.
    - Identifica, selecciona i ordena la informació que ha de contenir la base de dades, segons els requeriments de l'usuari.
    - Analitza la informació a representar i decideix el disseny per a la base de dades, segons els requeriments de l'usuari.
    - Defineix les entitats: nom, atributs, dominis dels atributs i camps claus.
    - Defineix les relacions: nom, atributs i grau.
    - Realitza el disseny lògic de la base de dades utilitzant el model Entitat-Relació.
    - Utilitza eines gràfiques per a representar el disseny lògic.

1. Conceptes del model Entitat-Relació. Entitats. Relacions

Les estructures bàsiques del model Entitat-Relació (model ER) es corresponen, fonamentalment, amb els conceptes proposats en la formulació original d'aquest model que va fer el Dr. Peter Pin Shan Chen en el seu treball *The Entity-Relationship Model - Toward a Unified View of Data* l'any 1976.

L'enginyeria del programari és aquella branca de l'enginyeria que permet elaborar programari de qualitat i amb un cost efectiu.

La notació d'aquestes construccions és fonamentalment diagramàtica, tot i que en alguns casos es pot afegir alguna especificació textual. Aquests diagrames són generalment coneguts com a **diagrames ER** (en referència al model) o **diagrames Chen** (en referència a l'autor).

Els diagrames ER són molt eficaços a l'hora de modelitzar la realitat (empresarial o de qualsevol índole) per obtenir un esquema conceptual entenedor. A causa d'això, moltes de les eines d'enginyeria del programari assistida per ordinador (eines CASE), que també ajuden en el disseny de BD, utilitzen els conceptes del model ER en els seus diagrames.

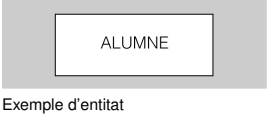
CASE, *computer aided software engineering*, en anglès

Actualment, tant en la bibliografia especialitzada com en les eines CASE de disseny de BD, es poden trobar petites variacions a partir de la notació original proposada inicialment pel Dr. Chen.

La utilització dels elements més simples del model ER, entitats, atributs i interrelacions, i potser d'alguna altra construcció addicional, com ara les entitats febles, poden ser de gran utilitat en la comunicació entre els dissenyadors de BD i els usuaris.

1.1 Entitats i atributs

Una **entitat** és alguna cosa que existeix en el món real, distingible de la resta de coses, i de la qual ens interessen algunes propietats.



Exemple d'entitat

Les entitats poden tenir una existència física, com per exemple una persona, un cotxe o un llibre, però també poden consistir en conceptes més abstractes, com ara una assegurança o un deute.

Exemple d'entitat

Imaginem que estem dissenyant la BD d'un institut de secundària, dedicat a l'ensenyament de diferents cicles formatius de formació professional. Cada persona concreta, alumna de l'institut esmentat, existeix en el món real i, per tant, es pot considerar una entitat.

Així, doncs, amb el terme *entitat* es pot fer referència a un objecte específic del món real, però també a un conjunt d'objectes semblants, dels quals ens interessen les mateixes característiques. Per tant, hem de distingir:

Les entitats en els diagrames ER es representen amb un rectangle.

- **Entitats-instància**, com a objectes concrets del món real (per exemple, l'alumne *Manel Riba* és una entitat-instància).
- **Entitats-tipus**, com a conjunts d'entitats-instància (per exemple, l'entitat tipus *alumne*).

Anomenem **atributs** les característiques que ens interessen de les entitats.

Habitualment, només ens interessarà modelitzar una part dels atributs d'una entitat, ja que hi podrà haver dades que només seran d'utilitat en àmbits molt específics.

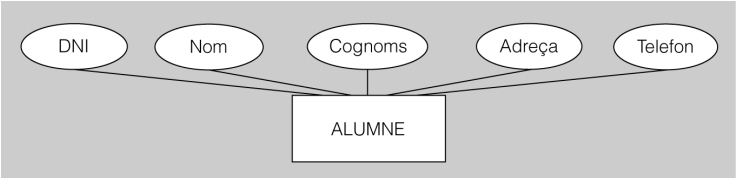
Exemples d'atributs

En una entitat-instància referent als alumnes d'un institut (figura 1.1), ens pot resultar interessant recollir certes dades personals, per tal d'identificar correctament els alumnes a l'hora de comunicar-nos amb ells, o d'expedir notes i títols acadèmics, com ara el DNI, el nom, els cognoms, l'adreça, el telèfon, etc.

En canvi, altres dades de la mateixa entitat no seran d'interès per a nosaltres, encara que sí que ho puguin ser per a una BD que pertanyi a un altre àmbit. Per exemple, des d'un punt de vista sanitari, podria ser interessant registrar l'alçada, el pes o el grup sanguini d'aquestes mateixes persones.

Els atributs en els diagrames ER es representen amb una el·lipse.

FIGURA 1.1. Exemples d'atributs



1.1.1 Domini dels atributs

Els atributs de cada entitat-instància adopten valors concrets. Aquests valors han de ser vàlids.

Perquè un valor d'un atribut sigui vàlid, ha de pertànyer al conjunt de valors acceptables per a l'atribut en qüestió. Aquest conjunt de valors vàlids s'anomena **domini**.

Exemples de domini i de valors vàlids

El domini de l'atribut Nom de l'entitat ALUMNE podria consistir en el conjunt de totes les cadenes de caràcters possibles d'una longitud determinada, tot excloent les xifres i els caràcters especials. Serien valors vàlids per a l'atribut Nom, definit d'aquesta manera,

"Laia", "Pol", etc. En canvi, no ho serien, per exemple, una data, un nombre o una cadena de caràcters que n'inclogués algun d'especial, com ara "Mariona".

1.1.2 Valor nul dels atributs

Els atributs d'una entitat-instància poden no tenir cap valor per a algun atribut concret. En aquests casos, també es diu que l'atribut té **valor nul**.

Exemple de valor nul

Pot passar que un alumne no tingui telèfon. Aleshores, l'atribut Telefon de l'entitat ALUMNE no contindrà cap valor o, dit d'una altra manera, tindrà un valor nul.

1.1.3 Atributs simples i compostos

Es poden considerar dos tipus diferenciats d'atributs: els atributs simples i els compostos.

Un **atribut simple** no es pot dividir en parts més petites sense que això comporti la pèrdua del seu significat.

Exemple d'atribut simple

L'atribut Nom és un atribut simple, perquè el seu significat és indivisible (encara que en alguns casos emmagatzemi noms compostos, com ara Joan Manel), i per tant no té sentit dividir el seu valor en cadenes de caràcters més petites per tractar-les per separat.

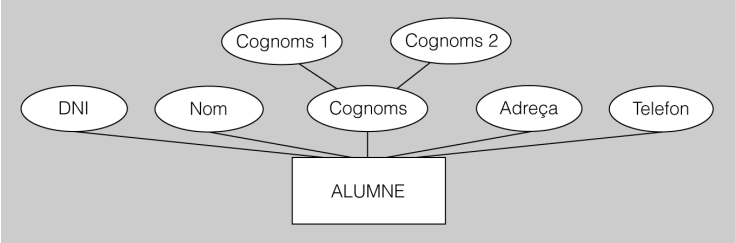
Un **atribut compost** és el que està subdividit en parts més petites (que també tenen la consideració d'atributs), les quals tenen un significat propi.

Exemple d'atribut compost

L'atribut Cognoms es pot tractar com un atribut compost (figura 1.2), perquè es pot dividir en dues parts més petites (dos atributs, en definitiva) que emmagatzemin, una, el primer cognom, i l'altra el segon cognom. Aquests dos atributs es poden tractar per separat sense problemes.

Com que moltes persones estrangeres només tenen un cognom, en aquest exemple, l'atribut Cognom1 sempre tindrà algun valor per a qualsevol entitat-instància, però l'atribut Cognom2 haurà d'admetre valors nuls.

FIGURA 1.2. Exemple d'atribut compost



Pot resultar interessant utilitzar atributs compostos si ens consta que els usuaris es referiran, de vegades, a l'atribut globalment considerat, i de vegades als seus components per separat.

D'altra banda, els atributs compostos agrupen els atributs relacionats, estructurant-los jeràrquicament, de manera que normalment contribueixen a la comprensibilitat dels models.

1.1.4 Atributs monovaluats i multivaluats

En el model relacional...

... els atributs resultants només poden ser simples i monovaluats. Però el model ER també pot servir per fer derivar el model conceptual resultant cap a altres models lògics que sí que acceptin els atributs compostos o els multivaluats.

Una altra manera de caracteritzar els atributs és en funció de si són atributs monovaluats o multivaluats.

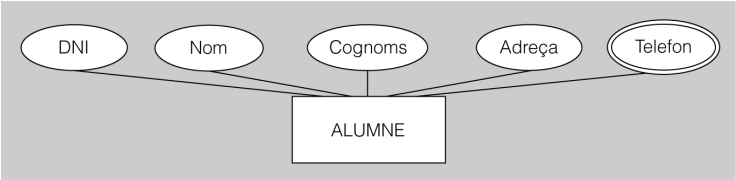
Un **atribut monovaluat** és el que només pot emmagatzemar, com a màxim, un sol valor per a cada entitat instància concreta, en un moment determinat.

Exemple d'atribut monovaluat

És evident que cada persona només pot tenir un DNI vàlid. Per tant, l'atribut DNI de l'entitat ALUMNE s'haurà de tractar necessàriament com un atribut monovaluat.

Un **atribut multivaluat** pot emmagatzemar, per a cada entitat instància concreta, diferents valors al mateix temps.

FIGURA 1.3. Exemple d'atribut multivaluat



Exemple d'atribut multivaluat

En el món real, una persona pot tenir més d'un telèfon (figura 1.3). Per exemple, pot disposar d'un telèfon fix al domicili particular, d'un altre a la feina, i a més pot tenir un telèfon mòbil. Per tant, l'atribut Telefon de l'entitat ALUMNE es pot tractar com un atribut multivaluat.

Els atributs multivaluats es representen en els diagrames ER amb una el·lipse de doble traç.

1.1.5 Cardinalitat dels atributs

Si cal, es poden especificar a continuació del nom de l'atribut, entre parèntesis i separats per comes, el límit màxim i el mínim de valors que s'han d'emmagatzemar, això és la **cardinalitat dels atributs**. I es poden presentar les opcions següents:

- NomAtribut (1, 1): atribut univaluat obligatori (valor per defecte, si no s'especifica res).
- NomAtribut (0, 1): atribut univaluat opcional (admet valors nuls).
- NomAtribut (1, n): atribut multivaluat obligatori (no admet valors nuls).
- NomAtribut (0, n): atribut multivaluat opcional (admet valors nuls).

Exemples de límits superior i inferior

Seguint amb el cas de l'atribut Telefon, es podria establir, per exemple, un límit inferior a 0 (ja que un alumne pot no disposar de cap telèfon durant un període de temps determinat) o a 1 (si volem obligar l'alumne a donar un telèfon de contacte, encara que no sigui el seu, sinó el d'un familiar, amic o veí).

I es podria limitar el nombre màxim de telèfons a emmagatzemar, per exemple, a 2 (si es preveu la possibilitat, força habitual, de tenir un fix i un mòbil) o a 3 (si, a més, considerem la possibilitat de registrar el telèfon del centre de treball).

1.1.6 Atributs derivats

Es diu que un **atribut** és **derivat** quan el seu valor es pot calcular a partir d'altres atributs o bé d'altres entitats interrelacionades.

Quan un atribut serveix per calcular el valor d'un atribut derivat, se'l considera atribut base d'aquest.

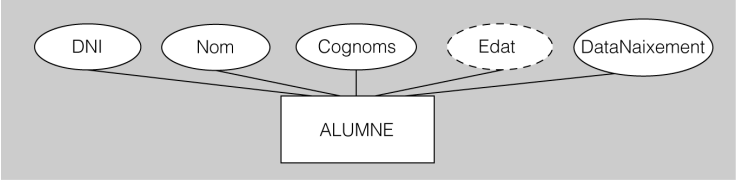
Exemples d'atribut derivat

Podríem necessitar saber quina és l'edat en anys dels alumnes, per tal de permetre'ls sortir o no de l'institut durant els períodes d'esbarjo, en funció d'aquella (figura 1.4). Si l'entitat ALUMNE té un atribut anomenat DataNaixement, en podríem modelitzar un altre de derivat, anomenat Edat, que es calculés a partir de la data actual (prenent la data del sistema) de la data de naixement (registrada en l'atribut DataNaixement).

Els atributs derivats en els diagrames ER es representen amb una el·lipse de traç discontinu.

També podríem necessitar saber el nombre total d'assignatures a les quals està matriculat cada alumne. Podríem establir un atribut derivat anomenat NombreAssignatures, el valor del qual es calcul és en funció del nombre d'ocurrències d'una altra entitat-tipus anomenada ASSIGNATURA interrelacionades amb cadascuna de les instàncies de l'entitat ALUMNE.

FIGURA 1.4. Exemple d'atribut derivat



Els atributs derivats constitueixen una redundància, és a dir, una repetició normalment innecessària de dades. Per aquest motiu, les dades dels atributs derivats inclosos en els diagrames ER no s'acostumen a emmagatzemen (i molt especialment si traduïm aquest esquema conceptual a l'esquema lògic més freqüentment utilitzat, és a dir, al model relacional), sinó que es calculen quan és necessari.

La clau primària en els diagrames ER es representa subratllant els atributs que la formen.

1.1.7 Clau primària

Una entitat instància concreta s'ha de poder distingir de la resta d'objectes del món real. Per tant, qualsevol modelització ER ha d'indicar, per a tota entitat tipus, un atribut o un conjunt d'atributs que la permeti identificar unívocament.

L'atribut o el conjunt d'atributs que identifiquen unívocament les entitats instància s'anomenen **clau primària** de l'entitat.

Exemples de clau primària

Podríem seleccionar l'atribut DNI de l'entitat ALUMNE com a clau primària (figura 1.5), ja que sabem que en el món real no han d'existir dos documents d'identitat iguals i, per tant, ens servirà amb tota seguretat per distingir qualsevol alumne de la resta.

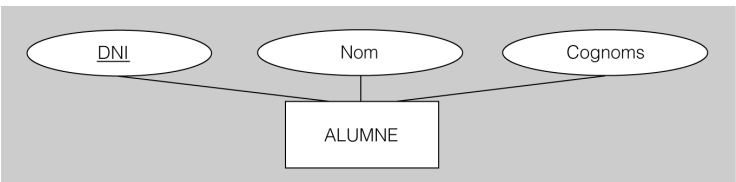
En els països on no existeixen documents d'identitat, com els anglosaxons, hauríem d'optar per una solució alternativa. Podríem afegir al nostre model un atribut identificador, de tipus codi, encara que aquest no existís al món real: CodiAlumne.

O també podríem considerar com a clau primària un conjunt d'atributs tal que fes impossible o, si més no, molt difícil, que es repetissin les combinacions dels seus valors per a diferents entitats-instància: Nom+Cognoms+Telefon.

Notacions ER alternatives

Actualment no existeix cap notació estanderditzada universalment per representar els esquemes del model ER. Cada recurs bibliogràfic o cada programari de disseny presenta, doncs, variacions i ampliacions sobre la reduïda notació proposada originàriament per Peter Chen.

FIGURA 1.5. Exemple de clau primària



1.1.8 Notació

El model ER ens permet representar entitats i atributs mitjançant una senzilla notació diagramàtica.

En aquesta representació respectarem les característiques següents:

- Com a regla general, no farem servir accents ni caràcters especials, només lletres i xifres.
- Representarem les entitats tipus escrivint el seu nom en majúscules i en singular, a dins d'un rectangle.
- Representarem cada atribut escrivint el seu nom amb la primera lletra en majúscula i la resta en minúscules, dins d'una el·lipse unida amb un guió amb el rectangle que representa l'entitat tipus de la qual formen part:
  - Si un atribut té un nom compost, cada nom començarà amb majúscula per tal de fer-lo més llegidor. Per exemple, TelefonFix, TelefonMobil.
  - Si el nom d'un atribut correspon a unes sigles, ha d'anar íntegrament en majúscules, com ara DNI (document nacional d'identitat).
  - Les el·lipses dels atributs en què es pot descompondre un atribut han d'anar unides amb un guió amb l'el·lipse de l'atribut compost.
  - L'el·lipse d'un atribut multivaluat estarà formada per un traç doble.
  - Els límits d'un atribut multivaluat, en cas d'existir, s'han d'especificar a continuació del nom de l'atribut, entre parèntesis i separats per una coma.
  - L'el·lipse d'un atribut derivat estarà formada per un traç puntejat.
  - Els atributs que formen part d'una clau primària han d'anar subratllats.

Si hem d'establir qualsevol altra característica de les dades que no tingui predefinida una notació diagramàtica concreta, haurem d'afegir al diagrama les especificacions textuals necessàries.

1.2 Interrelacions

Una **interrelació** consisteix en una associació entre dues o més entitats.

Amb el terme *interrelació* podem fer referència tant a una associació concreta entre diferents entitats instància, com també a una associació de caire més genèric, entesa com a un conjunt d'associacions de la mateixa tipologia, entre diferents entitats tipus.

Exemple d'interrelació

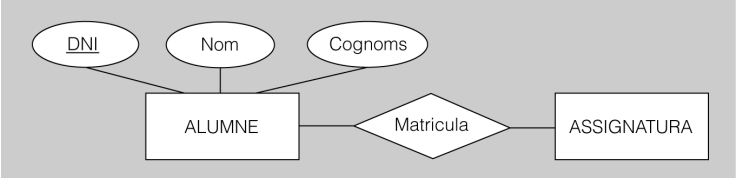
Ja coneixem l'entitat ALUMNE (figura 1.6). Però per dissenyar la BD del nostre institut necessitarem més entitats. Per exemple, serà convenient disposar d'una entitat per emmagatzemar les assignatures que conformin l'oferta formativa del centre. Podem anomenar, aquesta nova entitat, ASSIGNATURA.

En un centre educatiu, els alumnes es matriculen d'assignatures. Doncs bé, per modelitzar aquesta característica del món real, no necessitarem cap nova entitat. Només ens caldrà establir una associació entre les dues entitats de què disposem, ALUMNE i ASSIGNATURA, mitjançant una interrelació.

D'aquesta manera, modelitzarem l'associació de cada alumne amb totes les assignatures en què estigui matriculat, i, recíprocament, de cada assignatura amb tots els estudiants respectius. Podríem anomenar aquesta interrelació, per exemple, Matricula.

Les interrelacions en els diagrames ER es representen amb un rombe.

Figura 1.6. Exemple d'interrelació



1.2.1 Atributs de les interrelacions

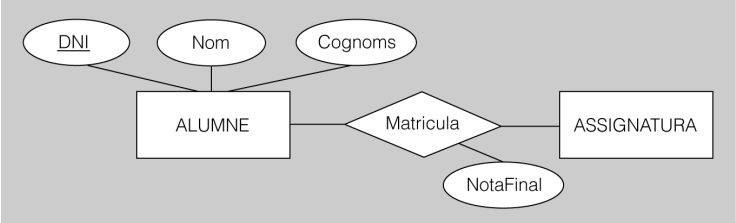
De vegades, ens pot interessar reflectir algunes característiques de determinades interrelacions. La manera de fer-ho és afegir els atributs necessaris, com faríem si treballéssim amb entitats. Aquests atributs són els **atributs de la interrelació**.

Exemple d'atribut d'interrelació

La secretaria del nostre institut necessitarà tenir constància, com a mínim, de la nota final obtinguda per cada alumne en cada assignatura en què s'hagi matriculat alguna vegada (figura 1.7).

La manera més senzilla de fer-ho seria afegir, a la interrelació Matricula, un atribut anomenat, per exemple, NotaFinal, que servís per emmagatzemar aquesta dada per a cada associació existent entre instàncies de les entitats ALUMNE i ASSIGNATURA.

Figura 1.7. Exemple d'atribut d'interrelació



Les propietats dels atributs de les interrelacions són idèntiques a les descrites prèviament en relació als atributs de les entitats.

1.2.2 Grau de les interrelacions

El **grau d'una interrelació** depèn del nombre d'entitats que aquesta associa.

Trobareu un exemple d'interrelació de grau dos (només associa dues entitats: ALUMNE i ASSIGNATURA) en la figura del subapartat "Atributs de les interrelacions".

Les interrelacions de grau dos també s'anomenen *binàries*. I les de grau superior a dos s'anomenen genèricament *n-àries*. Les interrelacions n-àries de grau tres també poden ser anomenades *ternàries*, i les de grau quatre, *quaternàries*.

Exemple d'interrelació de grau tres

Fins ara, la interrelació Matricula només permet emmagatzemar una matrícula de cada alumne en cada assignatura, i el seu atribut NotaFinal només permet reflectir una sola nota final de curs (figura 1.8).

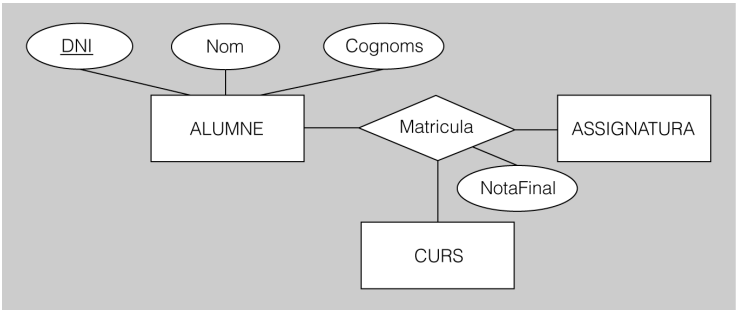
Però aquest esquema no permet modelitzar el fet que un alumne es pot haver de matricular més d'un cop d'una mateixa assignatura (i obtenir una nota final en cada nova matrícula) fins a obtenir una qualificació igual o superior a l'aprovat.

Una manera d'aconseguir representar aquesta característica del món real consistiria a afegir, en nostre disseny, una nova entitat que fes referència a l'element temporal. La podríem anomenar CURS, per exemple.

I, a continuació, només cal que la interrelació Matricula (tot conservant l'atribut NotaFinal) interrelacioni tres entitats: ALUMNE, ASSIGNATURA i CURS.

I el nou esquema ja permetrà registrar matrícules successives d'un mateix alumne en una mateixa assignatura, però al llarg de diferents cursos acadèmics, amb les respectives qualificacions obtingudes.

FIGURA 1.8. Exemple d'interrelació de grau tres



1.2.3 Connectivitat de les interrelacions

La **connectivitat** (també anomenada *cardinalitat*) d'una interrelació indica el tipus de correspondència que hi ha entre les ocurrències de les entitats que ella mateixa permet associar.

Interrelacions binàries

Tractant-se d'interrelacions binàries, la cardinalitat expressa el nombre màxim d'instàncies d'una de les entitats amb les quals una instància de l'altra entitat pot estar associada segons la interrelació en qüestió.

Una N en un costat de la interrelació també es representa freqüentment amb un asterisc (*)

Les interrelacions binàries poden oferir tres tipus de connectivitat:

- Un a un (1:1)
- Un a uns quants (1:N)
- Uns quants a uns quants (N:M)

Un 1 al costat d'una entitat indica que, com a màxim, només una de les seves instàncies (la qual podrà variar en cada cas) tindrà la possibilitat d'estar associada amb cadascuna de les instàncies de l'altra entitat.

Si més d'un extrem de la interrelació té una N, per raó d'elegància es representa amb consonants successives, començant per M: M, N, P, Q, etc.

La cardinalitat 1 també es pot representar convertint la línia que uneix la interrelació amb l'entitat en una fletxa que apunti cap a l'entitat.

En canvi, una N (o una M) al costat d'una entitat indica que serà una pluralitat de les seves instàncies (les quals també podran variar en cada cas) la que tindrà la possibilitat d'estar associada amb cadascuna de les instàncies de l'altra entitat.

La cardinalitat N (o M) també es pot representar amb una fletxa de doble punta que vagi de la interrelació cap a l'entitat.

És molt important adonar-se que, independentment del tipus de connectivitat, una interrelació només permet associar una sola vegada unes entitats instància determinades entre elles.

Exemple de connectivitat 1:1

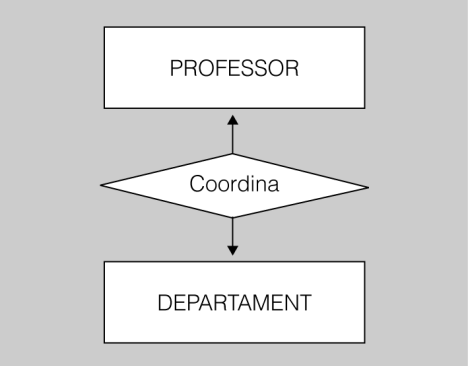
Com els alumnes, els professors també formen part de la comunitat educativa (a més d'altres col·lectius que de moment no necessitem tenir en compte, veure figura 1.9).

Els professors s'organitzen professionalment en departaments, en funció de la seva especialitat (per exemple: matemàtiques, filosofia, informàtica, etc.).

Per tal de reflectir aquestes dues realitats, haurem d'afegir al nostre model dues noves entitats: PROFESSOR i DEPARTAMENT.

Cada departament és coordinat per un sol professor, i un professor només pot coordinar un sol departament. Per tal de reflectir aquesta circumstància, haurem d'establir una interrelació entre les entitats PROFESSOR i DEPARTAMENT amb cardinalitat 1:1. Podem anomenar la nova interrelació Coordina.

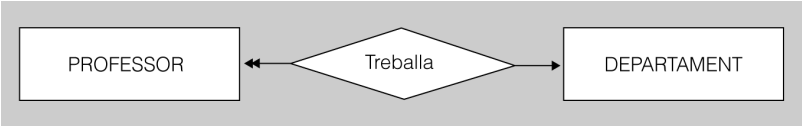
FIGURA 1.9. Exemple de connectivitat 1:1



Exemple de connectivitat 1:N

Ja sabem que tot professor d'institut està assignat a un departament (figura 1.10). Però encara ens falta establir una nova interrelació entre PROFESSOR i DEPARTAMENT que reflecteixi aquesta realitat. La podem anomenar Treballa.

FIGURA 1.10. Exemple de connectivitat 1:N



Com que cada professor només pot treballar a un departament, al costat de la interrelació que connecta l'entitat DEPARTAMENT, hi anirà un 1.

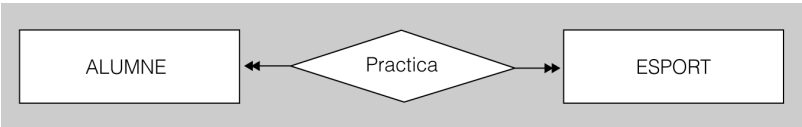
Inversament, com que cada departament pot tenir més d'un professor assignat, al costat de la interrelació que connecta l'entitat PROFESSOR, hi anirà una N.

Exemple de connectivitat N:M

Imaginem que al nostre institut s'organitzen activitats esportives extraescolars (figura 1.11). Cal incorporar al nostre model una nova entitat (que podem anomenar ESPORT, per exemple) i una nova interrelació que l'associï amb l'entitat ALUMNE (que podem anomenar Practica).

Els alumnes tenen la possibilitat d'inscriure's com a practicants d'un o més esports. I els esports, evidentment, poden ser practicats per més d'un alumne. Per tant, a un costat de la interrelació, hi anirà una N, i a l'altre una M.

FIGURA 1.11. Exemple de connectivitat M:N



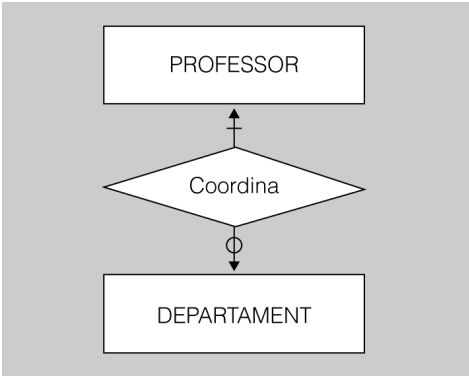
Dependències d'existència a les interrelacions binàries

Les entitats opcionals en els diagrames ER es representen superposant un cercle a la línia que uneix l'entitat a la relació.

Les entitats obligatòries en els diagrames ER es representen superposant un petit guió a la línia que uneix l'entitat a la relació.

De vegades, una entitat instància només té sentit si existeix com a mínim una altra entitat instància associada amb ella mitjançant una interrelació binària determinada. En aquests casos, es diu que la darrera entitat és una **entitat obligatòria** per a la interrelació. Altrament, es diu que es tracta d'una **entitat opcional** per a la interrelació.

FIGURA 1.12. Exemple de dependència d'existència



Un cercle en la línia de connexió entre una entitat i una interrelació indica que l'entitat és opcional en la interrelació. L'obligatorietat d'una entitat en una interrelació s'indica amb un guionet perpendicular a la línia que uneix l'entitat amb la interrelació. Si no es consigna ni un cercle ni una línia perpendicular, es considera que la dependència d'existència és desconeguda.

Tindrem en compte aquesta característica només pel que faci a les interrelacions binàries, però no a les n-àries.

Exemple de dependències d'existència

L'entitat PROFESSOR és obligatòria en la interrelació Coordina (figura 1.12). D'aquesta manera, s'indica que no pot existir un departament que no tingui cap professor que faci de coordinador del departament. L'entitat DEPARTAMENT, en canvi, és opcional en la interrelació Coordina, ja que la majoria dels professors no coordinaran cap departament.

Interrelacions ternàries i n-àries

La **cardinalitat de les interrelacions n-àries** expressa el nombre màxim d'instàncies d'una de les entitats amb les quals una combinació concreta d'instàncies de les altres entitats pot estar associada segons la interrelació en qüestió.

Les interrelacions ternàries poden oferir quatre tipus de connectivitat:

- 1:1:1
- 1:1:N
- 1:M:N
- M:N:P

En què 1 indica que com a màxim només una de les seves instàncies (la qual podrà variar en cada cas) tindrà la possibilitat d'estar associada amb cada combinació concreta d'instàncies de les altres entitats. I en què N, M o P indica que diverses

instàncies poden estar relacionades amb cada combinació d'instàncies de les altres entitats.

En general, les interrelacions n-àries poden oferir n + 1 tipus de connectivitat. Així, per exemple, una interrelació quaternària (és a dir, n-ària de grau 4) tindrà cinc tipus possibles de cardinalitat (perquè en aquest cas n + 1 = 4 + 1 = 5).

Exemple de connectivitat M:N:P

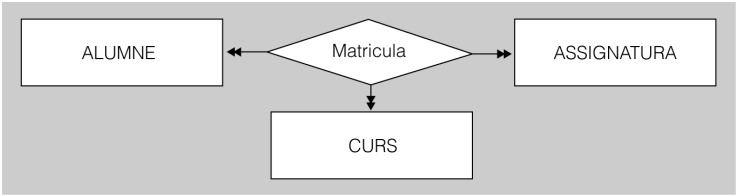
Ja coneixem la interrelació Matricula, que associa les entitats ASSIGNATURA, ALUMNE i CURS. Però encara no hem establert les seves cardinalitats (figura 1.13).

Un alumne, en un curs determinat, es pot matricular d'unes quantes assignatures. Per tant, al costat de l'entitat ASSIGNATURA, hi haurà una N (però si només es pogués matricular d'una sola assignatura, hi hauria d'haver un 1).

Un alumne es pot haver de matricular d'una mateixa assignatura durant més d'un curs acadèmic, fins que la superi. Per tant, al costat de l'entitat CURS, hi haurà una N (però si només fos possible matricular-se un cop d'una assignatura, hi hauria d'haver un 1).

I és evident que, durant un curs acadèmic, diferents alumnes poden estar matriculats en una mateixa assignatura. Per tant, al costat de l'entitat ALUMNE, també hi haurà una N (però si només s'acceptés la matrícula d'un alumne per assignatura i curs, hi hauria d'haver un 1).

FIGURA 1.13. Exemple de connectivitat M:N:P



Límits de cardinalitat

De vegades, pot resultar útil establir **límits mínims i màxims a les cardinalitats de les interrelacions**. Per fer-ho, només cal afegir una etiqueta del tipus *mín..màx*, per tal d'expressar els límits respectius, al costat de la línia que uneix cada entitat amb la interrelació.

Els valors *mín* i *màx* podran tenir els valors següents:

- Zero, per indicar la possibilitat que no existeixi cap associació entre instàncies.
- Qualssevol nombres enters, per indicar un límit mínim o màxim concret de possibilitats d'associació entre instàncies.
- Un asterisc (*), per indicar la possibilitat d'un nombre il·limitat d'associacions entre instàncies.

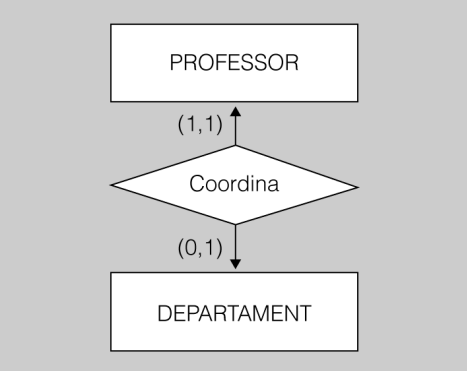
Exemple de límits de cardinalitat

Ja coneixem la interrelació Coordina, que associa les entitats PROFESSOR i DEPARTAMENT amb cardinalitat 1:1 (figura 1.14).

Cada departament ha de tenir assignat un, i només un, professor que el coordini. Per tal de reflectir aquesta limitació, haurem d'afegir l'etiqueta 1..1 al costat de la línia que uneix l'entitat PROFESSOR amb la interrelació Coordina.

D'altra banda, no tots els professors s'encarreguen de coordinar un departament (de fet, el més freqüent és que no se n'encarreguin). I si ho fan, només es poden encarregar de la coordinació d'un. Per tal de reflectir aquesta limitació haurem d'afegir l'etiqueta 0..1 al costat de la línia que uneix l'entitat DEPARTAMENT amb la interrelació Coordina.

FIGURA 1.14. Exemple de límits de cardinalitat



1.2.4 Interrelacions recursives

Tot i que altres interrelacions associen instàncies de diferents entitats, aquesta característica no és aplicable a les interrelacions recursives.

Una **interrelació recursiva** associa les instàncies d'una entitat amb altres instàncies de la mateixa entitat.

Es diu que una interrelació recursiva és de grau 2 (o binària) si només hi participa una entitat, la qual es relaciona amb ella mateixa.

Les interrelacions recursives en els diagrames ER es representen connectant una mateixa entitat més d'una vegada, mitjançant una única relació.

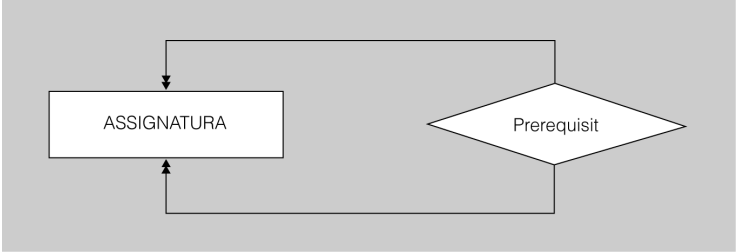
Exemple d'interrelació recursiva binària

Imaginem que al nostre institut s'estableix, com a requisit per cursar certes assignatures, el fet d'haver superat prèviament una altra o unes altres assignatures (figura 1.15).

Podríem modelitzar aquesta situació mitjançant una interrelació recursiva binària sobre l'entitat ASSIGNATURA, i anomenar-la, per exemple, Prerequisit.

Si considerem que cada assignatura pot tenir més d'una altra assignatura com a prerequisit, i que al mateix temps cada assignatura pot ser prerequisit d'una pluralitat d'assignatures, la cardinalitat hauria de ser M:N.

Figura 1.15. Exemple d'interrelació recursiva binària



Si en una interrelació recursiva participen, addicionalment, més entitats, parlarem d'interrelacions recursives de grau 3 (o ternàries), de grau 4 (o quaternàries), i així successivament.

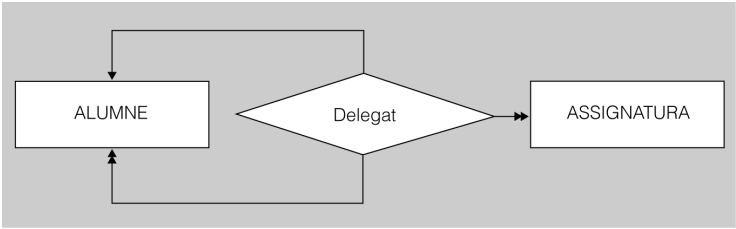
Exemple d'interrelació recursiva ternària

Cada alumne del nostre institut té un delegat per assignatura, que el representa davant del professorat que la imparteix, per tal de fer més fluïdes les comunicacions sobre les qüestions relatives al funcionament d'aquella que no siguin d'índole personal (figura 1.16).

Podríem modelitzar aquesta situació mitjançant una interrelació recursiva ternària, anomenada, per exemple, Delegat. Una ocurrència d'aquesta interrelació associarà un alumne que actuarà com a delegat en l'àmbit d'una assignatura, un altre alumne que actuarà com a estudiant de la mateixa assignatura, i l'assignatura en qüestió.

La connectivitat és 1:M:N. En els dos costats de l'entitat ALUMNE hi ha un 1 i una N, perquè, d'una banda, un delegat d'una assignatura pot representar més d'un estudiant (N), i, d'una altra banda, un estudiant d'una assignatura només pot tenir un sol representant en l'àmbit d'aquesta (1). I al costat de l'entitat ASSIGNATURA hi ha una M, perquè un alumne pot actuar com a representant d'un altre en diferents àmbits, corresponents a diferents assignatures.

Figura 1.16. Exemple d'interrelació recursiva ternària



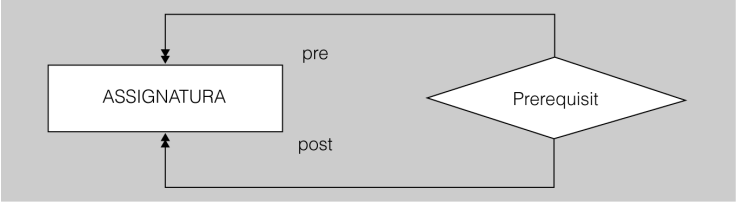
En una interrelació no recursiva, el paper, o rol, que interpreta cada entitat implicada se sobreentén i, per tant, no cal especificar-lo.

En el cas de les interrelacions recursives, pot tenir importància especificar els diferents papers o **rols** que interpreten les instàncies d'una mateixa entitat, si aquests rols no coincideixen plenament. Si el rol és exactament el mateix, no cal especificar-lo.

Exemple de diferenciació de rols

Podríem etiquetar les dues línies de la interrelació Prerequisit com a "pre" i "post", per exemple, per tal de modelitzar en el primer cas el rol de prerequisit acadèmic i, en el segon cas, el rol d'assignatura autoritzada (figura 1.17).

FIGURA 1.17. Exemple de diferenciació de rols



1.2.5 Notació

Com amb les entitats, la notació diagramàtica per representar les interrelacions i les seves propietats també és força senzilla:

- Tota interrelació es representa amb un rombe, que va unit, mitjançant línies, a totes les entitats que associa.
- Els atributs d'una interrelació, quan existeixen, es representen de la mateixa manera que els atributs d'una entitat.
- La connectivitat d'una interrelació es representa afegint una etiqueta amb un 1 o una N, segons calgui, a cadascuna de les línies que la uneix amb les entitats que hi participen.
- L'opcionalitat es representa superposant un cercle a la línia de connexió corresponent, i l'obligatorietat, superposant un petit guió perpendicular a la línia de connexió de què es tracti.
- Si cal establir límits (0, enter, *) a la cardinalitat d'una interrelació, s'ha d'afegir a cadascuna de les seves línies de connexió una etiqueta amb el límit inferior i el superior separats per dos punts seguits.
- La recursivitat d'una interrelació es representa fent arribar dues línies de connexió a la mateixa entitat. Si participen més entitats de la mateixa interrelació recursiva, s'hi han de fer arribar les línies de connexió corresponents des de la interrelació.
- Si cal fer una diferenciació dels rols d'una interrelació recursiva, s'ha d'afegir una etiqueta, amb l'especificació textual adequada, al costat de cadascuna de les línies de connexió.

1.3 Entitats febles

Les entitats que disposen d'un atribut o, si no, d'un conjunt d'atributs capaços d'establir una clau primària que serveixi per distingir cada instància de l'entitat de la resta d'ocurrències es poden anomenar, més específicament,*entitats fortes*.

Les **entitats febles** són aquelles que no disposen de prou atributs per a designar unívocament les seves instàncies. Per tal d'aconseguir-ho, han d'estar associades, mitjançant una interrelació, amb una entitat forta que les ajudi.

La interrelació entre una entitat feble i la seva forta associada és sempre de cardinalitat 1:N, i es resta l'1 al costat de l'entitat forta, i la N al costat de la feble.

Cada instància d'una entitat feble està associada amb una única ocurrència de l'entitat forta (per això és en el costat 1 de la interrelació), i així és possible completar-ne la identificació de manera única.

D'altra banda, l'entitat del costat 1 ha de ser obligatòria en la interrelació perquè, si no fos així, alguna instància de l'entitat feble podria no estar associada amb cap de les ocurrències de l'entitat forta i, aleshores, no es podria identificar completament.

Les entitats febles, doncs, no tenen clau primària, però sí un atribut (o un conjunt d'atributs) anomenat *discriminant*, que permet distingir entre elles totes les instàncies de l'entitat feble que depenen d'una mateixa instància de l'entitat forta.

Tot i no ser un cas gaire freqüent, es poden encadenar entitats febles, de tal manera que una entitat que actuï com a part feble en la interrelació que mantingui amb una altra entitat, pot actuar al mateix temps com a entitat forta respecte a una altra entitat que, al seu torn, la necessiti per identificar completament les seves instàncies.

Addicionalment a la interrelació que els serveix per identificar-se completament, les entitats febles poden participar en altres interrelacions, com qualsevol altra entitat.

Exemple d'entitat feble

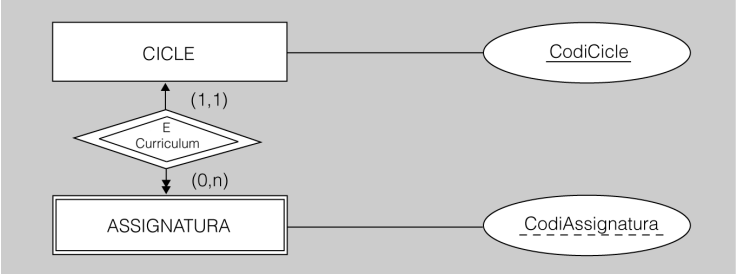
Ha arribat el moment d'establir una clau primària per a l'entitat ASSIGNATURA (figura 1.18). Podríem adoptar una codificació derivada de la utilitzada en els currículums oficials: C1, C2, C3, etc. (de Crèdit 1, Crèdit 2, i així successivament). Podríem anomenar aquest atribut CodiAssignatura. Però això no permetria distingir les assignatures dels diferents cicles formatius impartits en el nostre institut.

Per aconseguir la identificació inequívoca de cada crèdit, en primer lloc hauríem de comptar amb una nova entitat anomenada, per exemple, CICLE, per emmagatzemar tots els cicles impartits al centre. Aquesta entitat seria forta, i les seves instàncies es distingirien inequívocament les unes de les altres mitjançant una clau primària que es podria dir CodiCicle.

A continuació, hauríem d'establir una interrelació binària anomenada, per exemple, Curriculum, en la qual participés l'entitat ASSIGNATURA com a entitat feble, en el costat N de la interrelació, i l'entitat CICLE com a entitat forta, en el costat 1.

Les relacions febles en els diagrames ER es representen amb un rectangle de doble línia.

FIGURA 1.18. Exemple d'entitat feble



1.3.1 Notació

Per incorporar les entitats febles als diagrames ER, cal aplicar unes poques regles de notació addicionals:

- Les entitats febles es representen escrivint el seu nom en majúscules i en singular, dins d'un rectangle dibuixat amb una línia doble.
- La interrelació que uneix l'entitat feble amb la seva forta es representa amb un rombe també de línia doble.
- L'atribut o el conjunt d'atributs que actuïn com a discriminants han d'anar subratllats amb una línia discontínua.

2. Diagrames Entitat-Relació

Els diagrames Entitat-Relació són un estàndard actual en el disseny de bases de dades. De fet, es tracta d'una eina sense la qual, possiblement, les bases de dades tal com les entenem actualment no existirien.

Els diagrames Entitat-Relació són eines gràfiques clau en el disseny de bases de dades. La seva confecció ha de ser sistemàtica i rigorosa si es vol obtenir un sistema vàlid i eficient, ja que serà a partir d'aquests diagrames que es desenvoluparà tota la implementació de bases de dades en els sistemes gestors de bases de dades concrets que correspongui a cada empresa o organisme.

2.1 Disseny de bases de dades

El disseny de BD s'estructura, fonamentalment, en tres grans etapes:

- Disseny conceptual
- Disseny lògic
- Disseny físic

Encara que ens centrem en l'estudi i en la pràctica del disseny conceptual, no s'han de perdre de vista les altres fases del disseny, que també són importants, per tal d'obtenir una visió de conjunt de tots aquests processos.

Conèixer com s'estructura el model Entitat-Relació és important. I també ho són qüestions que ens han de permetre aprofitar la tecnologia proporcionada per les BD i els corresponents sistemes gestors, com ara els següents:

- Quines entitats ha d'incloure una BD determinada.
- Quines interrelacions s'han de considerar.
- Quins atributs han d'existir i en quines entitats o interrelacions s'han d'incorporar.
- Quines claus primàries ja es poden establir en la fase de disseny conceptual.

2.1.1 Fases del disseny de BD

Dissenyar BD no és una tasca senzilla. Encara que la porció del món real que es vulgui modelitzar en un cas concret sigui relativament petita, les estructures de

dades resultants poden arribar a tenir un cert grau de complexitat. Tanmateix, a mesura que augmenta la informació a considerar, i la seva complexitat, el model de dades necessari per representar-la es pot convertir, certament, en una construcció complicada.

Món real

Quan parlem de *món real*, ens referim a l'escenari o situació concreta que es vol modelitzar per tal de ser explotada mitjançant una BD.

Voler resoldre de cop tota la problemàtica que pot comportar la modelització d'una BD no és, doncs, una opció gaire realista. En afrontar una tasca d'aquesta envergadura, és preferible dividir-la en subtasques per tal de simplificar-la.

Així, doncs, resulta convenient descompondre el disseny de BD en diferents etapes, de manera que en cadascuna només s'examinin certs aspectes, o tipus de problemes, per tal de minimitzar la possibilitat d'error. Aleshores, a partir del resultat obtingut en cada fase, es pot continuar treballant en la fase següent, fins a arribar al resultat esperat, al final de l'última fase.

És habitual estructurar el disseny de BD en les tres etapes o fases següents:

1. Disseny conceptual.
2. Disseny lògic.
3. Disseny físic.

Fase de disseny conceptual

El primer que cal fer, durant la fase de disseny conceptual, és recopilar tota la informació necessària de la part del món real que ens proposem modelitzar amb una BD.

Aquesta recopilació d'informació es realitzarà per diferents vies, com ara aquestes:

- Entrevistes amb els futurs usuaris de la BD que s'està dissenyant.
- Examen de la documentació proporcionada per aquests mateixos usuaris.
- Observació directa dels processos a informatitzar.

A continuació, s'han d'estructurar convenientment les dades necessàries per tal de donar resposta a totes les necessitats derivades del conjunt d'informacions compendiades.

L'**objectiu del disseny conceptual** consisteix en l'obtenció d'una especificació sistemàtica.

L'esmentada especificació sistemàtica resultat del disseny conceptual ha de complir dos tipus de requisits:

- **De dades**. El model resultant ha de tenir en compte l'estructura completa de les dades i la seva integritat.
- **Funcionals**. Un bon esquema conceptual també haurà de preveure les necessitats bàsiques en matèria de manipulació de dades (és a dir, les operacions d'inserció, esborrament, consulta i modificació, d'aquestes). Durant les fases posteriors, pot ser convenient depurar el disseny per tal d'optimitzar les operacions a realitzar sobre les dades.

El model Entitat-Interrelació (o model Entitat-Relació) també es coneix de manera abreujada com a model ER (sigles corresponents a *entity-relationship*).

Finalment, cal triar un model de dades d'alt nivell i traduir els requisits anteriors a un esquema conceptual de la futura BD expressat amb els conceptes i la notació corresponents. Un dels models de dades d'alt nivell més utilitzats és el **model entitat-interrelació**.

Expressat en la terminologia del model ER, l'esquema de dades desenvolupat durant la fase de disseny conceptual ha d'especificar totes les entitats necessàries, i les interrelacions entre elles, amb les cardinalitats adequades, i també els atributs que corresponguin en cada cas.

El model ER

Una **entitat**, en el model ER, és una abstracció que ens interessa modelitzar, i mitjançant la qual s'agrupen les instàncies del món real que tenen unes característiques comunes.

Una **interrelació**, en el model ER, és l'associació entre instàncies de diferents entitats tipus. Aquesta associació es pot donar amb diverses cardinalitats.

Un atribut, en el model ER, és una característica d'una entitat que ens interessa tenir registrada.

El model resultant s'ha de revisar per tal de garantir la satisfacció de totes les necessitats detectades, d'una banda, i per evitar redundàncies de les dades (és a dir, repeticions indesitjades d'aquestes), d'una altra.

El resultat de la fase de disseny conceptual pertany a l'anomenat **món de les concepcions**, però encara no al **món de les representacions**, ja que no s'hi especifica cap representació informàtica concreta.

Com es pot veure, durant la fase de disseny conceptual no cal tenir en compte, encara, ni el tipus de BD que s'utilitzarà posteriorment ni, encara menys, el SGBD o el llenguatge concret amb el qual s'implementarà.

Fase de disseny lògic

En la fase de **disseny lògic**, es treballa amb el model abstracte de dades obtingut al final de l'etapa de disseny conceptual, per tal de traduir-lo al model de dades utilitzat pel sistema gestor de bases de dades (SGBD) amb el qual es vol implementar i mantenir la BD.

Les claus primàries serveixen per distingir entre si les diferents tuples d'atributs dins d'una mateixa relació.

Per tant, a partir d'aquesta fase de disseny, sí que cal tenir en compte la tecnologia concreta que s'ha d'emprar en la creació de la BD, ja que la BD resultant es pot

adequar a diferents models lògics, com ara els següents:

- Jeràrquic
- Relacional
- Distribuït
- Orientat a objectes

Malgrat la diversitat de possibilitats, el cert és que el més freqüent, a l'hora de dissenyar una BD, encara consisteix a expressar l'esquema conceptual en un model ER i, a continuació, traduir-lo a un model relacional.

Les claus foranes són uns instruments destinats a permetre la interrelació de la respectiva relació amb d'altres.

Quan el producte d'una fase de disseny lògic és una BD relacional, aquesta consisteix en un conjunt de relacions (altrament, anomenades *representacions tabulars*) compostes per atributs, alguns dels quals formen part de claus primàries o de claus foranes.

Resulta evident, doncs, que el resultat de la fase de disseny lògic ja se situa dins de l'anomenat **món de les representacions informàtiques**.

Fase de disseny físic

El **disseny físic** consisteix a fer certs tipus de modificacions sobre l'esquema lògic obtingut en la fase anterior de disseny lògic, per tal d'incrementar l'eficiència.

L'eficiència d'un esquema pot comportar la modificació d'algunes operacions que s'hagin de fer amb les dades, encara que comportin un cert grau de redundància d'aquestes, com per exemple:

- Afegir algun atribut calculable en alguna relació.
- Dividir una relació en altres dues o en més.
- Incloure en la BD una relació que sigui el producte de combinar dues o més relacions.

Però la fase de disseny físic també es caracteritza per la possibilitat d'adoptar altres decisions, relacionades amb aspectes d'implementació física a més baix nivell, i estretament vinculades amb el SGBD amb el qual es treballa en cada cas, com ara els següents:

- Definició d'índexs.
- Assignació de l'espai inicial per a les taules, i previsió del seu creixement ulterior.

- Selecció de la mida de les memòries intermèdies.
- Parametrització del SGBD segons les opcions que aquest ofereixi.

La volatilitat de les dades té a veure amb el volum d'insercions i esborraments de les mateixes.

Per tal de prendre encertadament aquests tipus de decisions, cal tenir en compte les característiques dels processos que operen amb les dades, la freqüència d'execució dels diferents tipus de consulta, el grau de volatilitat de les dades, els volums d'informació a emmagatzemar, etc.

2.1.2 Disseny conceptual d'una BD

Podem considerar qualsevol empresa, organització, institució, etc. com un sistema amb regles pròpies de funcionament. Aquest sistema, susceptible de ser informatitzat, està compost per tres subsistemes:

- **Subsistema de producció**. S'encarrega de realitzar les activitats pròpies de l'organització de què es tracti en cada cas (per exemple, fabricar cotxes, o reparar-los, o vendre'ls, etc.).
- **Subsistema de decisió**. S'encarrega de dirigir, coordinar i planificar les activitats realitzades dins de l'àmbit del subsistema de producció.
- **Subsistema d'informació**. S'encarrega de recollir, emmagatzemar, processar i distribuir, totes les informacions necessàries per al bon funcionament dels altres dos subsistemes.

Les BD serveixen per emmagatzemar les representacions de les informacions utilitzades dins de l'àmbit dels sistemes d'informació.

Els elements que conformen un **sistema d'informació** són de dos tipus:

- Dades: representacions de les informacions.
- Processos: accions exercides sobre les dades (consultes, modificacions, càlculs, etc.).

En funció de les observacions anteriors, podem afirmar que hi ha dues premisses que tot dissenyador de BD hauria de tenir ben assumides abans de començar a treballar en qualsevol projecte:

- No és competència del dissenyador de BD, com a tal, prendre decisions sobre la porció del món real que vol modelitzar.
- En principi, el dissenyador de BD tampoc no s'ha d'inventar característiques de la realitat a modelitzar: simplement les ha de reflectir de la manera més fidel possible en el model resultant.

Podem subdividir aquesta etapa de disseny conceptual en dues fases successives, les quals comporten tasques de diferents tipus: la **recollida i abstracció de les necessitats** de l'organització, d'una banda, i **l'elaboració d'un esquema conceptual** mitjançant un model de dades concret, de l'altra.

La **recollida i abstracció de les necessitats** de l'organització es duu a terme mitjançant diferents procediments (entrevistes, examen de documentació, etc.), i en aquesta fase cal recollir tota la informació necessària per tal de cobrir tots els requeriments de dades. Però, amb aquesta informació, s'ha de seguir un procés d'abstracció que ens permeti estructurar-la i diferenciar entre les qüestions essencials i les accessòries.

L'**elaboració d'un esquema conceptual** mitjançant un model de dades concret comporta que tota la informació recollida i degudament estructurada s'ha d'expressar en una notació estandarditzada (com ara els diagrames ER).

2.1.3 Captura i abstracció dels requeriments de dades

Per realitzar la captura i abstracció dels requeriments, en primer lloc, cal esbrinar quines necessitats tenen els usuaris de la futura BD. Sovint, aquests usuaris potencials només tenen una percepció molt general d'allò que necessiten.

Usuaris de les BD

Ho són tant els operadors que interactuen habitualment amb el sistema, com els destinataris finals de la informació que se n'ha d'extreure o que s'ha d'incloure en la BD.

El dissenyador de BD és el professional informàtic que s'encarrega de realitzar les tasques que implica el disseny de les BD.

El dissenyador ha de saber seleccionar les qüestions essencials, diferenciar-les dels aspectes accessoris, i descobrir quins són els vertaders interessos dels que han encarregat el disseny de la BD.

Ara bé, el dissenyador de BD, com a tal, no ha de decidir res. La seva tasca consisteix més aviat a ajudar els usuaris potencials a descobrir què necessiten exactament. És una feina feixuga i complicada, que serveix per descobrir el veritable flux de dades que s'ha de reflectir en el model conceptual resultant. Normalment, aquesta funció comporta el següent:

- Moltes entrevistes amb els futurs usuaris de tots els nivells i seccions de l'organització a informatitzar. Cal anotar tots els detalls sorgits durant les entrevistes, susceptibles d'implementació informàtica.
- Examen exhaustiu de la documentació proporcionada pel client, per tal de conèixer el funcionament intern de l'organització.
- Observacions directes dels diferents processos de l'organització.

Un bon dissenyador de BD ha d'arribar, com a mínim, a una solució plausible per a cada problema que se li hagi plantejat. De vegades, també podrà prendre en consideració solucions parcials alternatives. I, en tot cas, finalment, ha de presentar

una anàlisi completa dels requeriments en la qual concreti les especificacions de l'organització que li ha encarregat el projecte de disseny.

Un exemple concret: BD de la Xarxa de Biblioteques d'INS de Catalunya (XBIC)

L'XBIC com a tal no existeix. Aquest exemple només pretén simular la metodologia de treball habitual a l'hora de dissenyar conceptualment una base de dades.

Examinar els requisits de disseny que ens permetin establir un model conceptual per a una futura BD que doni servei conjuntament a les biblioteques de tots els instituts de Catalunya (mitjançant una aplicació web ulterior, per exemple), per tal de posar a l'abast de qualsevol membre de la comunitat educativa de secundària la totalitat dels fons bibliogràfics i d'altres recursos audiovisuals, encara que no es trobin físicament en el centre docent on treballi o estudiï l'usuari que sol·liciti el préstec, ens proveirà d'un bon escenari per desenvolupar un exemple de disseny de BD concret.

No es tracta ara tant d'aprofundir en qüestions de detall, com del fet de copsar globalment com es treballa durant la fase de disseny conceptual, i aplicar els coneixements adquirits prèviament -els quals són necessaris per elaborar els esquemes conceptuals-, en el nostre cas fonamentalment amb ajuda del model de dades ER.

Informe d'anàlisi i disseny de BD

La informació recollida pels dissenyadors de BD després de la captura i l'abstracció dels requeriments de les dades es recull en un informe.

Després d'entrevistar-nos amb l'alt càrrec del Departament d'Educació de la Generalitat de Catalunya -que promou la coordinació i col·laboració de les biblioteques de tots els instituts (INS) de Catalunya-, amb uns quants directors i directores de diferents biblioteques interessades a adherir-se a la futura Xarxa de Biblioteques d'INS de Catalunya (XBIC), i també amb uns quants bibliotecaris d'aquestes, i de recopilar i examinar diferent documentació que ens ha estat proporcionada, els dissenyadors de BD encarregats del projecte hem sintetitzat les informacions rebudes de la manera següent:

1. Servei de préstec

- Mitjançant aquest servei, s'ofereix la possibilitat de treure els diversos documents (llibres, revistes, vídeos, CD...) de la biblioteca.
- Per utilitzar el servei de préstec, els usuaris han de tenir el carnet de biblioteca.
- Aquest carnet serveix per utilitzar altres serveis de la biblioteca, com ara el servei d'accés a Internet, el servei Wi-Fi, consulta a BD o participació en algunes activitats que organitzin les biblioteques.
- El mateix carnet serveix per a tota la Xarxa de Biblioteques i permet gaudir d'avantatges interessants en el món de la cultura.
- La sol·licitud del carnet de biblioteca s'ha de fer des de la mateixa biblioteca o bé en línia. En el formulari de sol·licitud, es demanen a l'usuari les dades personals necessàries (nom i cognoms, adreça, data de naixement, etc.).

- A més, els usuaris poden fer altres gestions en línia relacionades amb el préstec: reserves i renovacions de documents.

2. Normativa d'ús del servei de préstec de la Xarxa de Biblioteques

- Tenir el carnet implica acceptar les normes de funcionament de la biblioteca.
- El carnet és personal i intransferible.
- Els usuaris menors de setze anys necessiten l'autorització dels pares per fer-se el carnet, i també els majors de setze anys que no tinguin DNI.
- Cal comunicar, a la biblioteca, qualsevol canvi de domicili o la pèrdua del carnet.
- Cada lector es pot emportar en préstec fins a deu documents entre llibres, revistes i audiovisuals, per un termini de tres setmanes, els llibres, i una setmana, les revistes i audiovisuals.
- Passat el termini, qualsevol document en préstec pot ser renovat, sempre que cap altre usuari no l'hagi reservat. La renovació es pot fer a la biblioteca, per telèfon, per correu electrònic o per Internet.
- Cal tenir cura dels documents deixats en préstec. Si un usuari no torna els documents en el termini fixat, pot ser exclòs del servei de préstec de les biblioteques de la Xarxa durant un temps equivalent al que s'ha retardat en la devolució. Si un usuari perd o fa malbé un document, ho ha de notificar a la biblioteca i ha de comprar el mateix document o abonar-ne l'import.
- Queden exclosos de préstec els documents següents:
  - Algunes obres de referència: enciclopèdies, diccionaris, atles, etc.
  - Fons de reserva.
  - Documents pertanyents a la col·lecció local.
  - Números corrents de revistes i de diaris.
  - Els documents que la biblioteca cregui convenient que no surtin de la biblioteca.
- A més del servei de préstec a l'usuari individual, les biblioteques ofereixen altres tipus de préstec:
  - Servei de préstec interbibliotecari. Mitjançant el servei de préstec interbibliotecari, les biblioteques de la Xarxa s'encarreguen de localitzar i proporcionar els documents que no té el fons propi i que estan disponibles en altres biblioteques de la Xarxa. El préstec entre les biblioteques de la Xarxa té un preu públic establert d'1,20 €.
  - Servei de préstec a domicili. Aquest servei s'adreça a persones amb problemes de mobilitat temporal o permanent, com ara malalts crònics, persones en període de convalescència, amb discapacitats físiques o gent gran. No totes les biblioteques de la Xarxa ofereixen aquest servei. El servei de préstec a domicili té un preu públic establert d'1,50 €.

3. Consulta del catàleg

- S'ha de poder consultar la disponibilitat dels fons bibliotecaris pels conceptes següents:
  - Signatura (identificador de l'exemplar físic objecte de préstec)
  - Títol
  - Paraula clau continguda en el títol
  - Autor (concretament, pel cognom)
  - Nom o cognoms incomplets de l'autor
  - Matèria
- A més, les cerques s'han de poder limitar als àmbits següents:
  - Una biblioteca concreta
  - Totes les biblioteques d'una sola població
  - Totes les biblioteques d'una sola comarca
- Finalment, s'han de poder limitar els resultats obtinguts en funció dels conceptes següents:
  - Idioma
  - Tipus de format
  - Any de publicació

2.1.4 Identificació d'entitats

L'especificació dels requeriments de dades serveix com a punt de partida per a l'elaboració de l'esquema conceptual de la futura BD. A continuació, el primer que s'ha de fer és identificar les entitats i els seus atributs.

La identificació de les entitats i dels seus atributs es pot documentar amb un llistat que segueixi el format següent:
ENTITAT1 (Atribut1, Atribut2, ...)
ENTITAT2 (Atribut1, Atribut2, ...)
....
ENTITATn (Atribut1, Atribut2, ...)

En el llistat, els noms de les entitats aniran amb majúscules, i els noms dels atributs començaran amb majúscules.

Seguint amb l'exemple BD de la XBIC, en una primera aproximació, tot repassant les especificacions que tenim, podríem obtenir una solució que comptés com a mínim amb les tres entitats següents, que incorporen els respectius atributs expressats entre parèntesis:

L'exemple parteix de la informació referida en l'apartat "Captura i abstracció dels requeriments de dades".

- INS(Poblacio, Comarca). Possibles atributs addicionals: Nom, Adreça i Telefon.
- DOCUMENT(Format, Import, ExclosPrestec, Signatura, Titol, Autor, Materia, Idioma, AnyPublicacio).
- USUARI(DNI, Nom, Cognoms, Adreça, DataNaixement, Carnet, DataFinalExclusioPrestec).

Però estem en condicions de depurar aquest resultat rudimentari. Fixem-nos en què hi haurà moltes poblacions i comarques que es repetiran per a diferents INS. Fóra millor, doncs, que aquests dos atributs constituïssin dues entitats independents, convenientment interrelacionades amb l'entitat INS: POBLACIO i COMARCA.

Un altre avantatge d'utilitzar aquestes dues entitats en comptes de considerar-les atributs d'una entitat concreta, és que les podrem reaprofitar interrelacionant-les amb altres entitats, en cas necessari.

Passa el mateix amb altres atributs de DOCUMENT: el format, l'autor la matèria i l'idioma.

Seria millor tenir una entitat per enregistrar els tipus de format, que sempre seran els mateixos (llibre, revista, CD, etc.), anomenada per exemple FORMAT, i interrelacionar-la amb DOCUMENT.

Estarem davant del mateix fenomen amb els idiomes (català, castellà, anglès, etc.) i les matèries (història, música, física, etc.). Per tant, serà convenient comptar amb dues entitats més: IDIOMA i MATERIA.

D'altra banda, els autors podran ser autors d'una pluralitat de documents i, a fi de simplificar les cerques ulteriors per autor, és preferible destinar una entitat pròpia per representar-los (podem anomenar-la, senzillament, AUTOR).

Després de fer aquestes consideracions, ens trobaríem amb unes quantes entitats més que no pas inicialment:

- INS(Nom, Adreça, Telefon)
- DOCUMENT(Signatura, Titol, AnyP ublicacio, Import, ExclosPrestec)
- USUARI(Carnet, DNI, Nom, Cognoms, Adreça, DataNaixement, DataFinalExclusioPrestec)
- POBLACIO(Nom)
- COMARCA(Nom)
- FORMAT(Descripcio)
- AUTOR(Nom, Cognoms)
- MATERIA(Descripcio)

- IDIOMA(Descripcio)

Hem de ser conscients que qualsevol especificació que no es desprengui directament dels documents de treball inicial en què hem sintetitzat els diferents requeriments de dades detectats com, per exemple, afegir els nous atributs a INS s'ha de validar mitjançant noves entrevistes, o l'examen de nova documentació o bé revisió de l'antiga, o, si no, observant *in situ* el procés en execució que pugui comportar una innovació en l'especificació.

En canvi, altres aspectes que poden semblar més agosarats (com ara representar el que era inicialment l'atribut d'una entitat com una altra entitat independent) no han d'implicar necessàriament uns processos de validació com els que acabem de descriure, si es tracten purament de decisions tècniques de disseny.

Cal fer notar que, per poc complicada que sigui la porció del món real a modelitzar, normalment és possible obtenir uns quants models conceptuals, en alguns casos alternatius i equivalents, i en d'altres orientats a satisfer en més o menys mesura diferents finalitats, però amb resultats igualment correctes.

2.1.5 Designació d'interrelacions

Després de la captura i abstracció dels requeriments de dades i de la identificació d'entitats, el pas següent consisteix a establir les interrelacions necessàries entre les entitats detectades.

Les interrelacions es poden documentar amb el format següent:
Interrelacio (Atribut1, Atribut2, ...), entre ENTITATi i ENTITATj
En què el nom de la interrelació començarà amb majúscules i, en cas de disposar d'atributs, aquests aniran entre parèntesis i amb la inicial del nom també amb majúscules. Caldrà especificar el nom de les entitats que interrelaciona.

L'exemple esmentat es refereix a les informacions que conté l'apartat "Captura i abstracció dels requeriments de dades", i es desenvolupa al llarg de les diferents fases de disseny conceptual de BD.

En algun cas, les interrelacions poden incorporar algun atribut (com passa amb la interrelació Prestec), de la mateixa manera que les entitats. Inicialment podem trobar, com a mínim, quatre interrelacions que només afecten les tres entitats originàries:

- Prestec(Data, Tipus, Preu), entre USUARI i DOCUMENT
- RenovacioPrestec(Data), entre USUARI i DOCUMENT
- Reserva(Data), entre USUARI i DOCUMENT
- Ubicació, entre INS i DOCUMENT

Però, després de considerar l'establiment de la resta d'entitats, haurem d'afegir les interrelacions corresponents:

- Esta, entre INS i POBLACIO
- Pertany, entre POBLACIO i COMARCA
- Te1, entre FORMAT i DOCUMENT
- Te2, entre AUTOR i DOCUMENT
- Versa, entre MATERIA i DOCUMENT
- Expressat, entre IDIOMA i DOCUMENT

Notem que quan es repeteix un mateix nom en més d'una interrelació (com aquí passa amb Te), cal numerar-les correlativament, per tal d'evitar confusions en fer referència a cadascuna.

I, a més, podríem considerar l'establiment d'una interrelació entre USUARI i POBLACIO per tal de completar-ne l'adreça degudament (com hem fet amb els INS). Quedaria així:

- Viu, entre USUARI i POBLACIO

2.1.6 Establiment de claus

Recordem que la **clau primària** d'una entitat està constituïda per un atribut, o per un conjunt d'atributs, els valors dels quals són capaços d'identificar unívocament les instàncies d'aquella.

De vegades, una entitat disposa de més d'un atribut, o conjunt d'atributs, que estan en condicions de constituir la clau primària. En aquest cas, parlem de **claus candidates**.

> I una vegada que el dissenyador tria un atribut o conjunt d'atributs concrets per identificar unívocament les instàncies, entre els que compleixen les condicions (**claus candidates**), aquest o aquests atributs passen a ser la **clau primària**, i els no seleccionats romanen com a **claus alternatives**.

Un criteri important a l'hora de decidir-se perquè un o més atributs formin la clau primària d'una entitat, és que el seu valor no canviï mai o, si més no, molt rarament.

> L'atribut o conjunt d'atributs que formen la clau primària han d'aparèixer subratllats en la documentació.

Per exemple, no seria gaire prudent decidir-se pel número de telèfon mòbil com a clau primària d'una entitat que emmagatzema persones, perquè, tot i ser un número habitualment d'ús personal, pot canviar al llarg del temps (per exemple, una persona pot regalar el seu mòbil a una altra). En canvi, el número de DNI pot ser, en general, una bona opció, ja que, en principi, aquest número és personal i intransferible.

L'exemple esmentat es refereix a les informacions contingudes en l'apartat "Captura i abstracció dels requeriments de dades", i es desenvolupa al llarg de les diferents fases de disseny conceptual de BD.

Continuant amb el nostre exemple, estem en condicions de trobar les claus següents, formades per només un atribut, les quals es mostren subratllades:

- DOCUMENT(Signatura , Titol, AnyPublicacio, Import, ExclosPrestec)
- COMARCA(Nom)
- POBLACIO(Nom)
- FORMAT(Descripcio)
- MATERIA(Descripcio)
- IDIOMA(Descripcio)

L'entitat AUTOR no té cap atribut que identifiqui plenament les seves instàncies. Aquí podríem haver optat per combinar els valors que adoptin els atributs Nom i Cognoms en cada tuple, per tal de no haver d'introduir cap tipus de codificació artificial, però aquesta no deixa de ser una opció arriscada, ja que pot passar que hi hagi dos autors amb el mateix nom i cognoms. Per tant, afegim un nou atribut, anomenat Codi, per tal que no hi hagi problemes amb la clau primària d'aquesta entitat. Per tant, l'entitat ens queda així:

- AUTOR(Codi, Nom, Cognoms)

En certa manera podríem considerar que USUARI té dues claus alternatives: Carnet i DNI. Però en realitat no és així, ja que hi poden haver usuaris menors d'edat que encara no tenen DNI. I ja sabem que cap clau primària pot admetre valors nuls, ja que aleshores no serviria per distingir unívocament les instàncies entre elles. En canvi, tot usuari disposarà d'un número de carnet. Per tant, ens quedarà l'estructura següent:

- USUARI(Carnet, DNI, Nom, Cognoms, Adreça, DataNaixement, DataFinalExclusioPrestec)

Finalment, hem d'examinar l'entitat INS. El nom dels instituts es pot repetir, i de fet es repeteix (per exemple, hi ha un INS anomenat Milà i Fontanals a Barcelona, un altre a Igualada, un altre a Vilafranca del Penedès, etc.). Per tant, l'atribut Nom no és suficient per constituir per si sol la clau primària de l'entitat. Una opció seria establir algun tipus de codificació. Però, en aquest cas, hem preferit considerar INS com una entitat feble que depèn de POBLACIO. Per tant, la clau primària d'INS estarà formada per la clau primària de l'entitat forta (l'atribut Nom de POBLACIO) més l'atribut discriminant de l'entitat feble (l'atribut Nom d'INS), que també mostrem subratllat:

- INS(Nom , Adreça, Telefon)

2.1.7 Establiment de cardinalitats

L'establiment correcte de les cardinalitats adequades per a cada interrelació depèn de les característiques del món real que es volen modelitzar.

Cal afegir, doncs, la informació referent a les cardinalitats de les interrelacions en la documentació de disseny.

L'exemple es planteja en l'apartat "Captura i abstracció dels requeriments de dades", i es va construint seguint les diferents fases de disseny conceptual de BD.

En el model que estem construint (seguint l'exemple de la BD de la XBIC) podem establir les cardinalitats següents:

- Un mateix usuari pot agafar en préstec diferents documents, i un mateix document es pot prestar a diferents usuaris, al llarg del temps:

Prestec(Data, Tipus, Preu), entre USUARI i DOCUMENT: M-N

- Passa el mateix en cas de renovar un préstec o de reservar un document:

RenovacioPrestec(Data), entre USUARI i DOCUMENT: M-N Reserva(Data), entre USUARI i DOCUMENT: M-N

- Tot document pertanyerà a la biblioteca d'un INS concret, la qual podrà disposar de molts documents:

Ubicació, entre INS i DOCUMENT: 1-N

- Dins d'una mateixa població podran coexistir diferents INS, però un INS estarà en una població concreta. A més no oblidem que, en tractarse d'una entitat feble, la cardinalitat ha de ser 1-N, amb l'entitat feble al costat de la N:

Esta, entre INS i POBLACIO: N-1

- Cada població pertany a una sola comarca, però cada comarca tindrà més d'una població a dins del seu territori:

Pertany, entre POBLACIO i COMARCA: N-1

- S'ha considerat que cada document només té un format, però evidentment cada format serà aplicable a molts documents diferents:

Te1, entre FORMAT i DOCUMENT: 1-N

- Amb una cardinalitat M-N, fem possible registrar més d'un autor per a un mateix document:

Te2, entre AUTOR i DOCUMENT: M-N

- En canvi, considerem que cada document només pot versar sobre una sola matèria:

Versa, entre MATERIA i DOCUMENT: 1-N

- Un document (com ara una pel·lícula en DVD) podrà estar expressat en diferents idiomes:

Expressat, entre IDIOMA i DOCUMENT: M-N

- Cada usuari té el domicili en una població concreta. Però en una mateixa població hi poden viure diferents usuaris:

Viu, entre USUARI i POBLACIO: N-1

2.1.8 Restriccions de participació i límits de cardinalitat

Es diu que la **participació d'una entitat en una interrelació**és **total** si cadascuna de les seves instàncies participa un cop, com a mínim, en la interrelació esmentada, i **parcial** en cas contrari.

- Prestec(Data, Tipus, Preu), entre USUARI (parcial) i DOCUMENT (parcial): M-N
- RenovacioPrestec(Data), entre USUARI (parcial) i DOCUMENT (parcial): M-N
- Reserva(Data), entre USUARI (parcial) i DOCUMENT (parcial): M-N
- Ubicació, entre INS (parcial) i DOCUMENT (total): 1-N
- Esta, entre INS (total) i POBLACIO (parcial): N-1
- Pertany, entre POBLACIO (total) i COMARCA (total): N-1
- Te1, entre FORMAT (parcial) i DOCUMENT (total): 1-N
- Te2, entre AUTOR (total) i DOCUMENT (total): M-N
- Versa, entre MATERIA (parcial) i DOCUMENT (total): 1-N
- Expressat, entre IDIOMA (total) i DOCUMENT (total): M-N

- Viu, entre USUARI (total) i POBLACIO (parcial): N-1

A més, hem d'establir un límit màxim a la cardinalitat de la interrelació Prestec perquè, en principi, un usuari no pot sol·licitar en préstec més de deu documents simultàniament.

Les interrelacions quedaran documentades, doncs, amb el format següent:
Interrelacio (Atribut1, Atribut2, ...), entre ENTITATi (participacio) i ENTITATj (participacio): tipus_de_cardinalitat
En què el nom de la interrelació començarà amb majúscules i, en cas de disposar d'atributs, aquests aniran entre parèntesis, separats per comes, i amb la inicial del nom també amb majúscules. Caldrà especificar el nom de les entitats que interrelaciona i el tipus de cardinalitat, que pot ser 1-1, 1-N, N-1 o M-N. La participació pot ser total o parcial.

2.1.9 Elaboració d'un esquema conceptual

Una vegada tenim els requeriments; i hem identificat les entitats, atributs i interrelacions; i gràcies al coneixement assolit de les necessitats que han de satisfer les dades, el dissenyador està en condicions d'elaborar un esquema conceptual complet d'aquestes i de les seves interrelacions.

Un diagrama ER és un esquema gràfic que serveix per representar un model ER.

Per tal de confeccionar aquest model conceptual, el dissenyador de BD utilitzarà algun model de dades que s'adapti a les necessitats del projecte, i permeti continuar treballant-hi durant la fase ulterior de disseny lògic.

UML

Acrònim de *unified modeling language* (llenguatge unificat de modelització). Notació gràfica que permet especificar dades i aplicacions orientades a objectes. S'entén per model de dades UML aquella part de l'UML destinada a descriure les dades.

El model de dades més utilitzat per elaborar l'esquema conceptual és el model ER, i la documentació del disseny conceptual culmina amb el **diagrama ER**.

Tot i que el model ER continua sent el model de dades més utilitzat, amb diferents variacions en aspectes de notació, cada vegada s'utilitza més el model UML de dades, molt útil en projectes que basin la implementació de la programació en el paradigma de l'orientació a objectes.

Per començar a treballar amb els diagrames ER, pot ser còmode establir l'esquelet del model, sense incloure encara tots els detalls (atributs, cardinalitats, etc.). En la figura 2.1, hi ha un esquema conceptual inacabat, expressat amb l'ajuda del model de dades ER, que només inclou les entitats i les interrelacions detectades inicialment.

Figura 2.1. Diagrama ER inicial de la XBIC

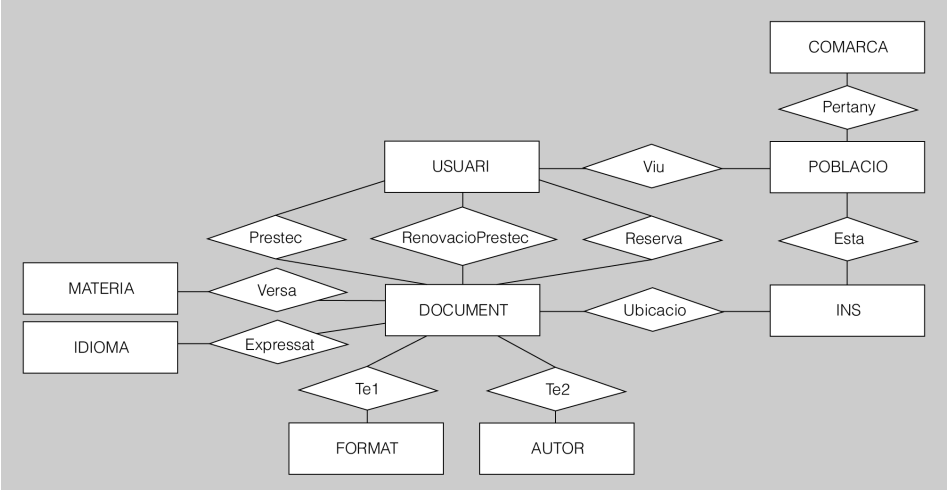
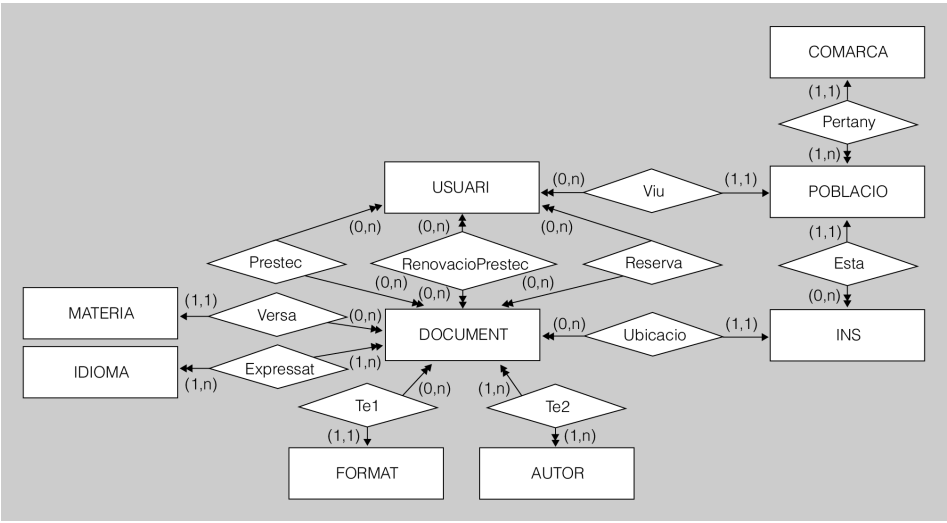


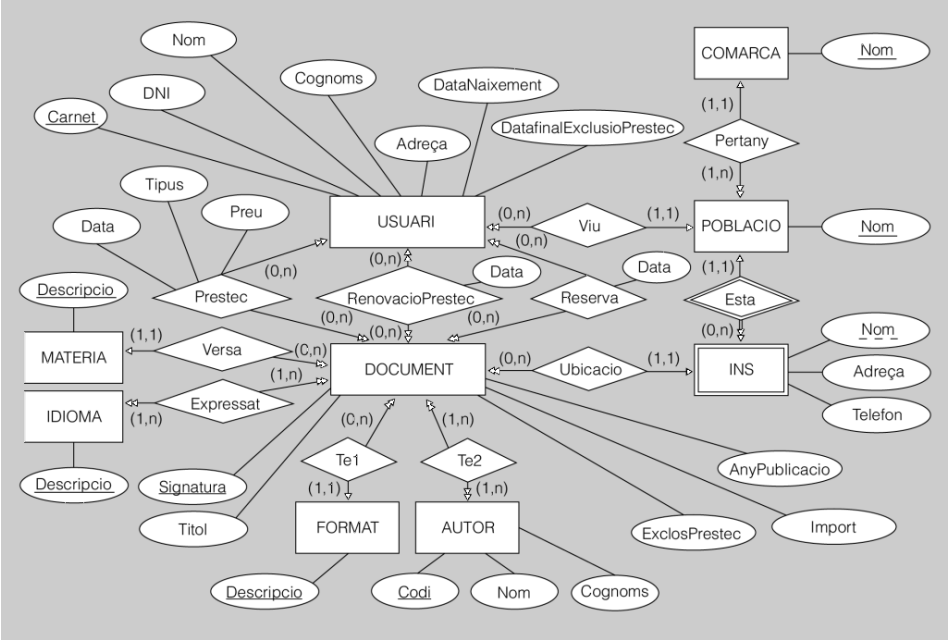
Figura 2.2. Diagrama ER de la XBIC a mig fer



A l'hora de dissenyar el diagrama ER, pot ser una bona pràctica partir d'un primer diagrama inacabat (com el de la figura 2.1) i incorporar-hi més detalls progressivament. En la figura 2.2, per exemple, ja han quedat establertes les cardinalitats i les restriccions de participació.

I l'últim pas que farà el dissenyador per completar el diagrama, de manera que reflecteixi tots els requeriments prèviament detectats, es pot veure en l'exemple de la figura 2.3, en què hi ha especificats tots els atributs i claus.

FIGURA 2.3. Diagrama ER de la XBIC totalment acabat



2.2 Extensions del model Entitat-Relació

Les estructures bàsiques del model Entitat-Relació (model ER) permeten representar la majoria de situacions del món real que habitualment cal incorporar en les BD. Però, de vegades, certs aspectes de les dades s'han de descriure mitjançant unes construccions més avançades del model ER, les quals comporten una extensió del model ER bàsic. Aquestes ampliacions del model ER consisteixen en l'especialització, la generalització i l'agregació, d'entitats.

2.2.1 Especialització i generalització

Ens podem trobar amb el cas d'alguna entitat tipus en què -a més de les característiques generals, comunes a totes les seves instàncies- ens interessi modelitzar, addicionalment, certes característiques específiques aplicables només a part de les seves instàncies.

Aleshores, podrem considerar que aquesta entitat tipus conté altres entitats tipus, de nivell inferior, amb característiques pròpies.

> L'**especialització** permet reflectir l'existència d'una entitat general, anomenada *entitat superclasse*, que es pot especialitzar en diferents entitats subclasse.

L'**entitat superclasse** permet representar les característiques comunes de l'entitat des d'un punt de vista general. Les **entitats subclasse**, en canvi, permeten representar les característiques pròpies de les especialitzacions de l'entitat superclasse.

Les instàncies de les subclasses han de ser, al mateix temps, instàncies de la superclasse respectiva.

El procés de designació de subclasses a partir d'una superclasse s'anomena *especialització*.

Exemple d'especialització

Fins ara comptàvem amb una única entitat PROFESSOR, que ens servia per treballar amb tots els docents del centre, ja que encara no havíem detectat cap subconjunt d'aquest col·lectiu que ens hagués fet pensar en implementar-ne una especialització.

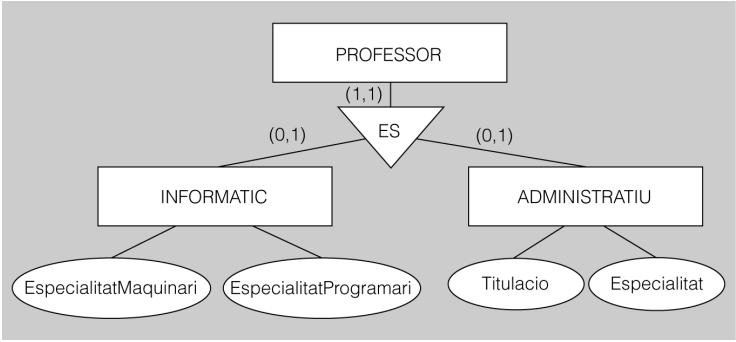
Però resulta que la direcció del centre vol implicar, en la gestió d'aquest i en el seu manteniment informàtic, el professorat de dues famílies professionals: l'administrativa i la informàtica, respectivament.

Per tant, ens interessa tenir constància, d'una banda, de la titulació dels professors de la família administrativa i de la seva especialitat, ja que en funció d'aquestes característiques podran assumir, o no, les responsabilitats que se'ls volen encomanar.

També pot resultar útil saber quina és l'especialitat principal, tant en maquinari com en programari, del professorat de la família d'informàtica, per assignar les tasques de manteniment amb una certa garantia d'èxit.

Com a conseqüència de tot això, implementarem una especialització de l'entitat PROFESSOR en dues subclasses: ADMINISTRATIU, que incorporarà dos nous atributs (Titulacio i Especialitat), i INFORMATIC, que n'incorporarà uns altres dos (EspecialitatMaquinari i EspecialitatProgramari).

Figura 2.4. Exemple d'especialització



L'especialització en els diagrames ER es representa amb un triangle.

L'especialització, doncs, permet reflectir les diferències entre les instàncies d'una mateixa entitat, mitjançant l'establiment de diferents entitats de nivell inferior, les quals agrupen els subconjunts d'instàncies amb característiques específiques comunes.

Aquestes característiques pròpies de les subclasses poden consistir tant en l'existència d'atributs com en la participació en interrelacions, però en cap cas no poden ser d'aplicació a totes les instàncies de la superclasse considerada com a tal.

La **generalització**, en canvi, és el resultat d'observar com diferents entitats preexistents comparteixen certes característiques comunes (és a dir, identitat d'atributs o d'interrelacions en les quals participen).

En funció de les similituds detectades entre diferents entitats, aquestes es poden arribar a sintetitzar en una sola entitat, de nivell superior, mitjançant un procés de generalització.

La generalització serveix per ressaltar les similituds entre entitats, per sobre de les diferències, i també per simplificar les representacions de les dades, en evitar la repetició d'atributs compartits per diferents subclasses.

Exemple de generalització

Fins ara, hem utilitzat dues entitats diferents que ens han servit per modelitzar dues categories, també diferents, existents al món real: ALUMNE i PROFESSOR.

Però, és evident que tant els alumnes com els professors són persones, tot i que amb rols diferents. Per tant, tindran una sèrie de característiques comunes, que es podran modelitzar de la mateixa manera.

Així, tant els uns com els altres tindran nom, cognoms, telèfons de contacte, etc., que es podran modelitzar mitjançant els mateixos atributs.

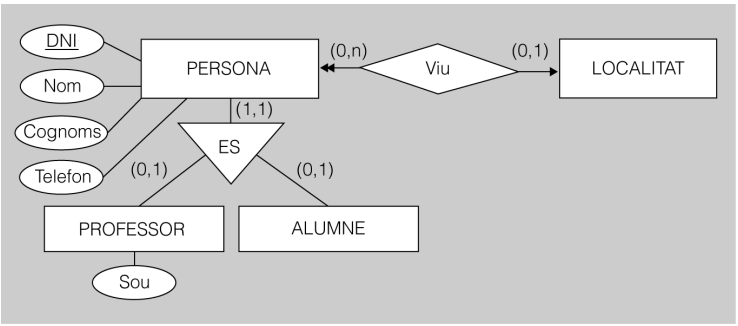
També és possible que totes dues tipologies puguin participar en les mateixes interrelacions. Per exemple, per tal d'indicar la localitat de residència, el més habitual és relacionar l'entitat que representa les persones amb una altra entitat que emmagatzema les diferents localitats.

En definitiva, partint de les entitats ALUMNE i PROFESSOR, podríem crear una altra entitat, superclasse de les anteriors, i anomenar-la, per exemple, PERSONA.

D'aquesta manera implementarem l'entitat PERSONA, com a generalització d'ALUMNE i PROFESSOR, la qual contindrà els atributs comuns a les seves subclasses, i a més participarà directament en les interrelacions que també siguin comunes a les subclasses esmentades.

La generalització en els diagrames ER es representa amb un triangle, com en l'especialització.

Figura 2.5. Exemple de generalització



El producte resultant de l'especialització i de la generalització és, doncs, idèntic. La diferència entre ambdós recau en el tipus de procés que condueix a cadascuna:

- L'especialització deriva d'un procés de disseny descendent, durant el qual, a partir d'una entitat preexistent, considerada com a superclasse es detecta la utilitat d'establir certes subclasses, a causa de l'existència de certes característiques (atributs i participacions en interrelacions) no aplicables a totes les instàncies de la superclasse.

- La generalització respon a un procés considerat de disseny ascendent. Durant aquest tipus de disseny es valora la utilitat de contemplar unes quantes entitats preexistents, anomenades subclasses, dependents d'una

mateixa superclasse comuna a totes elles. La superclasse presenta unes característiques comunes (atributs i participacions en interrelacions) a totes les subclasses que en depenen.

Herència de propietats

Tant en el cas de generalització com en el d'especialització, les característiques de l'entitat superclasse s'estenen cap a les entitats subclasse. Com ja sabem, aquestes característiques poden consistir o bé en atributs de l'entitat superclasse, o bé en la seva participació en diferents interrelacions.

Anomenem **herència de propietats** la transmissió de característiques (atributs i interrelacions) des de l'entitat superclasse cap a les entitats subclasse.

Jerarquia d'entitats

Quan s'encadenen diferents generalitzacions o especialitzacions de tal manera que una mateixa entitat és subclasse d'una estructura, i superclasse d'una altra, té lloc el que s'anomena *jerarquia d'entitats.*

Pot passar que una mateixa entitat adopti el rol de subclasse en un procés de generalització o especialització i que, al mateix temps, assumeixi el paper de superclasse en un altre d'aquests processos en què participi.

Quan es produeix una jerarquia d'entitats, les entitats dels nivells inferiors poden heretar característiques no solament de la superclasse respectiva, sinó també d'altres classes de nivells superiors.

Anomenem **herència múltiple** la recepció, per part d'una entitat subclasse, tant de les característiques (atributs i interrelacions) de la seva superclasse, com de les d'altres entitats de nivells superiors, dins d'una estructura jeràrquica d'entitats amb generalitzacions o especialitzacions encadenades.

Exemple de jerarquia d'entitats i d'herència múltiple

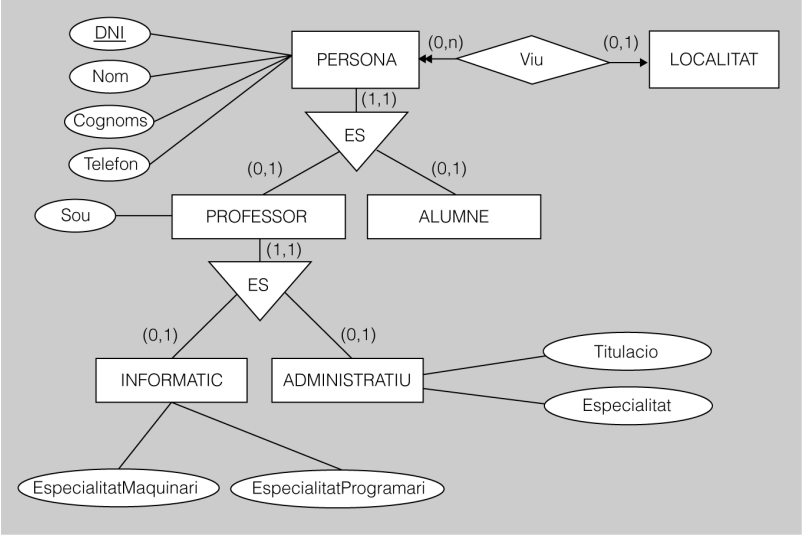
Com a resultat d'un procés de generalització hem conferit, a l'entitat PERSONA, la qualitat de superclasse de les entitats PROFESSOR i ALUMNE, considerades subclasses d'aquella.

Al mateix temps, però com a resultat d'un procés d'especialització, hem conferit a l'entitat PROFESSOR la categoria de superclasse de dues noves entitats, subclasses de la mateixa: INFORMATIC i ADMINISTRATIU.

A conseqüència de tot això, les entitats del nivell inferior (INFORMATIC i ADMINISTRATIU) no solament heretaran les característiques de la seva superclasse (PROFESSOR), sinó també les de les altres entitats de nivells superiors de les quals siguin descendents i, per tant, hereves.

En aquest cas, doncs, INFORMATIC i ADMINISTRATIU heretaran les característiques de la seva superclasse (PROFESSOR) i també les propietats de la superclasse d'aquella (PERSONA). Però no heretaran cap propietat d'ALUMNE, perquè no són descendents d'aquesta entitat.

FIGURA 2.6. Exemple d'herència múltiple



Restriccions

Per tal de modelitzar més exactament la parcel·la del món real que ens interessi, es poden establir certes restriccions sobre les especialitzacions o generalitzacions detectades.

Un primer tipus de restriccions defineix si les instàncies poden pertànyer simultàniament o no a més d'una subclasse d'una estructura simple (és a dir, que compti amb una sola superclasse i un sol nivell de subclasses) de generalització o especialització. En aquests casos, les entitats de tipus subclasse poden ser de dos tipus:

- **Disjuntes**. Una mateixa entitat instància no pot aparèixer en dues entitats subclasse diferents. Es representa en el diagrama afegint una etiqueta amb la lletra D.
- **Encavalcades**. Una mateixa entitat instància pot aparèixer en dues (o, fins i tot, en més de dues) entitats subclasse diferents. Es representa en el diagrama afegint una etiqueta amb la lletra E.

Un segon tipus de restriccions especifica si tota instància de la superclasse ha de pertànyer simultàniament a una o més de les subclasses o no. Aquí les entitats de tipus subclasse també poden ser de dos tipus:

- **Totals**. Tota instància de l'entitat superclasse ha de pertànyer simultàniament, com a mínim, a una de les seves entitats subclasse. Es denota amb l'etiqueta T.
- **Parcials**. Algunes instàncies de l'entitat superclasse poden no pertànyer simultàniament a cap de les seves entitats subclasse. Es denota amb l'etiqueta P.

Combinant aquestes restriccions obtenim, doncs, quatre possibilitats aplicables a les subclasses d'una generalització o especificació. Cal separar les lletres que s'inclouen en l'etiqueta amb una coma:

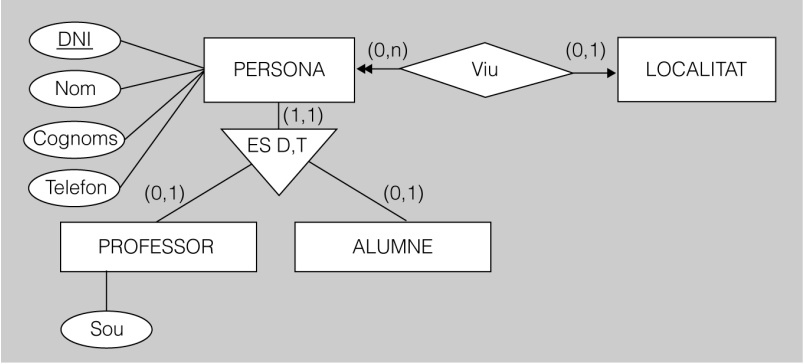
- D, T (disjuntes i totals)
- D, P (disjuntes i parcials)
- E, T (encavalcades i totals)
- E, P (encavalcades i parcials)

Exemple de subclasses D, T

Haurem de considerar disjuntes les subclasses de PERSONA si els reglaments de funcionament del centre no permeten que cap professor s'hi matriculi com a alumne, simultàniament amb l'exercici de la seva tasca docent.

Al mateix temps, les considerarem totals si la nostra BD registra exclusivament les dades de professors i d'alumnes, sense ocupar-se d'altres categories de persones (com podria ser el personal administratiu, de manteniment, de neteja, etc.).

Figura 2.7. Exemple de subclasses D, T

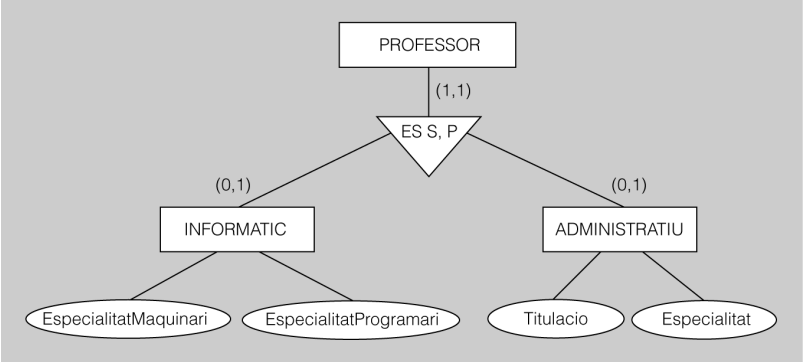


Exemple de subclasses E, P

Haurem de considerar encavalcades les subclasses de PROFESSOR si volem reflectir el fet que alguns professors, tot i exercir com a tals amb una especialitat concreta en un curs acadèmic, poden tenir altres especialitats. Per tant, un professor podrà ser simultàniament INFORMATIC i ADMINISTRATIU.

D'altra banda, les considerarem parcials perquè al nostre institut hi podrà haver, amb tota seguretat, professors d'altres especialitats (com ara electròniques, comercials, etc.), que no seran ni informàtics ni administratius.

Figura 2.8. Exemple de subclasses E, P



Per reflectir una combinació de característiques encara més complicada, s'ha de recórrer a una especificació textual que acompanyi el diagrama.

Exemple de subclasses amb diferents restriccions

Imaginem que volem afegir una nova subclasse de PERSONA, per tal d'incloure-hi el personal d'administració i serveis del centre. Anomenarem aquesta nova entitat ADMO_SERVEI.

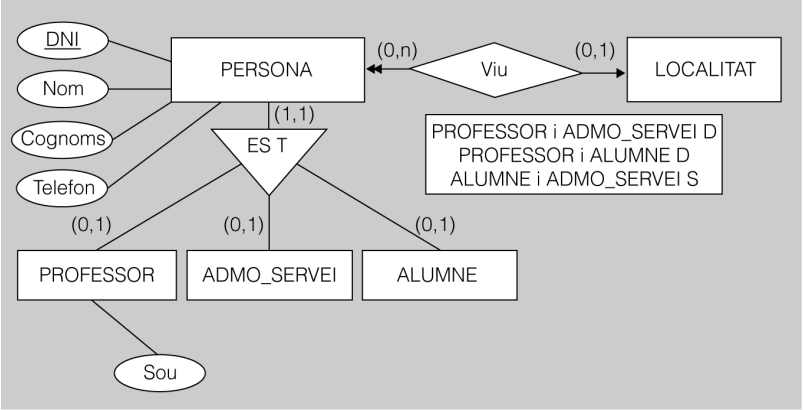
Ja hem exposat més amunt que els reglaments interns del nostre institut no permeten que cap professor sigui, simultàniament, alumne del centre. Però ara ens trobem que, al personal d'administració i serveis, sí que se li permet matricular-se com a alumne en algun dels estudis impartits al centre al mateix temps que exerceixen la seva tasca professional.

Totes tres subclasses seran totals, com abans, perquè tothom haurà de pertànyer a alguna de les tres categories reflectides.

PROFESSOR i ALUMNE seran disjuntes entre elles, igual que PROFESSOR i ADMO_SERVEI, ja que no es poden compatibilitzar les condicions esmentades. Però, al mateix temps, ALUMNE i ADMO_SERVEI seran encavalcades, perquè una persona podrà estar inclosa en aquestes dues categories simultàniament, segons hem vist.

Per reflectir una realitat com aquesta, no hi ha altre remei que fer servir una especificació textual que, tot acompanyant el diagrama, aclareixi degudament les característiques específiques de cada subclasse o agrupació d'aquestes.

FIGURA 2.9. Exemple de subclasses amb diferents restriccions



Notació

Tant l'especialització com la generalització es representen mitjançant un triangle que inclou, al seu interior, l'etiqueta ES. Aquesta etiqueta indica que tota instància de qualssevol de les subclasses és, al mateix temps, una instància de la superclasse corresponent (per exemple, tant un informàtic com un administratiu seran, al mateix temps, un professor).

Per distingir clarament la superclasse en els casos en què hi ha un gran nombre d'entitats subclasse implicades en l'estructura, o bé quan resulta difícil, per les característiques del diagrama, alinear clarament totes les subclasses, és convenient indicar els límits de cardinalitat de la generalització o especialització. Per a això, només cal afegir una etiqueta del tipus mín..màx, per tal d'expressar els límits respectius, al costat de la línia que uneix cada entitat amb el triangle que representa la generalització o especialització, en què mín i màx podran tenir els valors següents:

- 1..1 en la línia que enllaça la superclasse, perquè tota instància de qualsevol subclasse sempre constituirà, simultàniament, una i només una instància de la superclasse.
- 0..1 en la línia que enllaça cada subclasse, perquè no necessàriament tota instància de la superclasse haurà de ser, simultàniament, instància de la subclasse en qüestió (ho podrà ser d'una altra subclasse, o podrà no ser-ho de cap, si estem en presència d'una restricció de parcialitat).

Les entitats que formen part d'una estructura de generalització o especialització es representen com la resta d'entitats: cadascuna amb un rectangle que incorpora el nom respectiu, i els atributs respectius encerclats dins d'el·lipses lligades a la seva entitat amb una línia. Si els atributs formen una clau primària, el seu nom haurà d'anar subratllat.

En termes de notació diagramàtica, no s'estableix cap diferència entre una generalització i una especialització. Les diferències entre ambdós fenòmens es redueixen al procés que s'ha seguit per derivar en cadascun d'ells, però no en el resultat, que sempre és el mateix: l'establiment d'una superclasse i d'unes subclasses amb unes restriccions concretes, que es representen afegint, a l'etiqueta ES, les inicials de les dues restriccions aplicables separades per una coma:

- D, T (disjuntes i totals)
- D, P (disjuntes i parcials)
- E, T (encavalcades i totals)
- E, P (encavalcades i parcials)

2.2.2 Agregacions d'entitats

Amb les regles bàsiques del model ER, només es poden modelitzar interrelacions en què participen exclusivament entitats, però no es pot expressar la possibilitat que una interrelació participi directament en una altra interrelació. Però hi ha un mecanisme, anomenat *agregació*, que permet superar la limitació descrita anteriorment, tot considerant una interrelació entre entitats com si fos una entitat, i utilitzant-la com a tal.

Les agregacions també són conegudes com a entitats associatives.

L'**agregació d'entitats** és una abstracció, mitjançant la qual, una interrelació es tracta com una entitat de nivell més alt, que agrupa les entitats interrelacionades gràcies a ella. L'agregació ha de tenir el mateix nom que la interrelació sobre la qual es defineix.

La utilitat d'una agregació d'entitats, doncs, consisteix en el fet que la interrelació en què es basa es pot interrelacionar amb altres entitats. Una agregació d'entitats

es denota requadrant totes les entitats que participen en una interrelació determinada, per tal de construir una nova entitat que pot establir les pròpies interrelacions.

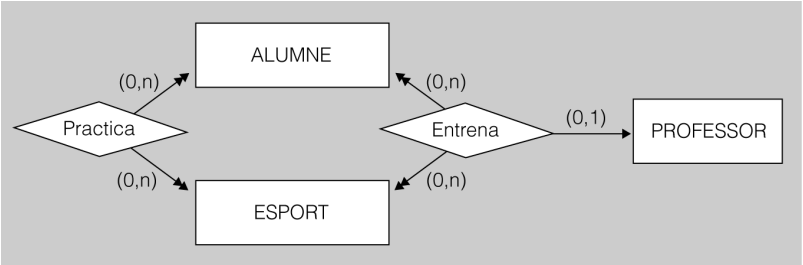
Exemple d'agregació d'entitats

Considerem la interrelació Practica, binària i de cardinalitat N-M, que té lloc entre les entitats ALUMNE i ESPORT, que ja coneixem.

Imaginem ara que es vol tenir constància del professor que, si és el cas, es dedica a entrenar un alumne que practica un esport determinat. I recordem que ja existeix en el nostre model una entitat anomenada PROFESSOR.

Una alternativa per representar aquesta realitat consistiria a crear una interrelació ternària, anomenada, per exemple, Entrena, entre les entitats ALUMNE, ESPORT i PROFESSOR (figura 2.10).

FIGURA 2.10. Exemple d'interrelacions redundants



D'aquesta manera, pot semblar que les interrelacions Practica i Entrena es poden combinar en una única interrelació. Però això no és del tot cert, ja que hi haurà interrelacions entre ALUMNE i ESPORT que no disposaran necessàriament d'un professor que actuï com a entrenador.

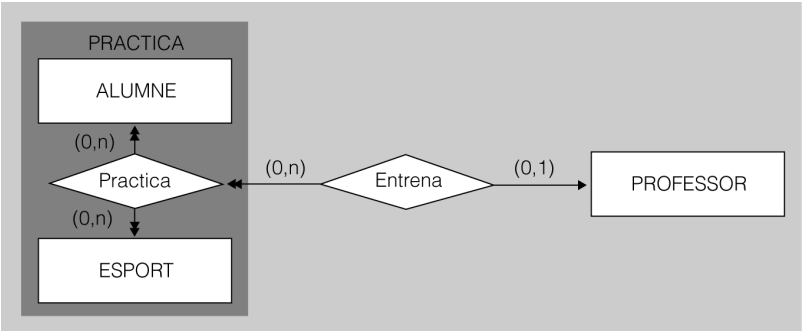
Ara bé, hi ha informació redundant en l'esquema proposat fins ara, ja que tota combinació entre instàncies de les entitats ALUMNE i ESPORT que hi ha a Entrena també és a Practica.

Si l'entrenador només fos un valor, ens podríem plantejar simplement afegir un atribut a la interrelació Practica, que es digués, per exemple, Entrenador. Però en existir una entitat (PROFESSOR) que conté la instància aplicable a cada cas, quan és necessari, hem de descartar aquesta possibilitat.

Així, doncs, la millor manera de reflectir totes aquestes circumstàncies és fer ús d'una agregació d'entitats. En aquest cas, cal considerar la interrelació Practica, entre ALUMNE i ESPORT, com una altra entitat de nivell més alt, anomenada PRACTICA. I, seguidament, es pot establir una interrelació binària amb cardinalitat 1-N entre PROFESSOR i l'agregació PRACTICA, i anomenar-la Entrena, i que inclogui les combinacions necessàries entre ambdues, per tal de modelitzar qui entrena la pràctica dels esports per part dels alumnes, quan es produeix aquesta circumstància (figura 2.11).

Les agregacions d'entitats en els diagrames ER es representen com una agrupació rectangular de les entitats i relacions que integren.

FIGURA 2.11. Exemple d'agregació d'entitats sense redundància



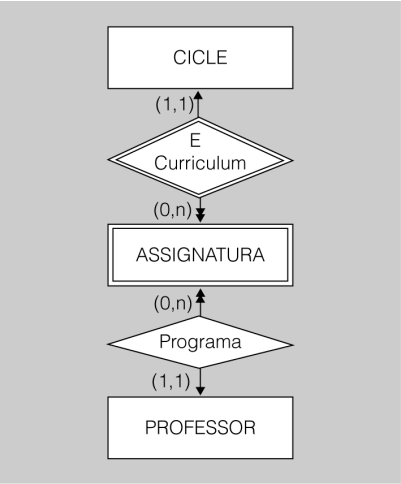
La tècnica de les agregacions engloba la de les entitats febles, però encara resulta més potent: sempre que fem servir una entitat dèbil, la podrem substituir per una agregació, però no a l'inrevés. Ara bé, cal mantenir les entitats febles en el model ER perquè, tot i que resulten menys complexes que les agregacions, normalment són suficients per modelitzar la majoria de les situacions que es produeixen al món real.

Exemple de substitució d'una entitat feble per una agregació

Recuperem l'entitat feble ASSIGNATURA. Ara imaginem que, per establir un cert control en matèria de coordinació pedagògica, es necessita saber qui és el professor responsable de realitzar la programació didàctica de cada assignatura.

Entre PROFESSOR i ASSIGNATURA es pot establir una interrelació binària de cardinalitat 1-N per representar aquest fet, que s'anomena, per exemple, Programa.

FIGURA 2.12



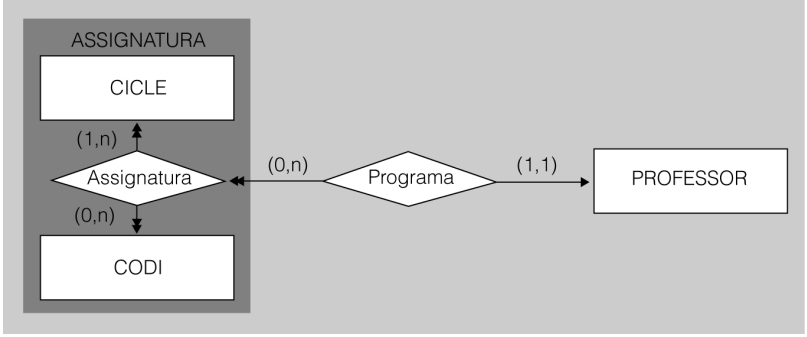
Però, alternativament, podríem modelitzar aquesta dada convertint ASSIGNATURA en una agregació.

Per aconseguir-ho, en primer lloc hauríem de considerar una nova entitat, i anomenar-la per exemple CODI, que emmagatzemés simplement codis d'assignatura (com ara C1, C2, C3, etc.).

A continuació, hauríem d'establir, d'una banda, una interrelació binària de cardinalitat N-M entre CICLE i CODI, i anomenar-la Assignatura. I, d'altra banda, també hauríem d'obtenir una agregació de la interrelació entre CICLE i CODI (és a dir, Assignatura).

Finalment, hauríem d'interrelacionar l'agregació resultant amb l'entitat PROFESSOR, amb una senzilla interrelació binària amb cardinalitat 1-N, anomenada Programa (figura 2.13).

FIGURA 2.13. Exemple de substitució d'una entitat feble per una agregació (resultat)



Notació

Les agregacions d'entitats es representen incloent dins d'un requadre totes les entitats que participen en una interrelació determinada.

Les agregacions també es representen freqüentment incloent, dins d'un requadre, només el rombe de la interrelació de la qual provenen les entitats implicades.

Des d'una interrelació, es pot fer arribar una una fletxa (de punta senzilla o doble, per tal d'expressar la cardinalitat 1 o N, respectivament) fins al rombe inclòs dins del requadre que indica l'existència d'una agregació (o bé fins al mateix requadre, exactament igual que si es tractés d'una simple entitat).

Tota agregació ha de tenir el mateix nom que la interrelació sobre la qual es defineix.

2.2.3 Exemple: BD d'un institut de formació professional

Ara desenvolupem un exemple de disseny conceptual de BD, corresponent a un institut de formació professional, per il·lustrar per separat els diferents conceptes i la seva respectiva modelització. Es tracta de dissenyar una BD per gestionar el personal de l'institut (compost pels professors, i pels treballadors d'administració i serveis) i el seu alumnat, a més dels estudis impartits.

Els requisits per dissenyar una BD...

... d'un institut real segurament serien molt més nombrosos que els que hem exposat aquí, els quals estan seleccionats amb finalitats educatives, i no necessàriament en funció de la seva importància o freqüència en el món real.

Les descripcions següents resumeixen els requisits dels usuaris de la futura BD:

- Les persones que formen part de la nostra comunitat educativa s'identifiquen mitjançant el DNI (o document equivalent, com ara la targeta de residència).
- També volem conèixer, d'aquestes persones, el nom i cognoms, l'adreça, un (i només un, de moment) telèfon de contacte, i la data de naixement.
- A més, hem de tenir registrada la localitat de residència, tot tenint en compte que la BD ha de poder emmagatzemar, per a altres usos, localitats on no visqui ningú.
- Tota persona de la comunitat educativa pertany, com a mínim, a un dels tres subtipus següents:
  - Professors
  - Alumnes
  - Personal d'administració i serveis
- Les persones només poden mantenir un tipus de relació laboral amb el centre educatiu. Per tant, els professors no poden pertànyer simultàniament al col·lectiu del personal d'administració i serveis.
- Tampoc no està permès que els professors es matriculin en el centre de treball en cap dels estudis impartits. Per tant, els professors no es poden considerar simultàniament alumnes.

- En canvi, sí està permès als integrants del col·lectiu d'administració i serveis que es matriculin, fora del seu horari laboral, en algun dels estudis impartits. Per tant, el personal d'administració i serveis pot pertànyer simultàniament a la categoria d'alumne.

- Hem de tenir constància del sou dels professors i del personal d'administració i serveis.

- Organitzativament, tot professor està assignat a un sol departament. I cada departament té assignat un dels seus professors com a coordinador.

- Tot professor té reconeguda una especialitat determinada (o més d'una). Però internament, la BD de l'institut només necessita registrar quins dels professors que té assignats el centre pertanyen a les especialitats d'informàtica o d'administració, per tal d'assignar-los tasques específiques addicionals a les docents que els són pròpies.

  – De cada informàtic, voldrem saber les especialitats professionals, quan n'hi hagin, tant en l'àmbit del maquinari com també en el del programari.
  – De cada administratiu, voldrem conèixer la titulació acadèmica i l'especialitat professional, si en té.

- Els alumnes poden practicar alguns esports a les instal·lacions del centre. I, fins i tot, poden disposar d'alguns professors com a entrenadors personals, que s'han ofert voluntàriament per realitzar aquesta tasca.

- Com és lògic, en tractar-se d'un centre de formació professional, l'institut del nostre exemple ofereix diferents estudis estructurats en cicles formatius, i cada cicle formatiu té les seves pròpies assignatures. Ens interessa, doncs, codificar les assignatures de la mateixa manera que es fa en el currículum oficial del cicle formatiu al qual pertanyen. El problema és que aquests codis es repeteixen per a tots els cicles formatius, ja que la codificació sempre consisteix en una C (per ser la inicial de la paraula *crèdit*) seguida d'un número enter (C1, C2, C3, i així successivament).

- Dins d'un mateix cicle formatiu, es pot exigir als alumnes que, per matricular-se en algunes assignatures, hagin superat alguna assignatura (o més d'una).

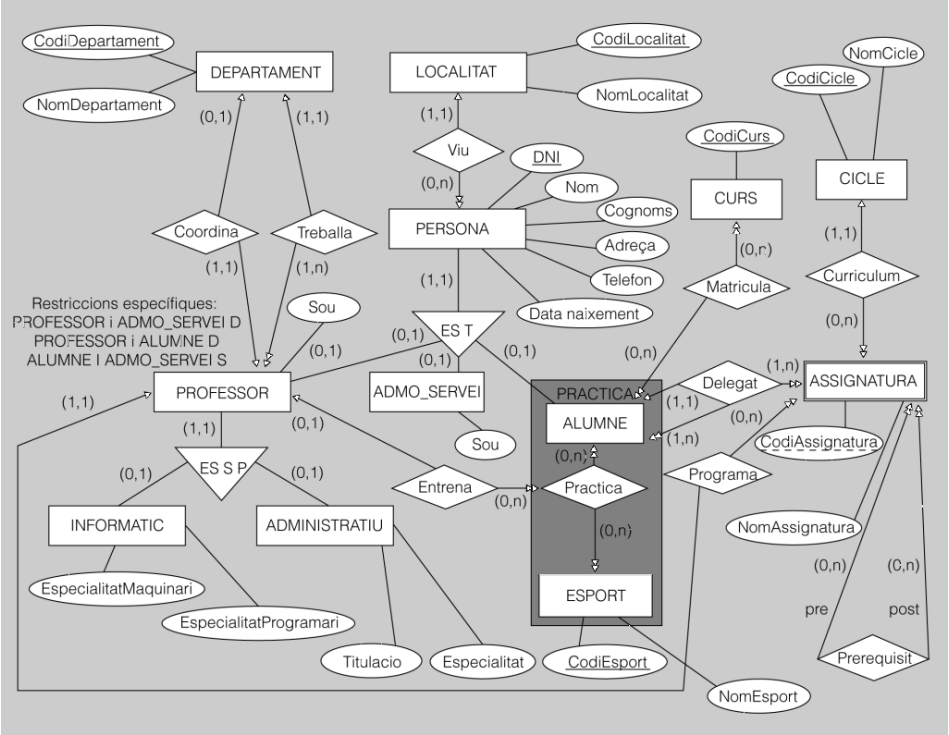
- D'altra banda, sempre hi ha un professor encarregat de realitzar la programació didàctica de cada assignatura. Un mateix professor, però, es pot encarregar de la programació didàctica de més d'una assignatura.

- Tots els alumnes del centre tenen un company que actua com a delegat en l'àmbit d'una assignatura i s'encarrega, per exemple, de distribuir els materials o les bateries d'exercicis. Un mateix alumne pot actuar com a delegat en l'àmbit de més d'una assignatura. Però cada alumne només tindrà un delegat en cada assignatura en què estigui matriculat.

- Finalment, la BD ha de recollir a quines assignatures està matriculat cada alumne en cada curs acadèmic, i la nota final obtinguda.

La figura 2.14 mostra un diagrama ER que compleix els requisits esmentats anteriorment.

FIGURA 2.14. Exemple: BD d'un institut de formació professional



A continuació, es mostra una llista de totes les entitats que apareixen en el diagrama, acompanyades dels respectius atributs (subratllats si formen part d'una clau primària).

- DEPARTAMENT
  - CodiDepartament, NomDepartament
- LOCALITAT
  - CodiLocalitat, NomLocalitat
- PERSONA
  - DNI, Nom, Cognoms, Adreça, Telefon, DataNaixement
- PROFESSOR (subclasse de PERSONA)
  - DNI, Sou
- ADMO_SERVEI (subclasse de PERSONA)
  - DNI, Sou
- ALUMNE (subclasse de PERSONA)
  - DNI
- INFORMATIC (subclasse de PROFESSOR)

  - DNI, EspecialitatMaquinari, EspecialitatProgramari

- ADMINISTRATIU (subclasse de PROFESSOR)
  - DNI, Titulacio, Especialitat
- ESPORT
  - CodiEsport, NomEsport
- CURS
  - CodiCurs
- CICLE
  - CodiCicle, NomCicle
- ASSIGNATURA (entitat feble: CodiAssignatura la identifica parcialment, i necessita el codi del cicle corresponent per tal d'identificar-se completament)
  - CodiAssignatura, NomAssignatura

Finalment, cal comentar alguns dels aspectes més complexos d'aquest model, proporcionat a tall d'exemple:

- Les subclasses en què s'especialitza PROFESSOR (INFORMATIC i ADMINISTRATIU) són encavalcades (E) entre elles, i a més a més parcials (P):
  - Són encavalcades perquè les instàncies de la superclasse poden pertànyer simultàniament a ambdues categories.
  - Són parcials, perquè no totes les instàncies de la superclasse han de pertànyer necessàriament a alguna de les dues categories.
- Les subclasses que donen lloc a la generalització de PERSONA (PROFESSOR, ADMO_SERVEI i ALUMNE) són totals, perquè tota instància de la superclasse ha de pertànyer simultàniament a alguna de les tres subclasses esmentades. Ara bé, respecte al fet de si poden pertànyer simultàniament a diferents subclasses o no, tenen restriccions específiques, i les combinen de dues en dues. Aquesta particularitat està especificada textualment dins del diagrama:
  - PROFESSOR i ADMO_SERVEI: les entitats són disjuntes entre elles, perquè les instàncies respectives no poden pertànyer al mateix temps a totes dues.
  - PROFESSOR i ALUMNE: es dóna la mateixa circumstància que en el cas anterior.
  - ALUMNE i ADMO_SERVEI: les entitats són encavalcades entre elles, perquè les instàncies respectives sí poden pertànyer al mateix temps a totes dues.

- Entre DEPARTAMENT i PROFESSOR hi ha dues interrelacions perquè serveixen per modelitzar dues realitats diferents: la coordinació del departament per part d'un professor (amb cardinalitat 1:1), i el fet que una pluralitat de professors estiguin adscrits al mateix (amb cardinalitat 1:N).

- La localitat de residència de les persones s'ha implementat mitjançant una entitat independent, i no com un simple atribut de l'entitat PERSONA. D'aquesta manera, s'evitaran redundàncies, perquè cada localitat només es registrarà un cop dins de la BD, tot i que després s'interrelacionarà amb totes les instàncies de PERSONA que calgui.

- S'ha optat per establir una agregació a partir de la interrelació Practica, per tal de permetre establir una altra interrelació (Entrena) entre aquesta i PROFESSOR, que evita la redundància de dades que hi hauria si s'hagués utilitzat una interrelació ternària entre ALUMNE, ESPORT i PROFESSOR, ja que contindria totes les combinacions de la interrelació entre ALUMNE i ESPORT. I no es podria implementar simplement una ternària entre ALUMNE, ESPORT i PROFESSOR, i suprimir la binària esmentada, perquè els alumnes també poden practicar els esports per lliure, sense cap professor que els entreni.

- Per representar la figura de l'alumne delegat d'assignatura, ha calgut recórrer a una interrelació recursiva ternària, ja que és necessari interrelacionar cada alumne que actua com a delegat amb els seus alumnes representats i, a més, amb l'assignatura de què es tracti en cada cas.

- Per representar els prerequisits de matriculació, hem afegit una altra interrelació recursiva, en aquest cas binària, que serveix per associar les assignatures entre elles quan és necessari.

- Fixem-nos que la interrelació ternària Matricula, entre CURS, ALUMNE i ASSIGNATURA, amb cardinalitat M:N:P, té un atribut propi per tal d'emmagatzemar la nota final de cada alumne.

3. Annex: Decisions de disseny

El disseny de les BD consisteix a definir una estructuració de les dades tal que satisfaci les necessitats dels futurs usuaris del sistema d'informació que es vol construir.

Per tal de satisfer com cal els requeriments funcionals dels usuaris, el dissenyador de BD haurà de considerar els diferents tipus d'operacions a realitzar sobre les dades.

El dissenyador haurà de prendre certes decisions en la modelització de les dades, que fins i tot poden comportar la revisió de l'esquema trobat inicialment, com per exemple en els àmbits següents:

- Ús d'entitats o d'atributs
- Ús d'entitats o d'interrelacions
- Ús d'una interrelació n-ària o de diferents interrelacions binàries
- Ubicació dels atributs de les interrelacions
- Ús de l'entitat DATA

El dissenyador també haurà de detectar i evitar els paranys de disseny que es puguin produir en fer conceptualitzacions errònies del món real, com ara:

- Encadenament erroni d'interrelacions binàries 1-N
- Ús incorrecte d'interrelacions binàries M-N
- Falses interrelacions ternàries

A continuació, cal plasmar totes aquestes decisions en una documentació que permeti continuar treballant en les fases de disseny posteriors.

Pot ser interessant per al dissenyador de BD conèixer algunes notacions alternatives que permetin representar els mateixos conceptes del model ER de manera equivalent, com ara l'estàndard UML (*unified modeling language*), o llenguatge unificat de modelització, el qual permet modelar simultàniament dades i funcionalitats, i que està especialment orientat al disseny global de dades i d'aplicacions que s'hagin d'implementar preferentment mitjançant llenguatges de programació orientats a objectes.

3.1 Alternatives de disseny

Una de les característiques fonamentals del model ER és que és molt flexible. Tant és així, que una mateixa realitat pot ser modelitzada de diferents maneres pel dissenyador, el qual de vegades disposa d'alternatives a l'hora de definir les entitats i les seves interrelacions.

3.1.1 Ús alternatiu d'entitats o d'atributs

De vegades, un mateix objecte del món real es pot representar mitjançat un atribut o una entitat.

L'exemple que s'utilitza parteix de la informació referida en l'apartat "Captura i abstracció dels requeriments de dades" del nucli d'activitat "Disseny de BD".

Considerem la ja coneguda entitat DOCUMENT, del model ER que hem elaborat per la XBIC, i que compta amb els atributs Signatura, Titol, AnyPublicacio, Import i ExclosPrestec. Es podria argumentar que el títol del document podria constituir una entitat per ell mateix. Els motius són els següents:

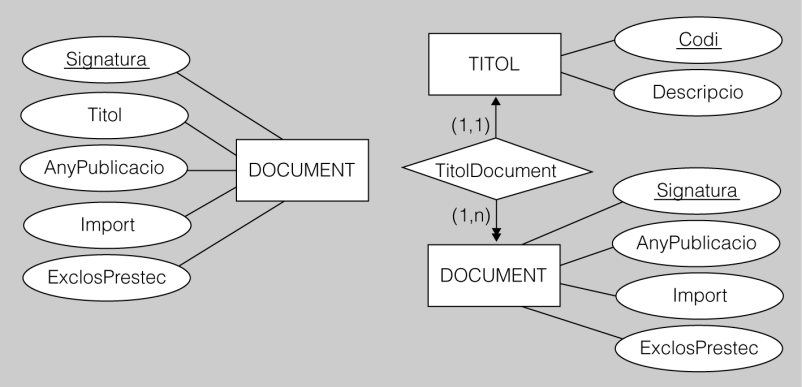
- Una biblioteca pot disposar de més d'un exemplar d'alguns títols (per exemple, una biblioteca podrà tenir dos o tres exemplars d'una mateixa novel·la si la demanda ho aconsella).
- Una biblioteca també pot disposar d'un mateix títol en diferents formats, cadascun constituirà un document diferent (pensem en el cas, per exemple, d'un mateix títol per a una obra que està disponible en format llibre, còmic i DVD).
- Finalment, un mateix títol pot estar disponible a més d'una biblioteca, i la futura BD ha de possibilitar el préstec interbibliotecari de documents i, per tant, la seva consulta.

Si s'accepta aquest punt de vista, l'entitat DOCUMENT originària s'hauria de tornar a definir de la manera següent:

- L'entitat DOCUMENT es quedaria amb els atributs Signatura, AnyPublicacio, Import i ExclosPrestec, i perdria l'atribut Titol.
- Hi hauria una nova entitat, anomenada TITOL. En previsió de la possible coincidència d'un mateix títol per a diferents obres, establim un atribut anomenat Codi, com a clau primària de l'entitat, i un altre atribut anomenat Descripcio, que emmagatzemarà els diferents títols de què disposin els fons bibliogràfic i documental de la XBIC.
- Seria necessari establir una interrelació entre les entitats TITOL i DOCUMENT, amb cardinalitat 1-N, i anomenar-la, per exemple, TitolDocument, per tal de reflectir l'associació entre cada títol i els documents respectius.

En la figura 3.1, es poden veure dos diagrames alternatius segons si es tracta el títol dels documents com un atribut o bé com una entitat.

Figura 3.1. Ús alternatiu d'entitats o d'atributs



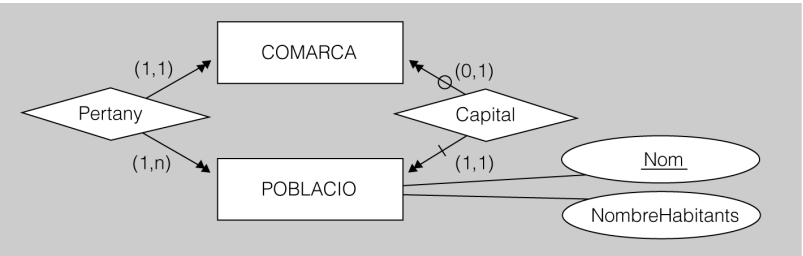
Quines són, aleshores, les diferències fonamentals entre les dues opcions considerades? Tractar un concepte del món real com una entitat en lloc de com un atribut comporta certs avantatges:

- Evita redundàncies de dades, ja que un mateix valor (com per exemple el títol d'un document) només s'introduirà un cop (i no un cop per l'atribut de cada exemplar), la qual cosa permet el següent:
  - Estalviar espai en la BD.
  - Minimitzar la possibilitat d'error dels usuaris (i al mateix temps facilitar-los la correcció).
  - Optimitzar les consultes sobre la BD i potenciar-ne la coherència dels resultats.

- Assigna una cardinalitat (1 o N), i uns límits sobre aquesta, sense recórrer a l'ús d'atributs multivaluats (els quals no són directament implementables en el model relacional, que continua essent el model lògic més utilitzat), de tal manera que podem assignar 0, 1 o més valors en cada cas, segons la realitat que correspongui modelitzar.
- Inclou informació addicional afegint nous atributs a l'entitat creada o, si no, relacionant aquesta amb altres entitats.

Així, doncs, estem en condicions d'afegir, per exemple, un nou atribut a l'entitat POBLACIO que ens indiqui el nombre d'habitants, o bé d'establir una nova interrelació amb COMARCA que ens indiqui quina és la capital de cadascun d'aquests ens territorials, tal com es pot veure en la figura 3.2.

Figura 3.2. Nous atributs



Evidentment, aquestes opcions no haurien estat possibles si haguéssim conceptualitzat les poblacions com a simples atributs d'algunes entitats del model (concretament de COMARCA, IES i USUARI).

Això no significa que sempre és recomanable l'ús d'entitats abans que no pas d'atributs. El primer que hauríem de fer abans d'adoptar una decisió en aquest sentit seria examinar si l'atribut en qüestió emmagatzemarà valors repetits, ja que en cas contrari, normalment, serà preferible obtenir només un objecte (una entitat) en lloc de tres (dues entitats i una interrelació), la qual cosa comporta un resultat molt més compacte.

En definitiva, el fet de tractar un concepte com a entitat és una opció més general que no pas tractar-lo com a atribut, la qual permet emmagatzemar informació addicional, afegint nous atributs o bé establint noves interrelacions.

Ara bé, decidir-se per aquesta opció només té sentit quan resulta d'alguna utilitat. Per exemple, difícilment es podria defensar el tractament del nom propi dels usuaris com a entitat per si mateix. Encara que, de ben segur, es produiran repeticions de valors en aquest atribut, utilitzar una entitat per representar-lo només complicaria l'esquema resultant, però no reflectiria millor la realitat que es vol modelitzar ni, en principi, aportaria cap avantatge respecte a l'opció inicial.

3.1.2 Ús alternatiu d'entitats o d'interrelacions

De vegades, és millor representar un objecte del món real mitjançant una entitat i, d'altres vegades, com una interrelació.

Com a regla general, podem fer les afirmacions següents:

- Les entitats consisteixen en objectes del món real, independentment del fet que existeixin físicament com, per exemple un cotxe, o que tinguin un caràcter més aviat abstracte, com ara una pòlissa d'assegurança. Habitualment, ens hi referim amb substantius.
- En canvi, les interrelacions, haurien de servir per representar accions o processos que tenen lloc entre entitats. És freqüent referir-s'hi utilitzant verbs (encara que sigui amb participis).

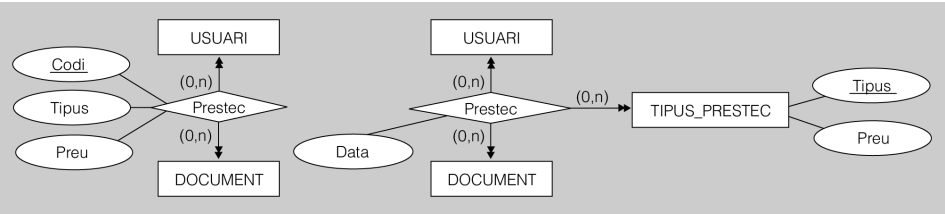
Considerar el préstec de documents als usuaris com a una interrelació amb tres atributs propis (Data, Tipus i Preu) provoca certs problemes, ja que els atributs descriptius Tipus i Preu tenen molts pocs valors possibles:

- préstec normal, gratuït

- préstec interbibliotecari, 1,20 €
- préstec a domicili, 1,50 €

Aquestes tres parelles de valors es repetiran per a cada préstec, ocuparan inútilment molt espai d'emmagatzemament i, encara pitjor, deixaran en les mans dels usuaris de la BD (o, com a màxim, dels informàtics que programin aplicacions contra la BD), a cada nova inserció, la responsabilitat de la consistència de les dades, ja que aquests atributs mai no haurien de tenir valors diferents dels esmentats.

Figura 3.3. Ús alternatiu d'entitats o d'interrelacions



Una possibilitat per evitar aquesta problemàtica consistiria a considerar l'existència d'una entitat, anomenada TIPUS_PRESTEC, amb dos atributs, que serien Tipus, com a clau primària, i Preu. Aleshores es podria establir una interrelació ternària de cardinalitat M-N-P entre USUARI, DOCUMENT i PRESTEC, que només incorporés l'atribut Data. Podem veure l'esquema plantejat inicialment i l'alternativa que acabem de descriure en la figura 3.3.

3.1.3 Ús alternatiu d'interrelacions binàries o ternàries

Les interrelacions més freqüents que es troben en les BD són binàries.

De vegades, certes interrelacions que en principi no semblen binàries es podrien plantejar més encertadament amb un conjunt d'interrelacions de grau 2.

Exemple d'una interrelació originàriament ternària

La interrelació d'un fill amb el seu pare i la seva mare (amb cardinalitat N-1-1).

Seria, doncs, més encertat plantejar dues interrelacions binàries que interrelacionessin per separat el fill i el pare, d'una banda, i el fill i la mare, d'una altra (amb cardinalitats N-1).

D'aquesta manera, encara que no constés la paternitat, es podria registrar correctament la maternitat. En canvi, fent servir una interrelació ternària no tindríem aquesta possibilitat.

D'altra banda, sempre és possible (la qual cosa no vol dir recomanable) representar les interrelacions ternàries amb cardinalitat M-N-P (i, per extensió, les n-àries de qualsevol ordre *n*, amb cardinalitat similar) amb un conjunt de tres interrelacions binàries (o de *n*, tractant-se d'una *n-ària* d'ordre superior).

Per aconseguir-ho, cal seguir els passos següents:

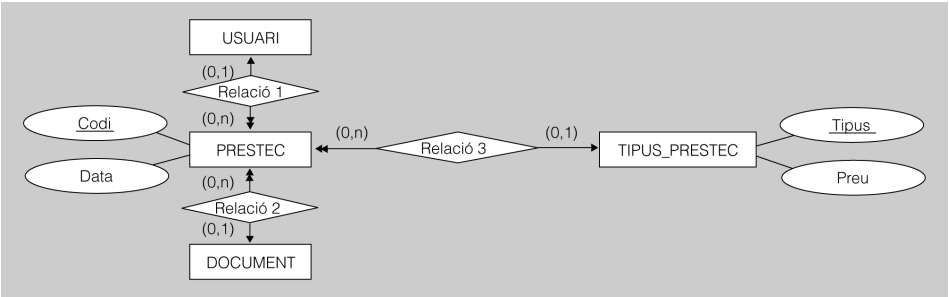
- Convertir la interrelació inicial en una entitat.
  - S'ha d'establir un atribut identificador que actuï com a clau primària.
  - Si la interrelació originària té atributs, aquests s'han d'incorporar a la nova entitat.

- Establir n interrelacions binàries entre la nova entitat i cadascuna de les n entitats preexistents.

Cal dir que aquest procés és reversible, és a dir, que es pot seguir de manera inversa.

En la figura 3.4, es mostra la conversió de la interrelació ternària Préstec (vegeu figura 3.6) en una nova entitat i tres noves interrelacions binàries.

FIGURA 3.4. Ús alternatiu d'interrelacions binàries o ternàries



Per tant, si no féssim cap altraries. Però això no seria desitjable gairebé mai pels motius següents:

- L'atribut identificador de l'entitat en què convertíssim l'entitat n-ària originària, juntament amb el conjunt d'interrelacions binàries necessàries, normalment comportarien un increment de la complexitat del disseny obtingut i, per tant, també comportarien un augment dels requeriments ulteriors d'emmagatzemament de la BD.
- Una interrelació n-ària mostra més clarament les entitats directament associades que no pas un conjunt d'interrelacions binàries.

Finalment, cal dir que quan alguna cardinalitat de la interrelació n-ària originalment plantejada no és N, sinó 1, no es pot utilitzar el mecanisme de traducció referit més amunt sense pèrdua de significat en el model resultant. Per tant, en aquests casos, mai no s'ha d'utilitzar aquesta alternativa.

Pensem, per exemple, en una interrelació ternària que modelitzés les destinacions del professorat als diferents centres d'ensenyament a l'inici de cada curs acadèmic. Seria una interrelació ternària entre PROFESSOR, CURS i CENTRE amb cardinalitat M-N-1. Doncs bé, si apliquéssim la metodologia que hem explicat, el model resultant (amb una nova entitat i tres noves interrelacions binàries) no podria reflectir la circumstància en què un professor, durant un curs concret, només pot ser destinat a un sol centre docent. En canvi, les cardinalitats de la interrelació ternària reflectirien aquest fet sense cap mena d'ambigüitat.

3.1.4 Ubicació dels atributs de les interrelacions

Les cardinalitats de les interrelacions poden afectar la situació dels seus atributs.

Tenim les possibilitats següents en les interrelacions binàries:

- Interrelacions binàries amb cardinalitat 1-1 i 1-N
- Interrelacions binàries amb cardinalitat M-N i interrelacions n-àries

Interrelacions binàries amb cardinalitat 1-1 i 1-N

Hi poden haver atributs adscrits directament a una interrelació binària amb cardinalitat 1-1, en lloc d'estar associats a alguna de les dues entitats participants.

Interrelacions binàries amb cardinalitat 1-1 i 1-N

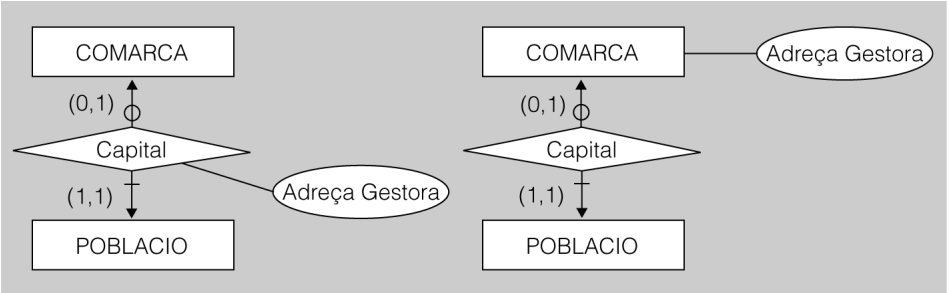
Per exemple, podem afegir un atribut a la interrelació Capital, entre les entitats COMARCA i POBLACIO, i anomenar-lo AdreçaGestora, perquè emmagatzemi l'adreça corresponent de la delegació (o gestora) territorial del Departament d'Educació.

L'altra opció consisteix a afegir l'atribut a una de les dues entitats interrelacionades:

- Quan no se sap de qui depèn l'existència de les entitats, resulta indiferent associar els atributs de la interrelació a qualsevol de les dues entitats implicades.
- Però quan una de les dues entitats és opcional en la interrelació, com en aquest cas l'entitat COMARCA, només podem optar entre associar l'atribut a la interrelació o bé a l'entitat opcional. En cap cas no hem d'associar-lo amb l'entitat obligatòria, ja que es generarien atributs amb valors nuls (en aquest exemple, seria el cas, d'altra banda majoritari, de les poblacions que no són capital de comarca).

La figura 3.5 mostra un exemple de cadascuna de les dues possibilitats esmentades.

Figura 3.5. Ubicació d'atributs a les interrelacions binàries amb cardinalitat 1-1



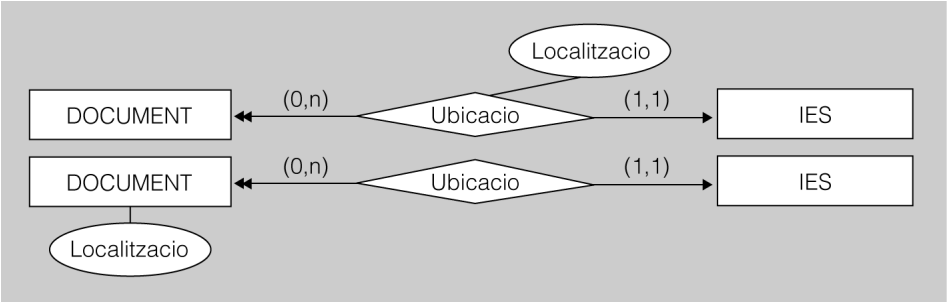
En el cas de les interrelacions binàries amb cardinalitat 1-N, també hi poden haver atributs directament associats amb la interrelació, en lloc d'estar-ho amb alguna de les dues entitats participants:

Per exemple, podem afegir un atribut, anomenat Localitzacio, a la interrelació Ubicació, existent entre les entitats DOCUMENT i IES, per tal de facilitar la localització física dels documents dins de cada institut (típicament estaran a la biblioteca, però alguns podran estar als departaments didàctics, als laboratoris, a la sala de professors, etc., en raó del seu ús habitual, encara que al mateix temps es puguin prestar).

L'altra opció vàlida consisteix a afegir l'atribut a l'entitat interrelacionada al costat de la N. En cap cas no l'hem d'associar amb l'entitat del costat de l'1, ja que aleshores només es podria indicar un mateix valor per a totes les interrelacions entre entitats:

En aquest exemple, si afegíssim el nou atribut considerat a l'entitat IES, que és al costat 1 de la interrelació, constaria que tots els documents de cada institut estarien a la mateixa ubicació, i això no reflectiria la realitat que es vol modelitzar. En canvi, si afegim el nou atribut a l'entitat DOCUMENT, que és al costat N, podrem indicar sense problemes la localització concreta de cada document dins de l'institut respectiu (vegeu figura 3.6).

Figura 3.6. Interrelacions binàries amb cardinalitat 1-N



En aquests casos, fer dependre els atributs descriptius de la interrelació o d'una de les entitats (sempre que l'equivalència sigui possible) és una decisió de disseny que pot contribuir a reflectir millor o pitjor les característiques pròpies de la porció del món real que es vol modelitzar, encara el model lògic resultant serà el mateix en tots dos casos.

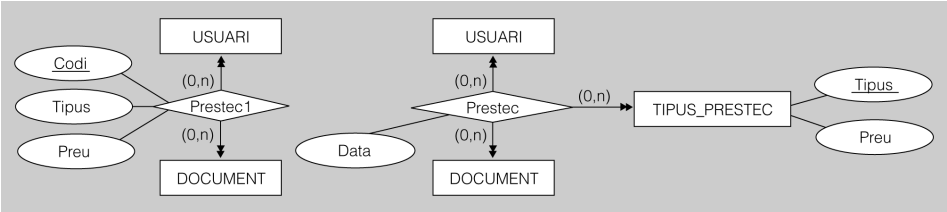
Interrelacions binàries amb cardinalitat M-N i interrelacions n-àries

En interrelacions binàries amb cardinalitats M-N, i en interrelacions ternàries o n-àries d'ordre superior, independentment de les cardinalitats, la ubicació dels atributs descriptius és molt més clara, i no hi ha equivalències:

- Sempre que un atribut descrigui una característica d'una entitat, ha de dependre directament d'aquesta.
- En canvi, quan el valor d'un atribut es determina en funció d'una combinació d'instàncies de les entitats que participen en la interrelació, només pot estar associat amb la interrelació.

Examinem, per exemple, la interrelació ternària Prestec de la figura 3.7. L'atribut Data no és una dada que respongui exclusivament dels usuaris de la xarxa de biblioteques, ni dels documents existents, ni tampoc dels tipus de préstec que es poden realitzar. Cada valor de l'atribut Data només tindrà sentit aplicat a una combinació d'instàncies de les tres entitats que participen en la interrelació (USUARI, DOCUMENT i TIPUS_PRESTEC), la qual constitueix una modalitat de préstec d'un document a un usuari en una data determinada. Per tant, en aquest cas, Data haurà d'acompanyar necessàriament la interrelació Prestec. En canvi, si l'afegíssim a una de les tres entitats abans esmentades, no ens serviria per modelitzar l'aspecte cronològic dels préstecs.

Figura 3.7. Ús alternatiu d'entitats o d'interrelacions



3.1.5 L'entitat DATA

Considerem una interrelació anomenada Prestec (amb un atribut per enregistrar la data) entre les entitats USUARI, DOCUMENT i TIPUS_PRESTEC, amb cardinalitat M-N-P.

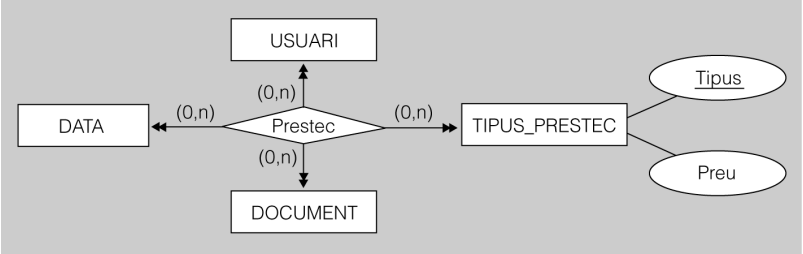
Concebuda la interrelació Prestec d'aquesta manera, permet representar la circumstància en què un document concret es presta, d'una manera determinada (segons el tipus de préstec de què es tracti), a un usuari de la xarxa de biblioteques, però només en una única data.

Això vol dir que si un usuari demana un document que ja se li ha prestat, no podrà formalitzar el préstec, encara que el document estigui disponible, perquè el sistema no ho permetrà.

Però la xarxa de biblioteques permet, evidentment, que un usuari pugui tornar a demanar en préstec un document, encara que se li hagi prestat en algun altre moment anterior. Per tant, l'estructura actual constitueix una errada de disseny, perquè la realitat no està ben modelitzada.

Una possible solució consistiria a afegir al diagrama una entitat abstracta, anomenada DATA, que participés de la interrelació Préstec amb cardinalitat N. D'aquesta manera, el sistema permetria registrar préstecs d'un mateix tipus, d'un mateix document, i a un mateix usuari, tantes vegades com fos necessari, això sí, en dates diferents. Podem observar el model resultant en la figura 3.8, on la interrelació ternària originària passa a convertir-se en quaternària.

FIGURA 3.8. Exemple d'ús de l'entitat abstracta DATA



DATA és una entitat abstracta que es fa servir molt sovint en els diagrames ER, afegint-la a una interrelació, per tal de modelitzar la possibilitat que una mateixa combinació d'instàncies de la resta d'entitats associades es pugui tornar a produir en més d'un instant.

Fixem-nos que hem fet servir una entitat molt útil, anomenada DATA, però que té una elaboració molt abstracta. A diferència de les altres entitats, no existeix com a tal en el món real. I també al contrari que la resta d'entitats, no acabarà donant lloc a una representació tabular, per si mateixa, en la futura BD.

Només cal fer servir l'entitat DATA quan la cardinalitat aplicable en connectar-la a la interrelació de què es tracti sigui N. En canvi, si ha de ser 1, es pot continuar utilitzant un atribut associat a la interrelació (i de fet, és millor així, perquè el diagrama resultant serà més compacte).

Si allò que ens interessa modelitzar no són les dates, sinó les hores, podem, simplement, anomenar aquesta entitat abstracta HORA. I, finalment, si el que en realitat volem modelitzar són dates i hores, també li podem dir DATA_HORA.

3.2 Paranys de disseny

Anomenem **paranys de disseny** les conceptualitzacions errònies del món real, produïdes durant la fase de disseny conceptual, que tenen repercussions negatives tant en la modelització inicial com en la implementació ulterior de la BD.

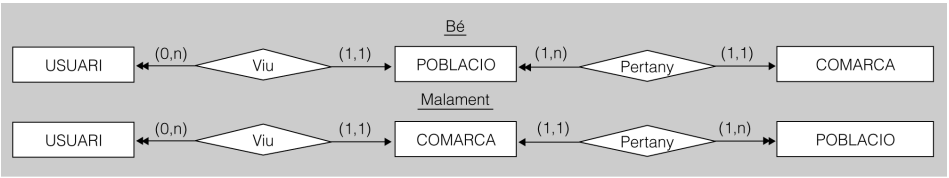
Aquests paranys poden comportar la impossibilitat de representar les dades tal com són, o bé la impossibilitat de realitzar determinades consultes sobre aquestes.

3.2.1 Encadenament erroni d'interrelacions binàries 1-N

L'**encadenament erroni d'interrelacions** es pot produir sempre que ens trobem amb dues (o més) interrelacions de cardinalitat 1-N mal encadenades. Considerem una entitat (A) que està associada amb una altra (B), que al mateix temps ho està amb una tercera entitat C. Aleshores, si en l'aplicació errònia de la transitivitat s'associa directament l'entitat A amb la C, es pot produir un error conceptual que provoqui pèrdua d'informació.

Amb el diagrama erroni de la figura 3.9, per exemple, no es reflecteix a quina població viu cada usuari, ja que a cada comarca pertany una pluralitat de poblacions. En canvi, amb l'esquema originari sí que es pot determinar, en primer lloc, a quina població viu cada usuari i, a continuació, si ens interessa, a quina comarca pertany cadascuna de les poblacions obtingudes.

Figura 3.9. Exemple d'encadenament erroni d'interrelacions amb pèrdua d'informació

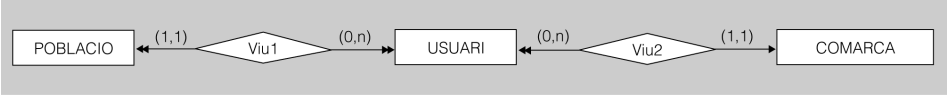


Aquest parany pot comportar la impossibilitat de resoldre correctament totes les consultes que s'haurien de poder fer sobre les dades. És molt important, doncs, triar correctament les associacions realment necessàries per tal de modelitzar correctament la realitat.

Altres vegades, l'encadenament erroni d'interrelacions no produeix una pèrdua d'informació, estrictament, ja que es poden realitzar totes les consultes necessàries sobre les dades, encara que sigui de manera ineficient. El problema principal rau en el fet que, en esborrar-se instàncies de l'entitat central, poden quedar desconnectades algunes de les instàncies de les entitats exteriors.

En la figura 3.10, podem veure una modelització errònia que ens permet registrar i consultar, tot i que de manera ineficient, l'associació entre instàncies de POBLACIO i COMARCA.

Figura 3.10. Exemple d'encadenament erroni d'interrelacions amb desconnexió d'instàncies



El problema principal deriva del fet que aquesta associació s'ha de fer mitjançant l'entitat USUARI. Si no hi ha cap usuari que visqui ni en una població ni en una comarca concretes, el sistema no permetrà associar aquestes dues instàncies (és a dir, no es podrà registrar a quina comarca està ubicada la població en qüestió). El

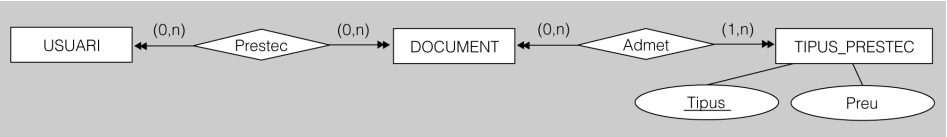
mateix impediment es produirà en cas que esborrem tots els usuaris que permeten associar una població amb una comarca concreta: l'associació entre ambdues deixarà d'existir.

3.2.2 Ús incorrecte d'interrelacions binàries M-N

L'ús de dues interrelacions binàries, encadenades i de cardinalitat M-N serà erroni sempre que en el món real existeixi algun tipus d'associació entre les instàncies de les entitats de tots dos extrems, ja que aquesta no quedarà reflectida en el model. La solució consistirà a substituir les dues interrelacions binàries per una de ternària, amb cardinalitat M-N-P.

El model proposat en la figura 3.11 només permetria enregistrar els préstecs de documents als usuaris, i els tipus de préstec que admet cada document. Però no permetria emmagatzemar el tipus de préstec que té lloc en cada cas. La manera de solucionar aquesta mancança consisteix a associar les tres entitats (USUARI, DOCUMENT i TIPUS_PRESTEC) en una interrelació ternària amb cardinalitat M-N-P.

FIGURA 3.11. Exemple d'ús incorrecte d'interrelacions binàries M-N

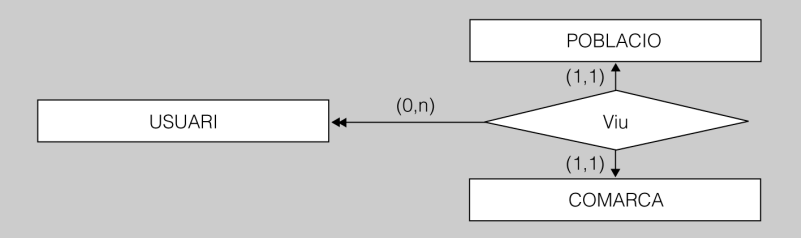


3.2.3 Falses interrelacions ternàries

Quan alguna interrelació ternària (o n-ària d'ordre superior) té associada alguna entitat amb cardinalitat 1, s'ha d'estudiar detingudament, ja que és possible que aquesta entitat estigui directament relacionada només amb una sola de les altres entitats i que, per tant, no hagi de participar en la interrelació examinada, sinó en una de binària amb l'entitat amb la qual manté realment una associació.

En la figura 3.12, es pot veure com s'utilitza innecessàriament una interrelació ternària per indicar la població i la comarca de residència dels usuaris. De fet, només caldria fer un encadenament (això sí, correcte) de dues interrelacions binàries (una entre USUARI i POBLACIÓ i una altra entre POBLACIÓ i COMARCA) amb les cardinalitats adients, tal com apareix al diagrama superior de la figura 3.9, per tal d'obtenir un model molt més compacte.

Figura 3.12. Exemple de falsa interrelació ternària



D'altra banda, si l'entitat connectada amb un 1 només té un atribut, normalment és preferible tractar-la com un atribut de la interrelació, ja que aquesta opció també contribueix a simplificar l'esquema resultant.

Model relacional i normalització

Isidre Guixà Miranda
Adaptació de continguts: Carlos Manuel Martí Hernández

Índex

Introducció

Els sistemes gestors de bases de dades són grans aplicacions informàtiques pensades per facilitar la gestió de dades, i perquè aquesta gestió sigui eficient és necessari i convenient que les dades tinguin un disseny adequat. Això s'aconsegueix seguint adequadament alguns mètodes de disseny com ara el disseny Entitat-Relació.

Partim, doncs, del supòsit que tenim bases de dades ben dissenyades i ens correspon implementar-les en un sistema gestor de bases de dades. Al llarg de la història de la informàtica s'han succeït diversos models de sistemes gestors de bases de dades: jeràrquics, en xarxa i relacionals. La majoria dels sistemes gestors de bases de dades actuals es basen en el model relacional i, per tant, ens hem de centrar en el seu estudi.

En l'apartat "Model relacional", es veurà com s'estructuren les dades i quines regles té associades aquest model. A banda d'això, cal destacar l'apartat destinat a la transformació del model ER en el model relacional, ja que aquesta serà una tècnica imprescindible per al disseny lògic de BD.

En l'apartat "Normalització" s'introdueix un seguit de conceptes que formen la coneguda teoria de la normalització, la qual permet detectar errors en el disseny de la base de dades i facilita mecanismes per a la seva correcció. En principi, tot el model relacional derivat d'un model Entitat-Relació previ, correctament elaborat, hauria de ser també correcte. Però en ocasions això no és així, i aleshores, en l'explotació d'una base de dades es poden detectar problemes que fan necessari un estudi del disseny del model relacional, per si cal introduir modificacions en el mateix. Així mateix, hi ha dissenyadors que no volen modelitzar la realitat amb el model Entitat-Relació i volen dissenyar directament el model relacional. Cal dir, d'altra banda, que aquesta problemàtica és molt freqüent quan les bases de dades d'una certa complexitat han estat dissenyades directament utilitzant el model relacional, sense haver dissenyat prèviament cap model entitat-relació.

El coneixement tan del model Relacional com de la normalització de bases de dades és molt important per a qualsevol persona que analitzi, dissenyi o administri bases de dades relacionals, o que implementi aplicacions que interactuïn amb bases de dades

Evidentment, per tal d'adquirir uns coneixements òptims en aquesta matèria és imprescindible efectuar totes les activitats proposades, així com els exercicis d'autoavaluació.

Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Dissenya models lògics normalitzats interpretant diagrames entitat/relació.
    - Identifica els principals elements del model relacional: relacions, atributs, domini dels atributs, diferents tipus de claus i cardinalitat de les relacions.
    - Identifica i interpreta les regles d'integritat associades a cadascuna de les claus primàries.
    - Identifica i interpreta les regles d'integritat associades a cadascuna de les claus foranies, tenint en compte les diferents possibilitats de modificar i/o esborrar (eliminació i/o modificació en cascada, restricció de l'eliminació i/o modificació, eliminació i/o modificació aplicant valors nuls als registres relacionats).
    - Identifica les taules, camps i les relacions entre taules, d'un disseny lògic.
    - Tradueix un model Entitat-Relació a model relacional aplicant les regles corresponents de traducció.
    - Aplica les regles de normalització en el model relacional.
    - Elabora la guia d'usuari i la documentació completa relativa al disseny físic (taules, atributs i relacions) de la base de dades relacional, de manera estructurada i clara; afegint les restriccions que no poden plasmar-se en el disseny lògic.

1. Model relacional

El model relacional és un model de dades basat en dues disciplines matemàtiques: la lògica de predicats i la teoria de conjunts.

Potser a causa d'aquest sòlid fonament teòric, que proporciona a aquest model una robustesa excepcional, els SGBD relacionals (o SGBDR) són actualment els que tenen una implantació més gran en el mercat.

SGBD

Acrònim de Sistema Gestor de Bases de Dades. És un programari especialitzat en la gestió de bases de dades (enteses, aquestes, com un conjunt estructurat d'informació).

El model relacional va ser proposat originàriament per Edgar Frank Codd en el seu treball *A Relational Model of Data for Large Shared Data Banks* ('Un model relacional de dades per a grans bancs de dades compartits') l'any 1970, tot i que no es va implementar comercialment fins al final de la dècada.

E. F. Codd

Codd treballava per a IBM, però no va ser aquesta multinacional qui va creure abans en les possibilitats del model relacional, sinó més aviat la competència, i molt especialment *Oracle*, empresa que va néixer, justament, amb el nom de Relational Software.

1.1 Estructuració de les dades

El **model relacional** permet construir estructures de dades per representar les diferents informacions del món real que tinguin algun interès.

Les estructures de dades construïdes seguint el model relacional estan formades per conjunts de relacions.

Les **relacions** poden ser concebudes com a representacions tabulars de les dades.

Tuple

En l'àmbit de les BD, podem definir *tuple* com una seqüència finita d'objectes que comprèn les diferents associacions entre cada atribut de la relació i un valor concret, admissible dins del domini respectiu.

Cal precisar els extrems següents:

- Tota relació ha de tenir un nom que la identifiqui unívocament dins de la base de dades.
- Cada fila està constituïda per un tuple de dades relacionades entre elles, anomenat també *registre*, que guarda les dades que ens interessa reflectir d'un objecte concret del món real.
- En canvi, cada columna conté, en cada cel·la, dades d'un mateix tipus, i se la pot anomenar *atribut* o *camp*.

- Cada cel·la, o intersecció entre fila i columna, pot emmagatzemar un únic valor.

Exemple de relació

La taula 1.1 reflecteix l'estructuració tabular de la relació ALUMNE, que conté les dades personals corresponents als individus matriculats en un centre docent.

Cada fila conté unes quantes dades relacionades que, en aquest cas, són les que pertanyen a un mateix alumne.

La relació té un nom (ALUMNE), com cadascuna de les columnes (DNI, Nom, Cognoms i Telefon). Si aquests noms són prou significatius, permeten copsar de seguida el sentit que tenen els valors de les dades emmagatzemades en la relació.

TAULA 1.1. Exemple de relació

ALUMNE			
DNI	**Nom**	**Cognoms**	**Telefon**
47126654F	Josep	Bel Rovira	453641282
51354897S	Anna	Pacheco Cuscó	723352151
56354981L	Xavier	Rius Montalvo	726922235

Tota base de dades relacional està formada per un conjunt de relacions.

Aquesta senzilla manera de visualitzar l'estructura de les bases de dades relacionals resulta molt entenedora per a la majoria d'usuaris. Però cal aprofundir en algunes característiques addicionals de les relacions, per talde poder-les distingir clarament dels fitxers tradicionals.

1.1.1 Domini

Pel que fa al model relacional, un **domini** consisteix en un conjunt finit de valors indivisibles.

Els atributs només poden prendre els valors que estiguin inclosos dins del domini respectiu. Altrament no són valors vàlids, i un SGBD relacional no en pot permetre l'emmagatzematge.

Exemples de dominis

Examinem l'atribut Telefon de la relació ALUMNE. Si el definim de tal manera que només pugui emmagatzemar nou caràcters (perquè els telèfons sempre consten de nou dígits) de tipus numèric (ja que les lletres no poden formar part d'un número de telèfon), el domini d'aquest atribut inclourà totes les combinacions possibles (en concret, 10^9, que és una magnitud gran, però finita).

Una altra cosa és que molts d'aquests valors no es podran correspondre mai amb valors existents en el món real (per exemple, difícilment un operador assignarà a un dels seus

abonats una cadena de nou zeros com a identificador telefònic). Per aconseguir-ho, caldria restringir força més el domini de l'atribut a l'hora de definir-lo.

Centrem-nos ara en l'atribut Cognoms. Contindrà els valors dels dos cognoms dels alumnes que els tinguin, separats per un espai en blanc. Per tant, aquest camp està definit per tal que pugui emmagatzemar dos objectes del món real: primer cognom i segon cognom.

Conceptualment, els usuaris podran distingir entre els dos objectes representats, i els programadors d'aplicacions podran truncar, en cas necessari, el resultat obtingut en fer una consulta del camp Cognoms. Però tot SGBD relacional considerarà el valor contingut en l'atribut Cognoms de manera atòmica, sense cap estructuració interna.

Hem de considerar dues tipologies de dominis:

- **Dominis predefinits**. Són els tipus de dades que admeti cada SGBD, com, per exemple (esmentats de manera genèrica, ja que hi ha moltes especificitats en funció dels diferents sistemes gestors), les cadenes de caràcters, els nombres enters, els nombres decimals, les dades de caire cronològic, etc.
- **Dominis definits pels usuaris**. Consisteixen en restriccions addicionals aplicades sobre el domini predefinit d'alguns atributs, establertes pels dissenyadors i pels administradors de bases de dades.

Exemple de domini definit per l'usuari

En una relació per emmagatzemar les dades dels aspirants a mosso d'esquadra, es podria establir el camp IMC, per registrar els índexs de massa corporal respectius.

Doncs bé, es podria restringir el domini d'aquest camp de tal manera que no admetés aspirants amb valors inferiors a dinou ni superiors a trenta, ja que la normativa no ho permet.

1.1.2 Esquema i extensió

Tota relació consta d'un esquema (també anomenat *intensió de la relació*) i de la seva extensió.

L'**esquema d'una relació** consisteix en un nom que la identifica unívocament dins de la base de dades, i en el conjunt d'atributs que aquella conté.

És molt recomanable, per tal d'evitar confusions en la implementació ulterior, seguir uniformement una notació concreta a l'hora d'expressar els esquemes de les relacions que formen una mateixa base de dades.

A continuació, es detallen les característiques d'un dels sistemes de notació més freqüents:

- Cal escriure el nom de les relacions amb majúscules i preferiblement en singular.

- S'ha d'escriure el nom dels atributs començant amb majúscula i continuant amb minúscules, sempre que no es tracti de sigles, ja que aleshores és més convenient deixar totes les lletres amb majúscules (com ara DNI). Per tal de fer els noms compostos més llegidors, es pot encapçalar cada paraula de les que formen el nom del camp amb una lletra majúscula (per exemple: DataNaixement, TelefonParticular, etc.).

Exemple d'esquema d'una relació

L'esquema de la relació que es mostra en la taula 1.2, conforme al sistema de notació proposat, quedaria com segueix:

ALUMNE(DNI, Nom, Cognoms, Telefon)

Cal precisar que l'ordre en què ens mostrin els atributs és indiferent, per definició del model relacional.

TAULA 1.2. Exemple de relació

ALUMNE			
DNI	**Nom**	**Cognoms**	**Telefon**
47126654F	Josep	Bel Rovira	453641282
51354897S	Anna	Pacheco Cuscó	723352151
56354981L	Xavier	Rius Montalvo	726922235

Els atributs d'una relació són únics dins d'aquesta. El seu nom no pot estar repetit dins d'una mateixa relació. Ara bé, diferents relacions sí que poden contenir atributs amb el mateix nom.

D'altra banda, cal dir que els dominis de diferents atributs d'una mateixa relació poden ser idèntics, malgrat que els camps respectius emmagatzemin els valors de diferents propietats de l'objecte (per exemple, seria perfectament lògic que els atributs TelefonFix, TelefonMobil i TelefonFeina, tot i pertànyer a una mateixa relació, tinguessin el mateix domini).

L'**extensió d'una relació** consisteix en els valors de les dades emmagatzemades en tots els tuples que aquesta conté.

Exemple d'extensió

Si prenem com a base, una vegada més, la relació amb esquema ALUMNE(DNI, Nom, Cognoms, Telefon) de la taula 2, la seva extensió seria una llista en què figurarien tots els alumnes de la base de dades:

Alumne 1: 47126654F, Josep, Bel Rovira, 453641282 Alumne 2: 51354897S, Anna, Pacheco Cuscó, 723352151 Alumne 3: 56354981L, Xavier, Rius Montalvo, 726922235

De vegades, els atributs de les relacions poden no contenir cap valor o, dit d'una altra manera, poden contenir valors nuls.

Exemple de valor nul

Imaginem que s'hi matricula un quart alumne que no té telèfon. Les seves dades en la coneguda relació amb esquema ALUMNE(DNI, Nom, Cognoms, Telefon) reflectiran aquesta circumstància amb l'absència de valor en l'atribut Telefon del tuple que li correspongui.

En utilitzar representacions tabulars per visualitzar els valors de les extensions de les relacions (en el pla teòric, no en implementacions reals amb SGBD), per tal d'indicar que una cel·la té valor nul s'hi pot incloure el mot NUL (com en la taula 1.3), o bé es pot deixar en blanc, simplement.

Taula 1.3. Exemple de relació amb valors nuls

ALUMNE			
DNI	**Nom**	**Cognoms**	**Telefon**
47126654F	Josep	Bel Rovira	453641282
51354897S	Anna	Pacheco Cuscó	723352151
56354981L	Xavier	Rius Montalvo	726922235
24583215W	Mariona	Castellví Mur	NUL

El **grau d'una relació** depèn del nombre d'atributs que inclou el seu esquema.

Exemple de grau d'una relació

La relació amb esquema ALUMNE(DNI, Nom, Cognoms, Telefon) de la taula 3 és de grau 4, perquè té quatre atributs.

La **cardinalitat d'una relació** ve donada pel nombre de tuples que en formen l'extensió.

Exemple de cardinalitat

Si ens fixem en la taula 1.3, la cardinalitat de la relació ALUMNE és 4, perquè la seva extensió conté quatre tuples corresponents als quatre alumnes que, de moment, hi ha matriculats.

1.1.3 Claus candidates, clau primària i claus alternatives

Per tal de resultar útil, l'emmagatzematge de la informació ha de permetre la identificació de les dades. En l'àmbit de les bases de dades relacionals, els tuples de les relacions s'identifiquen mitjançant les anomenades *superclaus*.

Una **superclau** és un subconjunt dels atributs que formen l'esquema d'una relació tal que no és possible que hi hagi més d'un tuple en l'extensió respectiva, amb la mateixa combinació de valors en els atributs que formen part del subconjunt esmentat.

Però una superclau pot contenir atributs innecessaris, que no contribueixen a la identificació inequívoca dels diferents tuples. El que habitualment interessa és treballar amb **superclaus mínimes**, tals que cap subconjunt propi sigui capaç per sí sol d'identificar els tuples de la relació.

Per definició, cap superclau mínima no pot admetre valors nuls en cap dels seus atributs, perquè si ho fes, no podria garantir la identificació inequívoca dels tuples que continguessin algun valor nul en alguns dels atributs de la superclau mínima en qüestió.

D'altra banda, cal dir que en una mateixa relació pot passar que hi hagi més d'una superclau mínima que permeti distingir els tuples unívocament entre ells.

S'anomenen **claus candidates** totes les superclaus mínimes d'una relació formades pels atributs o conjunts d'atributs que permeten identificar els tuples que conté la seva extensió.

Tria de la clau primària

Amb molta freqüència, és l'administrador de la BD qui tria la clau primària de la relació, d'entre les claus candidates disponibles, tot realitzant, en el fons, tasques de dissenyador lògic.

Però, a l'hora d'implementar una BD, entre totes les claus candidates de cada relació només se n'ha de triar una.

Quan parlem de **clau primària** ens referim a la clau que, finalment, el dissenyador lògic de la base de dades tria per distingir unívocament cada tuple d'una relació de la resta.

Aleshores, les claus candidates no triades com a clau primària resten presents en la relació.

Quan una relació ja té establerta una clau primària, la resta de claus presents en aquella, i que també podrien servir per identificar els diferents tuples de l'extensió respectiva, es coneixen com a **claus alternatives**.

Una forma de diferenciar els atributs que formen la clau primària de les relacions, dels altres atributs de l'esquema respectiu, és posar-los subratllats. Per aquest motiu, normalment es col·loquen junts i abans que la resta d'atributs, dins de l'esquema. Però només es tracta d'una qüestió d'elegància, ja que el model relacional no es basa ni en l'ordre dels atributs de l'esquema, ni tampoc en l'ordre dels tuples de l'extensió de la relació.

Exemples de claus candidates, primària i alternatives

Observant la taula 1.4, podem imaginar que la relació ALUMNE té uns quants atributs més, de manera que el seu esquema queda com segueix:

ALUMNE(DNI, NumSS, NumMatricula, Nom, Cognoms, Telefon)

Veurem fàcilment com els atributs DNI, NumSS (número de la Seguretat Social) i NumMatricula, en ser personals i irrepetibles, ens podrien servir per identificar unívocament els alumnes. Per tant, serien claus candidates.

Aleshores, el dissenyador de BD s'haurà de decidir per una clau candidata com a clau primària. Si, per exemple, tria DNI com a clau primària, les antigues claus candidates restants es passaran a considerar claus alternatives.

En aquest cas, doncs, l'esquema resultant haurà de reflectir quina és la clau primària de la relació, tot subratllant l'atribut DNI:

ALUMNE(<u>DNI</u>, NumSS, NumMatricula, Nom, Cognoms, Telefon)

Taula 1.4. Exemple de relació amb valors nuls

ALUMNE			
DNI	**Nom**	**Cognoms**	**Telefon**
47126654F	Josep	Bel Rovira	453641282
51354897S	Anna	Pacheco Cuscó	723352151
56354981L	Xavier	Rius Montalvo	726922235
24583215W	Mariona	Castellví Mur	NUL

Si no es disposa d'un atribut que sigui capaç d'identificar els tuples de la relació per si sol, cal buscar un subconjunt d'atributs, tals que la combinació dels valors que adoptin no es pugui repetir. Si aquesta possibilitat no existeix, cal afegir a la relació un atribut addicional que faci d'identificador.

Per definició, el model relacional no admet tuples repetits, és a dir, no permet l'existència de tuples en una mateixa relació que tinguin els mateixos valors en cadascun dels atributs.

Ara bé, les implementacions concretes dels diferents SGBD sí que permeten aquesta possibilitat, sempre que no s'estableixi cap clau primària en la relació amb tuples repetits.

Aquesta permissivitat de vegades permet solucionar certes eventualitats, però no hauria de ser la manera habitual de treballar amb BD relacionals.

1.1.4 Claus foranes

Qualsevol BD relacional, per petita que sigui, conté normalment més d'una relació. Per tal de reflectir correctament els vincles existents entre alguns objectes del món real, cal que els tuples de les diferents relacions d'una base de dades es puguin interrelacionar.

De vegades, fins i tot pot ser necessari relacionar els tuples d'una relació amb altres tuples de la mateixa relació.

El mecanisme que ofereixen les BD relacionals per interrelacionar les relacions que contenen es fonamenta en les anomenades *claus foranes*.

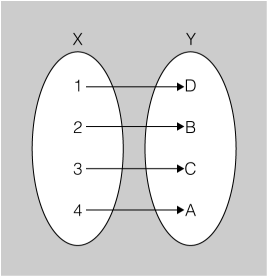
Una **clau forana** està constituïda per un atribut, o per un conjunt d'atributs, de l'esquema d'una relació, que serveix per relacionar els seus tuples amb els tuples d'una altra relació de la base de dades (o amb els tuples d'ella mateixa, en alguns casos).

Per tal d'aconseguir connectar els tuples d'una relació amb els d'una altra (o amb les seves pròpies), la clau forana utilitzada ha de referenciar la clau primària de la relació amb la qual es vol relacionar.

Les diferents combinacions de valors dels atributs de tota clau forana han d'existir en la clau primària a què fan referència, o bé han de ser valors nuls. Altrament, les referències serien errònies i, per tant, les dades serien incorrectes.

Cal parar atenció en les característiques següents de les claus foranes:

- Tota clau forana ha de tenir el mateix nombre d'atributs que la clau primària a la qual fa referència.
- Entre els atributs de l'esquema d'una clau forana i els de la clau primària respectiva s'ha de poder establir una correspondència (concretament, una bijecció).
- Els dominis dels atributs de tota clau forana han de coincidir amb els dominis dels atributs de la clau primària respectiva (o, com a mínim, cal que siguin compatibles dins d'un cert rang).



Exemple de bijecció

Una relació pot contenir més d'una clau forana, o bé no contenir-ne cap. I, en sentit invers, la clau primària d'una relació pot estar referenciada per una o més claus foràneies, o bé pot no estar referenciada per cap.

Finalment, cal dir que es pot donar el cas que un mateix atribut formi part tant de la clau primària de la relació com d'alguna de les seves claus foranes.

Exemples de claus foranes

La relació ALUMNE, tal com es mostra en la taula 1.5, incorpora dues claus foranes.

Una d'elles, CodiAula, fa referència a la clau primària de la relació AULA (formada per l'atribut Codi), exposada en la taula 1.6, per tal d'indicar quina aula correspon a cada alumne.

En canvi, DNIDelegat fa referència a la clau primària de la mateixa relació (formada per l'atribut DNI), i serveix per indicar quin és el delegat que representa cada alumne.

Fixem-nos que l'alumna Mariona Castellví encara no té assignat ni delegat ni aula i, per aquest motiu, el tuple que la representa conté, de moment, valors nuls en els atributs de les dues claus foranes.

TAULA 1.5. Exemple de relació amb claus foranes

ALUMNE					
DNI	**Nom**	**Cognoms**	**Telefon**	**DNIDelegat**	**CodiAula**
47126654F	Josep	Bel Rovira	453641282	47126654F	102
51354897S	Anna	Pacheco Cuscó	723352151	51354897S	201
56354981L	Xavier	Rius Montalvo	726922235	51354897S	201
24583215W	Mariona	Castellví Mur	NUL	NUL	NUL

TAULA 1.6. Exemple de relació amb clau primària referenciada

AULA	
Codi	**Capacitat**
101	40
102	36
201	30

La notació més habitual per designar les claus foranes de les relacions consisteix a afegir aquesta circumstància a continuació del seu esquema, incloent-hi entre dues claus el conjunt d'atributs que formen la clau forana de què es tracti, precedit de l'adverbi ON, i seguit de la forma verbal REFERENCIA i de la relació a què fa referència. Si hi ha més d'una clau forana, se separen per comes i l'última ha d'anar precedida de la conjunció I.

Exemple de notació per designar claus foranes

Les dues relacions que es mostren en les taules 5 i 6 s'expressaran de la manera següent:

ALUMNE(DNI, Nom, Cognoms, Telefon, DNIDelegat, CodiAula) ON {DNIDelegat} REFERENCIA ALUMNE i {CodiAula} REFERENCIA AULA

AULA(Codi, Capacitat)

1.1.5 Operacions amb relacions

El model relacional permet realitzar una sèrie d'operacions amb les dades emmagatzemades en les BD, les quals tenen diferents finalitats:

- **Actualització.** Aquestes operacions realitzen canvis en els tuples que queden reflectits en les relacions que contenen les BD. Poden ser de tres tipus:
  - **Inserció**. Consisteix a afegir un o més tuples nous a una relació determinada.
  - **Esborrat**. Consisteix a eliminar un o més tuples nous d'una relació determinada.
  - **Modificació.** Consisteix a canviar el valor d'un o més atributs d'un o més tuples d'una relació determinada.

- **Consulta.** Aquestes operacions només fan possible l'obtenció parametritzada de dades, sense que es vegin alterades les emmagatzemades en la BD.

La realització d'aquestes operacions comporta el coneixement previ de l'estructura formada per les relacions que sigui necessari utilitzar, és a dir, els esquemes de les relacions i les interrelacions entre elles, mitjançant les claus foranes.

Exemples d'operacions

Si prenem en consideració la relació ALUMNE que mostra la taula 1.7, un exemple d'inserció consistirà a afegir un nou alumne, com ara el següent:

```
<65618724G, Lídia, Bofarull Mora, 564628231, 47126654F, 102>
```

Un exemple d'esborrament seria eliminar el tuple que conté les dades d'un alumne donat d'alta, com ara el següent:

```
<56354981L, Xavier, Rius Montalvo, 726922235, 51354897S, 201>
```

Un exemple de modificació seria, per exemple, canviar el número de telèfon de Josep Bel Rovira que consta en la BD (453641282) per un altre (546022547), per tal de reflectir correctament la realitat, de manera actualitzada, o bé assignar-ne un de nou a algú que abans no en tenia, com la Mariona Castellví Mur, introduint 875261473 en lloc de l'anterior valor nul.

I, com a exemple de consulta, ens podria interessar obtenir una llista, ordenada alfabèticament pels cognoms, de tots els alumnes que són delegats d'aula i que, per tant, tenen valors coincidents en l'atribut DNI i en l'atribut DNIDelegat. En aquest cas, el resultat seria el següent:

```
<47126654F, Josep, Bel Rovira, 546022547, 47126654F, 102>
<51354897S, Anna, Pacheco Cuscó, 723352151, 51354897S, 201>
```

TAULA 1.7. Exemple de relació amb claus foranes

ALUMNE					
DNI	**Nom**	**Cognoms**	**Telefon**	**DNIDelegat**	**CodiAula**
47126654F	Josep	Bel Rovira	453641282	47126654F	102
51354897S	Anna	Pacheco Cuscó	723352151	51354897S	201
56354981L	Xavier	Rius Montalvo	726922235	51354897S	201
24583215W	Mariona	Castellví Mur	NUL	NUL	NUL

1.2 Regles d'integritat

Els valors que emmagatzemen les BD han de reflectir en tot moment, de manera correcta, la porció de la realitat que volem modelitzar.

Anomenem **integritat** la propietat de les dades que consisteix a representar correctament les situacions del món real que modelitzen.

Per tal que les dades siguin íntegres, cal garantir que siguin correctes, i també que estiguin senceres.

En atenció a l'objectiu esmentat, doncs, les dades han de complir certes condicions, que podem agrupar en dues tipologies diferents:

- **Restriccions d'integritat de l'usuari**. Són condicions específiques de cada BD. Els SGBD han de permetre als administradors establir certes restriccions aplicables a casos concrets, i han de garantir que es respectin durant l'explotació habitual del sistema.
- **Regles d'integritat del model**. Són condicions de caire general que han de complir totes les BD que segueixin el model relacional. No cal definir-les en implementar cada BD, perquè es consideren preestablertes.

Exemple de restricció d'integritat de l'usuari

En donar d'alta un nou alumne de la relació ALUMNE, que es mostra en la taula 8, podríem exigir al sistema que el validés, mitjançant l'algorisme corresponent, si la lletra introduïda del NIF es correspon amb les xifres introduïdes prèviament, i que denegués la inserció en cas contrari, per tal de no emmagatzemar una situació en principi no admissible en el món real.

Exemple de regla d'integritat del model

Com que la relació ALUMNE, que es mostra en la taula 1.8, té definit l'atribut DNI com a clau primària, el sistema validarà automàticament que no s'introdueixi més d'un alumne amb el mateix carnet d'identitat, ja que aleshores la clau primària no compliria el seu objectiu de garantir la identificació inequívoca de cada tuple, diferenciant-lo de la resta.

TAULA 1.8. Exemple de relació amb claus foranes

ALUMNE					
DNI	**Nom**	**Cognoms**	**Telefon**	**DNIDelegat**	**CodiAula**
47126654F	Josep	Bel Rovira	453641282	47126654F	102
51354897S	Anna	Pacheco Cuscó	723352151	51354897S	201
56354981L	Xavier	Rius Montalvo	726922235	51354897S	201
24583215W	Mariona	Castellví Mur	NUL	NUL	NUL

1.2.1 Unicitat de la clau primària

El valor d'una clau primària, globalment considerada, no pot estar repetit en més d'un tuple de la mateixa relació, ja que aleshores la clau primària no estaria en condicions d'assegurar la identificació inequívoca dels diferents tuples.

En cap moment, no hi pot haver dos o més tuples amb la mateixa combinació de valors en el conjunt dels atributs que formen la clau primària d'una relació.

Els SGBD relacionals han de garantir la regla d'unicitat de la clau primària en totes les insercions de nous tuples, i també en totes les modificacions que afectin el valor d'algun dels atributs que formin part de la clau primària.

Exemple d'unicitat de la clau primària

En la relació AULA, que es mostra en la taula 1.9, no s'hauria de poder inserir un nou tuple amb els valors <102, 40>, perquè la clau primària ja emmagatzema el valor 102, corresponent a un altre tuple.

Si volem donar d'alta una altra aula al primer pis de l'edifici amb capacitat per a quaranta alumnes, haurem d'utilitzar com a clau primària un altre valor no present en l'atribut Codi, i resultarà, per exemple, <103, 40>.

Tampoc no hauria de ser possible modificar la clau del tuple **<101, 40>** i assignar-hi el valor 102, perquè aquest valor ja el té assignat la clau primària d'un altre tuple.

Taula 1.9. Exemple de relació amb clau primària referenciada

AULA	
Codi	**Capacitat**
101	40
102	36
201	30

1.2.2 Entitat de la clau primària

Les claus primàries serveixen per diferenciar cada tuple d'una relació de la resta de tuples de la mateixa relació. Per garantir la consecució d'aquesta finalitat, cal que els atributs que formen part d'una clau primària no puguin tenir valor nul ja que, si s'admetés aquesta possibilitat, els tuples amb valors nuls en la clau primària no es podrien distingir d'alguns altres.

Cap atribut que formi part d'una clau primària no pot contenir mai valors nuls en cap tuple.

Els SGBD relacionals han de garantir la regla d'entitat de la clau primària en totes les insercions de nous tuples, com també en totes les modificacions que afectin el valor d'algun dels atributs que formin part de la clau primària.

Exemple d'entitat de la clau primària

En la relació AULA, que es mostra en la taula 1.10, no s'hauria de poder inserir un nou tuple amb els valors <NUL, 26>, perquè la clau primària, per definició, no pot contenir valors nuls.

Si volem donar d'alta una altra aula amb capacitat per a vint-i-sis alumnes, haurem d'utilitzar com a clau primària un altre valor no nul, i resultarà, per exemple, <202, 26>.

Tampoc no hauria de ser possible, per la mateixa raó que hem exposat més amunt, modificar la clau del tuple <101, 40> i assignar-hi el valor nul.

TAULA 1.10. Exemple de relació amb clau primària referenciada

AULA	
Codi	**Capacitat**
101	40
102	36
201	30

1.2.3 Integritat referencial

El model relacional no admet, per definició, que la combinació de valors dels atributs que formen una clau forana no sigui present en la clau primària corresponent, ja que això implicaria una connexió incorrecta.

La **integritat referencial** implica que, per a qualsevol tuple, la combinació de valors que adopta el conjunt dels atributs que formen la clau forana de la relació o bé ha de ser present en la clau primària a la qual fa referència, o bé ha d'estar constituïda exclusivament per valors nuls (si els atributs implicats admeten aquesta possibilitat, i així s'ha estipulat en definir-ne les propietats).

Els SGBD relacionals hauran de fer les comprovacions pertinents, de manera automàtica, per tal de garantir la integritat referencial, quan es produeixin dos tipus d'operacions amb relacions que tinguin claus foranes:

- Insercions de nous tuples.
- Modificacions que afectin atributs que formin part de qualsevol clau forana.

D'altra banda, els SGBD relacionals també hauran de validar la correcció d'uns altres dos tipus d'operacions amb relacions que tinguin la clau primària referenciada des d'alguna clau forana:

- Esborraments de tuples.
- Modificacions que afectin atributs que formin part de la clau primària.

Per tal de garantir la integritat referencial en aquests dos últims tipus d'operació, es pot seguir alguna de les tres polítiques següents: restricció, actualització en cascada i anul·lació.

Exemple de violació de la integritat referencial

Continuem especulant amb les relacions ALUMNE i AULA (reflectides en la taula 1.11 i taula 1.12 respectivament).

El tuple que conté les dades de Mariona Castellví Mur té un valor nul en l'atribut que forma la clau forana que fa referència a la relació AULA.

Si el volguéssim actualitzar amb el valor 316, per exemple, el sistema no ens ho hauria de deixar fer, perquè aquest valor no és present en la clau primària de cap tuple de la relació AULA i, per tant, aquesta operació contravindria la regla d'integritat referencial.

TAULA 1.11. Relació amb claus foranes

ALUMNE					
DNI	**Nom**	**Cognoms**	**Telefon**	**DNIDelegat**	**CodiAula**
47126654F	Josep	Bel Rovira	453641282	47126654F	102
51354897S	Anna	Pacheco Cuscó	723352151	51354897S	201
56354981L	Xavier	Rius Montalvo	726922235	51354897S	201
24583215W	Mariona	Castellví Mur	NUL	NUL	NUL

TAULA 1.12. Relació amb clau primària referenciada

AULA	
Codi	**Capacitat**
101	40
102	36
201	30

Política de restricció

La política de restricció consisteix a prohibir l'operació d'actualització de què es tracti:

- En cas d'esborrament, no permetrà eliminar un tuple si la seva clau primària està referenciada des d'alguna clau forana.
- En cas de modificació, no permetrà alterar el valor de cap dels atributs que formen la clau primària d'un tuple, si aquesta està referenciada des d'alguna clau forana.

Exemples de restriccions

Considerem una vegada més les relacions ALUMNE i AULA que es mostren en la taula 1.13 i taula 1.14 respectivament.

Aplicant la restricció tant en cas d'esborrament com de modificació, aquestes operacions no seran possibles amb l'aula 102 de la relació AULA perquè hi ha alumnes matriculats que han d'assistir a classe dins d'aquest espai i, per tant, la referencien des de la clau forana dels tuples que els representen. Sí seria possible, en canvi, esborrar l'aula 101, perquè no està referenciada des de la relació ALUMNE.

Taula 1.13. Relació amb claus foranes

ALUMNE					
DNI	**Nom**	**Cognoms**	**Telefon**	**DNIDelegat**	**CodiAula**
47126654F	Josep	Bel Rovira	453641282	47126654F	102
51354897S	Anna	Pacheco Cuscó	723352151	51354897S	201
56354981L	Xavier	Rius Montalvo	726922235	51354897S	201
24583215W	Mariona	Castellví Mur	NUL	NUL	NUL

Taula 1.14. Relació amb clau primària referenciada

AULA	
Codi	**Capacitat**
101	40
102	36
201	30

Actualització en cascada

La política d'actualització en cascada consisteix a permetre l'operació d'actualització de què es tracti sobre un tuple determinat, però disposant al mateix temps una sèrie d'operacions compensatòries que propaguin en cascada les actualitzacions necessàries per tal que es mantingui la integritat referencial dels tuples que referencien, des dels atributs que en formen la clau forana, el tuple objecte d'actualització:

- En cas d'esborrament, s'eliminaran tots els tuples que facin referència al tuple esborrat.
- En cas de modificació, els valors dels atributs que formin part de la clau forana dels tuples que facin referència al tuple modificat s'alteraran per tal de continuar coincidint amb els nous valors de la clau primària del tuple al qual fan referència.

Exemples d'actualització en cascada

Tornem a prendre com a punt de partida dels exemples les relacions ALUMNE i AULA que es mostren en la taula 1.15 i taula 1.16 respectivament.

Si apliquem l'actualització en cascada tot esborrant el tuple <201, 30> de la relació AULA, també s'esborraran els dos tuples de la relació ALUMNE que hi fan referència des de la clau forana respectiva (CodiAula).

En canvi, si apliquem l'actualització en cascada tot modificant el tuple <201, 30> de la relació AULA, canviant el valor de la seva clau primària per un altre, com ara 203, els dos tuples de la relació ALUMNE que hi fan referència actualitzaran en cascada el valor de l'atribut CodiAula de 201 a 203, per tal de mantenir la connexió correcta entre els tuples d'ambdues relacions.

TAULA 1.15. Relació amb claus foranes

ALUMNE					
DNI	**Nom**	**Cognoms**	**Telefon**	**DNIDelegat**	**CodiAula**
47126654F	Josep	Bel Rovira	453641282	47126654F	102
51354897S	Anna	Pacheco Cuscó	723352151	51354897S	201
56354981L	Xavier	Rius Montalvo	726922235	51354897S	201
24583215W	Mariona	Castellví Mur	NUL	NUL	NUL

TAULA 1.16. Relació amb clau primària referenciada

AULA	
Codi	**Capacitat**
101	40
102	36
201	30

Política d'anul·lació

La política d'anul·lació consisteix a permetre l'operació d'actualització de què es tracti en un tuple determinat, però disposant al mateix temps una sèrie d'operacions compensatòries que posin valors nuls en tots els atributs que formin part de les claus foranes dels tuples que facin referència al tuple objecte d'actualització:

- En cas d'esborrament, els atributs de la clau forana dels tuples que facin referència al tuple esborrat passaran a tenir valor nul, i no indicaran cap tipus de connexió.
- En cas de modificació, els atributs de la clau forana dels tuples que facin referència al tuple modificat passaran a tenir valor nul, i no indicaran cap tipus de connexió.

La política d'anul·lació només es pot aplicar si els atributs de les claus foranes implicades admeten els valors nuls.

Exemple d'anul·lació

Prenem una vegada més com a punt de partida dels exemples les relacions ALUMNE i AULA que es mostren en la taula 1.17 i taula 1.18 respectivament.

Aplicant la política d'anul·lació, tant si esborrem el tuple `<102, 36>` de la relació AULA, com si només canviem el valor de l'atribut de la seva clau primària (Codi) per un altre (com, per exemple, 105), el tuple de la relació ALUMNE que hi fa referència actualitzarà el valor de l'atribut CodiAula de 102 a valor nul, per tal d'evitar una connexió incorrecta entre els tuples d'ambdues relacions, i aleshores resultarà el tuple següent:

```
<47126654F, Josep, Bel Rovira, 453641282, 47126654F, NUL>
```

TAULA 1.17. Relació amb claus foranes

ALUMNE					
DNI	**Nom**	**Cognoms**	**Telefon**	**DNIDelegat**	**CodiAula**
47126654F	Josep	Bel Rovira	453641282	47126654F	102
51354897S	Anna	Pacheco Cuscó	723352151	51354897S	201
56354981L	Xavier	Rius Montalvo	726922235	51354897S	201
24583215W	Mariona	Castellví Mur	NUL	NUL	NUL

TAULA 1.18. Relació amb clau primària referenciada

AULA	
Codi	**Capacitat**
101	40
102	36
201	30

Selecció de la política que s'ha de seguir

Serà el dissenyador de cada BD qui escollirà la política més adequada que s'ha de seguir en cada cas concret. Com a orientació, convé saber que les opcions més freqüents, sempre que no calgui fer consideracions addicionals, són les següents:

- En cas d'esborrament, normalment s'opta per la restricció.
- En cas de modificació, el més habitual és optar per l'actualització en cascada.

La política d'anul·lació és molt menys freqüent, i es posa en pràctica quan es volen conservar certes dades, encara que hagin perdut la connexió que tenien abans, de vegades amb l'esperança que la puguin recuperar més endavant.

1.2.4 Integritat del domini

La regla d'integritat del domini implica que tots els valors no nuls que contenen els atributs de les relacions de qualsevol BD han de pertànyer als respectius dominis declarats per als atributs en qüestió.

Aquesta condició és aplicable tant pel que fa als dominis predefinits, com també pel que fa als dominis definits per l'usuari.

La regla d'integritat del domini també comporta que els operadors que és possible aplicar sobre els valors depenen dels dominis dels respectius atributs que els emmagatzemen.

Exemples d'integritat de domini

Fixem-nos, per últim cop, en la relació AULA que es mostra en la taula 1.19.

TAULA 1.19. Exemple de relació amb clau primària referenciada

AULA	
Codi	**Capacitat**
101	40
102	36
201	30

Si en la relació amb esquema AULA(Codi, Capacitat) definim el domini de l'atribut Codi com el dels nombres enters de 0 fins a 999, aleshores no podrem inserir, per exemple, un valor en l'atribut que forma la clau primària que no pertanyi al seu domini, com ara INF, o LAB.

Tampoc no podrem aplicar determinats operadors per comparar valors de la clau primària amb valors que no pertanyin al seu domini. Així, no podrem consultar les característiques d'una aula amb Codi='INF', ja que 'INF' és una cadena de caràcters.

1.3 Traducció del model Entitat-Relació al model relacional

Model Entitat Relació o model ER

El model ER és un model de dades que té com a resultat un diagrama ER o diagrama Chen on gràficament es poden identificar els principals elements de dades, les seves característiques més importants i les interrelacions entre els mateixos.

Una vegada conegudes les característiques d'un model de bases de dades relacional, caldrà partir del model conceptual general (del model Entitat-Relació) i fer un estudi del disseny lògic de bases de dades en aquest àmbit, el relacional.

En tots els exemples, pressuposarem que prèviament ha tingut lloc una fase de disseny conceptual de la qual ha resultat un model Entitat-Relació (o model ER) recollit en els diagrames Chen de què es tracti en cada cas.

Fases del disseny de BD

Les fases del disseny de BD són:

1. Disseny conceptual
2. Disseny lògic
3. Disseny físic

Abans d'implementar pròpiament la BD dins de l'entorn ofert pel SGBD utilitzat, cal transformar aquests diagrames en estructures de dades relacionals.

El model ER es basa en les entitats i en les interrelacions existents entre aquestes. Podem avançar uns quants aspectes generals sobre com s'han de traduir aquests elements al model relacional:

- Les entitats sempre donen lloc a relacions, siguin del tipus que siguin (a excepció de les entitats auxiliars de tipus DATA).
- Les interrelacions binàries de connectivitat 1-1 o 1-N originen claus foranes en relacions ja existents.
- Les interrelacions binàries de connectivitat M-N i totes les n-àries d'ordre superior a 2 sempre es transformen en noves relacions.

És convenient seguir un cert ordre a l'hora de dissenyar lògicament una base de dades. Una bona pràctica pot consistir a procedir de la manera següent:

1. En primer lloc, cal transformar les entitats del diagrama amb el qual treballem en relacions.

2. Després s'ha de continuar transformant en relacions les entitats que presenten algun tipus d'especificitat (és a dir, les febles, les associatives, o les derivades d'un procés de generalització o especialització).

3. A continuació, s'han d'afegir a les anteriors relacions els atributs necessaris per formar les claus foranes derivades de les interrelacions binàries amb connectivitat 1-1 i 1-N presents en el diagrama ER.

4. I, finalment, ja pot començar la transformació de les interrelacions binàries amb connectivitat M-N i de les interrelacions n-àries.

Cap SGBD no pot resoldre una referència a una taula que encara no ha estat creada.

D'aquesta manera evitarem que hi hagi claus foranes que facin referència a relacions que encara no s'han descrit. Això fa més llegidor el model relacional obtingut, certament, però també estalvia la feina d'haver d'ordenar les relacions a l'hora d'escriure (típicament en llenguatge SQL) i les instruccions pertinents per tal que el SGBD utilitzat creï les taules de la base de dades.

Les tècniques necessàries per realitzar correctament el disseny lògic de bases de dades, segons el tipus de conceptualització de què es tracti en cada cas.

1.3.1 Entitats

Cada entitat del model ER es transforma en una relació del model relacional:

- Els atributs de l'entitat originària seran els atributs de la relació resultant.

- La clau primària de l'entitat originària serà la clau primària de la relació resultant.

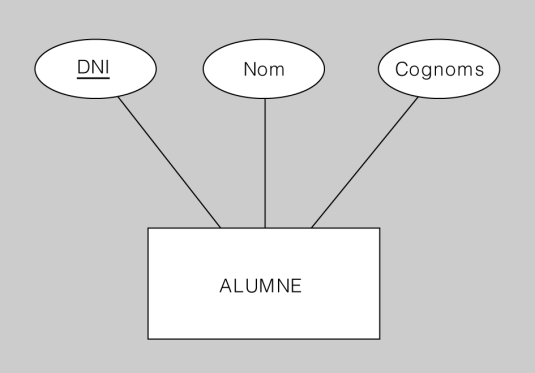
- Quan una entitat intervé en alguna interrelació binària 1-1 o 1-N, pot ser necessari afegir ulteriorment nous atributs, per tal que actuïn com a claus foranes de la relació.

Exemple de transformació d'entitat

El diagrama ER de la figura 1.1 es tradueix al model relacional de la manera següent:

ALUMNE(DNI, Nom, Cognoms)

FIGURA 1.1. Entitat



1.3.2 Interrelacions

Un cop transformades totes les entitats en relacions, cal traduir les interrelacions en què aquelles participen.

1. Binàries. Per traduir les interrelacions binàries cal tenir en compte la seva connectivitat, així com també les dependències d'existència.

a. Connectivitat 1-1 i dependències d'existència. Cal afegir a qualsevol de les dues relacions una clau forana que faci referència a l'altra relació.

Dependències d'existència

De vegades, una entitat instància només té sentit si hi ha com a mínim una altra entitat instància que hi està associada mitjançant una interrelació binària determinada. En aquests casos, es diu de la darrera entitat que és una entitat obligatòria en la interrelació. Altrament, es diu que es tracta d'una entitat opcional en la interrelació.

Però si una de les dues entitats és opcional en la relació, aleshores és ella qui ha d'acollir la clau forana, per tal d'evitar, en cas contrari, l'emmagatzematge de valors nuls en aquesta, i estalviar-se així espai d'emmagatzematge.

Els atributs de la interrelació (si n'hi ha) acompanyen la clau forana.

Exemple de transformació d'interrelació binària amb connectivitat 1-1

El diagrama ER de la figura 1.2 representa una interrelació binària amb connectivitat 1-1. Per tant, en principi hi hauria dues possibilitats de transformació, segons si es col·loca la clau forana en l'entitat PROFESSOR o en l'entitat DEPARTAMENT:

DEPARTAMENT(Codi, Descripcio)

PROFESSOR(DNI, Nom, Cognoms, CodiDepartament) ON {CodiDepartament} REFERENCIA DEPARTAMENT i CodiDepartament ADMET VALORS NULS

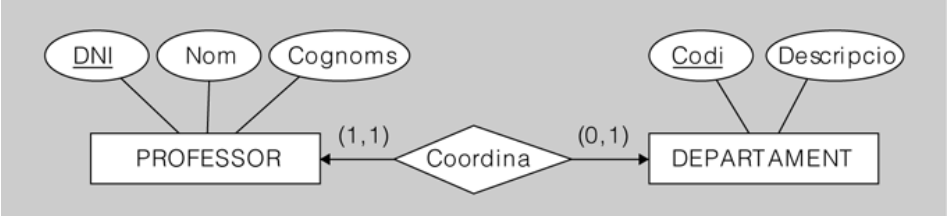
O bé:

PROFESSOR(DNI, Nom, Cognoms)

DEPARTAMENT(Codi, Descripcio, DNIProfessor) ON {DNIProfessor} REFERENCIA PROFESSOR

Ara bé, l'entitat DEPARTAMENT és opcional en la interrelació Coordina. Això vol dir que hi pot haver professors que no coordinin cap departament. Per tant, l'opció més correcta consisteix a afegir la clau forana a la relació DEPARTAMENT, ja que si s'afegís a la relació PROFESSOR hauria de prendre el valor nul en molts casos, i ocuparia un espai d'emmagatzematge innecessari.

Figura 1.2. Interrelació binària amb connectivitat 1-1



b. Connectivitat 1-N. En aquests casos cal afegir una clau forana a la relació que resulta de traduir l'entitat ubicada al costat N de la interrelació, que faci referència a l'altra relació.

Si es col·loqués la clau forana en l'altra relació, l'atribut que la forma hauria de ser multivalent per tal de poder representar totes les connexions possibles, i això no està permès dins del model relacional.

Els atributs de la interrelació (si n'hi ha) acompanyen la clau forana.

Exemple de transformació d'interrelació binària amb connectivitat 1-N

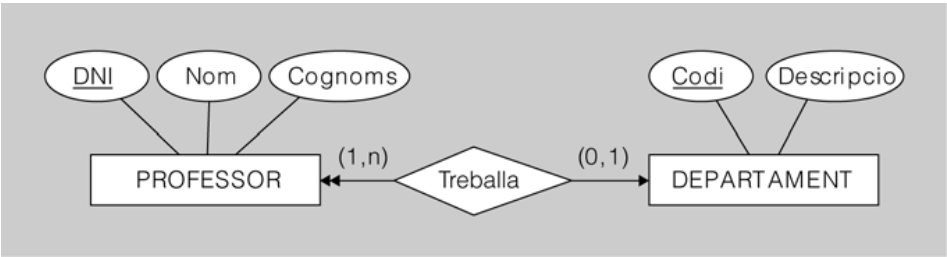
El diagrama ER de la figura 1.3 representa una interrelació binària amb connectivitat 1-N. Per tant, la clau forana s'haurà d'afegir necessàriament a l'entitat derivada de l'entitat del costat N, i resulta el model següent:

DEPARTAMENT(Codi, Descripcio)

PROFESSOR(DNI, Nom, Cognoms, CodiDepartament) ON {CodiDepartament} REFERENCIA DEPARTAMENT i CodiDepartament ADMET VALORS NULS

L'entitat del costat 1 (DEPARTAMENT) és opcional en la interrelació Treballa. Això implica que l'entitat PROFESSOR admetrà valors nuls en la seva clau forana que fa referència a DEPARTAMENT, ja que hi podrà haver professors no assignats a cap departament. Però, al contrari del que passava amb les interrelacions 1-1, aquí no es podran evitar aquests valors nuls, ja que la clau forana ha d'anar necessàriament a l'entitat que resulta de traduir al model relacional l'entitat ubicada al costat N de la interrelació.

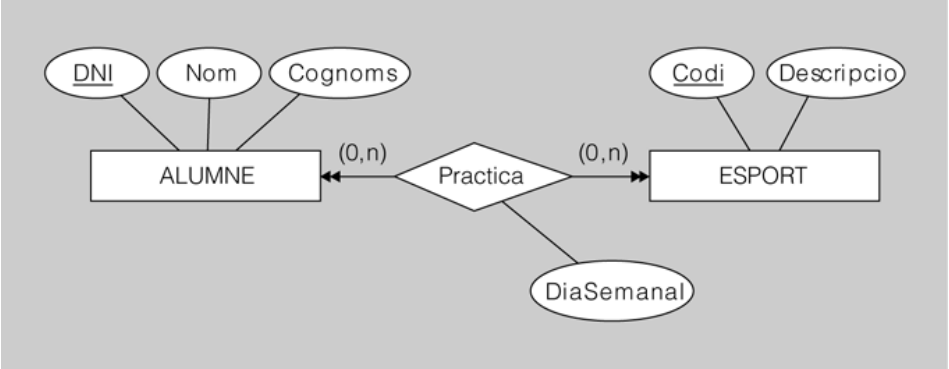
Figura 1.3. Interrelació binària amb connectivitat 1-N



c. Connectivitat M-N. Cada interrelació M-N es transforma en una nova relació amb les característiques següents:

- La seva clau primària estarà formada pels atributs de les claus primàries de les dues entitats interrelacionades.
- Els atributs de la interrelació (si n'hi ha) es convertiran en atributs de la nova relació.

FIGURA 1.4. Interrelació binària amb connectivitat M-N



Exemple de transformació d'interrelació binària amb connectivitat M-N

El diagrama ER de la figura 1.4 es tradueix al model relacional de la manera següent:

ALUMNE(DNI, Nom, Cognoms)

ESPORT(Codi, Descripcio)

PRACTICA(DNIAlumne, CodiEsport, DiaSemanal) ON {DNIAlumne} REFERENCIA ALUMNE i {CodiEsport} REFERENCIA ESPORT

2. Ternàries. Tota interrelació ternària es transforma en una nova relació, que tindrà per atributs els de les claus primàries de les tres entitats interrelacionades, més els atributs propis de la interrelació, si en té.

La composició de clau primària de la nova relació depèn de la connectivitat de la interrelació ternària originària.

a. Connectivitat M-N-P. En aquest cas, la clau primària està formada per tots els atributs que formen les claus primàries de les tres entitats interrelacionades (si no fos així, la clau primària hauria de repetir algunes combinacions dels seus valors per tal de modelitzar totes les possibilitats, però aquesta possibilitat no està permesa dins del model relacional).

Exemple de transformació d'interrelació ternària amb connectivitat M-N-P

El diagrama ER de la figura 1.5 es tradueix al model relacional de la manera següent:

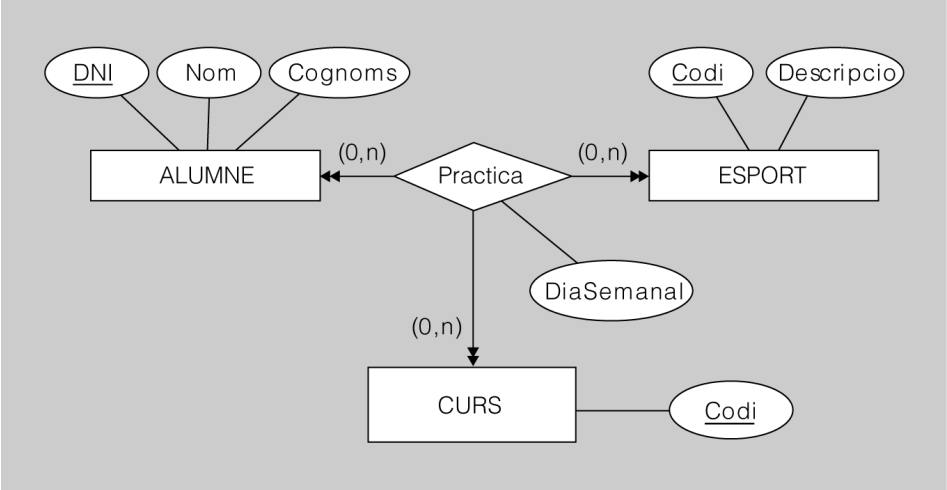
ALUMNE(DNI, Nom, Cognoms)

ESPORT(Codi, Descripcio)

CURS(Codi)

PRACTICA(DNIAlumne, CodiEsport, CodiCurs, DiaSemanal) ON {DNIAlumne} REFERENCIA ALUMNE {CodiEsport} REFERENCIA ESPORT i {CodiCurs} REFERENCIA CURS

Figura 1.5. Interrelació ternària amb connectivitat M-N-P



b. Connectivitat 1-M-N. La clau primària està composta per tots els atributs que formen les claus primàries de les dues entitats que són a tots dos costats de la interrelació etiquetats amb una N (o amb el que és equivalent, una fletxa de punta doble).

Exemple de transformació d'interrelació ternària amb connectivitat 1-M-N

El diagrama ER de la figura 1.6 es tradueix al model relacional de la manera següent:

ALUMNE(DNI, Nom, Cognoms)

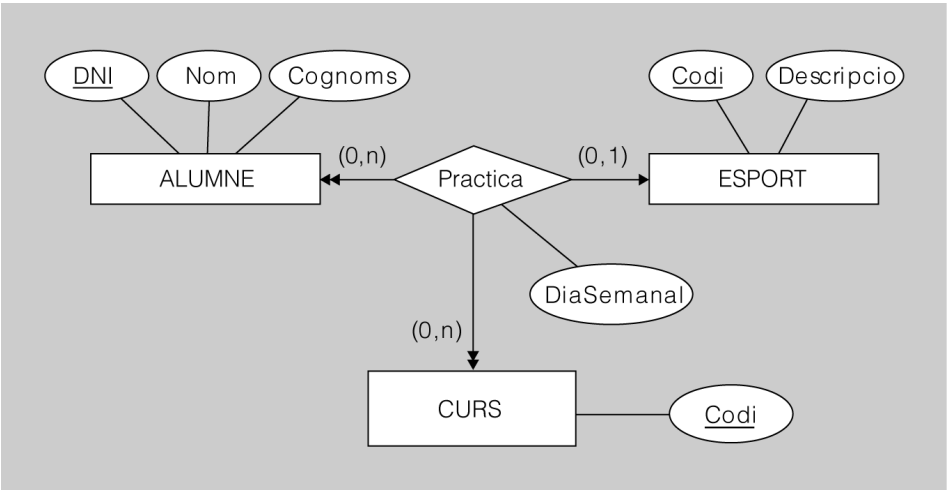
ESPORT(Codi, Descripció)

CURS(Codi)

PRACTICA(DNIAlumne, CodiEsport, CodiCurs, DiaSemanal) ON {DNIAlumne} REFERENCIA ALUMNE {CodiEsport} REFERENCIA ESPORT, I {CodiCurs} REFERENCIA CURS

Fixem-nos que, en aquest cas, un alumne només pot practicar un esport en cada curs acadèmic i, per tant, no cal incorporar la clau de l'entitat ESPORT a la clau de la relació PRACTICA.

Figura 1.6. Interrelació ternària amb connectivitat 1-M-N



c. Connectivitat 1-1-N. En aquests casos, la clau primària està composta pels atributs que formen la clau primària de l'entitat del costat N de la interrelació, més els atributs que formen la clau primària de qualsevol de les altres dues entitats connectades amb cardinalitat 1.

Així, doncs, tota nova relació derivada d'una interrelació ternària amb connectivitat 1-1-N disposarà de dues claus candidates. L'elecció d'una d'aquestes com a clau primària de la nova relació quedarà al criteri del dissenyador lògic de BD.

Exemple de transformació d'interrelació ternària amb connectivitat 1-1-N

El diagrama ER de la figura 1.7 es pot traduir al model relacional de dues maneres:

ALUMNE(DNI, Nom, Cognoms)

ESPORT(Codi, Descripció)

CURS(Codi)

COORDINACIO(CodiCurs, DNIAlumne, CodiEsport, DiaSemanal) ON {DNIAlumne} REFERENCIA ALUMNE {CodiEsport} REFERENCIA ESPORT i {CodiCurs} REFERENCIA CURS

O bé:

ALUMNE(DNI, Nom, Cognoms)

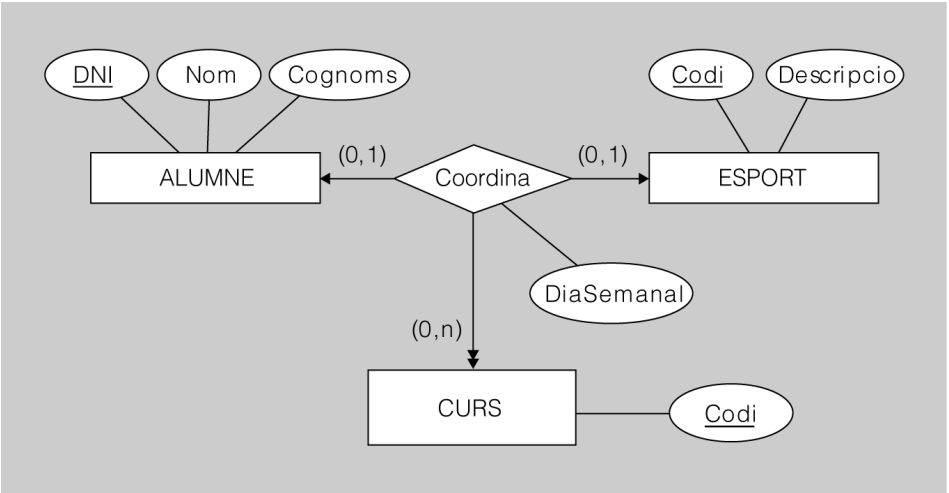
ESPORT(Codi, Descripció)

CURS(Codi)

COORDINACIO(CodiCurs, CodiEsport, DNIAlumne, DiaSemanal) ON {DNIAlumne} REFERENCIA ALUMNE {CodiEsport} REFERENCIA ESPORT i {CodiCurs} REFERENCIA CURS

Es pot veure que hem modificat el nom de la relació derivada de la interrelació per tal de convertir el verb originari en un substantiu, que normalment és més adequat per designar relacions.

FIGURA 1.7. Interrelació ternària amb connectivitat 1-1-N



d. Connectivitat 1-1-1. En aquests casos, la clau primària està composta pels atributs que formen la clau primària de dues entitats qualssevol, ja que totes tres estan connectades amb cardinalitat 1.

Així, doncs, tota nova relació derivada d'una interrelació ternària amb connectivitat 1-1-1 disposarà de tres claus candidates. L'elecció d'una d'aquestes com a clau primària de la nova relació quedarà al criteri del dissenyador lògic de BD.

Exemple de transformació d'interrelació ternària amb connectivitat 1-1-1

El diagrama ER de la figura 1.8 es pot traduir al model relacional de tres maneres:

ALUMNE(DNI, Nom, Cognoms)

ESPORT(Codi, Descripció)

CURS(Codi)

COORDINACIO(CodiCurs, DNIAlumne, CodiEsport, DiaSemanal) ON {DNIAlumne} REFERENCIA ALUMNE {CodiEsport} REFERENCIA ESPORT i {CodiCurs} REFERENCIA CURS

O bé:

ALUMNE(DNI, Nom, Cognoms)

ESPORT(Codi, Descripció)

CURS(Codi)

COORDINACIO(CodiCurs, CodiEsport, DNIAlumne, DiaSemanal) ON {DNIAlumne} REFERENCIA ALUMNE {CodiEsport} REFERENCIA ESPORT i {CodiCurs} REFERENCIA CURS

O bé:

ALUMNE(DNI, Nom, Cognoms)

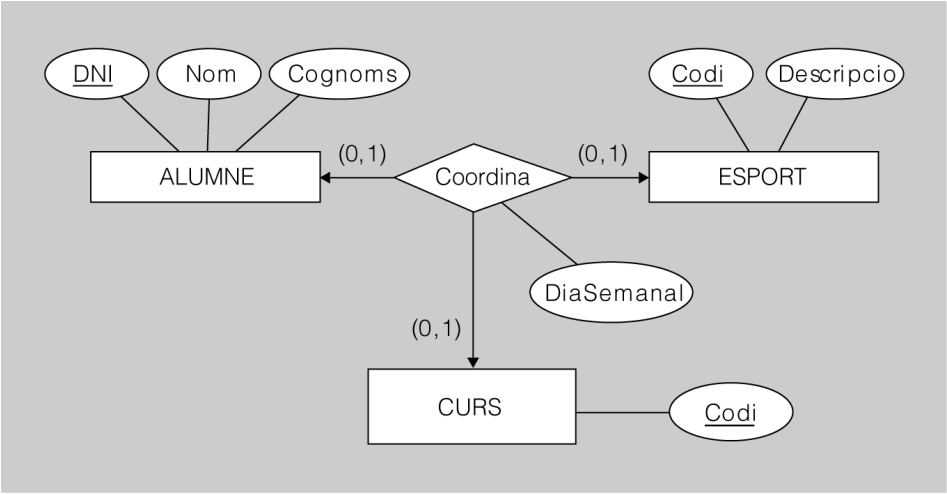
ESPORT(Codi, Descripció)

CURS(Codi)

COORDINACIO(CodiEsport, DNIAlumne, CodiCurs, DiaSemanal) ON {DNIAlumne} REFERENCIA ALUMNE {CodiEsport} REFERENCIA ESPORT i {CodiCurs} REFERENCIA CURS

Fixem-nos que hem canviat el significat del diagrama respecte al que hem representat en la figura 1.7: ara un alumne només pot coordinar la pràctica d'un esport durant un sol curs acadèmic, al llarg dels seus estudis, per tal d'afavorir la rotació en els càrrecs de coordinació del centre.

Figura 1.8. Interrelació ternària amb connectivitat 1-1-1



3. n-àries. Cada interrelació n-ària es transforma en una nova relació, que té com a atributs les claus primàries de totes les entitats relacionades, més els atributs propis de la interrelació originària, si en té.

La composició de la clau primària de la nova relació depèn de la connectivitat de la interrelació n-ària.

a. Connectivitat de totes les entitats amb cardinalitat N. La clau primària està formada per tots els atributs que formen les claus primàries de totes les entitats interrelacionades (n).

Cal seguir el mateix mecanisme que amb les interrelacions ternàries amb connectivitat M-N-P.

b. Connectivitat d'una o més entitats amb cardinalitat 1. La clau primària està formada per tots els atributs que formen les claus primàries de totes les entitats interrelacionades excepte una (n-1). L'entitat que no incorpora la seva clau primària a la de la nova relació ha d'estar forçosament connectada amb un 1.

Cal seguir el mateix mecanisme que amb les interrelacions ternàries amb connectivitat 1-M-N, 1-1-N i 1-1-1.

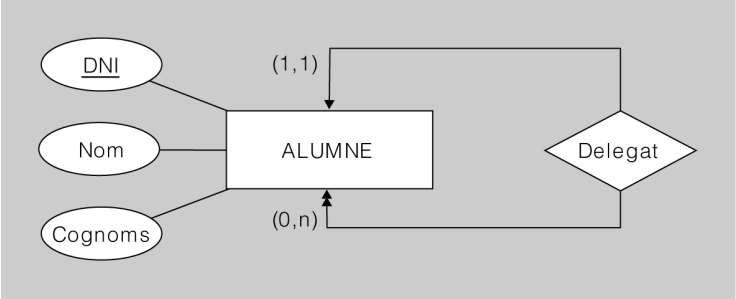
4. Recursives. Les interrelacions recursives traduïdes es comporten de la mateixa manera que la de la resta d'interrelacions:

- Les binàries amb connectivitat 1-1 i 1-N donen lloc a una clau forana.
- Les binàries amb connectivitat M-N i les n-àries originen una nova relació.

a. Binàries amb connectivitat 1-1 o 1-N. En aquestes situacions, cal afegir a la relació sorgida de l'entitat originària que es relaciona amb ella mateixa una clau forana que faci referència a la pròpia clau primària.

Evidentment, els atributs de la clau forana no poden tenir els mateixos noms que els de la clau primària als quals fan referència, ja que tots dos es troben en la mateixa relació, i això atemptaria contra els principis del model relacional.

FIGURA 1.9. Interrelació recursiva binària amb connectivitat 1-N



Exemple de transformació d'interrelació recursiva binària amb connectivitat 1-N

El diagrama ER de la figura 1.9 es tradueix al model relacional de la manera següent:

ALUMNE(DNI, Nom, Cognoms, DNIDelegat) ON {DNIDelegat} REFERENCIA ALUMNE

b. Binàries amb connectivitat M-N. Quan la interrelació recursiva binària té connectivitat M-N s'origina una nova relació, la qual té com a clau primària els atributs que formen la clau primària de l'entitat originària, però dos cops, ja que cal modelitzar el fet que l'única entitat que intervé en la conceptualització prevista s'interrelaciona amb ella mateixa (i no pas amb una altra de diferent).

Cal modificar convenientment els noms d'aquests atributs que són presents dos cops en la nova relació perquè no coincideixin, i respectar així les directrius del model relacional.

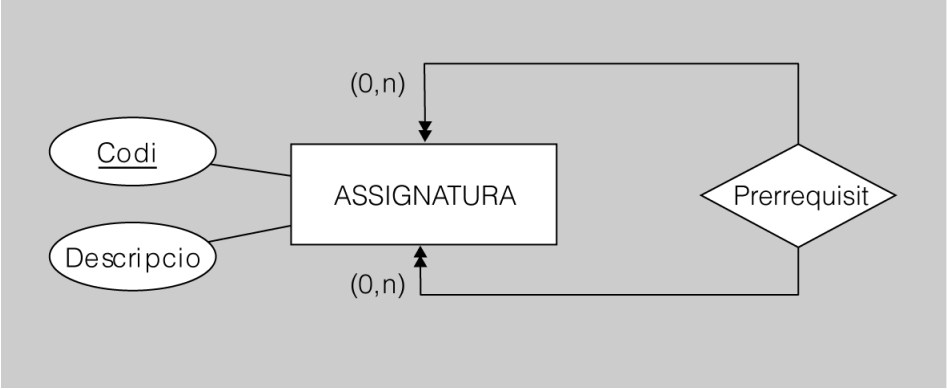
Exemple de transformació d'interrelació recursiva binària amb connectivitat M-N

El diagrama ER de la figura 1.10 es tradueix al model relacional de la manera següent:

ASSIGNATURA (Codi, Descripcio)

PRERREQUISIT(CodiAssignatura, CodiPrerrequisit) ON {CodiAssignatura} REFERENCIA ASSIGNATURA (Codi) i {CodiPrerrequisit} REFERENCIA ASSIGNATURA (Codi)

Figura 1.10. Interrelació recursiva binària amb connectivitat M-N



c. n-àries. S'origina una nova relació, la clau primària de la qual es construeix de manera diferent en funció de la connectivitat:

Quan la connexió de totes les entitats es produeix amb cardinalitat N, la clau primària de la nova relació es compon de tots els atributs que formen part de les claus primàries de totes les entitats interrelacionades (n).

Quan la connexió d'una o més de les entitats es produeix amb cardinalitat 1, la clau primària de la nova relació es compon de tots els atributs que formen les claus primàries de totes les entitats interrelacionades excepte una (n-1). L'entitat que no incorpora la seva clau primària a la de la nova relació ha d'estar forçosament connectada amb un 1.

Exemple de transformació d'interrelació recursiva n-ària

El diagrama ER de la figura 1.11 es tradueix al model relacional de la manera següent:

ALUMNE(DNI, Nom, Cognoms)

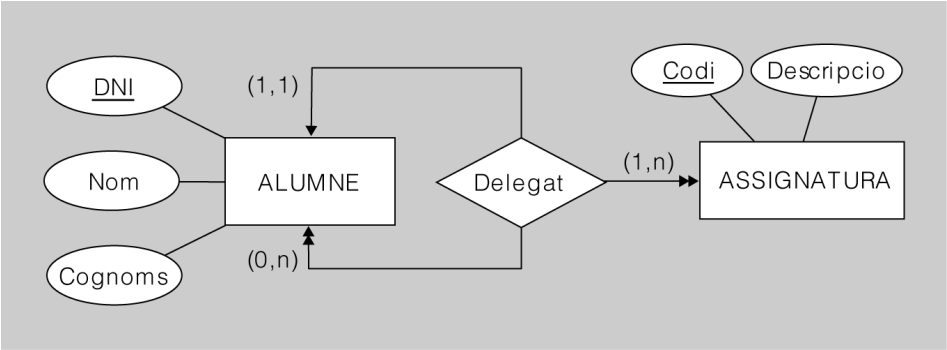
ASSIGNATURA(Codi, Descripcio)

DELEGAT(DNIAlumne, CodiAssignatura, DNIDelegat) ON {DNIAlumne} REFERENCIA ALUMNE, {CodiAssignatura} REFERENCIA ASSIGNATURA i {DNIDelegat} REFERENCIA ALUMNE

Fixem-nos que hem incorporat a la clau primària de la nova relació els atributs que formen les claus primàries de les dues entitats connectades amb cardinalitat N, és a dir, ASSIGNATURA i ALUMNE, però des de la posició dels alumnes que no són delegats.

D'aquesta manera, es modelitza el fet que cada alumne té un delegat per a cada assignatura, i que el delegat de cada assignatura representa una pluralitat d'alumnes.

FIGURA 1.11. Interrelació recursiva n-ària



1.3.3 Entitats febles

Com que les entitats febles sempre estan situades en el costat N d'una interrelació 1-N que els serveix per completar la identificació inequívoca de les seves instàncies, la relació derivada de l'entitat feble ha d'incorporar a la seva clau primària els atributs que formen la clau primària de l'entitat de la qual són tributàries. Els atributs esmentats constitueixen, simultàniament, una clau forana que fa referència a l'entitat de la qual depenen.

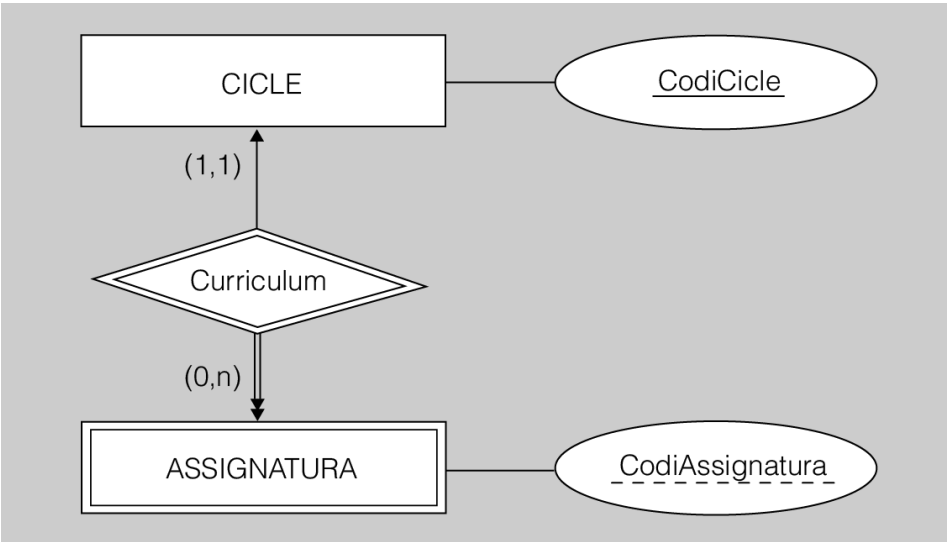
Exemple de transformació d'entitat feble

El diagrama ER de la figura 1.12 es tradueix al model relacional de la manera següent:

CICLE(CodiCicle)

ASSIGNATURA(CodiCicle, CodiAssignatura) ON {CodiCicle} REFERENCIA CICLE

FIGURA 1.12. Entitat feble



1.3.4 Generalització i especialització

En aquests casos, tant l'entitat superclasse com les entitats de tipus subclasse es transformen en noves relacions.

La relació derivada de la superclasse hereta d'aquesta la clau primària. A més, s'encarrega d'emmagatzemar els atributs comuns a tota l'especialització o generalització.

Les relacions derivades de les entitats de tipus subclasse també tenen, com a clau primària, la clau de l'entitat superclasse, que al mateix temps actua com a clau forana, en referenciar l'entitat derivada de la superclasse.

Exemple de transformació de generalització o especialització

La figura 1.13 mostra un encadenament de generalitzacions o especialitzacions. Si el traduïm a un model relacional obtenim el resultat següent:

PERSONA(DNI, Nom, Cognoms, Telefon)

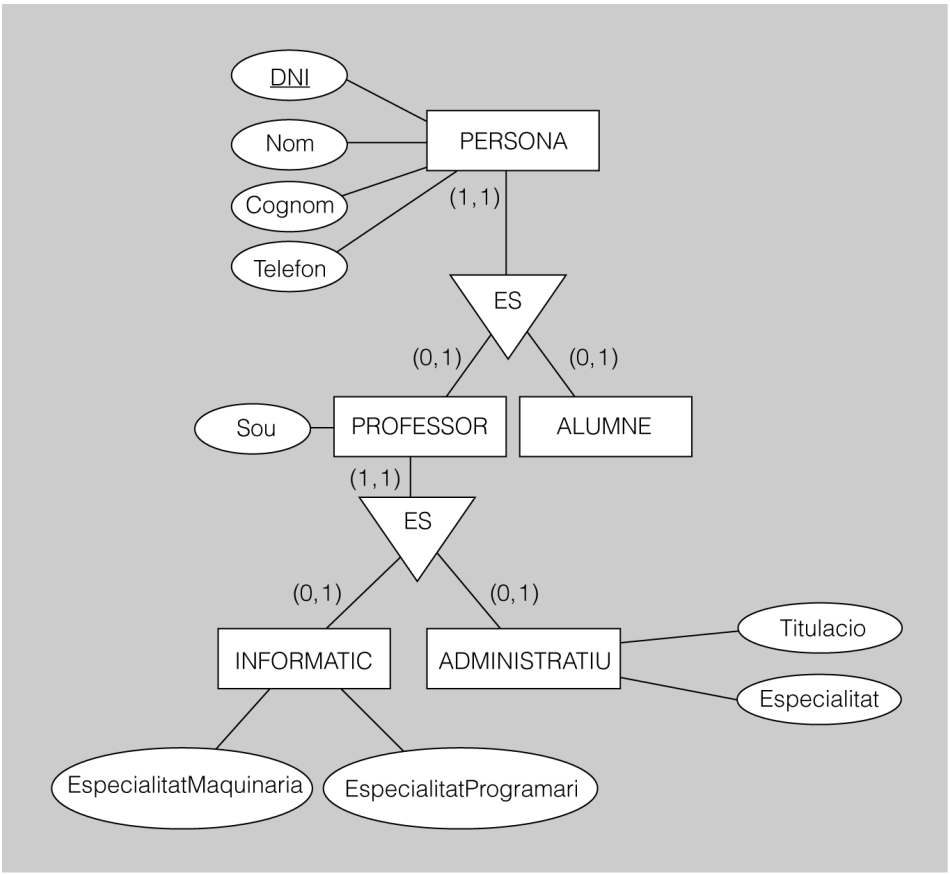
PROFESSOR(DNI, Sou) ON {DNI} REFERENCIA PERSONA

ALUMNE(DNI) ON {DNI} REFERENCIA PERSONA

INFORMATIC(DNI, EspecialitatMaquinari, EspecialitatProgramari) ON {DNI} REFERENCIA PROFESSOR

ADMINISTRATIU(DNI, Titulacio, Especialitat) ON {DNI} REFERENCIA PROFESSOR

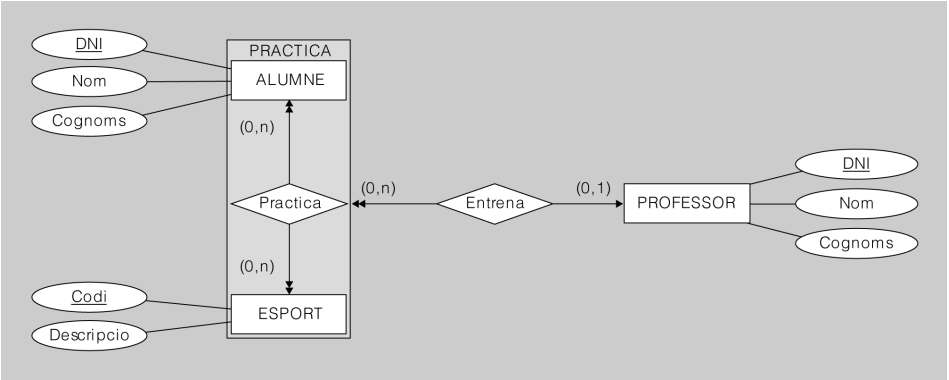
Figura 1.13. Generalització i especialització



1.3.5 Entitats associatives

Les entitats associatives es basen en una interrelació entre entitats. La traducció d'aquesta interrelació a un model relacional equival a la traducció de l'entitat associativa.

FIGURA 1.14. Entitat associativa



Exemple de transformació d'entitat associativa

El diagrama ER de la figura 1.14 es tradueix al model relacional de la manera següent:

ALUMNE(DNI, Nom, Cognoms)

ESPORT(Codi, Descripcio)

PROFESSOR(DNI, Nom, Cognoms)

PRACTICA(DNIAlumne, CodiEsport, DNIProfessor) ON {DNIAlumne} REFERENCIA ALUMNE, {CodiEsport} REFERENCIA ESPORT i {DNIProfessor} REFERENCIA PROFESSOR

Fixem-nos com la relació PRACTICA, derivada de l'entitat associativa, incorpora una clau forana que fa referència a la relació PROFESSOR, ja que l'entitat associativa originària és al costat N d'una interrelació binària amb l'entitat PROFESSOR.

2. Normalització

El disseny d'una base de dades pot ser una tasca extremadament complexa. Hi ha diferents metodologies que permeten abordar el problema de trobar l'esquema relacional que representi millor la realitat que es vol modelitzar.

Coneixem el model Entitat-Relació per establir models per a qualsevol realitat, del qual s'obté, com a resultat, el diagrama Entitat-Relació, altrament anomenat *diagrama de Chen*. També coneixem el procés de traducció d'un diagrama Entitat-Relació a un esquema relacional.

Per tant, si per arribar a l'esquema relacional que ha de modelitzar la realitat hem seguit el camí que consisteix a, primerament, efectuar el diagrama Entitat-Relació per després efectuar-ne la traducció al model relacional, i el diagrama Entitat-Relació era correcte, haurem obtingut un esquema relacional del tot correcte. Aquest seria el camí aconsellable.

En l'apartat del "Model relacional" d'aquesta unitat, es presenta el procés de traducció d'un diagrama Entitat-Relació a un esquema relacional.

Però no sempre és així i ens trobem dissenys efectuats directament en l'esquema relacional. Hi ha diferents causes que ho provoquen:

- D'entrada, el model Entitat-Relació és posterior al model relacional i, per tant, hi ha bases de dades que van ser formulades directament en la terminologia relacional. No hi havia cap altra opció!
- Hi ha dissenyadors que "no volen perdre el temps" en un model Entitat-Relació i dissenyen directament en el model relacional. Quin error més gran!
- De vegades, s'ha de modificar la base de dades a causa de noves necessitats, i el disseny s'efectua directament sobre aquesta en lloc d'analitzar-se i realitzar-se sobre el model Entitat-Relació per després transferir els canvis a l'esquema relacional. Quin error més gran!

Fixeu-vos que donem un suport absolut al fet d'utilitzar el model Entitat-Relació per obtenir-ne posteriorment el model relacional. Un bon disseny en el model Entitat-Relació acostuma a proporcionar una base de dades relacional ben dissenyada, cosa que no passarà si el disseny Entitat-Relació incorpora errors. D'altra banda, si no hi ha hagut el disseny Entitat-Relació previ, hi ha més possibilitats de tenir una base de dades relacional mal dissenyada.

La **teoria de la normalització** és un mètode que permet assegurar si un disseny relacional (tant si prové de la traducció d'un diagrama Entitat-Relació com si s'ha efectuat directament en el model relacional) és més o menys correcte.

En general, els mals dissenys poden originar les situacions següents:

- Repetició de la informació
- Impossibilitat de representar certa informació:
  - Anomalies en les insercions
  - Anomalies en les modificacions
  - Anomalies en els esborraments

Un bon disseny ha d'aconseguir el següent:

- Emmagatzemar tota la informació necessària amb el mínim d'informació redundant.
- Mantenir el mínim de lligams entre les relacions de la base de dades per tal de facilitar-ne la utilització.
- Millorar la consultabilitat de les dades emmagatzemades.
- Minimitzar els problemes d'actualització (altes, baixes i modificacions) que poden sorgir en haver d'actualitzar simultàniament dades de diferents relacions.

Exemple de disseny relacional inadequat

Considerem el disseny relacional de la taula 2.1 per enregistrar la informació dels professors amb els alumnes de cadascun i la qualificació que han obtingut en els diversos crèdits.

TAULA 2.1. Exemple de disseny relacional inadequat

DniProf	NomProfessor	DniAlum	NomAlumne	Edat	Credit	Nota
33.333.333	Joan Finestra	77.777.777	Anna Taula	20	ADBD	4.5
33.333.333	Joan Finestra	88.888.888	Miquel Cadira	19	ADBD	5.7
33.333.333	Joan Finestra	77.777.777	Anna Taula	20	SGBD	6
33.333.333	Joan Finestra	88.888.888	Miquel Cadira	19	SGBD	7
44.444.444	Maria Porta	77.777.777	Anna Taula	20	MET	6
44.444.444	Maria Porta	88.888.888	Miquel Cadira	19	MET	5
44.444.444	Maria Porta	77.777.777	Anna Taula	20	LLC	4
44.444.444	Maria Porta	88.888.888	Miquel Cadira	19	LLC	3

Oi que convindreu que aquest disseny està pensat amb els peus? Ràpidament, hi veiem els problemes següents:

- Hi ha informació repetida, fet que pot provocar inconsistències. Fixem-nos que en cas d'haver de modificar qualsevol dels valors dels camps que formen la clau primària (*DniProf,*

NomProfessor, DniAlum, NomAlumne, Edat, Credit), el canvi s'ha d'efectuar en totes les files en què apareix aquest valor.

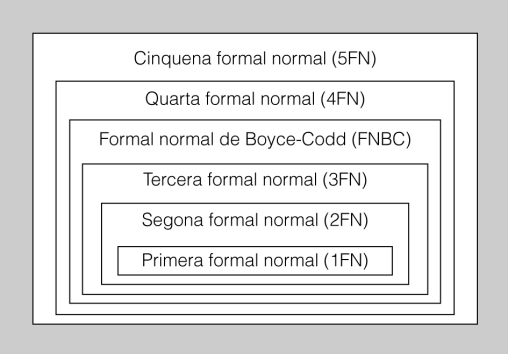
- No hi pot haver valors nuls en les columnes que formen la clau primària.
- Així, si no coneixem l'edat d'un alumne, tenim un greu problema.
- En cas d'arribar a la conclusió que necessitem emmagatzemar més informació dels professors o dels alumnes, caldrà afegir més columnes i repetir la informació per a cada fila en què aparegui el professor o alumne.
- Consultar la informació en la taula 2.1 pot esdevenir feixuc atesa la gran quantitat d'informació diferent que conté.

El mètode que proposa la teoria de la normalització per determinar si un disseny relacional és correcte consisteix a avaluar el disseny de totes les relacions (taules) per tal de veure en quin grau de normalitat es troba cadascuna i, així, poder decidir si el disseny ja és correcte o si cal refinar-lo.

La teoria de la normalització defineix les **formes normals** com a indicadors per avaluar el grau de normalitat de les relacions, i es diu que una relació està en una forma normal determinada quan satisfà un conjunt determinat de condicions.

Hi ha diferents graus de normalitat i, per tant, de formes normals, les quals compleixen la relació d'inclusió de la figura 2.1, que s'ha d'interpretar en el sentit que a mesura que augmenta el nivell de la forma normal, la relació ha de complir un conjunt de condicions més restrictiu i, per tant, continua verificant les condicions de les formes normals de nivell inferior.

Figura 2.1. Relació d'inclusió entre les diverses formes normals



Així, doncs, l'objectiu hauria de ser aconseguir un esquema relacional en què totes les relacions tinguessin el grau màxim de normalitat, és a dir, en què totes es trobessin en la cinquena forma normal (5FN).

El **procés de normalització** per aconseguir que una relació que es troba en una forma normal X passi a estar en una forma normal Y superior a X consisteix sempre en la descomposició o subdivisió de la relació original (forma normal X) en dues o més relacions que verifiquin el nivell de forma normal Y.

Per tant, el procés de normalització augmenta el nombre de relacions presents en la base de dades. Amb això, segur que s'aconsegueix una disminució de redundàncies i una disminució de les anomalies en els problemes d'actualització de la informació, però, en canvi, es penalitzen les consultes, ja que la seva execució haurà d'anar a cercar la informació en moltes taules relacionades entre elles.

Així, doncs, cal trobar un equilibri, i de vegades pot ser convenient renunciar al nivell màxim de normalització (5FN) i, per tant, permetre una certa redundància en els esquemes amb la finalitat d'alleugerir els costos de les consultes. En aquestes situacions, es parla d'un procés de desnormalització.

El nostre objectiu final és conèixer les condicions que han de complir les relacions per assolir cadascun dels nivells de forma normal, i el procés per dividir les relacions en noves relacions que verifiquin les condicions desitjades. Per aconseguir-ho, hem de conèixer els conceptes de *relació universal* i *dependència funcional*.

2.1 La relació universal

En efectuar directament el disseny relacional d'una base de dades, el dissenyador es troba amb un conjunt de conceptes que tradueix en atributs, els quals, pel seu significat, agruparà en una o més relacions.

Anomenem **relació universal** la relació consistent en l'agrupament dels atributs corresponents a **tots** els conceptes que constitueixen una base de dades relacional.

TAULA 2.2. Relació universal per a un esquema relacional ideat per a una gestió de comandes de compra

Num	DataComanda	Article	Descripcio	Qtat	Preu	DataPrevista	NomProv	PaisProv	Moneda
22.523	25-05-2000	PC3-500	PC Pentium III a 500	5	150	1-06-2000	ARKANSAS	XINA	EUR
22.523	25-05-2000	PRO-15	Protector pantalla 15"	5	8	1-06-2000	ARKANSAS	XINA	EUR
22.524	27-05-2000	PC3-500	PC Pentium III a 500	15	145	5-06-2000	MELISSA	ITÀLIA	USD
22.524	27-05-2000	PRO-15	Protector pantalla 15"	15	50	5-06-2000	MELISSA	ITÀLIA	USD
22.525	27-05-2000	INK430	Cartutx de tinta 430	20	25	31-5-2000	ARKANSAS	XINA	EUR

Així, imaginem que es vol dissenyar una base de dades per al control de les comandes de compra d'una organització determinada. Imaginem que cal incloure-hi els conceptes corresponents a número i data de la comanda; codi, descripció,

quantitat i preu pactat per cada article sol·licitat; data prevista de lliurament de la comanda; nom (*NomProv*) i país (*PaisProv*) del proveïdor; i moneda en què es pacta la comanda. La relació universal es representa en la taula 2.2.

Oi que hi ha molta redundància i poca organització? Evidentment, el disseny relacional d'una base de dades basat en la relació universal acostuma a ser del tot incorrecte, i fa necessari aplicar un procés de normalització per tal d'anar dividint la relació en altres relacions de manera que assoleixin graus de normalitat millors, és a dir, compleixin les restriccions corresponents a les formes normals més elevades.

Molt poques vegades es parteix de la relació universal. L'experiència dels dissenyadors provoca que, d'entrada, ja es pensi en relacions que assoleixen un cert grau de normalitat.

2.2 Dependències funcionals

Les definicions de les diferents formes normals, és a dir, el conjunt de condicions que les defineixen, es basen en el concepte de **dependència funcional**.

> Donats dos atributs (o conjunts d'atributs) A i B d'una relació R, direm que **B depèn funcionalment de A** si per cada valor de A existeix un, i només un, valor de B associat amb ell. També direm que **A implica B**. Ho simbolitzarem per A B.

El concepte de dependència funcional estableix lligams entre atributs o conjunt d'atributs d'una mateixa relació.

Taula 2.3. Relació universal per a un esquema relacional ideat per a una gestió de comandes de compra

Num	DataComanda	Article	Descripcio	Qtat	Preu	DataPrevista	NomProv	PaisProv	Moneda
22.523	25-05-2000	PC3-500	PC Pentium III a 500	5	150	1-06-2000	ARKANSAS	XINA	EUR
22.523	25-05-2000	PRO-15	Protector pantalla 15"	5	8	1-06-2000	ARKANSAS	XINA	EUR
22.524	27-05-2000	PC3-500	PC Pentium III a 500	15	145	5-06-2000	MELISSA	ITÀLIA	USD
22.524	27-05-2000	PRO-15	Protector pantalla 15"	15	50	5-06-2000	MELISSA	ITÀLIA	USD
22.525	27-05-2000	INK430	Cartutx de tinta 430	20	25	31-5-2000	ARKANSAS	XINA	EUR

En la relació universal de la taula 2.3 diríem que, entre d'altres, la data de comanda depèn funcionalment del número de comanda, igual que la data prevista, el nom i

el país del proveïdor i la moneda. Ho podríem escriure com segueix:

Num –> DataComanda

Num –> DataPrevista

Num –> NomProv

Num –> PaisProv

Num –> Moneda

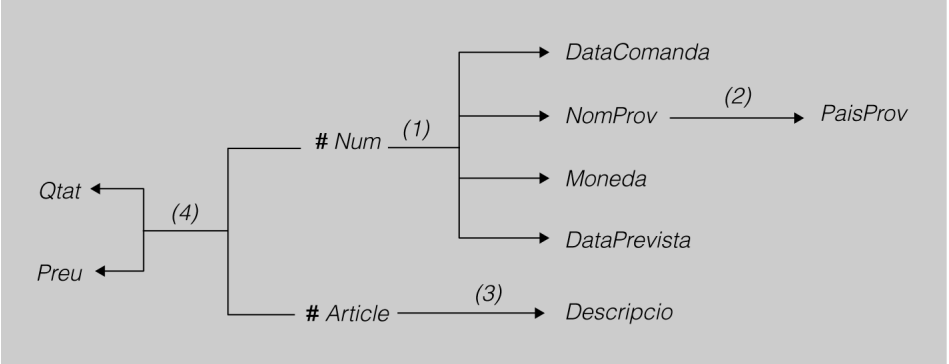
En tractar-se de diferents atributs que depenen funcionalment d'un mateix atribut, escriurem:

Num –> DataComanda, DataPrevista, NomProv, PaisProv, Moneda

> Donats dos atributs (o conjunts d'atributs) A i B d'una relació R, direm que B té una **dependència funcional completa o total** de A si B depèn funcionalment de A però no depèn funcionalment de cap subconjunt de A.

És molt convenient representar les dependències funcionals d'una relació mitjançant un **esquema de dependències funcionals**. L'esquema per a la relació universal de la taula 2.3 seria el que es mostra en la figura 2.2.

FIGURA 2.2. Exemple d'esquema de dependències funcionals



Fixem-nos que hem marcat els atributs que són clau d'alguna de les entitats que formen part de la relació: *Article* identifica l'article i *Num* identifica la comanda. Fixem-nos, també, que la parella (*Num, Article*) identifica la quantitat i preu dels articles demanats en la comanda.

En aquest esquema, es poden veure les dependències funcionals entre els atributs. Es veu que *DataComanda, NomProv, Moneda* i*DataPrevista* depenen funcionalment (1) de *Num*, que *PaisProv* depèn funcionalment (2) de *NomProv*, i que *Descripció* depèn funcionalment (3) d'*Article*. Així mateix, *Qtat*i*Preu* depenen funcionalment (4) de *Num* i *Article*.

És evident que les dependències (1), (2) i (3) són totals, ja que la part esquerra de la dependència (l'**implicador**) està formada per un únic atribut i, per tant, és impossible que la part dreta de la dependència (l'**implicat**) pugui dependre d'un

subconjunt de l'implicador. La dependència (4) també és total, ja que *Qtat*i *Preu* depenen de la parella (*Num*, *Article*) i no pas de cap subconjunt d'aquesta.

Una darrera apreciació sobre la dependència funcional (4) de la figura 2.2: veiem que en una mateixa comanda (*Num*) no és possible tenir diverses vegades el mateix article (*Article*), ja que els atributs *Qtat* i *Preu* depenen funcionalment de (*Num, Article*). En cas que fos necessari tenir diverses vegades el mateix article en una comanda, caldria utilitzar algun altre atribut per identificar l'article dins la comanda com, per exemple, el *NumeroDeLinia* de la comanda.

Donat un atribut o conjunt d'atributs A d'una relació, direm que **A és un determinant**de la relació si hi ha algun altre atribut o conjunt d'atributs B que té dependència funcional total de A.

En el cas anterior, veiem que *Num*, *Article, NomProv* i la parella (*Num, Article*) són determinants de la relació.

Donats A, B i C atributs o conjunts d'atributs d'una relació, direm que **C depèn transitivament** de A a través de B si B depèn funcionalment de A, C depèn funcionalment de B, i A no depèn funcionalment de B.

En l'exemple de la figura 2.2, podem dir que *PaisProv* depèn transitivament de *Num* a través de *NomProv*.

2.3 Primera forma normal

En aplicar el procés de normalització a una relació, es comença per comprovar si la relació està en primera forma normal i, si no és el cas, s'efectuen les modificacions oportunes per aconseguir-ho.

Una relació està en **primera forma normal** (**1FN**) si cap atribut pot contenir valors no atòmics (indivisible).

Considerem la relació universal de la taula 2.4.

La relació de la taula 2.4 no està en 1FN, ja que té atributs que poden contenir més d'un valor. Veiem que aquest exemple inclou tres files (comandes 22.523, 22.524 i 22.525), i alguns atributs (*Article, Descripcio, Qtat*i*Preu*) tenen diversos valors per a algunes de les files.

El procés que s'ha de seguir per assolir una 1FN és afegir tantes files com sigui necessari per a cadascun dels diferents valors del camp o camps que tinguin valors no atòmics.

Així, en el nostre cas, obtenim la relació en 1FN de la taula 2.5.

Taula 2.4. Relació que té atributs multivalor i, per tant, no es troba en 1FN

Num	DataComanda	Article	Descripcio	Qtat	Preu	DataPrevista	NomProv	PaisProv	Moneda
22.523	25-05-2000	PC3-500 , PRO-15	PC Pentium III a 500 , Protector Pantalla 15”	5, 5	150, 8	1-06-2000	ARKANSAS	XINA	EUR
22.524	27-05-2000	PC3-500 , PRO-15	PC Pentium III a 500 , Protector Pantalla 15”	15, 15	145, 50	5-06-2000	MELISSA	ITÀLIA	USD
22.525	27-05-2000	INK430	Cartutx de tinta 430	20	25	31-5-2000	ARKANSAS	XINA	EUR

Taula 2.5. Relació en 1FN

Num	DataComanda	Article	Descripcio	Qtat	Preu	DataPrevista	NomProv	PaisProv	Moneda
22.523	25-05-2000	PC3-500	PC Pentium III a 500	5	150	1-06-2000	ARKANSAS	XINA	EUR
22.523	25-05-2000	PRO-15	Protector pantalla 15”	5	8	1-06-2000	ARKANSAS	XINA	EUR
22.524	27-05-2000	PC3-500	PC Pentium III a 500	15	145	5-06-2000	MELISSA	ITÀLIA	USD
22.524	27-05-2000	PRO-15	Protector pantalla 15”	15	50	5-06-2000	MELISSA	ITÀLIA	USD
22.525	27-05-2000	INK430	Cartutx de tinta 430	20	25	31-5-2000	ARKANSAS	XINA	EUR

De fet, la restricció que persegueix la 1FN forma part de la definició del model relacional i, per tant, tota relació, per definició, ha d'estar en 1FN. És a dir, aquesta forma normal és redundant amb la definició del model relacional i no caldria considerar-la. Es manté, però, per assegurar que les relacions dissenyades tenen un punt de partida correcte.

En general, les relacions en 1FN poden tenir molta informació redundant. Això no ens ha de preocupar, ja que la solució rau en les formes normals de nivell superior.

Natural Join de R1*R2

L'operació de Natural Join és la combinació de totes les dades de la primera relació (R1) amb totes les de la segona (R2), sempre que els valors de les dades de les columnes anomenades de forma idèntica a les dues relacions coincideixin.

2.4 Preservació d'informació i dependències en la normalització

En el procés de normalització d'una relació R s'apliquen processos de descomposició per aconseguir relacions R1, R2, ..., Rn que verifiquin un nivell de

normalització superior al de la relació R. La descomposició consisteix a efectuar projeccions de la relació R sobre atributs que verifiquen certes condicions, la qual cosa dóna lloc a l'aparició de R1, R2, ..., Rn.

Cal garantir que la descomposició de R en R1, R2, ..., Rn preservi la informació existent, és a dir, que el *natural-join* R1 * R2 * ... * Rn proporcioni exactament la mateixa informació que tenia la relació R original, tant enintensió (quantitat d'atributs) com en extensió (quantitat de files).

De la mateixa manera caldria garantir la conservació de les dependències, és a dir, el conjunt de dependències associades a la relació R original ha de ser equivalent al conjunt de dependències associat a les relacions R1, R2, ..., Rn.

Ja podem avançar que la conservació de les dependències no es pot garantir en tots els processos de normalització.

2.5 Segona forma normal

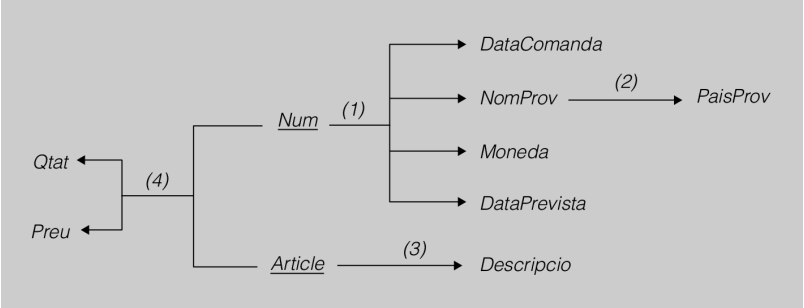
La segona forma normal persegueix l'eliminació dels problemes motivats per la presència de dependències funcionals no totals dels atributs que no formen part de la clau primària respecte a la clau primària.

> Una relació està en **segona forma normal** (**2FN**) si està en 1FN i tot atribut que no pertany a la clau té dependència funcional total de la clau.

Considerem la relació següent en 1FN ideada per a la gestió de comandes de compra en una organització:

```
R (__Num__, DataComanda, __Article__, Descripcio, Qtat, Preu,
DataPrevista, NomProv, PaisProv, Moneda)
```

Figura 2.3. Esquema de dependències funcionals



Veiem que la clau primària està formada per la parella (*Num, Article*) i que, en l'esquema de dependències funcionals associat (figura 2.3), hi ha atributs fora de la clau primària que no tenen dependència funcional completa de la clau.

En efecte, les dependències funcionals (1), (2) i (3) ens presenten atributs que no tenen dependència funcional total de la clau, formada per la parella (*Num, Article*).

El procés que s'ha de seguir per assolir una 2FN és dividir la relació (conservant la informació i les dependències) en tantes relacions com sigui necessari de manera que cada relació verifiqui que els seus atributs no-clau tenen dependència funcional total de la clau.

L'esquema de dependències funcionals ajuda a veure les relacions que han d'aparèixer. Així, en el nostre cas, de les dependències (1), (3) i (4) obtenim les relacions en 2FN:

```
COMANDA (__Num__, DataComanda, DataPrevista, Moneda, NomProv, PaisProv)
ARTICLE (__Article__, Descripcio)
DETALL (__Num__, __Article__, Qtat, Preu)
         comanda  article
```

És molt probable que aquest disseny fos el proposat com a punt de partida, és a dir: sovint, en efectuar un disseny ja obtindrem relacions que estan en 2FN i, fins i tot, en formes normals de nivell superior.

La informació que hi ha en la taula 2.6 corresponent a la relació que acabem de normalitzar ara passa a estar repartida en tres taules (taula 2.7, taula 2.8 i taula 2.9).

Taula 2.6. Relació en 1FN

Num	DataComanda	Article	Descripcio	Qtat	Preu	DataPrevista	NomProv	PaisProv	Moneda
22.523	25-05-2000	PC3-500	PC Pentium III a 500	5	150	1-06-2000	ARKANSAS	XINA	EUR
22.523	25-05-2000	PRO-15	Protector pantalla 15"	5	8	1-06-2000	ARKANSAS	XINA	EUR
22.524	27-05-2000	PC3-500	PC Pentium III a 500	15	145	5-06-2000	MELISSA	ITÀLIA	USD
22.524	27-05-2000	PRO-15	Protector pantalla 15"	15	50	5-06-2000	MELISSA	ITÀLIA	USD
22.525	27-05-2000	INK430	Cartutx de tinta 430	20	25	31-5-2000	ARKANSAS	XINA	EUR

Taula 2.7. Relació en 2FN que emmagatzema les comandes

COMANDA					
Num	**DataComanda**	**DataPrevista**	**NomProv**	**PaisProv**	**Moneda**
22.523	25-05-2000	1-06-2000	ARKANSAS	XINA	EUR
22.524	27-05-2000	5-06-2000	MELISSA	ITÀLIA	USD
22.525	27-05-2000	31-5-2000	ARKANSAS	XINA	EUR

Taula 2.8. Relació en 2FN pels articles

ARTICLE	
Article	**Descripcio**
PC3-500	PC Pentium III a 500
PRO-15	Protector pantalla 15”
INK430	Cartutx de tinta 430

Taula 2.9. Relació en 2FN pel detall de comanda

DETALL			
Num	**Article**	**Qtat**	**Preu**
22.523	PC3-500	5	150
22.523	PRO-15	5	8
22.524	PC3-500	15	145
22.524	PRO-15	15	50
22.525	INK430	20	25

Per acabar, fixem-nos que amb el nou disseny s'ha aconseguit eliminar molta redundància i, per tant, es redueixen els problemes en les operacions d'actualització i consulta. Però no desapareixen tots.

2.6 Tercera forma normal

Considerem el disseny de les relacions següents en 2FN ideades per a la gestió de les comandes de compra d'una organització:

```
COMANDA (__Num__, DataComanda, DataPrevista, Moneda, NomProv, PaisProv)
ARTICLE (__Article__, Descripcio)
DETALL (__Num__, __Article__, Qtat, Preu)
         comanda  article
```

Fixem-nos que el país del proveïdor apareix en cada comanda. Si partim de la base que el país on resideix el proveïdor és únic, oi que encara hi ha informació redundant?

La tercera forma normal persegueix l'eliminació dels problemes motivats per la presència de dependències transitives dels atributs que no formen part de la clau primària, respecte de la clau primària.

> Una relació està en **tercera forma normal** (**3FN**) si està en 2FN i cap atribut que no pertany a la clau depèn transitivament de la clau.

Les relacions *ARTICLE* i *DETALL* que ens ocupen ja estan en 3FN, però considerem la relació *comanda* que conté el país del proveïdor.

```
COMANDA (__Num__, DataComanda, DataPrevista, Moneda, NomProv, PaisProv)
```

L'atribut *PaisProv* depèn transitivament de *Num* a través de *NomProv*. Per tant, aquesta relació no està en 3FN.

> El procés que s'ha de seguir per assolir una 3FN és dividir la relació (conservant la informació i les dependències) en noves relacions més simples, de manera que cada relació verifiqui que cap dels seus atributs no-clau depèn transitivament de la clau.

En el nostre cas, obtenim les relacions en 3FN:

```
PROVEIDOR (__CodProv__, NomProv, PaisProv)
COMANDA (__Num__, DataComanda, DataPrevista, Moneda, CodProv)
                                                    proveïdors
```

Fixem-nos que, en trencar la relació inicial *COMANDA*, ha semblat oportú considerar un nou atribut (*CodProv*) que identifiqui millor la nova relació*PROVEÏDOR*. Aquest fet no és imprescindible i no sempre serà convenient. Podríem haver considerat el trencament següent:

```
PROVEIDOR (__NomProv__, PaisProv)
COMANDA (__Num__, DataComanda, DataPrevista, Moneda, NomProv)
                                                 proveïdors
```

Ara bé, per a aquesta darrera possibilitat hem escollit el nom del proveïdor com a clau primària de la nova relació *PROVEIDOR*, i l'experiència ens aconsella definir un codi que ens permeti identificar-los de manera més clara que la que proporciona el seu nom.

Tenint en compte el disseny que incorpora l'atribut *CodProv*, tindríem la conversió de la taula 2.7 en la taula 2.10 i taula 2.11.

TAULA 2.10. Relació en 3FN que emmagatzema les comandes

COMANDA				
Num	**DataComanda**	**DataPrevista**	**CodProv**	**Moneda**
22.523	25-05-2000	1-06-2000	ARK	EUR
22.524	27-05-2000	5-06-2000	MEL	USD
22.525	27-05-2000	31-5-2000	ARK	EUR

TAULA 2.11. Relació en 3FN pels proveïdors

PROVEIDOR		
CodProv	**NomProv**	**PaisProv**
ARK	ARKANSAS	XINA
MEL	MELISSA	ITÀLIA

2.7 Forma normal de Boyce-Codd

Considerem les naus d'emmagatzematge que hi ha en un gran mercat dedicades a guardar les mercaderies dels venedors del mercat. Imaginem que cada nau guarda mercaderia d'un tipus concret (carn fresca, peix fresc, congelats, vegetals, basar...) i que cada venedor pot dipositar mercaderia en diferents naus segons el tipus de mercaderia de cada nau (una parada de peix del mercat es pot dedicar a vendre peix fresc i peix congelat, per exemple). Ara bé, tota la mercaderia d'unes mateixes característiques d'un venedor es troba concentrada en una ubicació dins una mateixa nau per minimitzar al màxim els desplaçaments del venedor.

Per tenir constància de quin tipus de material hi ha a cada nau, es dissenya aquesta relació:

```
DIPOSIT (__Venedor__, __TipusMaterial__, Nau, Ubicacio)
```

La taula 2.12 ens exemplifica la situació. Fixem-nos que es troba en 3FN:

- Tots els atributs no pertanyents a la clau (*Nau* i *Ubicacio*) tenen dependència funcional total de la clau (2FN).

Taula 2.12. Relació en 3FN

DIPOSIT			
Venedor	**TipusMaterial**	**Nau**	**Ubicacio**
JOSEP	Peix fresc	15	S4-C3-U1:10
MARIA	Peix fresc	25	S3-C5-U5:22
RAMON	Congelats	17	S2-C4-U1:25
ANNA	Vegetals	20	S1-C6-U7:10
ANNA	Peix fresc	25	S2-C5-U12:15
MARIA	Basar	10	S3-C4-U20:25

En efecte, la *Nau* i la *Ubicacio* dins la nau depenen del *Venedor* i del*TipusMaterial*, ja que hi pot haver diverses naus dedicades a un tipus de material, però tot el material similar d'un venedor es troba en una nau determinada. Al mateix temps, hi pot haver diverses naus amb material d'un venedor a causa de la diferent tipologia del material.

- Cap atribut no pertanyent a la clau (*Nau* i *Ubicacio*) no depèn transitivament de la clau. En efecte, és impossible que hi hagi cap dependència transitiva de la clau, ja que no hi ha cap atribut que pugui servir de pont per a la transitivitat. Però aquesta relació, tot i estar en 3FN, presenta anomalies:
  - Si en un moment donat una nau no té material de cap venedor, es perd la informació referent al tipus de mercaderia que correspon a la nau.
  - Si canvia la descripció del tipus de mercaderia assignada a una nau,

cal modificar tantes files com venedors amb dipòsits d'aquell tipus de mercaderia hi hagi a la nau.

> Una relació està en la **forma normal de Boyce-Codd** (**FNBC**) si està en 2FN i tots els seus determinants són claus candidates.

L'anterior relació *DIPOSIT* no es troba en FNBC, ja que l'atribut *Nau* és un determinant de la relació i *TipusMaterial* té dependència funcional total de *Nau*, i en canvi *Nau* no és clau candidata.

És a dir, es dóna aquesta situació:

Nau –> TipusMaterial

Es verifica que tota relació en FNBC està en 3FN però no a l'inrevés, com hem pogut comprovar en el nostre cas.

> El procés que s'ha de seguir per assolir una FNBC és apartar de la relació els atributs que depenen dels determinants que no són claus candidates, i formar noves relacions que recullen els atributs apartats i que preserven la informació inicial.

En el nostre cas, apartarem de la relació l'atribut *TipusMaterial* i obtindrem les relacions en FNBC:

```
DIPOSIT (__venedor__, nau, ubicacio)
                    nauTipus
NAU_TIPUS (__nau__, tipusMaterial)
```

En els trencaments efectuats sobre una relació no normalitzada per assolir relacions 2FN i 3FN, cal efectuar la divisió de manera que es preservin la informació i les dependències funcionals, fet sempre possible en el pas a 2FN i 3FN. En el pas a FNBC, també sempre és possible efectuar la divisió mantenint la informació, però no sempre és possible el manteniment de les dependències funcionals.

En el nostre cas, la relació inicial *DIPOSIT* contenia les dependències funcionals següents:

Venedor, TipusMaterial –> Nau, Ubicacio

Nau –> TipusMaterial

I, en les relacions finals *DIPOSITS* i *NAU_TIPUS*, s'ha perdut la dependència funcional que indicava que la *Nau* depenia de la parella *Venedor* i*TipusMaterial*:

DIPOSIT: Venedor –> Nau, Ubicacio

NAU_TIPUS: Nau –> TipusMaterial

La dependència funcional s'ha perdut perquè el concepte de dependència funcional en el model relacional es defineix únicament entre atributs d'una mateixa

relació, i en aquest s'hauria de poder definir entre atributs de relacions diferents. De tota manera, el concepte d'integritat referencial intenta superar aquesta limitació.

A causa de la pèrdua de dependències funcionals, que no sempre es produeix, sovint no es normalitza a FNBC i es treballa amb relacions en 3FN.

2.8 Quarta forma normal

Considerem la relació *ESTUDIANT* (taula 2.13) dissenyada per emmagatzemar els diversos crèdits que cursa i les diverses activitats esportives que practica.

```
1 ESTUDIANT(__Dni__, __Credit__, __Esport__)
```

És a dir, la relació estudiant recull la possibilitat que un estudiant cursi diversos crèdits i practiqui diverses activitats esportives.

Taula 2.13. Relació FNBC amb redundància

ESTUDIANT		
Dni	**Credit**	**Esport**
10.000.000	SGBD	Bàsquet
10.000.000	ADBD	Bàsquet
10.000.000	SGBD	Futbol
10.000.000	ADBD	Futbol
20.000.000	PEM	Natació
20.000.000	ADBD	Natació
20.000.000	SGBD	Natació
20.000.000	PEM	Esgrima
20.000.000	ADBD	Esgrima
20.000.000	SGBD	Esgrima
15.000.000	PEM	Natació
15.000.000	PEM	Bàsquet

Aquesta relació es troba en FNBC i, tot i així, hi ha redundància, provocada per un nou concepte: les dependències multivalents.

> Donats A i B atributs o conjunts d'atributs d'una relació, direm que B té una **dependència multivalent** de A si un valor de A pot determinar un conjunt de valors de B. Ho simbolitzarem amb la notació A –>> B.

Les dependències multivalents no són dependències funcionals. En canvi, però, una dependència funcional es pot arribar a considerar una dependència multivalent en què per cada valor de l'implicant hi ha un únic valor de l'implicat.

Els problemes provocats per les dependències funcionals han causat la definició de la 1FN, 2FN, 3FN i FNBC. La redundància provocada per les dependències multivalents ens porten a definir la 4FN.

Una relació es troba en **quarta forma normal** (**4FN**) si està en 3FN i l'implicant de tota dependència multivalent és una clau candidata.

Quan té lloc una dependència multivalent A –>> B, també existeix la dependència multivalent A –>> X - (A U B), on X indica el conjunt de tots els atributs de la relació. És a dir, les dependències multivalents es presenten per parelles.

En el nostre cas (taula 2.13) es verifica el següent:

Dni –> Credit

Dni –> Esport

Per assolir la 4FN a partir d'una relació R (A, B, C) que té una dependència multivalent A –>> B, cal descompondre la relació R en dues relacions R1 (A, B) i R2 (A, C).

Taula 2.14. Relació en 4FN

CREDIT_EN_CURS	
Dni	**Credit**
10.000.000	SGBD
10.000.000	ADBD
20.000.000	PEM
20.000.000	ADBD
20.000.000	SGBD
15.000.000	PEM

Taula 2.15. Relació en 4FN

ESPORT_EN_PRACTICA	
Dni	**Esport**
10.000.000	Bàsquet
10.000.000	Futbol
20.000.000	Natació
20.000.000	Esgrima
15.000.000	Natació
15.000.000	Bàsquet

En aquest cas, obtenim les relacions següents (taula 2.14 i taula 2.15):

```
1 CREDIT_EN_CURS (__Dni, Credit__)
2 ESPORT_EN_PRACTICA (__Dni, Esport__)
```

Sovint, la descomposició causada per les dependències multivalents s'efectua abans de les descomposicions per assolir els nivells 2FN, 3FN i FNBC. En aquesta situació, en les relacions obtingudes, cal aplicar les comprovacions per aconseguir que estiguin en 2FN, 3FN i FNBC.

2.9 Cinquena forma normal

Considerem la relació *PROFESSOR* (taula 2.16) dissenyada per gestionar els professors d'una determinada institució escolar que té diferents centres de docència. Cada professor està autoritzat a impartir unes determinades especialitats docents que pot posar en pràctica en qualsevol dels centres docents de la institució escolar. Així mateix, cada professor pot exercir, a més de la docència, diferents tasques (càrrecs, tutoria pedagògica, tutoria tècnica...) en diversos centres de la institució escolar.

```
PROFESSOR (__CodiProf, Centre, Especialitat, Tasca__)
```

Taula 2.16. Relació en FNBC amb dependències multivalents

PROFESSOR			
CodiProf	**Centre**	**Especialitat**	**Tasca**
P1	C1	Matemàtiques	Tutor
P1	C2	Matemàtiques	Tutor
P1	C2	Informàtica	Aula Informàtica
P2	C1	Català	Coordinador
P2	C2	Castellà	Tutor

Aquesta relació és FNBC i s'hi aprecia una espècie de dependència multivalent, la qual no es pot solucionar per la via de la descomposició. En efecte, és molt fàcil pensar en una descomposició en les tres relacions següents (exemplificades en la taula 2.17, taula 2.18 i taula 2.19):

```
CENTRE_DE_PROFESSOR (__CodiProf, Centre__)
ESPECIALITAT_DE_PROFESSOR (__CodiProf, Especialitat__)
TASCA_DE_PROFESSOR (__CodiProf, Tasca__)
```

Taula 2.17. Relació en 4FN

CENTRE_DE_PROFESSOR	
CodiProf	**Centre**
P1	C1
P1	C2
P2	C1
P2	C2

TAULA 2.18. Relació en 4FN

ESPECIALITAT_DE_PROFESSOR	
CodiProf	**Especialitat**
P1	Matemàtiques
P1	Informàtica
P2	Català
P2	Castellà

TAULA 2.19. Relació en 4FN

TASCA_DE_PROFESSOR	
CodiProf	**Tasca**
P1	Tutor
P1	Aula informàtica
P2	Coordinador
P2	Tutor

Aquesta descomposició és errònia, ja que si apliquem el *natural-join* de les tres relacions (taula 2.17, taula 2.18 i taula 2.19) no obtenim la relació inicial (taula 2.16) sinó que obtenim una relació (taula 2.20) amb moltes més instàncies.

TAULA 2.20. Relació obtinguda del "natural-join" de les relacions de la :table:Taula36:, :table:Taula37: i :table:Taula38:

PROFESSOR			
CodiProf	**Centre**	**Especialitat**	**Tasca**
P1	C1	Matemàtiques	Tutor
P1	C1	Matemàtiques	Aula informàtica
P1	C1	Informàtica	Tutor
P1	C1	Informàtica	Aula informàtica
P1	C2	Matemàtiques	Tutor
P1	C2	Matemàtiques	Aula informàtica
P1	C2	Informàtica	Tutor
P1	C2	Informàtica	Aula informàtica
P2	C1	Català	Coordinador
P2	C1	Català	Tutor
P2	C1	Castellà	Coordinador
P2	C1	Castellà	Tutor
P2	C2	Català	Coordinador
P2	C2	Català	Tutor
P2	C2	Castellà	Coordinador
P2	C2	Castellà	Tutor

Queda clar, doncs, que el mètode utilitzat en aquest cas no és correcte i això és degut al fet que, en aquesta situació, hi ha el que s'anomena **dependències mútues** entre els atributs de la relació. Les dependències mútues provoquen que la descomposició de la relació en altres relacions (projeccions de l'original) no verifiqui que el seu *natural-join* coincideix amb la relació original.

Direm que una relació R descomposta en relacions R1, R2, ..., Rn satisfà una **dependència de reunió**, també anomenada **dependència de projecció-join**, respecte a R1, R2, ..., Rn únicament si R és igual al *natural-join* de R1, R2, ..., Rn. La notarem com a DR*(R1, R2, ..., Rn).

Tornem a l'exemple de descomposició anterior: la relació *PROFESSOR* s'ha descompost en tres relacions, *CENTRE_DE_PROFESSOR*,*ESPECIALITAT_DE_PROFESSOR* i *TASCA_DE_PROFESSOR*, i hem pogut comprovar que la relació *professor* no satisfà una dependència de reunió respecte a *CENTRE_DE_PROFESSOR*, *ESPECIALITAT_DE_PROFESSOR* i*TASCA_DE_PROFESSOR*.

Ens cal trobar una dependència de reunió per a la relació *PROFESSOR*, és a dir, trobar una descomposició tal que el seu *natural-join* recuperi la relació original. Fixem-nos en la descomposició següent (taula 2.21, taula 2.22 i taula 2.23):

```
PROFESSOR (__CodiProf, Centre, Especialitat, Tasca__)
PCE = PROFESSOR [__CodiProf, Centre, Especialitat__]
PCT = PROFESSOR [__CodiProf, Centre, Tasca__]
PET = PROFESSOR [__CodiProf, Especialitat, Tasca__]
```

On PCE, PCT i PET són seccions de la taula original.

Taula 2.21. Relació 4FN

PCE		
CodiProf	**Centre**	**Especialitat**
P1	C1	Matemàtiques
P1	C2	Matemàtiques
P1	C2	Informàtica
P2	C1	Català
P2	C2	Castellà

Taula 2.22. Relació 4FN :table:Taula42:. Relació 4FN

PCT		
CodiProf	**Centre**	**Tasca**
P1	C1	Tutor
P1	C2	Tutor
P1	C2	Aula informàtica
P2	C1	Coordinador
P2	C2	Tutor

TAULA 2.23. Relació 4FN

PET		
Professor	**Especialitat**	**Tasca**
P1	Matemàtiques	Tutor
P1	Informàtica	Aula informàtica
P2	Català	Coordinador
P2	Castellà	Tutor

En aquesta situació, veiem que si efectuem el *natural-join* de les tres relacions *PCE*, *PCT* i *PET* obtenim la relació original.

> Direm que una relació està en **cinquena forma normal** (**5FN**), també anomenada **forma normal projecció-join** (**FNPJ**), si està en 4FN i tota dependència de reunió és conseqüència de claus candidates.

Per tant, la relació *PROFESSOR* del nostre exemple no es troba en 5FN, ja que hem trobat la dependència de reunió *PROFESSOR*(PCE, PCT, PET)* en què les relacions *PCE*, *PCT* i *PET* no estan constituïdes per claus candidates de *PROFESSOR*.

> Una relació 4FN que no sigui 5FN a causa d'una dependència de reunió es pot descompondre, sense pèrdua d'informació, en les relacions sobre les quals es defineix la dependència de reunió, les quals estan en 5FN.

Així, en el nostre exemple, la relació *PROFESSOR* desapareixeria per donar pas a les tres relacions *PCE*, *PCT* i *PET* en què es basa la dependència de reunió trobada.

```
1 PCE (__CodiProf, Centre, Especialitat__)
2 PCT (__CodiProf, Centre, Tasca__)
3 PET (__CodiProf, Especialitat, Tasca__)
```

2.10 Desnormalització

Pot semblar estrany plantejar-se la desnormalització d'una base de dades, després d'argumentar substancialment la importància que té disposar de bases de dades normalitzades, però en alguns casos una desnormalització controlada pot ser molt útil i, fins i tot, desitjable.

La desnormalització és pot definir com la introducció de redundàncies de forma controlada en una bases de dades, per tal de fer més eficients alguns processos que, altrament, farien que globalment el rendiment del sistema resultés poc òptim.

Tot i que per a un sistema donat, es podrien preveure certes modificacions sobre la BD per tal de millorar el rendiment una vegada en funcionament, el més típic és detectar aquestes necessitats a posteriori. Així, doncs, aquests tipus de modificacions sobre la base de dades es duen a terme, habitualment, després d'haver observat certes ineficiències en algunes operacions habituals sobre una base de dades.

A continuació es descriuen algunes situacions, a mode d'exemple, on pot ser útil la desnormalització:

Trigger

Un `trigger` és un procediment de BD que s'executa automàticament quan s'esdevenen situacions donades. Per exemple, inserció d'un nou registre en una taula, modificació d'una dada en un camp donat, d'una taula, etc.

- En BD on hi ha consultes intensives sobre dades d'una taula que tenen referenciades altres taules on s'hi emmagatzemen descripcions, en algun cas pot ser útil mantenir la descripció en la mateixa taula original, per tal de fer més ràpida la consulta mencionada. Pensem, per exemple, en una taula d'adreces d'empleats que té el codi de la província i que fa referència a una altra taula que té la descripció de la província. Cada cop que calgui consultar la província dels empleats caldrà processar l'operació de `join` de les dues taules. En canvi, si es disposa del nom de la província en la taula d'adreces, s'evitarà aquesta operació que resulta costosa.
- Afegir camps que són calculables, com ara el total d'una factura, pot reduir, també, el temps de consulta de dades d'aquesta taula, per exemple.

La introducció de redundàncies que desnormalitzin la BD habitualment va lligat a la implementació de certs mecanismes que intenten solucionar els possibles problemes que es podrien generar d'aquestes accions. La forma més típica de controlar els canvis per a evitar inconsistències en en aquests casos és la programació de `triggers`. Així doncs, es pot programar un `trigger`, per exemple, perquè en inserir, modificar o eliminar una línia de factura, es recalculi i actualitzi l'import del total de la factura, per així tenir aquest camp calculat sempre consistent.

Llenguatge SQL. Consultes

Cristina Obiols Llopart
Adaptació de continguts: Isidre Guixà Miranda

Bases de dades

Índex

Introducció

Les aplicacions informàtiques utilitzades en l'actualitat per a la gestió de qualsevol organització mouen una quantitat considerable de dades que s'emmagatzemen en bases de dades gestionades per sistemes gestors de bases de dades (SGBD).

Hi ha bases de dades ofimàtiques, com ara Ms-Access o Base de l'OpenOffice, que donen prou prestacions per emmagatzemar informació que podríem anomenar *domèstica*. En podem trobar molts exemples: base de dades per organitzar els nostres volums musicals (discos, CD...), base de dades per portar la comptabilitat domèstica, base de dades per gestionar les fotos que tenim a casa, base de dades per gestionar els llibres de la nostra biblioteca... I, també, per què no, gestionar les dades de petites organitzacions empresarials (botigues, tallers...). Però les bases de dades ofimàtiques acostumen a no tenir prou recursos quan es tracta de gestionar grans volums d'informació a la qual han de poder accedir molts usuaris simultàniament des de la xarxa local i també des de llocs de treball remots i, per aquest motiu, apareixen les bases de dades corporatives.

Tots els SGBD (ofimàtics i corporatius) incorporen el llenguatge SQL (*structured query language*) per poder donar instruccions al sistema gestor i així poder efectuar altes, baixes, consultes i modificacions, i crear les estructures de dades (taules i índexs), i als usuaris perquè puguin accedir a les bases de dades, concedir-los i revocar-los permisos d'accés... El treball en SGBD ofimàtics s'acostuma a efectuar sense utilitzar aquest llenguatge, ja que la interfície gràfica que aporta l'entorn acostuma a permetre qualsevol tipus d'operació. Els SGBD corporatius també aporten (cada vegada més) interfícies gràfiques potents que permeten efectuar moltes operacions però, tot i així, es fa necessari el coneixement del llenguatge SQL per efectuar múltiples tasques.

En aquesta unitat, "Llenguatge SQL. Consultes", farem els primers passos en el coneixement del llenguatge SQL, i ens introduirem en els tipus de dades que pot gestionar i en el disseny de consultes senzilles a la base de dades, per passar a ampliar el nostre coneixement sobre el llenguatge SQL per tal d'aprofitar tota la potència que dóna en l'àmbit de la consulta d'informació. Així, per exemple, aprendrem a ordenar la informació, a agrupar-la efectuant-hi filtratges per reduir el nombre de resultats, a combinar resultats de diferents consultes... És a dir, que aquest llenguatge és una meravella que possibilita efectuar qualsevol tipus de consulta sobre una base de dades per obtenir la informació volguda.

Concretament, en l'apartat "Consultes de selecció simples" es fa una breu introducció als diferents tipus de sentències SQL per a passar a veure els tipus de dades que suporta MySQL (SGBD amb el que es treballa en aquests materials) i per acabar veient l'estructura bàsica de la sentència de selecció SELECT.

En l'apartat "Consultes de selecció complexes" es donaran a conèixer les funcions més importants que incorpora MySQL, així com algunes clàusules opcionals de la

sentència de selecció SELECT que permeten ordenar o agrupar files o combinar diverses taules.

També trobareu un "Annex 1. MySQL" que us guiarà durant la instal·lació de MySQL i dóna unes indicacions per a donar els primers passos en aquest entorn on es treballarà al llarg de tota la unitat.

Per adquirir un bon coneixement del llenguatge SQL, és necessari que aneu reproduint en el vostre ordinador tots els exemples incorporats en el text, i també les activitats i els exercicis d'autoavaluació. I per a això, utilitzarem l'SGBD MySQL i les eines adequades seguint les instruccions del material web d'aquest mòdul.

Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Consulta la informació emmagatzemada en una base de dades emprant assistents, eines gràfiques i el llenguatge de manipulació de dades.
    - Identifica les funcions, la sintaxi i les ordres bàsiques del llenguatge SQL per consultar i modificar les dades de la base de dades de manera interactiva.
    - Empra assistents, eines gràfiques i el llenguatge de manipulació de dades sobre un SGBDR corporatiu de manera interactiva i tenint en compte les regles sintàctiques.
    - Fa consultes simples de selecció sobre una taula (amb restricció i ordenació) per consultar les dades d'una base de dades.
    - Fa consultes utilitzant funcions afegides i valors nuls.
    - Fa consultes amb diverses taules mitjançant composicions internes.
    - Fa consultes amb diverses taules mitjançant composicions externes.
    - Fa consultes amb subconsultes.

1. Consultes de selecció simples

La millor manera d'iniciar l'estudi del llenguatge SQL és executar consultes senzilles en la base de dades. Abans, però, ens convé conèixer els orígens i evolució que ha tingut aquest llenguatge, els diferents tipus de sentències SQL existents (subdivisió del llenguatge SQL), com també els diferents tipus de dades (nombres, dates, cadenes...) que ens podem trobar emmagatzemades en les bases de dades. I, llavors, ja ens podrem iniciar en l'execució de consultes simples.

1.1 Orígens i evolució del llenguatge SQL sota el guiatge dels SGBD

El model relacional en què es basen els SGBD actuals va ser presentat el 1970 pel matemàtic Edgar Frank Codd, que treballava en els laboratoris d'investigació de l'empresa d'informàtica IBM. Un dels primers SGBD relacionals a aparèixer va ser el System R d'IBM, que es va desenvolupar com a prototip per provar la funcionalitat del model relacional i que anava acompanyat del llenguatge SEQUEL (acrònim de *Structured English Query Language*) per manipular i accedir a les dades emmagatzemades en el System R. Posteriorment, el mot SEQUEL es va condensar en SQL (acrònim de *Structured Query Language*).

Per què SQL en lloc de SEQUEL?

S'utilitza l'acrònim *SQL* en lloc de *SEQUEL* perquè el mot *SEQUEL* ja estava registrat per la companyia anglesa d'avions Hawker-Siddeley.

Una vegada comprovada l'eficiència del model relacional i del llenguatge SQL, es va iniciar una dura cursa entre diferents marques comercials. Així, tenim el següent:

- IBM comercialitza diversos productes relacionals amb el llenguatge SQL: System/38 el 1979, SQL/DS el 1981, i DB2 el 1983.
- Relational Software, Inc. (actualment, *Oracle* Corporation) crea la seva pròpia versió de SGBD relacional per a la Marina dels EUA, la CIA i d'altres, i l'estiu del 1979 allibera *Oracle* V2 (versió 2) per a les computadores VAX (les grans competidores de l'època amb les computadores d'IBM).

El llenguatge SQL va evolucionar (cada marca comercial seguia el seu propi criteri) fins que els principals organismes d'estandardització hi van intervenir per obligar els diferents SGBD relacionals a implementar una versió comuna del llenguatge i, així, el 1986 l'ANSI (American National Standards Institute) publica l'estàndard SQL-86, que el 1987 és ratificat per l'ISO (Organització Internacional per a la Normalització, o International Organization for Standardization en anglès).

Pronunciació d'SQL

L'ANSI ha decidit que la pronunciació anglesa de l'acrònim *SQL* és /s kju: l/, i la corresponent catalana seria /ésku él/. Avui en dia es troben molts professionals que, erròniament, pronuncien *sequel*.

La taula 1.1 presenta les diferents revisions de l'estàndard SQL que han aparegut des de 1986. No ens cal saber què ha aportat cada revisió, sinó que aquestes revisions han existit.

TAULA 1.1. Revisions de l'estàndard SQL

Any	Revisió	Àlies	Comentaris
1986	SQL-86	SQL-87 / SQL1	Publicat per l'ANSI el 1986, i ratificat per l'ISO el 1987.
1989	SQL-89		Petita revisió.
1992	SQL-92	SQL2	Gran revisió.
1999	SQL:1999	SQL3	Introdueix consultes recursives, disparadors...
2003	SQL:2003		Introdueix temes d'XML, funcions Windows...
2006	SQL:2006		

1.2 Tipus de sentències SQL

Termes en anglès

En l'argot informàtic s'utilitzen els termes següents:

- *Data definition language*, abreujat DDL
- *Data control language*, abreujat DCL
- *Query language*, abreujat QL
- *Data manipulation language*, abreujat DML

Els SGBD relacionals incorporen el llenguatge SQL per executar diferents tipus de tasques en les bases de dades: definició de dades, consulta de dades, actualització de dades, definició d'usuaris, concessió de privilegis... Per aquest motiu, les sentències que aporta el llenguatge SQL s'acostumen a agrupar en les següents:

1. Sentències destinades a la definició de les dades (LDD), que permeten definir els objectes (taules, camps, valors possibles, regles d'integritat referencial, restriccions...).
2. Sentències destinades al control sobre les dades (LCD), que permeten concedir i retirar permisos sobre els diferents objectes de la base de dades.
3. Sentències destinades a la consulta de les dades (LC), que permeten accedir a les dades en mode consulta.
4. Sentències destinades a la manipulació de les dades (LMD), que permeten actualitzar la base de dades (altes, baixes i modificacions).

En alguns SGBD no hi ha distinció entre LC i LMD, i únicament es parla d'LMD per a les consultes i actualitzacions. De la mateixa manera, a vegades s'inclouen les sentències de control (LCD) juntament amb les de definició de dades (LDD). No té cap importància que s'incloguin en un grup o que siguin un grup propi: és una simple classificació.

Tots aquests llenguatges acostumen a tenir una sintaxi senzilla, similar a les ordres de consola per a un sistema operatiu, anomenada **sintaxi auto-suficient**.

SQL allotjat

Les sentències SQL poden presentar, però, una segona sintaxi, sintaxi allotjada, consistent en un conjunt de sentències que són admeses dins d'un llenguatge de programació anomenat *llenguatge amfitrió*.

Així, podem trobar LC i LMD que es poden allotjar en llenguatges de tercera generació com C, Cobol, Fortran..., i en llenguatges de quarta generació.

Els SGBD acostumen a incloure un llenguatge de tercera generació que permet allotjar sentències SQL en petites unitats de programació (funcions o procediments). Així, l'SGBD Oracle incorpora el llenguatge PL/SQL, l'SGBD SQLServer incorpora el llenguatge Transact-SQL, l'SGBD MySQL 5.x segueix la sintaxi SQL 2003 per a la definició de rutines de la mateixa manera que l'SGBD DB2 d'IBM.

1.3 Tipus de dades

L'evolució anàrquica que ha seguit el llenguatge SQL ha fet que cada SGBD hagi pres les seves decisions quant als tipus de dades permeses. Certament, els diferents estàndards SQL que han anat apareixent han marcat una certa línia i els SGBD s'hi apropen, però tampoc no poden deixar de donar suport als tipus de dades que han proporcionat al llarg de la seva existència, ja que hi ha moltes bases de dades repartides pel món que les utilitzen.

De tot això hem de deduir que, per tal de treballar amb un SGBD, hem de conèixer els principals tipus de dades que facilita (numèriques, alfanumèriques, moments temporals...) i ho hem de fer centrant-nos en un SGBD concret (la nostra elecció ha estat MySQL) tenint en compte que la resta de SGBD també incorpora tipus de dades similars i, en cas d'haver-hi de treballar, sempre haurem de fer una ullada a la documentació que cada SGBD facilita.

Cada valor manipulat per un SGBD determinat correspon a un tipus de dada que associa un conjunt de propietats al valor. Les propietats associades a cada tipus de dada fan que un SGBD concret tracti de manera diferent els valors de diferents tipus de dades.

En el moment de creació d'una taula, cal especificar un tipus de dada per a cadascuna de les columnes. En la creació d'una acció o funció emmagatzemada en la base de dades, cal especificar un tipus de dada per a cada argument. L'assignació correcta del tipus de dada és fonamental perquè els tipus de dades defineixen el domini de valors que cada columna o argument pot contenir. Així, per exemple, les columnes de tipus DATE no podran acceptar el valor '*30 de febrer*' ni el valor 2 ni la cadena *Hola*.

Dins els tipus de dades bàsics, en podem distingir els següents:

- Tipus de dades per gestionar informació alfanumèrica.
- Tipus de dades per gestionar informació numèrica.
- Tipus de dades per gestionar moments temporals (dates i temps).
- Altres tipus de dades.

MySQL és el SGBD amb què es treballa en aquests materials i el llenguatge SQL de MySQL el que es descriu. La notació que s'utiliza, en quant a sintaxi

de definició del llenguatge, habitual, consisteix a posar entre claudàtors([ ]) els elements opcionals, i separar amb el caràcter | els elements alternatius.

1.3.1 Tipus de dades string

Els tipus de dades *string* emmagatzemen dades alfanumèriques en el conjunt de caràcters de la base de dades. Aquests tipus són menys restrictius que altres tipus de dades i, en conseqüència, tenen menys propietats. Així, per exemple, les columnes de tipus caràcter poden emmagatzemar valors alfanumèrics -lletres i xifres-, però les columnes de tipus numèric només poden emmagatzemar valors numèrics.

MySQL proporciona els tipus de dades següents per gestionar dades alfanumèriques:

- `CHAR`
- `VARCHAR`
- `BINARY`
- `VARBINARY`
- `BLOB`
- `TEXT`
- `ENUM`
- `SET`

El tipus CHAR [(llargada)]

Aquest tipus especifica una cadena de longitud fixa (indicada per llargada) i, per tant, MySQL assegura que tots els valors emmagatzemats en la columna tenen la longitud especificada. Si s'hi insereix una cadena de longitud més curta, MySQL l'emplena amb espais en blanc fins a la llargada indicada. Si s'intenta inserir-hi una cadena de longitud més llarga, es trunca.

La llargada mínima i per defecte (no és obligatòria) per a una columna de tipus `CHAR` és d'1 caràcter, i la llargada màxima permesa és de 255 caràcters.

Per indicar la llargada, cal especificar-la amb un nombre entre parèntesis, que indica el nombre de caràcters, que tindrà l'*string*. Per exemple `CHAR(10)`.

El tipus VARCHAR (llargada)

Aquest tipus especifica una cadena de longitud variable que pot ser, com a màxim, la indicada per llargada, valor que és obligatori introduir.

Els valors de tipus VARCHAR emmagatzemen el valor exacte que indica l'usuari sense afegir-hi espais en blanc. Si s'intenta inserir-hi una cadena de longitud més llarga, VARCHAR retorna un error.

La llargada màxima d'aquest tipus de dades és de 65.535 caràcters.

La llargada es pot indicar amb un nombre, que indica el nombre de caràcters màxim que contindrà l'*string*. Per exemple: VARCHAR(10).

En comparació del tipus de dades CHAR, VARCHAR funciona tal com es mostra en la taula 1.2.

TAULA 1.2. Comparació entre el tipus CHAR i VARCHAR

Valor	CHAR(4)	VARCHAR(4)
'ab'	'ab '	'ab'
'abcd'	'abcd'	'abcd'
'abcdefgh'	'abcd'	'abcd'

El tipus de dades alfanumèric més habitual per emmagatzemar *strings* en bases de dades MySQL és VARCHAR.

El tipus BINARY (llargada)

El tipus de dada BINARY és similar al tipus CHAR, però emmagatzema caràcters en binari. En aquest cas, la llargada mpre s'indica en bytes. La llargada mínima per a una columna BINARY és d'1 byte. La llargada màxima permesa és de 255.

El tipus VARBINARY (llargada)

El tipus de dada VARBINARY és similar al tipus VARCHAR, però emmagatzema caràcters en binari. En aquest cas, la llargada sempre s'indica en bytes. Els bytes que no s'emplenen explícitament s'emplenen amb **'\0'**.

Així, doncs, per exemple, una columna definida com a VARBINARY (4) a la qual s'assigni el valor **'a'** contindrà, realment, **'a\0\0\0'** i caldrà tenir-ho en compte a l'hora de fer, per exemple, comparacions, ja que no serà el mateix comparar la columna amb el valor **'a'** que amb el valor **'a\0\0\0'**.

El valor **'\0'** en hexadecimal es correspon amb **0x00**.

El tipus BLOB

El tipus de dades BLOB és un objecte que permet contenir una quantitat gran i variable de dades de tipus binari.

De fet, es pot considerar una dada de tipus BLOB com una dada de tipus VARBINARY, però sense limitació quant al nombre de bytes. De fet, els valors de tipus BLOB

s'emmagatzemen en un objecte separat de la resta de columnes de la taula, a causa dels seus requeriments d'espai.

Realment, hi ha diversos subtipus de `BLOG`: `TINYBLOB`, `BLOB`, `MEDIUMBLOB` i `LONGBLOB`.

- `TINYBLOB` pot emmagatzemar fins a 2^8 + 2 bytes
- `BLOB` pot emmagatzemar fins a 2^16 + 2 bytes
- `MEDIUMBLOB` pot emmagatzemar fins a 2^24 + 3 bytes
- `LONGBLOB` pot emmagatzemar fins a 2^32 + 4 bytes

El tipus TEXT

El tipus de dades `TEXT` és un objecte que permet contenir una quantitat gran i variable de dades de tipus caràcter.

De fet, es pot considerar una dada de tipus `TEXT` com una dada de tipus `VARCHAR`, però sense limitació quant al nombre de caràcters. De manera similar al tipus `BLOB`, els valors tipus `TEXT` també s'emmagatzemen en un objecte separat de la resta de columnes de la taula, a causa dels seus requeriments d'espai.

Realment, hi ha diversos subtipus de `TEXT`: `TINYTEXT`, `TEXT`, `MEDIUMTEXT` i `LONGTEXT`:

- `TINYTEXT` pot emmagatzemar fins a 2^8 + 2 bytes
- `TEXT` pot emmagatzemar fins a 2^16 + 2 bytes
- `MEDIUMTEXT` pot emmagatzemar fins a 2^24 + 3 bytes
- `LONGTEXT` pot emmagatzemar fins a 2^32 + 4 bytes

El tipus ENUM ('cadena1' [,'cadena2'] ... [,'cadena_n'])

El tipus `ENUM` defineix un conjunt de valors de tipus *string* amb una llista prefixada de cadenes que es defineixen en el moment de la definició de la columna i que es correspondran amb els valors vàlids de la columna.

Exemple de columna tipus ENUM

```
CREATE TABLE sizes ( \\
    name ENUM('small', 'medium', 'large')\\
);
```

El conjunt de valors són obligatòriament literals entre cometes simples.

Si una columna es declara de tipus `ENUM` i s'hi especifica que no admet valors nuls, aleshores, el valor per defecte serà el primer de la llista de cadenes.

El nombre màxim de cadenes diferents que pot suportar el tipus `ENUM` és 65535.

El tipus SET ('cadena1' [,'cadena2'] ... [,'cadena_n'])

Una columna de tipus SET pot contenir zero o més valors, però tots els elements que contingui han de pertànyer a una llista especificada en el moment de la creació.

El nombre màxim de valors diferents que pot suportar el tipus SET és 64.

Per exemple, es pot definir una columna de tipus `SET('one', 'two')`. I un element concret pot tenir qualsevol dels valors següents:

- ' '
- 'one'
- 'two'
- 'one,two' (el valor 'two,one' no es preveu perquè l'+++++++ dels elements no afecta les llistes)

1.3.2 Tipus de dades numèriques

MySQL suporta tots els tipus de dades numèriques de SQL estàndard:

- `INTEGER` (també abreujat per `INT`)
- `SMALLINT`
- `DECIMAL` (també abreujat per `DEC` o `FIXED`)
- `NUMERIC`

També suporta:

- `FLOAT`
- `REAL`
- `DOUBLE PRECISION` (també anomenat `DOUBLE`, simplement, o bé **REAL**)
- `BIT`
- `BOOLEAN`

Els tipus de dades INTEGER

El tipus `INTEGER` (comunament abreujat com a INT) emmagatzema valors enters. Hi ha diversos subtipus d'enters en funció dels valors admesos (vegeu la taula 1.3).

Taula 1.3. Tipus de dades INTEGER

Tipus d'enter	Emmagatzemament (en bytes)	Valor mínim (amb signe / sense signe)	Valor màxim (amb signe)
TINYINT	1	-128 / 0	127 / 255
SMALLINT	2	-32768 / 0	32767 / 65535
MEDIUMINT	3	-8388608 / 0	8388607 / 167772
INT	4	-2147483648 / 0	2147483647 / 4294
BIGINT	8	-9223372036854775808 / 0	922337203685477 184467440737095

Els tipus de dades enteres admeten l'especificació del nombre de dígits que cal mostrar d'un valor concret, utilitzant la sintaxi:

`INT (N)`, en què N és el nombre de dígits visibles.

Així, doncs, si s'especifica una columna de tipus `INT (4)`, en el moment de seleccionar un valor concret, es mostraran tan sols 4 dígits. Cal tenir en compte que aquesta especificació no condiciona el valor emmagatzemat, tan sols fixa el valor que cal mostrar.

També es pot especificar el nombre de dígits visibles en els subtipus d'`INTEGER`, utilitzant la mateixa sintaxi.

> Per emmagatzemar dades de tipus enter en bases de dades MySQL el més habitual és utilitzar el tipus de dades `INT`.

Tipus FLOAT, REAL i DOUBLE

`FLOAT`, `REAL` i `DOUBLE` són els tipus de dades numèriques que emmagatzemen valors numèrics reals (és a dir, que admeten decimals).

Els tipus `FLOAT` i `REAL` s'emmagatzemen en 4 bytes i els `DOUBLE` en 8 bytes.

Els tipus `FLOAT`, `REAL` o `DOUBLE PRECISION` admeten que s'especifiquin els dígits de la part entera (E) i els dígits de la part decimal (D, que poden ser 30 com a màxim, i mai més grans que E-2). La sintaxi per a aquesta especificació seria:

- `FLOAT(E,D)`
- `REAL(E,D)`
- `DOUBLE PRECISION(E,D)`

Exemple d'emmagatzematge en FLOAT

Per exemple, si es defineix una columna com a `FLOAT(7,4)` i s'hi vol emmagatzemar el valor **999.00009**, el valor emmagatzemat realment serà **999.0001**, que és el valor més proper (aproximat) a l'original.

Tipus de dades DECIMAL i NUMERIC

`DECIMAL` i `NUMERIC` són els tipus de dades reals de punt fix que admet MySQL. Són sinònims i, per tant, es poden utilitzar indistintament.

Els valors en punt fix no s'emmagatzemaran mai de manera arrodonida, és a dir, que si cal emmagatzemar un valor en un espai que no és adequat, emetrà un error. Aquest tipus de dades permeten assegurar que el valor és exactament el que s'ha introduït. No s'ha arrodonit. Per tant, es tracta d'un tipus de dades molt adequat per representar valors monetaris, per exemple.

Ambdós tipus de dades permeten especificar el total de dígits (T) i la quantitat de dígits decimals (D), amb la sintaxi següent:

```
DECIMAL (T,D)
```

```
NUMERIC (T,D)
```

Valors possibles per a NUMERIC

Per exemple un `NUMERIC (5,2)` podria contenir valors des de **-999.99** fins a **999.99**.

També s'admet la sintaxi `DECIMAL (T)` i `NUMERIC (T)` que és equivalent a `DECIMAL (T,0)` i `NUMERIC (T,0)`.

El valor per defecte de T és 10, i el seu valor màxim és 65.

Tipus de dades BIT

`BIT` és un tipus de dades que permet emmagatzemar bits, des d'1 (per defecte) fins a 64. Per especificar el nombre de bits que emmagatzemarà cal definir-lo, seguint la sintaxi següent:

`BIT (M)`, en què M és el nombre de bits que s'emmagatzemaran.

Els valors literals dels bits es donen seguint el format següent: **b'valor_binari'**. Per exemple, un valor binari admissible per a un camp de tipus BIT seria **b'0001'**.

Si donem el valor **b'1010'** a un camp definit com a `BIT(6)` el valor que emmagatzemarà serà **b'001010'**. Afegirà, doncs, zeros a l'esquerra fins a completar el nombre de bits definits en el camp.

`BIT` és un sinònim de `TINYINT(1)`.

Altres tipus numèrics

`BOOL` o `BOOLEAN` és el tipus de dades que permet emmagatzemar tipus de dades booleanes (que permeten els valors cert o fals). `BOOL` o `BOOLEAN` és sinònim de `TINYINT(1)`. Emmagatzemar un valor de zero es considera fals. En canvi, un valor diferent de zero s'interpreta com a cert.

Modificadors de tipus numèrics

Hi ha algunes paraules clau que es poden afegir a la definició d'una columna numèrica (entera o real) per tal de condicionar els valors que contindran.

- `UNSIGNED`: amb aquest modificador tan sols s'admetran els valors de tipus numèric no negatius.
- `ZEROFILL`: s'hi afegiran zeros a l'esquerra fins a completar el total de dígits del valor numèric, si cal.
- `AUTO_INCREMENTv`: quan s'afegeix un valor 0 o NULL en aquella columna, el valor que s'hi emmagatzema és el valor més alt incrementat en 1. El primer valor per defecte és 1.

1.3.3 Tipus de dades per a moments temporals

El tipus de dades que MySQL disposa per tal d'emmagatzemar dades que indiquin moments temporals són:

- `DATETIME`
- `DATE`
- `TIMESTAMP`
- `TIME`
- `YEAR`

> Tingueu en compte que quan cal referir-se a les dades referents a un any és important explicitar-ne els quatre dígits. És a dir, que per expressar l'any 98 del segle XX el millor és referir-s'hi explícitament així: 1998.

Si s'utilitzen dos dígits en lloc de quatre per expressar anys, cal tenir en compte que MySql els interpretarà de la manera següent:

- Els valors dels anys que van entre 00-69 s'interpreten com els anys: 2000-2069.

- Els valors dels anys que van entre 70-99 s'interpreten com els anys: 1970-1999.

El tipus de dada DATE

`DATE` permet emmagatzemar dates. El format d'una data en MySQL és 'AAAA-MM-DD', en què AAAA indica l'any expressat en quatre dígits, MM indica el mes expressat en dos dígits i DD indica el dia expressat en dos dígits.

La data mínima suportada pel sistema és '1000-01-01'. I la data màxima admissible en MySQL és '9999-12-31'.

El tipus de dada DATETIME

`DATATIME` és un tipus de dada que permet emmagatzemar combinacions de dies i hores.

El format de `DATETIME` en MySQL és 'AAAA-MM-DD HH:MM:SS', en què AAAA-MM-DD és l'any, el mes i el dia, i HH:MM:SS indiquen l'hora, minut i segon, expressats en dos dígits, separats per ':'.

Els valors vàlids per als camps de tipus `DATETIME` van des de '1000-01-01 00:00:00' fins a '9999-12-31 23:59:59'.

El tipus de dada TIMESTAMP

`TIMESTAMP` és un tipus de dada similar a `DATETIME` i té el mateix format per defecte: 'AAAA-MM-DD HH:MM:SS'. `TIMESTAMP`, però, permet emmagatzemar l'hora i data actuals en un moment determinat.

Si s'assigna el valor NULL a una columna `TIMESTAMP` o no se n'hi assigna cap explícitament, el sistema emmagatzema per defecte la data i hora actuals. Si s'especifica una data-hora concretes a la columna, aleshores la columna prendrà aquell valor, com si es tractés d'una columna `DATETIME`.

El rang de valors que admet `TIMESTAMP` és de '1970-01-01 00:00:01' a '2038-01-19 03:14:07'.

Una columna `TIMESTAMP` és útil quan es vol emmagatzemar la data i hora en el moment d'afegir o modificar una dada en la base de dades, per exemple.

Si en una taula hi ha més d'una columna de tipus `TIMESTAMP`, el funcionament de la primera columna `TIMESTAMP` és l'esperable: s'hi emmagatzema la data i hora de l'operació més recent, llevat que explícitament s'hi assigni un valor concret, i, aleshores, preval el valor especificat. La resta de columnes `TIMESTAMP` d'una mateixa taula no s'actualitzaran amb aquest valor. En aquest cas, si no s'especifica un valor concret, el valor emmagatzemat serà zero.

Exemple de creació de taula utilitzant TIMESTAMP

Per crear una taula amb una columna de tipus TIMESTAMP són equivalents les sintaxis següents:

- `CREATE TABLE t (ts TIMESTAMP);`
- `CREATE TABLE t (ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP);`
- `CREATE TABLE t (ts TIMESTAMP ON UPDATE CURRENT_TIMESTAMP\newline DEFAULT CURRENT_TIMESTAMP);`

Tipus de dades TIME

`TIME` és un tipus de dada específic que emmagatzema l'hora en el format 'HH:MM:SS'.

`TIME`, però, també permet expressar el temps transcorregut entre dos moments (diferència de temps). Per això, els valors que permet emmagatzemar són de '-838:59:59' a '838:59:59'. En aquesta cas, el format serà 'HHH:MM:SS'.

Altres formats també admesos per a una dada de tipus `TIME` són: 'HH:MM', 'D HH:MM:SS', 'D HH:MM', 'D HH', o 'SS', en què D indica els dies. És possible, també, un format que admet microsegons 'HH:MM:SS.uuuuuu' en què uuuuuu són els microsegons.

Tipus de dades YEAR

`YEAR` és una dada de tipus `BYTE` que emmagatzema dades de tipus any. El format per defecte és AAAA (l'any expressat en quatre dígits) o bé 'AAAA', expressat com a *string*.

També es poden utilitzar els tipus `YEAR(2)` o `YEAR(4)`, per especificar columnes de tipus any expressat amb dos dígits o any amb quatre dígits.

S'admeten valors des de 1901 fin a 2155. També s'admet 0000. En el format de dos dígits, s'admeten els valors del 70 al 69, que representen els anys del 1970 al 2069.

1.3.4 Altres tipus de dades

En MySQL hi ha extensions que permeten emmagatzemar altres tipus de dades: les dades poligonals.

Així, doncs, MySQL permet emmagatzemar dades de tipus objecte poligonal i dóna implementació al model geomètric de l'estàndard OpenGIS.

Per tant, podem definir columnes MySQL de tipus Polygon, Point, Curve o Line, entre d'altres.

1.4 Consultes simples

Un cop ja coneixem els diferents tipus de dades que ens podem trobar emmagatzemades en una base de dades (nosaltres ens hem centrat en *MySQL*, però en la resta de SGBD és similar), estem en condicions d'iniciar l'explotació de la base de dades, és a dir, de començar la gestió de les dades. Evidentment, per poder gestionar dades, prèviament cal haver definit les taules que han de contenir les dades, i per poder consultar dades cal haver-les introduït abans. L'aprenentatge del llenguatge SQL s'efectua, però, en sentit invers; és a dir, començarem coneixent les possibilitats de consulta de dades sobre taules ja creades i amb dades ja introduïdes. Necessitem, però, conèixer l'estructura de les taules que s'han de gestionar i les relacions existents entre elles.

Les taules que gestionarem formen part de dos dissenys diferents, és a dir, són taules de temes disjunts. Vegem-les.

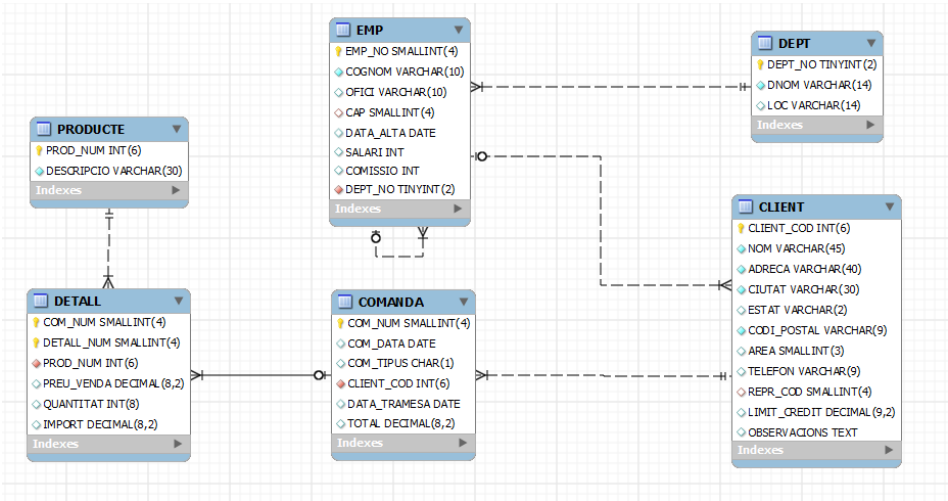
1. Temàtica empresa

La figura 1.1 mostra el disseny de l'esquema Empresa, implementat amb la utilitat *Data modeling* del programari *MySQL Workbench*, que utilitza una notació força intuïtiva:

- Les claus primàries s'indiquen amb el símbol d'una clau.
- Els atributs que no admeten valors nuls van precedits del símbol *.
- Els atributs que admeten valor nul estan marcats amb un rombe blanc.
- Les interrelacions 1:N s'indiquen amb una línia que acaba amb tres branques al costat de l'entitat interrelacionada en el costat N.
- L'opcionalitat s'indica amb un cercle al costat de l'entitat opcional i l'obligatorietat amb una petita línia perpendicular a la interrelació.

Les dades del disseny EntitatRelació del tema *empresa*, les gestionarem al llarg d'aquesta unitat.

FIGURA 1.1. Disseny de l'esquema Empresa



Tingueu present que la figura 1.1 mostra un esquema similar als diagrames Entitat-Relació o *disseny Chen*. Fent una ullada ràpida a aquest disseny, hem d'interpretar el següent:

- Tenim sis entitats diferents: departaments (DEPT), empleats (EMP), clients (CLIENT), productes (PRODUCTE), ordres (COMANDA) i detall de les ordres (DETALL).
- Entre les sis entitats s'estableixen relacions:
  - Entre DEPT i EMP (relació 1:N), ja que un empleat és assignat obligatòriament a un departament, i un departament té assignats zero o diversos empleats.
  - Entre EMP i EMP (relació reflexiva 1:N), ja que un empleat pot tenir per cap un altre empleat de l'empresa, i un empleat pot ser cap de zero o diversos empleats.
  - Entre EMP i CLIENT (relació 1:N), ja que un empleat pot ser el representant de zero o diversos clients, i un client pot tenir assignat un representant que ha de ser un empleat de l'empresa.
  - Entre CLIENT i COMANDA (relació 1:N), ja que un client pot tenir zero o diverses ordres a l'empresa, i una ordre és obligatòriament d'un client.
  - Entre COMANDA i DETALL (relació forta-feble 1:N), ja que l'ordre està formada per diverses línies, anomenades *detall de l'ordre*.
  - Entre DETALL i PRODUCTE (relació N:1), ja que cada línia de detall correspon a un producte.
- De vegades, algun alumne no expert en dissenys Entitat-Relació es pregunta el perquè de l'entitat DETALL i pensa que no hi hauria de ser, i la substitueix per una relació N:N entre les entitats COMANDA i PRODUCTE. Gran error. L'error rau en el fet que en una relació N:N entre COMANDA i PRODUCTE, un mateix producte no pot estar més d'una vegada en la mateixa ordre. En certs negocis, això pot ser una decisió encertada, però no sempre és així, ja que es poden donar situacions similars a les següents:
  - Per raons comercials o d'una altra índole, en una mateixa ordre hi ha certa quantitat d'un producte amb un preu i descomptes determinats, i una altra quantitat del mateix producte amb unes condicions de venda (preu i/o descomptes) diferents.
  - Pot ser que una quantitat de producte s'hagi de lliurar en una data, i una altra quantitat del mateix producte en una altra data. En aquesta situació, la data de tramesa hauria de residir en cada línia de detall.

La traducció corresponent al model relacional, considerant els atributs subratllats com a clau primària i el símbol (VN) que indica que admet valors nuls, és la següent:

```
DEPT (__Dept_no__, Dnom, Loc(VN))

EMP (__Emp_No__, Cognom, Ofici(VN), Cap(VN), Data_Alta(VN), Salari(VN), Comissió(VN), Dept_No) on Dept_No REFERENCIA DEPT

CLIENT (__Client_Cod__, Nom, Direcció, Ciutat, Estat(VN), Codi_Postal, Àrea(VN), Telèfon(VN), Repr_Cod(VN), Límit_Crèdit(VN), Observacions(VN)) on Repr_Cod REFERENCIA EMP

PRODUCTE (__Prod_Num__, Descripció)

COMANDA (__Com_Num__, Com_Data(VN), Com_Tipus(VN), Client_Cod, Data_Tramesa(VN), Total(VN))
 on Client_Cod REFERENCIA CLIENT

DETALL (__Com_Num, Detall_Num__, Prod_Num, Preu_Venda(VN), Quantitat(VN), Import(VN)) on Com_Num REFERENCIA COMANDA i Prod_Num REFERENCIA PRODUCTE
```

La implementació d'aquest model relacional en MySQL ha provocat les taules següents:

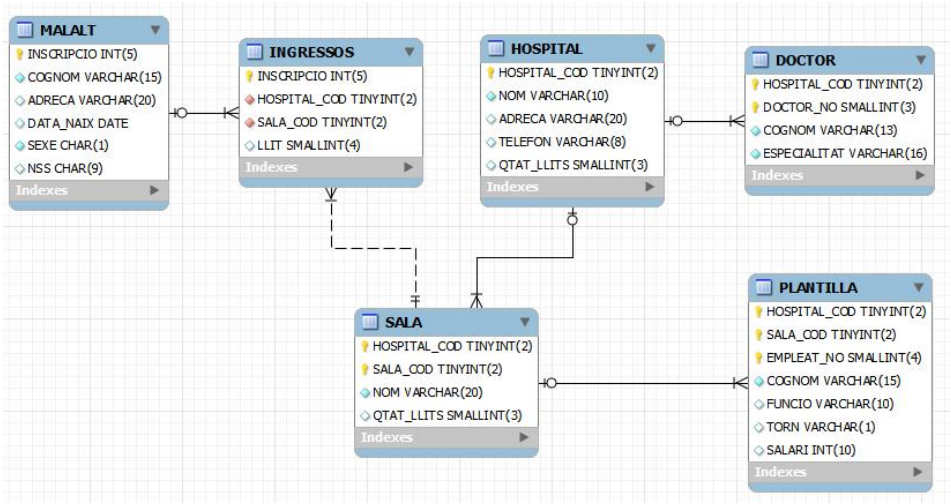
```
>> Taula DEPT, que conté els departaments de l'empresa

Nom          Null?     Tipus         Descripció
──────────── ──────── ──────────── ──────────────────────────────────────────
DEPT_NO      NOT NULL INT(2)        Número de departament de l'empresa
DNOM         NOT NULL VARCHAR(14)   Descripció del departament
LOC                   VARCHAR(14)   Localitat del departament

>> Taula EMP, que conté els empleats de l'empresa

Nom          Null?     Tipus         Descripció
──────────── ──────── ──────────── ──────────────────────────────────────────
EMP_NO       NOT NULL INT(4)        Número d'empleat de l'empresa
COGNOM       NOT NULL VARCHAR(10)   Cognom de l'empleat
OFICI                 VARCHAR(10)   Ofici de l'empleat
CAP                   INT(4)        Número de l'empleat que és el cap directe (taula EMP)
DATA_ALTA             DATE          Data d'alta
SALARI                INT(10)       Salari mensual
COMISSIO              INT(10)       Import de les comissions
DEPT_NO      NOT NULL INT(2)        Departament al qual pertany (taula DEPT)

>> Taula CLIENT, que conté els clients de l'empresa

Nom          Null?     Tipus         Descripció
──────────── ──────── ──────────── ──────────────────────────────────────────
CLIENT_COD   NOT NULL INT(6)        Codi de client
NOM          NOT NULL VARCHAR(45)   Nom del client
ADRECA       NOT NULL VARCHAR(40)   Direcció del client
CIUTAT       NOT NULL VARCHAR(30)   Ciutat del client
ESTAT                 VARCHAR(2)    País del client
CODI_POSTAL  NOT NULL VARCHAR(9)    Codi postal del client
AREA                  INT(3)        Àrea telefònica
TELEFON               VARCHAR(9)    Telèfon del client
REPR_COD              INT(4)        Codi del representant del client
                                    És un dels empleats de l'empresa (taula EMP)
LIMIT_CREDIT          DECIMAL(9,2)  Límit de crèdit de què disposa el client
OBSERVACIONS          TEXT          Observacions

>> Taula PRODUCTE, que conté els productes a vendre

Nom          Null?     Tipus         Descripció
──────────── ──────── ──────────── ──────────────────────────────────────────
PROD_NUM     NOT NULL INT(6)        Codi de producte
DESCRIPCIO   NOT NULL VARCHAR(30)   Descripció del producte
```

```
>> Taula COMANDA, que conté les ordres de venda

Nom          Null?     Tipus         Descripció
-----------  --------  ------------  ------------------------------------------------------
COM_NUM      NOT NULL  INT(4)        Número d'ordre de venda
COM_DATA               DATE          Data de l'ordre de venda
COM_TIPUS              VARCHAR(1)    Tipus d'ordre – Valors vàlids: A, B, C
CLIENT_COD   NOT NULL  INT(6)        Codi del client que efectua l'ordre (taula CLIENT)
DATA_TRAMESA           DATE          Data d'enviament de l'ordre
TOTAL                  DECIMAL(8,2)  Import total de l'ordre

>> Taula DETALL, que conté el detall de les ordres de venda

Nom          Null?     Tipus         Descripció
-----------  --------  ------------  ------------------------------------------------------
COM_NUM      NOT NULL  INT(4)        Número d'ordre (taula COMANDA)
DETALL_NUM   NOT NULL  INT(4)        Número de línia per cada ordre
PROD_NUM     NOT NULL  INT(6)        Codi del producte de la línia (taula PRODUCTE)
PREU_VENDA             DECIMAL(8,2)  Preu de venda del producte
QUANTITAT              INT(8)        Quantitat de producte a vendre
IMPORT                 DECIMAL(8,2)  Import total de la línia
```

2. Temàtica sanitat

La figura 1.2 mostra el disseny del tema Sanitat. Aquest disseny està efectuat amb l'eina d'anàlisi i disseny *Data Modeling* de *MySQL Workbench*

FIGURA 1.2. Disseny de l'esquema Sanitat



Les dades del disseny EntitatRelació del tema *sanitat* les gestionarem al llarg d'aquest mòdul.

Fent una ullada ràpida a aquest disseny, hem d'interpretar el següent:

- Tenim sis entitats diferents: hospitals (HOSPITAL), sales dels hospitals (SALA), doctors dels hospitals (DOCTOR), empleats de les sales dels hospitals (PLANTILLA), malalts (MALALT) i malalts ingressats actualment (INGRESSOS).
- Entre les sis entitats s'estableixen relacions:
  - Entre HOSPITAL i SALA (relació forta-feble 1:N), ja que les sales s'identifiquen amb un codi de sala dins cada hospital; és a dir, podem tenir una sala identificada amb el codi 1 a l'hospital X, i una sala identificada també amb el codi 1 a l'hospital Y.

  - Entre HOSPITAL i DOCTOR (relació forta-feble 1:N), ja que els doctors s'identifiquen amb un codi de doctor dins cada hospital; és a dir, podem tenir un doctor identificat amb el codi 10 a l'hospital X, i un doctor identificat també amb el codi 10 a l'hospital Y.
  - Entre SALA i PLANTILLA (relació forta-feble 1:N), ja que els empleats s'identifiquen amb un codi dins de cada sala; és a dir, podem tenir un empleat identificat amb el codi 55 a la sala 10 de l'hospital X, i un empleat identificat també amb el codi 55 en una altra sala de qualsevol hospital.
  - Entre MALALT i INGRESSOS (relació forta-feble 1:1), ja que un malalt pot estar ingressat o no.
  - Entre SALAi INGRESSOS (relació 1:N), ja que en una sala hi pot haver zero o diversos malalts ingressats, i un malalt ingressat només ho pot estar en una única sala.

- Ben segur que no és el millor disseny per a una gestió correcta d'hospitals, però ens interessa mantenir aquest disseny per a les possibilitats que ens donarà amb vista a l'aprenentatge del llenguatge SQL. Aprofitem, però, l'ocasió per comentar els punts foscos en el disseny:

  - Potser no és gaire normal que els empleats d'un hospital s'identifiquin dins de cada sala. És a dir, en el disseny, l'entitat PLANTILLA és feble de l'entitat SALA i, potser, seria més lògic que fos feble de l'entitat HOSPITAL de manera similar a l'entitat DOCTOR.
  - Per poder gestionar els pacients (MALALT) que actualment estan ingressats, és necessari establir una relació entre MALALT i SALA, la qual seria d'ordre N:1, ja que en una sala hi pot haver diversos pacients ingressats i un pacient, si està ingressat, ho està en una sala. La traducció corresponent al model relacional provocaria el següent:

```
SALA (__Hospital_Cod, Sala_Cod__, Nom, Qtat_Llits(VN)) on Sala_Cod REFERENCIA
    HOSPITAL

MALALT (__Inscripció__, Cognom, Adreça(VN), Data_Naix(VN), Sexe, Nss(VN),
    Hosp_Ingrés(VN), Sala_Ingrés(VN)) on {Hosp_Ingrés, Sala_Ingrés} REFERENCIA
     SALA
```

Fixem-nos que la relació (taula) MALALT conté la parella d'atributs (Hosp_Ingrés, Sala_ingrés) que conjuntament són clau forana de la relació (taula) SALA i que poden tenir valors nuls (VN), ja que un pacient no ha d'estar necessàriament ingressat. Si pensem una mica en la gestió real d'aquestes taules, ens trobarem que la taula MALALT normalment contindrà moltes files i que, per sort per als pacients, moltes d'aquestes tindran buits els camps Hosp_Ingrés i Sala_Ingrés, ja que, del total de pacients que passen per un hospital, un conjunt molt petit hi està ingressat en un moment determinat. Això pot arribar a provocar una pèrdua greu d'espai en la base de dades.

En aquestes situacions és lícit pensar en una entitat que aglutini els pacients que estan ingressats actualment (INGRESSOS), la qual ha de ser feble de l'entitat

que engloba tots els pacients (MALALT). Aquesta és l'opció adoptada en aquest disseny.

També seria adequat disposar d'una entitat que aglutinés les diferents especialitats mèdiques existents, de manera que poguéssim establir una relació entre aquesta entitat i l'entitat DOCTOR. No és el cas i, per tant, l'especialitat de cada doctor s'introdueix com un valor alfanumèric.

Així mateix, de manera similar, seria adequat disposar d'una entitat que aglutinés les diferents funcions que pot dur a terme el personal de la plantilla, de manera que poguéssim establir una relació entre aquesta entitat i l'entitat PLANTILLA. Tampoc és el cas i, per tant, la funció que cada empleat duu a terme s'introdueix com un valor alfanumèric.

La traducció corresponent al model relacional és la següent:

```
HOSPITAL (__Hospital_Cod__, Nom, Direcció(VN), Telèfon(VN), Qtat_Llits(VN))

SALA (__Hospital_Cod, Sala_Cod__, Nom, Qtat_Llits(VN)) on Hospital_Cod
    REFERENCIA HOSPITAL

PLANTILLA (__Hospital_Cod, Sala_Cod, Empleat_No__, Cognom, Funció(VN), Torn(VN)
    , Salari(VN)) ON {Hospital_Cod, Sala_Cod} REFERENCIA SALA

MALALT (__Inscripció__, Cognom, Direcció(VN), Data_Naix(VN), Sexe, Nss(VN))

INGRESSOS (__Inscripció__, Hospital_Cod, Sala_Cod, Llit(VN)) ON Inscripció
    REFERENCIA MALALT, {Hospital_Cod, Sala_Cod} REFERENCIA SALA

DOCTOR (__Hospital_Cod, Doctor_No__, Cognom, Especialitat) ON Hospital_Cod
    REFERENCIA HOSPITAL
```

La implementació d'aquest model relacional en MySQL ha provocat les taules següents:

```
>> Taula HOSPITAL, que conté l'enumeració dels hospitals

Nom          Null?     Tipus         Descripció
------------ --------- ------------- ------------------------------------------------------------
HOSPITAL_COD NOT NULL  INT(2)        Codi de l'hospital
NOM          NOT NULL  VARCHAR(10)   Nom de l'hospital
ADRECA                 VARCHAR(20)   Direcció de l'hospital
TELEFON                VARCHAR(8)    Telèfon de l'hospital
QTAT_LLITS             INT(3)        Quantitat de llits de l'hospital

>> Taula SALA, que conté les sales de cada hospital

Nom          Null?     Tipus         Descripció
------------ --------- ------------- ------------------------------------------------------------
HOSPITAL_COD NOT NULL  INT(2)        Codi de l'hospital (taula HOSPITAL)
SALA_COD     NOT NULL  INT(2)        Codi de la sala dins cada hospital
NOM          NOT NULL  VARCHAR(20)   Nom de la sala
QTAT_LLITS             INT(3)        Quantitat de llits de la sala

>> Taula DOCTOR, que conté els doctors dels diferents hospitals

Nom          Null?     Tipus         Descripció
------------ --------- ------------- ------------------------------------------------------------
HOSPITAL_COD NOT NULL  INT(2)        Codi de l'hospital (taula HOSPITAL)
DOCTOR_NO    NOT NULL  INT(3)        Codi de doctor dins cada hospital
COGNOM       NOT NULL  VARCHAR(13)   Cognom del doctor
ESPECIALITAT NOT NULL  VARCHAR(16)   Especialitat del doctor
```

```
>> Taula PLANTILLA, que conté els treballadors no doctors de les sales dels hospitals

Nom          Null?     Tipus         Descripció
──────────── ──────── ──────────── ──────────────────────────────────────────────
HOSPITAL_COD NOT NULL INT(2)        Codi de l'hospital
SALA_COD     NOT NULL INT(2)        Codi de la sala dins cada hospital
                                    La parella (HOSPITAL_COD, SALA_COD) és clau forana de
                                    la taula SALA
EMPLEAT_NO   NOT NULL INT(4)        Codi de l'empleat (independent d'hospital i sala)
COGNOM       NOT NULL VARCHAR(15)   Cognom de l'empleat
FUNCIO                VARCHAR(10)   Tasca de l'empleat
TORN                  VARCHAR(1)    Torn de l'empleat
                                    Valors possibles: (M)atí – (T)arda – (N)it
SALARI                INT(10)       Salari anual de l'empleat

>> Taula MALALT, que conté els malalts

Nom          Null?     Tipus         Descripció
──────────── ──────── ──────────── ──────────────────────────────────────────────
INSCRIPCIO   NOT NULL INT(5)        Identificació del malalt
COGNOM       NOT NULL VARCHAR(15)   Cognom del malalt
ADRECA                VARCHAR(20)   Adreça del malalt
DATA_NAIX             DATE          Data de naixement del malalt
SEXE         NOT NULL VARCHAR(1)    Sexe del malalt
                                    Valors possibles: (H)ome – (D)ona
NSS                   CHAR(9)       Número de Seguretat Social del malalt

>> Taula INGRESSOS, que conté els malalts ingressats als hospitals

Nom          Null?     Tipus         Descripció
──────────── ──────── ──────────── ──────────────────────────────────────────────
INSCRIPCIO   NOT NULL INT(5)        Codi de malalt
HOSPITAL_COD NOT NULL INT(2)        Codi d'hospital
SALA_COD     NOT NULL INT(2)        Codi de sala d'hospital
LLIT                  INT(4)        Número de llit que ocupa dins la sala
```

Així, doncs, ja estem en condicions d'introduir-nos en l'aprenentatge de les instruccions de consulta SQL, les quals practicarem en les taules dels temes empresa i sanitat presentats.

Per poder practicar el llenguatge SQL en un SGBD MySQL, vegeu l'Annex per tal d'instal·lar aquest programari i incorporar-hi les bases de dades d'exemple descrites (empresa i sanitat).

Totes les consultes en el llenguatge SQL es fan amb una única sentència, anomenada SELECT, que es pot utilitzar amb diferents nivells de complexitat. I totes les instruccions SQL finalitzen, obligatòriament, amb un punt i coma.

Tal com ho indica el seu nom, aquesta sentència permet **seleccionar** allò que l'usuari demana, el qual no ha d'indicar on ho ha d'anar a cercar ni com ho ha de fer.

La sentència SELECT consta de diferents apartats que s'acostumen a anomenar **clàusules**. Dos d'aquests apartats són sempre obligatoris i són els primers que presentarem. La resta de clàusules s'han d'utilitzar segons els resultats que es vulguin obtenir.

És molt important que practiqueu sobre un SGBD MySQL totes les instruccions SQL que es van explicant. Per fer-ho, seguiu les instruccions de l'Annex i instal·leu-vos, si no ho heu fet encara, el programari MySQL, i importeu les bases de dades d'exemple **empresa** i **sanitat** per tal de poder provar els exemples que es van mostrant al llarg del material.

1.4.1 Clàusules SELECT i FROM

La sintaxi més simple de la sentència `SELECT` utilitza aquestes dues clàusules de manera obligatòria:

```
select <expressió/columna>, <expressió/columna>,...
from <taula>, <taula>,...;
```

> La clàusula `SELECT` permet escollir columnes i/o valors (resultats de les expressions) derivats d'aquestes.
> La clàusula `FROM` permet especificar les taules en què cal anar a cercar les columnes o sobre les quals es calcularan els valors resultants de les expressions.

Una sentència SQL es pot escriure en una única línia, però per fer la sentència més llegible s'acostumen a utilitzar diferents línies per a les diferents clàusules.

Exemple d'utilització simple de les clàusules select i from

En el tema *empresa*, es volen mostrar els codis, cognoms i oficis dels empleats.

Aquest és un exemple clar de les consultes més simples: cal indicar les columnes a visualitzar i la taula d'on visualitzar-les. La sentència és la següent:

```
select emp_no, cognom, ofici from emp;
```

El resultat que s'obté és el següent:

```
EMP_NO                 COGNOM      OFICI
---------------------- ----------- ----------
7369                   SÁNCHEZ     EMPLEAT
7499                   ARROYO      VENEDOR
7521                   SALA        VENEDOR
7566                   JIMÉNEZ     DIRECTOR
7654                   MARTÍN      VENEDOR
7698                   NEGRO       DIRECTOR
7782                   CEREZO      DIRECTOR
7788                   GIL         ANALISTA
7839                   REY         PRESIDENT
7844                   TOVAR       VENEDOR
7876                   ALONSO      EMPLEAT
7900                   JIMENO      EMPLEAT
7902                   FERNÁNDEZ   ANALISTA
7934                   MUÑOZ       EMPLEAT

14 rows selected
```

Exemple d'utilització d'expressions en la clàusula select

En el tema *empresa*, es volen mostrar els codis, cognoms i salari anual dels empleats.

Com que saben que en la taula EMP consta el salari mensual de cada empleat, sabem calcular, mitjançant el producte pel nombre de pagues mensuals en un any (12, 14, 15...?), el seu salari anual. Suposarem que l'empleat té catorze pagues mensuals, com la majoria dels mortals! Per tant, en aquest cas, alguna de les columnes a visualitzar és el resultat d'una expressió:

```
select emp_no, cognom, salari*14 from emp;
```

El resultat que s'obté és el següent:

```
EMP_NO                         COGNOM      SALARI*14
------------------------------ ----------- --------------------------
7369                           SÁNCHEZ     1456000
7499                           ARROYO      2912000
7521                           SALA        2275000
7566                           JIMÉNEZ     5414500
7654                           MARTÍN      2275000
7698                           NEGRO       5187000
7782                           CEREZO      4459000
7788                           GIL         5460000
7839                           REY         9100000
7844                           TOVAR       2730000
7876                           ALONSO      2002000
7900                           JIMENO      1729000
7902                           FERNÁNDEZ   5460000
7934                           MUÑOZ       2366000

14 rows selected
```

Fixem-nos que el llenguatge SQL utilitza els noms reals de les columnes com a títols en la presentació del resultat i, en cas de columnes que corresponguin a expressions, ens mostra l'expressió com a títol.

El llenguatge SQL permet donar un nom alternatiu (anomenat **àlies**) a cada columna. Per a això, es pot emprar la sintaxi següent:

```
select <expressió/columna> [as àlies], <expressió/columna> [as àlies],...
from <taula>, <taula>,...;
```

Tingueu en compte el següent:

- Si l'àlies és format per diverses paraules, cal tancar-lo entre cometes dobles.
- Hi ha alguns SGBD que permeten la no utilització de la partícula `as` (com l'Oracle i el MySQL) però en d'altres és obligatòria (com l'MS-Access).

Exemple d'utilització d'àlies en la clàusula select

En el tema *empresa*, es volen mostrar els codis, cognoms i salari anual dels empleats.

La instrucció per assolir l'objectiu pot ser, amb la utilització d'àlies:

```
select emp_no, cognom, salari*14 as "Salari Anual" from emp;
```

Obtindríem el mateix resultat sense la partícula `as`:

```
select emp_no, cognom, salari*14 "Salari Anual" from emp;
```

El resultat que s'obté en aquest cas és el següent:

```
EMP_NO                  COGNOM     Salari Anual
----------------------- ---------- -------------------------

7369                    SÁNCHEZ    1456000
7499                    ARROYO     2912000
7521                    SALA       2275000
7566                    JIMÉNEZ    5414500
7654                    MARTÍN     2275000
7698                    NEGRO      5187000
7782                    CEREZO     4459000
7788                    GIL        5460000
7839                    REY        9100000
7844                    TOVAR      2730000
7876                    ALONSO     2002000
7900                    JIMENO     1729000
7902                    FERNÁNDEZ  5460000
7934                    MUÑOZ      2366000

14 rows selected
```

El llenguatge SQL facilita una manera senzilla de mostrar totes les columnes de les taules seleccionades en la clàusula from (i perd la possibilitat d'emprar un àlies) i consisteix a utilitzar un asterisc en la clàusula select.

Exemple d'utilització d'asterisc en la clàusula select

Se'ns demana de mostrar, en el tema *empresa*, tota la informació que hi ha en la taula que conté els departaments.

La instrucció que ens permet assolir l'objectiu és la següent (fixem-nos que la instrucció SELECT amb asterisc ens mostra les dades de totes les columnes de la taula):

```
select * from dept;
```

I obtenim el resultat esperat:

```
DEPT_NO    DNOM           LOC
---------- -------------- --------------

10         COMPTABILITAT  SEVILLA
20         INVESTIGACIÓ   MADRID
30         VENDES         BARCELONA
40         PRODUCCIÓ      BILBAO
```

Tot i que disposem de l'asterisc per visualitzar totes les columnes de les taules de la clàusula from, de vegades ens interessarà conèixer les columnes d'una taula per dissenyar una sentència SELECT adequada a les necessitats i no utilitzar l'asterisc per visualitzar totes les columnes.

Els SGBD acostumen a facilitar mecanismes per visualitzar un **descriptor** breu d'una taula. En MySQL (i també en Oracle), disposem de l'ordre **desc** (no és sentència del llenguatge SQL) per a això. Cal emprar-la acompanyant el nom de la taula.

Exemple d'obtenció del descriptor d'una taula

Si necessitem conèixer les columnes que formen una taula determinada (i les seves característiques bàsiques), en podem obtenir el descriptor: **desc**

```
SQL> desc dept;

Nom                                       Null?     Tipus
----------------------------------------- --------- ------------
DEPT_NO                                   NOT NULL INT(2)
DNOM                                      NOT NULL VARCHAR(14)
LOC                                                VARCHAR(14)
```

L'ordre DESC no és exactament una ordre SQL, sinó una ordre que faciliten els SGBD per tal de visualitzar l'estructura o el diccionari de les dades emmagatzemades en una BD concreta. Per tant, com que no es tracta estrictament d'una ordre SQL, admet la no-utilització del punt i coma (;) al final de la sentència. D'aquesta manera, els codis següents són equivalents:

- `SQL> desc dept;`
- `SQL> desc dept`

Suposem que estem connectats amb l'SGBD a la base de dades o l'esquema (*schema*, en anglès) per defecte que conté les taules corresponents al tema *empresa* i que necessitem accedir a taules d'una base de dades que conté les taules corresponents a l'esquema *sanitat*. Ho podem aconseguir?

La clàusula from pot fer referència a taules d'una altra base de dades. En aquesta situació, cal anotar la taula com a `<nom_esquema>.<nom_taula>`.

L'accés a objectes d'altres esquemes (bases de dades) per a un usuari connectat a un esquema només és possible si té concedits els permisos d'accés corresponents.

Bases de dades gestionades per una instància d'un SGBD corporatiu

Un SGBD és un conjunt de programes encarregats de gestionar bases de dades.

En els SGBD ofimàtiques (MS-Access) podem posar en marxa l'SGBD sense l'obligació d'obrir (engegar) cap base de dades. En un moment concret, podem tenir diferents execucions de l'SGBD (instàncies), cadascuna de les quals pot donar servei a una única base de dades.

Els SGBD corporatius (Oracle, MsSQLServer, MySQL, PostgreSQL...), en posar-los en marxa, obliguen a tenir definit el conjunt de bases de dades al qual donen servei, conjunt que s'acostuma a anomenar *cluster database*. En una mateixa màquina, podem tenir diferents execucions d'un mateix SGBD (instàncies), cadascuna de les quals dóna servei a un conjunt diferent de bases de dades (*cluster database*). Així, doncs, cada instància d'un SGBD permet gestionar un *cluster database*.

En les màquines en què hi ha SGBD corporatius instal·lats, s'acostuma a configurar un servei de sistema operatiu per a cada instància configurada perquè sigui gestionada per l'SGBD. Així, en màquines amb sistema operatiu MS-Windows, aquesta situació es pot constatar ràpidament fent una ullada als serveis instal·lats (Tauler de control\Eines administratives\Serveis), en què podríem trobar diversos serveis de l'Oracle, MySQL, SQLServer... identificats per noms que es decideixen en el moment de creació de la instància (*cluster database*).

Així, doncs, per exemple, en una empresa en què és molt usual tenir una base de dades per a la gestió comercial, una base de dades per al control de la producció, una base de dades per a la gestió de personal, una base de dades per a la gestió financera..., en SGBD com MySQL, PostgreSQL i SQLServer podrien ser diferents bases de dades gestionades per una mateixa instància de l'SGBD.

Exemple d'accés a taules d'altres esquemes

Si, estant connectats a l'esquema (base de dades) *empresa*, volem mostrar els hospitals existents en l'esquema *sanitat*, caldrà fer el següent:

```
select * from sanitat.hospital;
```

El resultat que s'obté és el següent:

```
HOSPITAL_COD  NOM        ADREÇA      TELÈFON  QTAT_LLITS
____________ ___________ ____________________ _________
    ___________

13            Provincial O Donell 50          964-4264 88
18            General    Atocha s/n           595-3111 63
22            La Paz     Castellana 1000      923-5411 162
45            San Carlos Ciudad Universitaria 597-1500 92

4 rows selected
```

El llenguatge SQL efectua el producte cartesià de totes les taules que troba en la clàusula from. En aquest cas, hi pot haver columnes amb el mateix nom en diferents taules i, si és així i cal seleccionar-ne una, cal utilitzar obligatòriament la sintaxi `<nom_taula>.<nom_columna>` i, fins i tot, la sintaxi `<nom_esquema>.<nom_taula>.<nom_columna>` si s'accedeix a una taula d'un altre esquema.

Exemple de sentència "SELECT" amb diverses taules i coincidència en noms de columnes

Si des de l'esquema *empresa* volem mostrar el producte cartesià de totes les files de la taula `DEPT` amb totes les files de la taula `SALA` de l'esquema *sanitat* (visualització que no té cap sentit, però que fem a tall d'exemple), mostrant únicament les columnes que formen les claus primàries respectives, executaríem el següent:

```
select dept.dept_no, sanitat.sala.hospital_cod,
   sanitat.sala.sala_cod
from dept, sanitat.sala;
```

El resultat obtingut és format per quaranta files. En mostrem només algunes:

```
DEPT_NO                  HOSPITAL_COD             SALA_COD
_____________________ _____________________ ________________

10                       13                       3
10                       13                       6
10                       18                       3
...
40                       45                       1
40                       45                       2
40                       45                       4

40 rows selected
```

En aquest cas, la sentència s'hauria pogut escriure sense utilitzar el prefix sanitat en les columnes de la taula SALA en la clàusula select, ja que en la clàusula from no apareix més d'una taula anomenada SALA i, per tant, no hi ha problemes d'ambigüitat. Hauríem pogut escriure el següent:

```
select dept.dept_no, sala.hospital_cod, sala.sala_cod
from dept, sanitat.sala;
```

El llenguatge SQL permet definir **àlies** per a una taula. Per aconseguir-ho, cal escriure l'àlies en la clàusula from després del nom de la taula i abans de la coma que la separa de la taula següent (si existeix) de la clàusula from.

Exemple d'utilització d'àlies per a noms de taules

Si estem connectats a l'esquema *empresa*, per obtenir el producte cartesià de totes les files de la taula DEPT amb totes les files de la taula SALA de l'esquema *sanitat*, que mostri únicament les columnes que formen les claus primàries respectives, podríem executar la instrucció següent:

```
select d.dept_no, s.hospital_cod, s.sala_cod
from dept d, sanitat.sala s;
```

El llenguatge SQL permet utilitzar el resultat d'una sentència SELECT com a taula dins la clàusula from d'una altra sentència SELECT.

Exemple de sentència "SELECT" com a taula en una clàusula from

Així, doncs, una altra manera d'obtenir, estant connectats a l'esquema *empresa*, el producte cartesià de totes les files de la taula DEPT amb totes les files de la taula SALA de l'esquema *sanitat*, que mostri únicament les columnes que formen les claus primàries respectives, seria la següent:

```
select d.dept_no, h.hospital_cod, h.sala_cod
from dept d, (select hospital_cod, sala_cod from sanitat.sala) h;
```

MySQL (igual que Oracle) incorpora una taula fictícia, anomenada `DUAL`, per efectuar càlculs independents de qualsevol taula de la base de dades aprofitant la potència de la sentència SELECT.

Així, doncs, podem emprar aquesta taula per tal de fer el següent:

1. Efectuar càlculs matemàtics

```
SQL> select 4 * 3 - 8 / 2 as "Resultat" from dual;


RESULTAT
------------------------
8
1 rows selected
```

2. Obtenir la data del sistema, sabent que la proporciona la funció sysdate()

```
SQL> select sysdate() from dual;

SYSDATE()
------------------------
09/02/08
1 rows selected
```

La taula DUAL també pot ser elidida. D'aquesta manera, les sentències següents serien equivalents a les exposades anteriorment:

```
select 4 * 3 - 8 / 2 as "Resultat";

select sysdate();
```

1.4.2 Clàusula ORDER BY

La sentència SELECT té més clàusules a banda de les conegudes select i from. Així, té una clàusula `order by` que permet ordenar el resultat de la consulta.

Exemple d'ordenació de les dades utilitzant la clàusula ORDER BY

Si es volen obtenir totes les dades de la taula departaments, ordenades pel nom de la localitat, podem executar la sentència següent:

```
SELECT * FROM DEPT ORDER BY loc;
```

I n'obtindrem el resultat següent:

```
DEPT_NO     DNOM            LOC
─────────── ─────────────── ──────────
30          VENDES          BARCELONA
40          PRODUCCIÓ       BILBAO
20          INVESTIGACIÓ    MADRID
10          COMPTABILITAT   SEVILLA
```

1.4.3 Clàusula Where

La clàusula where s'afegeix darrere de la clàusula from de manera que ampliem la sintaxi de la sentència SELECT:

```
select <expressió/columna>, <expressió/columna>,...
from <taula>, <taula>,...
[where <condició_de_cerca>];
```

> La clàusula where permet establir els criteris de cerca sobre les files generades per la clàusula from.

La complexitat de la clàusula where és pràcticament il·limitada gràcies a l'abundància d'operadors disponibles per efectuar operacions.

1. Operadors aritmètics

Són els típics operadors +,-,*,/ utilitzables per formar expressions amb constants, valors de columnes i funcions de valors de columnes.

2. Operadors de dates

Per tal d'obtenir la diferència entre dues dates:

- Operador -, per restar dues dates i obtenir el nombre de dies que les separen.

3. Operadors de comparació

Disposem de diferents operadors per efectuar comparacions:

- Els típics operadors =, !=, >, <, >=,<=, per efectuar comparacions entre dades i obtenir-ne un resultat booleà: cert o fals.
- L'operador [NOT] LIKE, per comparar una cadena (part esquerra de l'operador) amb una cadena patró (part dreta de l'operador) que pot contenir els caràcters especials següents:
  - % per indicar qualsevol cadena de zero o més caràcters.
  - _ per indicar qualsevol caràcter.

Així:

```
LIKE 'Torres' compara amb la cadena 'Torres'.
LIKE 'Torr%'  compara amb qualsevol cadena iniciada per 'Torr'.
LIKE '%S%' compara amb qualsevol cadena que contingui 'S'.
LIKE '_o%' compara amb qualsevol cadena que tingui per segon caràcter una 'o'.
LIKE '%%__%%' compara amb qualsevol cadena de dos caràcters.
```

Un últim conjunt d'operadors lògics:

```
[NOT] BETWEEN valor_1 AND valor_2
```

que permet efectuar la comparació entre dos valors.

```
[NOT] IN (llista_valors_separats_per_comes)
```

que permet comparar amb una llista de valors.

```
IS [NOT] NULL
```

que permet reconèixer si ens trobem davant d'un valor null.

```
<comparador genèric> ANY (llista_valors)
```

que permet efectuar una comparació genèrica (=, !=, >, <, >=, <=) amb **qualsevol** dels valors de la dreta. Els valors de la dreta seran el resultat d'execució d'una altra consulta (SELECT), per exemple:

```
select * from emp where cognom != any (select 'Alonso' from dual);
```

```
<comparador genèric> ALL (llista_valors)
```

que permet efectuar una comparació genèrica (=, !=, >, <, >=, <=) amb **tots** els valors de la dreta. Els valors de la dreta seran el resultat d'execució d'una altra consulta (SELECT), per exemple:

```
select * from emp where cognom != all (select 'Alonso' from dual);
```

Fixem-nos que:

- =ANY és equivalent a IN.
- =ANY és equivalent a NOT IN.
- = ALL sempre és fals si la llista té més d'un element diferent.
- != ANY sempre és cert si la llista té més d'un element diferent.

Exemple de filtratge simple en la clàusula where

En el tema *empresa*, es volen mostrar els empleats (codi i cognom) que tenen un salari mensual igual o superior a 200.000 i també el seu salari anual (suposem que en un any hi ha catorze pagues mensuals).

La instrucció que permet assolir l'objectiu és aquesta:

```
select emp_no as "Codi", cognom as "Empleat", salari*14 as "
    Salari anual" from emp where salari >= 200000;
```

El resultat obtingut és aquest:

```
Codi                  Empleat    Salari anual
--------------------- ---------- ------------------------
7499                  ARROYO     2912000
7566                  JIMENEZ    5414500
7698                  NEGRO      5187000
7782                  CEREZO     4459000
7788                  GIL        5460000
7839                  REY        9100000
7902                  FERNANDEZ  5460000

7 rows selected
```

Exemple de filtratge de dates utilitzant l'especificació ANSI per indicar una data

En el tema *empresa*, es volen mostrar els empleats (codi, cognom i data de contractació) contractats a partir del mes de juny del 1981.

La instrucció que permet assolir l'objectiu és aquesta:

```
select emp_no as "Codi", cognom as "Empleat", data_alta as "
    Contracte" from emp where data_alta >= '1981-06-01';
```

El resultat obtingut és aquest:

```
Codi                  Empleat    Contracte
--------------------- ---------- ------------------------
7654                  MARTIN     29/09/81
7782                  CEREZO     09/06/81
7788                  GIL        09/11/81
7839                  REY        17/11/81
7844                  TOVAR      08/09/81
7876                  ALONSO     23/09/81
7900                  JIMENO     03/12/81
7902                  FERNANDEZ  03/12/81
7934                  MUÑOZ      23/01/82

9 rows selected
```

Exemple d'utilització d'operacions lògiques en la clàusula where

En el tema *empresa*, es volen mostrar els empleats (codi, cognom) resultat de la intersecció dels dos darrers exemples, és a dir, empleats que tenen un sou mensual igual o superior a 200.000 i contractats a partir del mes de juny del 1981.

La instrucció per aconseguir el que se'ns demana és aquesta:

```
select emp_no as "Codi", cognom as "Empleat" from emp where
    data_alta >= '1981-06-01' and salari >= 200000;
```

El resultat obtingut és aquest:

```
Codi                   Empleat
---------------------- ----------
7782                   CEREZO
7788                   GIL
7839                   REY
7902                   FERNANDEZ

4 rows selected
```

Exemple 1 d'utilització de l'operador like

En el tema *empresa*, es volen mostrar els empleats que tenen com a inicial del cognom una 'S'.

La instrucció demanada pot ser aquesta:

```
select cognom as "Empleat" from emp where cognom like 'S%';
```

Aquesta instrucció mostra els empleats amb el cognom començat per la lletra 'S' majúscula, i se suposa que els cognoms estan introduïts amb la inicial en majúscula, però, per tal d'assegurar la solució, en l'enunciat es pot utilitzar la funció incorporada upper(), que retorna una cadena en majúscules:

```
select cognom as "Empleat" from emp where upper(cognom) like 'S%'
    ;
```

Exemple 2 d'utilització de l'operador like

En el tema *empresa*, es volen mostrar els empleats que tenen alguna **S** en el cognom.

La instrucció demanada pot ser aquesta:

```
select cognom as "Empleat" from emp where upper(cognom) like '%S%
    ';
```

Exemple 3 d'utilització de l'operador like

En el tema *empresa*, es volen mostrar els empleats que no tenen la **R** com a tercera lletra del cognom.

La instrucció demanada pot ser aquesta:

```
select cognom as "Empleat" from emp where upper(cognom) not like
    '__R%';
```

Exemple d'utilització de l'operador between

En el tema *empresa*, es volen mostrar els empleats que tenen un salari mensual entre 100.000 i 200.000.

La instrucció demanada pot ser aquesta:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari"
    from emp where salari >= 100000 and salari <= 200000;
```

Podem, però, utilitzar l'operador **between**:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari"
    from emp where salari between 100000 and 200000;
```

Exemple d'utilització dels operadors in o =any

En el tema *empresa*, es volen mostrar els empleats dels departaments 10 i 30.

La instrucció demanada pot ser aquesta:

```
select emp_no as "Codi", cognom as "Empleat", dept_no as "
    Departament" from emp where dept_no = 10 or dept_no = 30;
```

Podem, però, utilitzar l'operador `in`:

```
select emp_no as "Codi", cognom as "Empleat", dept_no as "
    Departament" from emp where dept_no in (10,30);
```

Exemple d'utilització de l'operador "not in"

En el tema *empresa*, es volen mostrar els empleats que no treballen en els departaments 10 i 30.

La instrucció demanada pot ser aquesta:

```
select emp_no as "Codi", cognom as "Empleat", dept_no as "
    Departament" from emp where dept_no !=10 and dept_no != 30;
```

2. Consultes de selecció complexes

Una vegada coneixem els tipus de dades que facilita l'SGBD i sabem utilitzar la sentència SELECT per a l'obtenció d'informació de la base de dades amb la utilització de les clàusules select (selecció de columnes i/o expressions), from (selecció de les taules corresponents) i where (filtratge adequat de les files que interessen), estem en condicions d'aprofundir en les possibilitats que té la sentència SELECT, ja que només en coneixem les clàusules bàsiques.

A l'hora d'obtenir informació de la base de dades, ens interessa poder incorporar, en les expressions de les clàusules select i where, càlculs genèrics que els SGBD faciliten amb funcions incorporades (càlculs com el valor absolut, arrodoniments, truncaments, extracció de subcadenes en cadenes de caràcters, extracció de l'any, mes o dia en dates...), com també poder efectuar consultes més complexes que permetin classificar la informació, efectuar agrupaments de files, realitzar combinacions entre diferents taules i incloure els resultats de consultes dins altres consultes.

2.1 Funcions incorporades a MySQL

Funcions d'agrupament

La sentència SELECT té més clàusules a banda de les conegudes select, from i where. Així, té una clàusula que permet agrupar les files resultants de la consulta i aplicar-hi funcions d'agrupament: max() per al càlcul del valor més gran de cada grup, min() per al càlcul del valor més petit de cada grup, etc.

Els SGBD acostumen a incorporar funcions utilitzables des del llenguatge SQL. El llenguatge SQL, en si mateix, no incorpora funcions genèriques, a excepció de les anomenades *funcions d'agrupament*.

> Les funcions incorporades proporcionades pels SGBD es poden utilitzar dins d'expressions i actuen amb els valors de les columnes, variables o constants en les clàusules select, where i order by.

També és possible utilitzar el resultat d'una funció com a valor per utilitzar en una altra funció.

Les funcions principals facilitades per MySQL són les de matemàtiques, de cadenes de caràcters, de gestió de moments temporals, de control de flux, però disposem de més funcions. Sempre caldrà consultar la documentació de MySQL.

2.1.1 Funcions matemàtiques

La taula 2.1 ens presenta les funcions i els operadors matemàtics principals proporcionats per l'SGBD MySQL.

TAULA 2.1. Funcions matemàtiques principals proporcionades per l'SGBD MySQL

Funció o operador matemàtic	Descripció	Exemples
ABS(x)	Retorna el valor absolut de x	
ACOS(x)	Retorna l'arc cosinus de x	SELECT ACOS(1);
ASIN(x)	Retorna l'arc sinus de x	SELECT ASIN(0.2);
ATAN(x), ATAN2(x,y)	Retorna l'arc tangent de x i l'arc tangent de les coordenades (x,y), respectivament	SELECT ATAN(2); SELECT ATAN(-2,2);
CEIL(x)	Retorna el valor enter immediatament superior a x, o x si ja és un valor enter	SELECT CEILING(1.23); Retorna 2. SELECT CEILING(3); Retorna 3
CEILING(x)	Sinònim de CEIL(x)	
CONV(x,b1,b2)	Converteix el nombre x, considerat expressat en base b1, a base b2.	SELECT CONV(10,10,2); Retorna 1010
COS(x)	Retorna el cosinus de x	
COT(x)	Retorna la cotangent de x	
CRC32(x)	Calcula el valor CRC32 (*cyclic redundancy check*)	
DEGREES(x)	Converteix x, expressat en radians, a graus	SELECT DEGREES(PI());
DIV	Calcula la divisió entera	SELECT 5 DIV 2; Retorna 2.
/	Operador de divisió	SELECT 3/5; Retorna: 0.60
EXP(x)	Calcula e elevat a x	
FLOOR(x)	Retorna el valor enter immediatament inferior a x. O x, si ja és un valor enter.	SELECT FLOOR(-1.23); Retorna: -2.
LN(x)	Retorna el logaritme de base e de x	
LOG10(x)	Retorna el logaritme de base 10 de x	
LOG2(x)	Retorna el logaritme de base 2 de x	
LOG(b,x)	Retorna el logaritme de base b de x	
-	Operador resta	SELECT 5-3;
MOD(x,y)	Retorna la resta de la divisió de x i y. És equivalent a N % M, i també a N MOD M	SELECT MOD(29,9); Retorna: 2
OCT(x)	Retorna la representació octal del nombre x, expressat en decimal.	
PI()	Retorna el valor pi	
+	Operador de suma	SELECT 5+3;
POW(x,y)	Retorna x elevat a y	
POWER(x,y)	Sinònim de POW	
RADIANS(x)	Retorna x, expressat en graus, a radians	
RAND()	Retorna un valor real aleatori entre 0 i 1	

TAULA 2.1 (continuació)

Funció o operador matemàtic	Descripció	Exemples
ROUND(x)	Arrodoneix x al nombre enter més proper.	SELECT ROUND (1.9); Retorna 2. SELECT ROUND(1.5); Retorna 2. SELECT ROUND (1.2); Retorna 1.
SIGN(x)	Retorna -1 si x és negatiu. Retorna 1 si x és positiu. I retorna 0 si x = 0	
SIN(x)	Retorna el valor del sinus de x	
SQRT(x)	Retorna l'arrel quadrada de x	
TAN(x)	Retorna la tangent de x	
*	Operador de multiplicació	SELECT 5*3;
TRUNCATE(x,d)	Trunca x a d decimals	SELECT TRUNCATE(-1.999,1); Retorna -1.9
-	Operador canvi de signe	SELECT - 2; Retorna: -2

En negreta s'han destacat les funcions més utilitzades a l'hora de manipular expressions numèriques.

2.1.2 Funcions de cadenes de caràcters

La taula 2.2 ens presenta les funcions principals per a la gestió de cadenes de caràcters proporcionades per l'SGBD MySQL.

TAULA 2.2. Principals funcions de gestió de cadenes de caràcters proporcionades per l'SGBD MySQL

Funció	Descripció	Exemple
ASCII(s)	Retorna el valor numèric del caràcter de més a l'esquerra.	SELECT ASCII('2'); Retorna: 50
BIN(n)	Retorna un *string* amb la representació en binari del nombre n.	SELECT BIN(12); Retorna: '1100'
BIT_LENGTH(s)	Retorna la longitud en bits de la cadena s.	SELECT BIT_LENGTH('text'); Retorna: 32
CHAR_LENGTH(s)	Retorna el nombre de caràcters de la cadena s.	SELECT BIT_LENGTH('text'); Retorna: 4
CHAR(n,... [USING nom_charset])	Retorna la llista de caràcters per cada enter.	SELECT CHAR(77,121,83,81,'76'); Retorna: 'MySQL'. Explicitant el tipus de caràcters: SELECT CHAR(83 USING utf8);
CHARACTER_LENGTH(s)	Sinònim de CHAR_LENGTH(s).	
CONCAT(s1,s2, ...)	Retorna la concatenació de s1, s2, etc. Si un dels paràmetres és null, la funció retorna null.	SELECT CONCAT('My', 'S', 'QL'); Retorna: 'MySQL' . Que és equivalent a SELECT 'My' 'S' 'QL';
CONCAT_WS(s,s1,s2, ...)	Retorna la concatenació de s1, s2, etc, separats per s.	SELECT CONCAT_WS(', ','Nom','Telèfon','Adreça'); Retorna: 'Nom, Telèfon, Adreça'
ELT(n,s1,s2,...)	Si n = 1, retorna s1. Si n = 2, retorna s2, etc	SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo'); Retorna: 'foo'

Taula 2.2 (continuació)

Funció	Descripció	Exemple
EXPORT_SET (bits,on,off [,separador[,nombre_de_bits]])	Retorna una cadena en què, per cada bit del valor bits, pot obtenir una cadena on i per cada bit reassignat obté una cadena off. Els bits en bits s'examinen de dreta a esquerra (de bits més petits a més grans). Les cadenes s'afegeixen al resultat d'esquerra a dreta, separats per la cadena separador (',' per defecte). El nombre de bits examinats s'obté pel nombre_de_bits (por defecte 64).	SELECT EXPORT_SET (5,'Y','N',',',4); Retorna: 'Y,N,Y,N'
FIELD(s,s1,s2,s3,...)	Retorna l'índex (posició) del primer argument en els arguments següents s1, s2, etc.	SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo'); Retorna: 2
FIND_IN_SET(s1,s2)	Retorna la posició en què es troba s1 dins de s2.	SELECT FIND_IN_SET('b','a,b,c,d'); Retorna: 2
FORMAT(x,d)	Retorna el nombre x, com un *string* formatat amb la plantilla següent '#,###,###.##' i amb d decimals.	SELECT FORMAT(12332.1,4); Retorna: '12,332.1000'
HEX(x)	Retorna la representació hexadecimal d'un nombre decimal o d'un *string*.	SELECT HEX('abc'); Retorna: 616263
INSERT(s1,p,l,s2)	Insereix s2 en la posició p i longitud l de s1	SELECT INSERT('Hola', 5, 3, 'món'); Retorna: 'Holamón'
INSTR(s1,s2)	Retorna la primera posició en què es troba s2 dins de s1.	SELECT INSTR('peu de pàgina', 'pàgina'); Retorna: 8
LCASE(s)	Sinònim de LOWER(s).	
LEFT(s,l)	Retorna els l caràcters de més a l'esquerra de s.	SELECT LEFT('Hola món',4); Retorna: 'Hola'
LENGTH(s)	Retorna la longitud de s en bytes.	SELECT LENGTH('Hola'); Retorna: 4
LIKE	Operador de comparació entre expressions, utilitzat, sobretot en clàusules WHERE.	SELECT 'David' LIKE 'Dav%'; Retorna cert (1)
LOAD_FILE(s)	Obre el fitxer anomenat s i en retorna el contingut.	
LOCATE(s1,s2,[p])	Retorna la posició de la primera ocurrència de s1 dins de s2, a partir de la posició p o de la primera posició, si no s'especifica p.	SELECT LOCATE('bar', 'foobarbar'); Retorna 4
LOWER(s)	Retorna s en minúscules.	SELECT LOWER('Hola Món'); Retorna: 'hola món'
LPAD(s1,n,s2)	Retorna la cadena s1, alineada a l'esquerra amb la cadena s2 a una longitud de n caràcters.	SELECT LPAD('hi',4,'??'); Retorna: '??hi'
LTRIM(s)	Esborra els caràcters blancs que hi ha a l'esquerra de l'*string*.	SELECT LTRIM(' hola'); Retorna: 'hola'
MAKE_SET(bits,s1,s2,...)	Retorna el conjunt de *strings* que seleccionen els índexs indicats en bits	SELECT MAKE_SET(14,'hello','nice','world'); Retorna: 'hello,world'
MATCH	Operador de cerca.	

TAULA 2.2 (continuació)

Funció	Descripció	Exemple
MID(s,p,l)	Retorna la subcadena de s, de longitud l, que comença en la posició p. És sinònim de SUBSTRING(s,p,l).	SELECT MID('Hola Món',1,4); Retorna: 'Hola'
NOT LIKE	Negació de l'operador LIKE.	
NOT REGEXP	Negació de l'operador REGEXP.	
OCTET_LENGTH(s)	Sinònim de LENGTH(s).	
ORD(s)	Retorna el codi acsii del primer caràcter de s.	
POSITION(s1 IN s2)	Sinònim de LOCATE(s1,s2).	
QUOTE(s)	Retorna la cadena s formatada amb els caràcters especials per tal de poder ser utilitzada correctament com a codi SQL.	SELECT QUOTE('Don\'t!'); Retorna: 'Don\'t!'
REGEXP	Operador per emprar expressions regulars.	
REPEAT(s,n)	Retorna la cadena s repetida n vegades.	SELECT REPEAT('MySQL', 3); Retorna: 'MySQLMySQLMySQL'
REPLACE(s,s1,s2)	Canvia les ocurrències de s1 per s2 dins de s.	SELECT REPLACE('www.mysql.com', 'w', 'Ww'); Retorna: 'WwWwWw.mysql.com'
REVERSE(s)	Retorna la cadena s amb l'ordre invers dels caràcters.	SELECT REVERSE('abc'); Retorna: 'cba'
RIGHT(s,l)	Retorna els l caràcters de més a la dreta de s.	SELECT RIGHT('Hola món',3); Retorna: 'món'
RLIKE	Sinònim de REGEXP	
RPAD(s1,n,s2)	Retorna la cadena s, alineada a la dreta amb la cadena s2 amb longitud n.	SELECT RPAD('hi',5,'?'); Retorna: 'hi???'
RTRIM(s)	Retorna s sense espais en blanc a la dreta.	SELECT RIGHT('Hola '); Retorna: 'Hola'
SOUNDEX(s)	Retorna una cadena soudex de s.	
SOUNDS LIKE	Operador equivalent a SOUNDEX(exp1) = SOUNDEX(exp2).	
SPACE(n)	Retorna una cadena de n espais en blanc.	SELECT SPACE(6); Retorna: ' '
STRCMP(s1,s2)	Compara dues cadenes.	
SUBSTR(s,p[,l])	Retorna la subcadena de s de longitud l a partir de la posició p. També admet les sintaxis següents: SUBSTR(str,pos), SUBSTR(str FROM pos), SUBSTR(str,pos,len), SUBSTR(str FROM pos FOR len). Sinònim també de SUBSTRING.	SELECT SUBSTR('Hola Món',1,4); Retorna: 'Hola'
SUBSTRING_INDEX(s,d,n)	Retorna la subcadena de la cadena s abans de n ocurrències del delimitador d.	SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2); Retorna: 'www.mysql'

Taula 2.2 (continuació)

Funció	Descripció	Exemple
SUBSTRING(s,p[,l])	Retorna la subcadena de s de longitud l a partir de la posició p. També admet les sintaxis següents: SUBSTRING(str,pos) , SUBSTRING(str FROM pos), SUBSTRING(str,pos,len) , SUBSTRING(str FROM pos FOR len).	SELECT SUBSTR('Hola Món',1,4); Retorna: 'Hola'
TRIM([{BOTH o LEADING o TRAILING} [remstr] FROM] s)	S'utilitza per eliminar els espais en blanc de l'inici i el final de la cadena s. Retorna la cadena s amb tots els prefixos i/o sufixos remstr eliminats. Per defecte, remstr és l'espai en blanc.	SELECT TRIM(' hola '); Retorna: 'hola'
UCASE(s)	Sinònim d'UPPER(s).	
UNHEX(s)	Converteix els codis hexadecimals a caràcters. És la funció contrària a HEX(s).	SELECT UNHEX('4D7953514C'); Retorna: 'MySQL'
UPPER(s)	Converteix s a majúscules.	SELECT UPPER('Hola Món'); Retorna: 'HOLA MÓN'

En negreta s'han destacat les funcions més utilitzades a l'hora de manipular expressions numèriques.

2.1.3 Funcions de gestió de dates

La taula 2.3 ens presenta algunes de les funcions per a la gestió de dades DATE proporcionades per l'SGBD MySQL.

Taula 2.3. Funcions per a la gestió de dades DATE proporcionades per l'SGBD MySQL

Funció	Descripció	Exemple
ADDDATE(d,n)	Suma n dies a la data d	SELECT DATE_ADD('1998-01-02',31); Retorna: '1998-02-02'
ADDTIME(t1,t2)	Suma t1 i t2	SELECT ADDTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002'); Retorna: '2008-01-02 01:01:01.000001'
CONVERT_TZ(t,z1,z2)	Converteix t d'una zona a una altra	SELECT CONVERT_TZ('2004-01-01 12:00:00','GMT','MET'); Retorna: '2004-01-01 13:00:00'
CURDATE()	Retorna la data actual en format 'AAAA-MM-DD'	SELECT CURDATE();
CURRENT_DATE(), CURRENT_DATE	Sinònims de CURDATE()	
CURRENT_TIME(), CURRENT_TIME	Sinònims de CURTIME()	
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Sinònims de NOW()	
CURTIME()	Retorna l'hora actual en format 'HH:MM:SS'	SELECT CURTIME();

TAULA 2.3 (continuació)

Funció	Descripció	Exemple
DATE_FORMAT(d,f)	Retorna la data d en format f, en què f està codificat seguint la taula 2.4	
DATE_SUB(d,n)	De manera similar a DATE_ADD, en aquest cas, resta de la data el valor del segon paràmetre	
DATE(d)	Obté la data d'una dada que conté data-hora	SELECT DATE('2003-12-31 01:02:03'); Retorna: '2003-12-31'
DATEDIFF(d1,d2)	Resta dues dates	
DAY(d)	Sinònim de DAYOFMONTH()	SELECT DAY('2007-02-03'); Retorna: 3
DAYNAME(d)	Retorna el nom del dia de la setmana	SELECT DAYNAME('2007-02-03'); Retorna: 'Saturday'
DAYOFMONTH(d)	Retorna el nom del dia del mes (0-31)	SELECT DAYOFMONTH('2007-02-03'); Retorna: 3
DAYOFWEEK(d)	Retorna el número d'ordre del dia de la setmana a què correspon d (1 = Sunday, 2 = Monday, ..., 7 = Saturday)	SELECT DAYOFWEEK('2007-02-03'); Retorna: 7
DAYOFYEAR(d)	Retorna el dia de l'any	SELECT DAYOFYEAR('2007-02-03'); Retorna: 34
EXTRACT(u FROM d)	Extreu part de la data	SELECT EXTRACT(YEAR FROM '2009-07-02'); Retorna: 2009
FROM_DAYS(n)	Atès un nombre n el converteix a tipus data	SELECT FROM_DAYS(730669); Retorna: '2007-07-03'
FROM_UNIXTIME()	Formata les dates-hores de UNIX com una data	
GET_FORMAT({DATE o TIME o DATETIME}, {'EUR' o 'USA' o 'JIS' o 'ISO' o 'INTERNAL'})	Retorna el format de data vàlid sol·licitat. Se sol combinar amb DATE_FORMAT	SELECT DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR')); Retorna: '03.10.2003'
HOUR(d)	Extreu l'hora	SELECT HOUR('10:05:03'); Retorna: 10
LAST_DAY(d)	Retorna l'últim dia del mes de la data d	SELECT LAST_DAY('2003-02-05'); Retorna: '2003-02-28'
LOCALTIME(), LOCALTIME	Sinònim de NOW()	
LOCALTIMESTAMP, LOCALTIMESTAMP()	Sinònim de NOW()	
MAKEDATE(d, dia)	Crea una data a partir de l'any i el dia de l'any	SELECT MAKEDATE(2011,32); Retorna: '2011-02-01'
MAKETIME MAKETIME(h,m,s)	Crea una hora a partir de les hores, minuts i segons donats	SELECT MAKETIME(12,15,30); Retorna '12:15:30'
MICROSECOND(d)	Retorna els microsegons	
MINUTE(d)	Retorna els minuts de d	SELECT MINUTE('2008-02-03 10:05:03'); Retorna: 5
MONTH(d)	Retorna el mes de d	SELECT MONTH('2008-02-03'); Retorna: 2
MONTHNAME(d)	Retorna el nom del mes	SELECT MONTH('2008-02-03'); Retorna:'February'

Taula 2.3 (continuació)

Funció	Descripció	Exemple
NOW()	Retorna la data i hora actuals	SELECT NOW();
PERIOD_ADD(p,n)	Afegeix n mesos al període p	SELECT PERIOD_ADD(200801,2); Retorna: 200803
PERIOD_DIFF(p1,p2)	Retorna el nombre de mesos entre p1 i p2	
QUARTER(d)	Retorna el trimestre de d	SELECT QUARTER('2008-04-01'); Retorna: 2
SEC_TO_TIME(n)	Converteix n segons al format d'hores 'HH:MM:SS'	SELECT SEC_TO_TIME(2378); Retorna: '00:39:38'
SECOND(t)	Retorna el nombre de segons de l'hora determinada t (0-59)	SELECT SEC_TO_TIME(2378); Retorna: '00:39:38'
STR_TO_DATE(s,f)	Converteix una cadena s a una data en format f	SELECT STR_TO_DATE('May 1, 2013','%M %d,%Y'); Retorna: '2013-05-01'
SUBDATE(d,n)	Sinònim de DATE_SUB()	
SUBTIME(d1,d2)	Resta d1 i d2	
SYSDATE()	Retorna el dia i hora en el moment d'executar la funció	SELECT SYSDATE();
TIME_FORMAT(d,f)	Formata d segons el format f, de manera similar a DATE_FORMAT	
TIME_TO_SEC(t)	Retorna t convertit a segons	
TIME(d)	Extreu l'hora de d	SELECT TIME('2003-12-31 01:02:03'); Retorna: '01:02:03'
TIMEDIFF(t1,t2)	Resta t1 - t1	
TIMESTAMP(d)	Retorna la data-hora d'una data d	SELECT TIMESTAMP('2003-12-31'); Retorna: '2003-12-31 00:00:00'
TIMESTAMPADD(n,i,d)	Suma n intervals i a una data d	SELECT TIMESTAMPADD(WEEK,1,'2003-01-02'); Retorna: '2003-01-09'
TIMESTAMPDIFF(i,d1,d2)	Resta d1 i d2, i obté el resultat segons la unitat i	SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01'); Retorna: 3
TO_DAYS(d)	Retorna d convertit a dies	SELECT TO_DAYS('2007-10-07'); Retorna: 733321
TO_SECONDS(d)	Retorna d convertit a segons	SELECT TO_SECONDS('2009-11-29'); Retorna: 63426672000
UNIX_TIMESTAMP()	Retorna l'hora en format UNIX	
UTC_DATE()	Retorna la data UTC	
UTC_TIME()	Retorna l'hora UTC	
UTC_TIMESTAMP()	Retorna la data i hora UTC	
WEEK(d)	Retorna el nombre de setmana	SELECT WEEK('2008-02-20'); Retorna: 7
WEEKDAY(d)	Retorna l'índex de dia de la setmana (0 = Monday, 1 = Tuesday, ... 6 = Sunday)	SELECT WEEKDAY('2007-11-06'); Retorna: 1

Taula 2.3 (continuació)

Funció	Descripció	Exemple
WEEKOFYEAR(d)	Retorna la setmana del calendari de d (0-53)	SELECT WEEKOFYEAR('2008-02-20'); Retorna: 8
YEAR(d)	Retorna l'any	SELECT YEAR('1987-01-01'); Retorna: 1987
YEARWEEK(d)	Retorna l'any i setmana de d	SELECT YEARWEEK('1987-01-01'); Retorna: 198653

En negreta s'han destacat les funcions més utilitzades a l'hora de manipular expressions numèriques.

Per tal d'especificar les dates i hores en diversos formats, cal seguir les notacions descrites en la taula 2.4.

Taula 2.4. Notacions per formatar les dates en MySQL

Especificador	Descripció
%a	Dia de la setmana abreujat (Sun..Sat)
%b	Mes abreujat (Jan..Dec)
%c	Mes, numèric (0..12)
%D	Dia del mes amb sufixe anglès (0th, 1st, 2nd, 3rd...)
%d	Dia del mes numèric (00..31)
%e	Dia del mes numèric (0..31)
%f	Microsegons (000000..999999)
%H	Hora (00..23)
%h	Hora (01..12)
%I	Hora (01..12)
%i	Minuts, numèric (00..59)
%j	Dia de l'any (001..366)
%k	Hora (0..23)
%l	Hora (1..12)
%M	Nom del mes (January..December)
%m	Mes, numèric (00..12)
%p	AM o PM
%r	Hora, 12 hores (hh:mm:ss seguit de AM o PM)
%S	Segons (00..59)
%s	Segons (00..59)
%T	Hora, 24 hores (hh:mm:ss)
%U	Setmana (00..53), en què diumenge és el primer dia de la setmana
%u	Setmana (00..53), en què dilluns és el primer dia de la setmana
%V	Setmana (01..53), en què diumenge és el primer dia de la setmana; utilitzat amb %X
%v	Setmana (01..53), en què dilluns és el primer dia de la setmana; utilitzat amb %x

TAULA 2.4 (continuació)

Especificador	Descripció
%W	Nom del dia de la setmana (Sunday..Saturday)
%w	Dia de la setmana (0=Sunday..6=Saturday)
%X	Any per a la setmana en què diumenge és el primer dia de la setmana, numèric, quatre dígits; usat amb %V
%x	Any per a la setmana, en què dilluns és el primer dia de la setmana, numèric, quatre dígits; usat amb %v
%Y	Any, numèric, quatre dígits
%y	Any, numèric (dos dígits)

Així, per exemple, es pot mostrar la data i/o l'hora utilitzant expressions com ara les següents, obtenint els resultats indicats:

```
1  SELECT DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');
2  Retorna: 'Saturday October 1997'
3
4  SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H:%i:%s');
5  Retorna: '22:23:00'
6
7  SELECT DATE_FORMAT('1997-10-04 22:23:00', '%D %y %a %d %m %b %j');
8  Retorna: '4th 97 Sat 04 10 Oct 277'
9
10 SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H %k %I %r %T %S %w');
11 Retorna:  '22 22 10 10:23:00 PM 22:23:00 00 6'
12
13 SELECT DATE_FORMAT('1999-01-01', '%X %V');
14 Retorna: '1998 52'
```

2.1.4 Funcions de control de flux

Tot i que el llenguatge SQL no és un llenguatge de programació d'aplicacions estrictament, sí que, en algunes operacions sobre la base de dades és últim realitzar una operació o considerar uns valors o uns altres en funció d'un estat inicial o de partida. Per aquest motiu MySQL ofereix la possibilitat d'incloure, dintre de la sintaxi de les sentències SQL, uns operadors i unes funcions que permetin realitzar accions diferents en funció d'uns estats.

Operador CASE

L'operador `CASE` permet obtenir diversos valors en funció d'unes condicions o comparacions prèvies. Hi ha dues maneres d'utilitzar l'operador `CASE`:

- `CASE` valor `WHEN` [valor_comparació] `THEN` resultat [`WHEN` [valor_comparació] `THEN` resultat ...] [`ELSE` resultat] **END**
- `CASE WHEN` [condició] `THEN` resultat [`WHEN` [condició] `THEN` resultat ...] [`ELSE` resultat] `END`

Aquestes dues formes es poden utilitzar com en els exemples:

```
SELECT CASE 1 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'more' END;
Retorna: 'one'

SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
Retorna: 'true'
```

Construcció IF

La construcció `IF` permet, de manera similar a `CASE`, obtenir valors diferenciats en funció d'una condició.

La sintaxi de la construcció `IF` és la següent:

```
IF (condició, expressió1, expressió2)
```

I funciona de la manera següent: si la condició és certa, es retorna l'expressió1, altrament, l'expressió2. Vegem l'exemple següent:

```
SELECT IF(1<2,'si','no');
Retorna: 'si'
```

Funció IFNULL

La funció `IFNULL` (expressió) és una funció que retorna el valor de l'expressió si aquest no és nul, i zero en cas que s'avalués com a nul.

Aquesta funció es pot utilitzar per evitar que expressions complexes en què hi ha valors nuls s'avaluïn globalment com a nul, si és possible assignar-los el valor de zero en cas de nul. Fixeu-vos-hi amb l'exemple següent:

En l'esquema *empresa*, es volen mostrar els empleats (codi, cognom, salari i comissió) acompanyats d'una columna que contingui la suma del salari i la comissió. En un principi, consideraríem la sentència següent:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari",
       comissio as "Comissió", salari+comissio as "Suma"
from emp;
```

I se n'obtindrà el resultat següent:

```
Codi    Empleat    Salari    Comissió     Suma
------- ---------- --------- ------------ ------------------
7369    SÁNCHEZ    104000
7499    ARROYO     208000    39000        247000
7521    SALA       162500    65000        227500
7566    JIMÉNEZ    386750
7654    MARTÍN     162500    182000       344500
7698    NEGRO      370500
7782    CEREZO     318500
7788    GIL        390000
7839    REY        650000
7844    TOVAR      195000    0            195000
7876    ALONSO     143000
```

```
7900    JIMENO     123500
7902    FERNÁNDEZ  390000
7934    MUÑOZ      169000

14 rows selected
```

El resultat no és el volgut, ja que tots els empleats tenen salari però no tots tenen comissió assignada (per als qui no tenen comissió hi ha un valor `NULL` en la columna corresponent), i aquest fet motiva que la suma només es calculi per als qui tenen comissió. Possiblement esperaríem que l'SGBD considerés que un `NULL` en la comissió dels empleats és equivalent a zero. L'hi hem d'especificar, tal com es veu en la sentència següent:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari",
       comissio as "Comissió", salari+ifnull(comissio,0) as "Suma"
from emp;
```

Ara sí que el resultat és el volgut:

```
Codi   Empleat    Salari    Comissió     Suma
______ _________ _________ ____________ ____________________

7369   SÁNCHEZ    104000                 104000
7499   ARROYO     208000    39000        247000
7521   SALA       162500    65000        227500
7566   JIMÉNEZ    386750                 386750
7654   MARTÍN     162500    182000       344500
7698   NEGRO      370500                 370500
7782   CEREZO     318500                 318500
7788   GIL        390000                 390000
7839   REY        650000                 650000
7844   TOVAR      195000    0            195000
7876   ALONSO     143000                 143000
7900   JIMENO     123500                 123500
7902   FERNÁNDEZ  390000                 390000
7934   MUÑOZ      169000                 169000

14 rows selected
```

Funció NULLIF

La sintaxi que utilitza la funció **NULLIF** és la següent:

```
NULLIF(expressió1,expressió2)
```

Si l'expressió1 i l'expressió2 s'avaluen com a iguals la funció retorna NULL; altrament, retorna l'expressió1.

Vegem-ne els exemples següents:

```
SELECT NULLIF(1,1);
Retorna: NULL

SELECT NULLIF(1,2);
Retorna:  1
```

Altres funcions de MySQL

MySQL proporciona altres funcions pròpies que enriqueixen el llenguatge. Per exemple, en la versió del programari 5.5 es proporcionen funcions per accedir a codi XML i obtenir-ne dades, suportant el llenguatge *XPath 1.0* d'accés a dades XML.

També proporciona operadors que permeten treballar a nivell d'operacions de bits (inversió de bits, operacions de `AND`, `OR` o `XOR` o desplaçaments de bits, per exemple).

MySQL disposa de funcions per comprimir (`ENCODE()`) i descomprimir (`DECODE()`) dades, i també per encriptar-ne (`ENCRYPT()`) i desencriptar-ne (`DES_DECRYPT()`).

Evidentment, MySQL també ofereix tot un seguit d'operacions diverses d'administració de l'SGBD que permeten accedir al diccionari de dades.

Per a totes aquestes altres operacions caldrà consultar la guia de referència del llenguatge que es pot trobar al lloc web oficial de MySQL.

2.2 Classificació de files. Clàusula ORDER BY

La clàusula `select` permet decidir quines columnes se seleccionaran del producte cartesià de les taules especificades en la clàusula `from`, i la clàusula `where` en filtra les files corresponents. No es pot assegurar, però, l'ordre en què l'SGBD donarà el resultat.

> La clàusula `order by` permet especificar el criteri de classificació del resultat de la consulta.

Aquesta clàusula s'afegeix darrere de la clàusula `where` si n'hi ha, de manera que ampliem la sintaxi de la sentència SELECT:

```
1 select <expressió/columna>, <expressió/columna>,...
2 from <taula>, <taula>,...
3 [where <condició_de_recerca>]
4 [order by <expressió/columna> [asc|desc], <expressió/columna> [asc|desc],...];
```

Com es pot veure, la clàusula `order by` permet ordenar la sortida segons diferents expressions i/o columnes, que han de ser calculables a partir dels valors de les columnes de les taules de la clàusula from encara que no apareguin en les columnes de la clàusula `select`.

Les expressions i/o columnes de la clàusula `order by` que apareixen en la clàusula select es poden referenciar pel nombre ordinal de la posició que ocupen en la clàusula select en lloc d'escriure'n el nom.

El criteri d'ordenació depèn del tipus de dada de l'expressió o columna i, per tant, podrà ser numèric o lexicogràfic.

Quan hi ha més d'un criteri d'ordenació (diverses expressions i/o columnes), es classifiquen d'esquerra a dreta.

La seqüència d'ordenació per defecte és ascendent per a cada criteri. Es pot, però, especificar que la seqüència d'ordenació per a un criteri sigui descendent amb la partícula desc a continuació del criteri corresponent. També es pot especificar la partícula asc per indicar una seqüència d'ordenació ascendent, però és innecessari perquè és la seqüència d'ordenació per defecte.

Exemple de classificació de resultats segons diverses columnes

En l'esquema *empresa*, es volen mostrar els empleats ordenats de manera ascendent pel seu salari mensual, i ordenats pel cognom quan tinguin el mateix salari.

La instrucció per assolir l'objectiu és aquesta:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari"
from emp
order by salari, cognom;
```

En cas que hi hagi criteris d'ordenació que també apareixen en la clàusula select i tinguin un àlies definit, es pot utilitzar aquest àlies en la clàusula order by. Així, doncs, tindríem el següent:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari"
from emp
order by "Salari", "Empleat";
```

I, com hem dit més amunt, també podríem utilitzar l'ordinal:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari"
from emp
order by 3, 2;
```

Exemple de classificació de resultats segons expressions

En l'esquema *empresa*, es volen mostrar els empleats amb el seu salari i comissió ordenats, de manera descendent, pel sou total mensual (salari + comissió).

La instrucció per assolir l'objectiu és aquesta:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari",
   comissio as "Comissió"
from emp
order by salari+IFNULL(comissio,0) desc;
```

Fixeu-vos que si no utilitzem la funció IFNULL també apareixen tots els empleats, però tots els que no tenen comissió assignada apareixen agrupats a l'inici, ja que el valor NULL es considera superior a tots els valors i el resultat de la suma salari+comissio és NULL per a les files que tenen NULL en la columna comissio.

2.3 Exclusió de files repetides. Opció DISTINCT

La clàusula `select` permet decidir quines columnes se seleccionaran del producte cartesià de les taules especificades en la clàusula `from`, i la clàusula `where` en filtra les files corresponents. El resultat, però, pot tenir files repetides, per a les quals pot interessar tenir només un exemplar.

> L'opció `distinct` acompanyant la clàusula `select` permet especificar que es vol un únic exemplar per a les files repetides.

La sintaxi és la següent:

```
select [distinct] <expressió/columna>, <expressió/columna>,...
from <taula>, <taula>,...
[where <condició_de_recerca>]
[order by <expressió/columna> [asc|desc], <expressió/columna> [asc|desc],...];
```

La utilització de l'opció `distinct` implica que l'SGBD executi obligatòriament una `order by` sobre totes les columnes seleccionades (encara que no s'especifiqui la clàusula `order by`), fet que implica un cost d'execució addicional. Per tant, l'opció `distinct` s'hauria d'utilitzar en cas que hi pugui haver files repetides i interessi un únic exemplar, i en estar segur s'hauria d'evitar que no hi pot haver files repetides.

Exemple de la necessitat d'utilitzar l'opció distinct

Com exemple en què cal utilitzar l'opció `distinct` vegem com es mostren, en l'esquema *empresa*, els departaments (només el codi) en els quals hi ha algun empleat.

La instrucció per assolir l'objectiu és aquesta:

```
select distinct dept_no as "Codi"
from emp;
```

Evidentment, la consulta no es pot efectuar sobre la taula dels departaments, ja que hi pot haver algun departament que no tingui cap empleat. Per aquest motiu, l'executem sobre la taula dels empleats i hem d'utilitzar l'opció distinct, ja que altrament un mateix departament apareixeria tantes vegades com empleats tingués assignats.

2.4 Agrupaments de files. Clàusules GROUP BY i HAVING

Sabent com se seleccionen files d'una taula o d'un producte cartesià de taules (clàusula `where`) i com quedar-nos amb les columnes interessants (clàusula `select`), cal veure com agrupar les files seleccionades i com filtrar per condicions sobre els grups.

La clàusula `group by` permet agrupar les files resultat de les clàusules `select`, `from` i `where` segons una o més de les columnes seleccionades. La clàusula `having` permet especificar condicions de filtratge sobre els grups assolits per la clàusula `group by`.

Les clàusules `group by` i `having` s'afegeixen darrere la clàusula `where` (si n'hi ha) i abans de la clàusula `order by` (si n'hi ha), de manera que ampliem la sintaxi de la sentència SELECT:

```
select [distinct] <expressió/columna>, <expressió/columna>,...
from <taula>, <taula>,...
[where <condició_de_recerca>]
[group by <àlies/columna>, <àlies/columna>,...]
[having <condició_sobre_grups>]
[order by <expressió/columna> [asc|desc], <expressió/columna> [asc|desc],...];
```

Recordeu que els elements que es posen entre corxets (`[]`) indiquen opcionalitat.

La taula 2.5 ens mostra les funcions d'agrupament més importants que es poden utilitzar en les sentències SELECT de selecció de conjunts.

Taula 2.5. Funcions d'agrupament més importants que es poden utilitzar en les sentències SELECT d'agrupament de files

Funció	Descripció	Exemples
AVG (n)	Retorna el valor mitjà de la columna n ignorant els valors nuls.	AVG (salari) retorna el salari mitjà de tots els empleats seleccionats que tenen salari (els nuls s'ignoren).
COUNT ([* o expr])	Retorna el nombre de vegades que expr avalua alguna dada amb valor no nul. L'opció * comptabilitza totes les files seleccionades.	COUNT (dept_no) (sobre la taula d'empleats) compta quants empleats estan assignats a algun departament.
MAX (expr)	Retorna el valor màxim de expr.	MAX (salari) retorna el salari més alt.
MIN (expr)	Retorna el valor mínim de expr.	MIN (salari) retorna el salari més baix.
STDDEV (expr)	Retorna la desviació típica de expr sense tenir en compte els valors nuls.	STDDEV (salari) retorna la desviació típica dels salaris.
SUM (expr)	Retorna la suma dels valors de expr sense tenir en compte els valors nuls.	SUM (salari) retorna la suma de tots els salaris.
VARIANCE (expr)	Retorna la variància de expr sense tenir en compte els valors nuls.	VARIANCE (salari) retorna la variància dels salaris.

L'expressió sobre la qual es calculen les funcions d'agrupament pot anar precedida de l'opció `distinct` per indicar que s'avaluï sobre els valors diferents de l'expressió, o de l'opció `all` per indicar que s'avaluï sobre tots els valors de l'expressió. El valor per defecte és `all`.

Una sentència SELECT és una sentència de selecció de conjunts quan apareix la clàusula `group by` o la clàusula `having` o una funció d'agrupament; és a dir, una sentència SELECT pot ser sentència de selecció de conjunts encara que no hi hagi clàusula `group by`, cas en què es considera que hi ha un únic conjunt format per totes les files seleccionades.

Les columnes o expressions que no són funcions d'agrupament i que apareixen en una clàusula `select` d'una sentència `SELECT` de selecció de conjunts han d'aparèixer obligatòriament en la clàusula `group by` de la sentència. Ara bé, no totes les columnes i expressions de la clàusula `group by` han d'aparèixer necessàriament en la clàusula `select`.

Exemple d'utilització de la funció count() sobre tota la consulta

En l'esquema *empresa*, es vol comptar quants empleats hi ha.

La instrucció per assolir l'objectiu és aquesta:

```
select count(*) as "Quants empleats" from emp;
```

Aquesta sentència SELECT és una sentència de selecció de conjunts malgrat que no aparegui la clàusula group by. En aquest cas, l'SGBD ha agrupat totes les files en un únic conjunt per tal de poder-les comptar.

Exemple d'utilització de l'opció distinct en una funció d'agrupament

En l'esquema *empresa*, es vol comptar quants oficis diferents hi ha.

La instrucció per assolir l'objectiu és aquesta:

```
select count(distinct ofici)  as "Quants oficis" from emp;
```

En aquest cas és necessari indicar l'opció distinct, ja que altrament comptaria totes les files que tenen algun valor en la columna ofici, sense descartar els valors repetits.

Exemple d'utilització de la funció count() sobre una consulta amb grups

En l'esquema *empresa*, es vol mostrar quants empleats hi ha de cada ofici.

La instrucció per assolir l'objectiu és aquesta:

```
select ofici as "Ofici", count(*) as "Quants empleats"
from emp
group by ofici;
```

El resultat obtingut és aquest:

```
Ofici      Quants empleats
---------- ---------------
EMPLEAT    4
VENEDOR    4
ANALISTA   2
PRESIDENT  1
DIRECTOR   3

5 rows selected
```

Exemple de coexistència de les clàusules group by i order by

En l'esquema *empresa*, es volen mostrar els departaments que tenen empleats, acompanyats del salari més alt dels seus empleats i ordenats de manera ascendent pel salari màxim.

La instrucció per assolir l'objectiu és aquesta:

```
select dept_no, max(salari)
from emp
group by dept_no
order by max(salari);
```

O també:

```
select dept_no as "Codi", max(salari) as "Màxim salari"
from emp
group by dept_no
order by "Màxim salari";
```

O també:

```
select dept_no as "Codi", max(all salari) as "Màxim salari"
from emp
group by dept_no
order by "Màxim salari";
```

Exemple de coexistència de les clàusules group by i order by

En l'esquema *empresa*, es vol comptar quants empleats de cada ofici hi ha en cada departament, i veure el resultats ordenats per departament de manera ascendent i per nombre d'empleats de manera descendent.

La instrucció per assolir l'objectiu és aquesta:

```
select dept_no as "Codi", ofici as "Ofici",
    count(*) as "Quants empleats"
from emp
group by dept_no, ofici
order by dept_no, 3 desc;
```

Exemple de coexistència de les clàusules group by i where

En l'esquema *empresa*, es vol mostrar quants empleats de cada ofici hi ha en el departament 20.

La instrucció per assolir l'objectiu és aquesta:

```
select ofici as "Ofici", count(*) as "Quants empleats"
from emp
where dept_no=20
group by ofici;
```

Exemple d'utilització de la clàusula having

En l'esquema *empresa*, es vol mostrar el nombre d'empleats de cada ofici que hi ha per als oficis que tenen més d'un empleat.

La instrucció per assolir l'objectiu és aquesta:

```
select ofici as "Ofici", count(*) as "Quants empleats"
from emp
group by ofici
having count(*)>1;
```

2.5 Unió, intersecció i diferència de sentències SELECT

El llenguatge SQL permet efectuar operacions sobre els resultats de les sentències SELECT per tal d'obtenir un resultat nou.

Tenim tres operacions possibles: unió, intersecció i diferència. Els conjunts que cal unir, intersecar o restar han de ser compatibles: igual quantitat de columnes i columnes compatibles –tipus de dades equivalents– dos a dos.

2.5.1 Unió de sentències SELECT

El llenguatge SQL proporciona l'operador `union` per combinar totes les files del resultat d'una sentència `SELECT` amb totes les files del resultat d'una altra sentència `SELECT`, i elimina qualsevol duplicació de files que es pogués produir en el conjunt resultant.

La sintaxi és aquesta:

```
sentència_select_sense_order_by
union
sentència_select_sense_order_by
[order by ...]
```

El resultat final mostrarà, com a títols, els corresponents a les columnes de la primera sentència `SELECT`. Així, doncs, en cas de voler assignar àlies a les columnes, només cal definir-los en la primera sentència `SELECT`.

Exemple de l'operació union entre sentències SQL

En l'esquema *sanitat*, es vol presentar el personal que treballa en cada hospital, incloent-hi el personal de la plantilla i els doctors, i mostrant l'ofici que hi exerceixen.

Una possible instrucció per assolir l'objectiu és aquesta:

```
select nom as "Hospital", 'Doctor' as "Ofici", doctor_no as "Codi
    ",
   cognom "Empleat"
from hospital, doctor
where hospital.hospital_cod=doctor.hospital_cod
**union**
select nom, funcio, empleat_no, cognom
from hospital, plantilla
where hospital.hospital_cod=plantilla.hospital_cod
order by 1,2;
```

Fixem-nos que hem assignat als doctors com a ofici la constant 'Doctor'. El resultat obtingut és aquest:

```
Hospital   Ofici      Codi       Empleat
__________ __________ __________ _______________
General    Doctor     585        Miller G.
General    Doctor     982        Cajal R.
General    Intern     6357       Karplus W.
La Paz     Doctor     386        Cabeza D.
La Paz     Doctor     398        Best K.
La Paz     Doctor     453        Galo D.
La Paz     Infermer   8422       Bocina G.
La Paz     Infermera  1009       Higueras D.
La Paz     Infermera  6065       Rivera G.
La Paz     Infermera  7379       Carlos R.
La Paz     Intern     9901       Adams C.
Provincial Doctor     435        López A.
Provincial Infermer   3106       Hernández J.
Provincial Infermera  3754       Díaz B.
San Carlos Doctor     522        Adams C.
San Carlos Doctor     607        Nico P.
San Carlos Infermera  8526       Frank H.
San Carlos Intern     1280       Amigó R.

18 rows selected
```

2.5.2 Intersecció i diferència de sentències SELECT

Altres SGBDR (no pas MySQL, en aquest cas) proporcionen operacions d'intersecció i diferència de consultes.

La intersecció consisteix a obtenir un resultat de files comú (idèntic) entre dues sentències SELECT concretes. La sintaxi més habitual per a la intersecció és:

```
sentència_select_sense_order_by
**intersect**
sentència_select_sense_order_by
[order by ...]
```

La diferència entre sentències `SELECT` consisteix a obtenir les files que es troben a la primera sentència `SELECT` que no es trobin en la segona. La sintaxi habitual és utilitzant l'operador minus:

```
sentència_select_sense_order_by
**minus**
sentència_select_sense_order_by
[order by ...]
```

2.6 Combinacions entre taules

La clàusula `from` efectua el producte cartesià de totes les taules que apareixen en la clàusula. El producte cartesià no ens interessarà gairebé mai, sinó únicament un subconjunt d'aquest.

Els tipus de subconjunts que ens poden interessar coincideixen amb els resultat de les combinacions *join*, *equi-join*, *natural-join* i *outer-join*.

Operacions per combinar taules

Donades dues taules R i S, es defineix el *join* de R segons l'atribut A, i de S segons l'atribut Z, i s'escriu R[AZ]S com el subconjunt de files del producte cartesià R S que verifiquen A Z en què és qualsevol dels operadors relacionals (>,>=,<,<=,==,!=).

L'*equi-join* és un *join* en què l'operador és la igualtat. S'escriu R[A=Z]S.

El *natural-join* és un *equi-join* en què l'atribut per al qual s'executa la combinació només apareix una vegada en el resultat. S'escriu R[A*Z]S.

La notació R*S indica el *natural-join* per a tots els atributs del mateix nom en totes dues relacions.

De vegades, cal tenir el resultat de l'*equi-join* ampliat amb totes les files d'una de les relacions que no tenen tuple corresponent en l'altra relació. Ens trobem davant un *outer-join* i tenim dues possibilitats (left o right) segons on es trobi (esquerra o dreta) la taula per la qual han d'aparèixer totes les files encara que no tinguin correspondència en l'altra taula.

Donades dues relacions R i S, es defineix el *left-outer-join* de R segons l'atribut A, i de S segons l'atribut Z, i s'escriu per R[A=Z]S, com el subconjunt de files del producte cartesià R S que verifiquen A = Z (resultat de R[A=Z]S) més les files de R que no tenen, per a l'atribut

A, correspondència amb cap tuple de S segons l'atribut Z, les quals presenten valors NULL en els atributs provinents de S.

Donades dues relacions R i S, es defineix el *right-outer-join* de R segons l'atribut A, i de S segons l'atribut Z, i s'escriu R[A=Z]S, com el subconjunt de files del producte cartesià R S que verifiquen A = Z (resultat de R[A=Z]S) més les files de S que no tenen, per a l'atribut Z, correspondència amb cap tuple de R segons l'atribut A, les quals presenten valors NULL en els atributs provinents de R.

També podem considerar el *full-outer-join* de R segons l'atribut A, i de S segons l'atribut Z, i s'escriu per R[A=Z]S, com la unió d'un *right-outer-join* i d'un *left-outer-join*. Recordem que la unió de conjunts no té en compte les files repetides. És a dir, amb un *full-outer-join* aconseguiríem tenir totes les files de totes dues taules: les files que tenen correspondència per als atributs de la combinació i les files que no hi tenen correspondència.

Actualment, tenim diverses maneres d'efectuar combinacions entre taules, producte de l'evolució dels estàndards SQL i dels diversos SGBD comercials existents: les combinacions segons la norma SQL-87 i les combinacions segons la norma SQL-92.

2.6.1 Combinacions entre taules segons la norma SQL-87 (SQL-ISO)

El llenguatge SQL-86, ratificat per l'ISO el 1987, establia que els diferents tipus de combinacions es podien assolir afegint, en la clàusula `where`, els filtres corresponents a les combinacions entre les columnes de les taules que s'han de combinar.

Exemple de combinació entre dues taules segons l'SQL-87

En l'esquema *empresa*, es volen mostrar els empleats (codi i cognom) juntament amb el codi i nom del departament al qual pertanyen.

La instrucció per assolir l'objectiu és aquesta:

```
select emp.emp_no as "Codi empleat", emp.cognom as "Empleat",
   emp.dept_no as "Codi departament", dnom as "Descripció"
from emp, dept
where emp.dept_no = dept.dept_no;
```

Fixem-nos que l'accés a la taula DEPT és necessari per aconseguir el nom del departament. Tinguem en compte, també, que, com que el camp deptno de la taula EMP no permet valors NULL, tots els empleats tindran valor en aquest camp i, a causa de la integritat referencial, no poden tenir un valor que no sigui en la taula DEPT. Així, doncs, una vegada executat el filtre de la clàusula where*, totes les files de la taula EMP estaran combinades amb una fila de la taula DEPT.

El resultat obtingut és aquest:

```
Codi empleat       Empleat    Codi departament       Descripció
_____________________ _____________ _____________________________
    _________________

7369               SANCHEZ    20                     INVESTIGACIÓ

7499               ARROYO     30                     VENDES
7521               SALA       30                     VENDES
7566               JIMENEZ    20                     INVESTIGACIÓ

7654               MARTIN     30                     VENDES
```

```
7698               NEGRO      30                    VENDES
7782               CEREZO     10
    COMPTABILITAT
7788               GIL        20                    INVESTIGACIÓ

7839               REY        10
    COMPTABILITAT
7844               TOVAR      30                    VENDES
7876               ALONSO     20                    INVESTIGACIÓ

7900               JIMENO     30                    VENDES
7902               FERNANDEZ  20                    INVESTIGACIÓ

7934               MUÑOZ      10
    COMPTABILITAT

14 rows selected
```

Aplicant els filtres adequats en la clàusula **where**, aconseguim implementar el *join*, l'*equi-join* i el *natural-join*, però no arribem a poder implementar els *outer-join*, ja que el producte cartesià no pot "inventar" files amb valors nuls.

L'any 1987, l'estàndard SQL-ISO va proposar (potser pressionat per Oracle, que ja tenia implementada la solució) la utilització d'una marca en la condició de combinació entre les columnes de les taules R i S que calgués combinar que indiqués la necessitat de fer aparèixer totes les files d'una taula (per exemple R), malgrat que per a la columna de combinació no hi hagués correspondència en l'altra taula (S), fent aparèixer valors nuls en les columnes de la taula S indicades en la clàusula select. La marca adoptada va ser el símbol (+) enganxat a la dreta de la taula (S) per a la qual cal generar valors nuls.

És a dir, un *left-outer-join* de la taula R amb la taula S segons les columnes A (taula R) i Z (taula S) respectives, que s'escriuria com a **R[A=Z]S**, es convertiria en SQL en una condició com la següent:

```
where R.A = S.Z(+)
```

De la mateixa manera, un *right-outer-join* de la taula R amb la taula S segons les columnes A (taula R) i Z (taula S) respectives, que en àlgebra relacional s'escriuria com a **R[A=Z]S**, es convertiria en SQL en una condició com la següent:

```
where R.A(+) = S.Z
```

L'estàndard SQL-ISO no proporciona cap instrucció específica per assolir un *full-outer-join* entre dues taules i **no** és permès escriure el següent:

```
where R.A(+) = S.Z(+)
```

La solució en SQL-ISO per aconseguir un *full-outer-join* passa per una operació union entre una sentència SELECT amb el *left-outer-join* i una sentència SELECT amb el *right-outer-join*.

MySQL no suporta aquest estàndard d'implementació de les *left-outer-join*, *right-outer-join* i *full-outer-join*.

2.6.2 Combinacions entre taules segons la norma SQL-92

No tots els SGBD van seguir la modalitat de la marca (+) per implementar les dues variants d'*outer-join*. I és comprensible, ja que hi ha una manera més entenedora d'explicar el perquè de les combinacions entre diferents taules.

Sovint, en la combinació entre dues taules hi ha una taula que es pot considerar la taula principal (on hem d'anar a cercar la informació) i una altra taula que es pot considerar secundària (on hem d'anar a cercar informació que complementi la informació cercada en la taula principal).

Per aconseguir el nostre propòsit, des de la revisió del 92 (SQL-92), el llenguatge SQL facilita les operacions *join* en la clàusula `from` indicant la condició de combinació, la qual ja no s'haurà d'indicar (excepte en un cas) dins la clàusula `where`.

És a dir, passem d'una sentència `SELECT` que inclou una part similar a:

```
...
from taula1, taula2
where <condició combinació entre taula1 i taula2>
...
```

a una sentència `SELECT` en què la condició de combinació entre taules ja no s'indica en la clàusula `where`, sinó que acompanya la clàusula `from`:

```
...
from taula1 [ inner | left | right ] join taula2
on <condició combinació entre taula1 i taula2>
where ...
```

Com es veu en la sintaxi anterior, hi ha diferents opcions de *join*: `inner`, `left` i `right`.

Els SGBD relacionals actuals (MySQL, Oracle, SQLServer, PostgreSQL, MS-Access...) incorporen les combinacions `inner`, `left` i `right` amb les quals es poden aconseguir tots els tipus de combinacions entre taules. Hi ha altres opcions que alguns SGBD també suporten, però no sempre seguint una sintaxi idèntica. La sintaxi aquí presentada és la proporcionada per l'SGBD MySQL a partir de la versió 5.

La combinació `inner` és la més comuna i, de fet, és la que s'executa si no s'indica cap de les opcions. S'anomena *combinació interna* i combina files de dues taules sempre que hi hagi valors coincidents en el camp o camps de combinació.

Exemple de combinació inner entre dues taules

En l'esquema *empresa*, es volen mostrar els empleats (codi i cognom) juntament amb el codi i nom del departament al qual pertanyen.

```
select emp.emp_no as "Codi empleat", emp.cognom as "Empleat",
   emp.dept_no as "Codi departament", dnom as "Descripció"
from emp inner join dept on emp.dept_no = dept.dept_no;
```

Fixem-nos que la columna corresponent al codi de departament només apareix una vegada, ja que només l'hem indicat una vegada en la clàusula select. Recordem també que no és obligatori utilitzar la paraula inner.

Les combinacions **left join**, **right join** i **full join** (anomenades *combinacions externes*) són les opcions que proporciona l'SQL-92 per assolir les diverses opcions d'*outer-join*.

La combinació **left join** permet combinar totes les files de la taula de l'esquerra del *join* amb les files amb valors coincidents de la taula de la dreta, i proporciona valors nuls per a les columnes de la taula de la dreta quan no hi ha files amb valors coincidents.

La combinació **right join** permet combinar totes les files de la taula de la dreta del *join* amb les files amb valors coincidents de la taula de l'esquerra, i proporciona valors nuls per a les columnes de la taula de l'esquerra quan no hi ha files amb valors coincidents.

La combinació **full join** és la unió del **left join** i **right join** eliminant la duplicitat de files a causa de les files de les dues taules que tenen valors coincidents.

Exemple 1 de right-outer-join i left-outer-join entre taules segons l'SQL-92

En l'esquema *empresa*, es volen mostrar tots els departaments (codi i descripció) acompanyats del salari més alt dels seus empleats.

```
select d.dept_no as "Codi", dnom as "Departament",
    max(salari) as "Major salari"
from dept d left join emp e on d.dept_no = e.dept_no
group by d.dept_no, dnom
order by 1;
```

El resultat obtingut és aquest:

```
Codi       Departament    Salari més alt
---------- -------------- --------------------
10         COMPTABILITAT  650000
20         INVESTIGACIÓ   390000
30         VENDES         370500
40         PRODUCCIÓ

4 rows selected
```

En la sentència anterior hem utilitzat un left join, el qual es pot convertir en un right join si canviem l'ordre de les taules en la clàusula `from`:

```
select d.dept_no as "Codi", dnom as "Departament",
   max(salari) as "Major salari"
from emp e right join dept d on e.dept_no = d.dept_no
group by d.dept_no, dnom
order by 1;
```

El resultat obtingut en aquest cas és el mateix que en la sentència anterior.

Exemple 2 de right-outer-join i left-outer-join entre taules segons l'SQL-92

Es volen mostrar, en l'esquema *empresa*, tots els empleats acompanyats dels clients de qui són representants.

La instrucció per assolir l'objectiu és aquesta:

```
select emp_no as "Codi", cognom as "Empleat", client_cod as "
    Client", nom as "Raó social"
from emp left join client on emp_no = repr_cod
order by 1,2;
```

O també:

```
select emp_no as "Codi", cognom as "Empleat", client_cod as "
    Client", nom as "Raó social"
from client right join emp on repr_cod = emp_no
order by 1,2;
```

El resultat que s'obté, en ambdós casos, és aquest:

```
Codi       Empleat    Client     Raó social
---------- ---------- ---------- --------------------------------------------
7369       SANCHEZ
7499       ARROYO     107        WOMEN SPORTS
7499       ARROYO     104        EVERY MOUNTAIN
7521       SALA       101        TKB SPORT SHOP
7521       SALA       103        JUST TENNIS
7521       SALA       106        SHAPE UP
7566       JIMENEZ
7654       MARTIN     102        VOLLYRITE
7698       NEGRO
7782       CEREZO
7788       GIL
7839       REY
7844       TOVAR      108        NORTH WOODS HEALTH AND FITNESS SUPPLY CENTER
7844       TOVAR      105        K + T SPORTS
7844       TOVAR      100        JOCKSPORTS
7876       ALONSO
7900       JIMENO
7902       FERNANDEZ
7934       MUÑOZ

19 rows selected
```

Fixem-nos que, per assegurar l'aparició de tots els empleats, cal utilitzar un `outer-join`, ja que altrament els empleats que no tenen assignat cap client no apareixerien.

Exemple 3 de right-outer-join i left-outer-join entre taules segons l'SQL-92

En l'esquema *empresa*, es volen mostrar tots els clients acompanyats de l'empleat que tenen com a representant.

La instrucció per assolir l'objectiu és aquesta:

```
select client_cod as "Client", nom as "Raó social", emp_no as "
    Codi", cognom as "Empleat"
from client left join emp on repr_cod = emp_no;
```

O també:

```
select client_cod as "Client", nom as "Raó social", emp_no as "
    Codi", cognom as "Empleat"
from emp right join client on emp_no = repr_cod;
```

El resultat que s'obté és aquest:

```
Client     Raó social                                    Codi       Empleat
---------- --------------------------------------------- ---------- ----------
107        WOMEN SPORTS                                  7499       ARROYO
104        EVERY MOUNTAIN                                7499       ARROYO
106        SHAPE UP                                      7521       SALA
103        JUST TENNIS                                   7521       SALA
101        TKB SPORT SHOP                                7521       SALA
102        VOLLYRITE                                     7654       MARTIN
108        NORTH WOODS HEALTH AND FITNESS SUPPLY CENTER  7844       TOVAR
105        K + T SPORTS                                  7844       TOVAR
100        JOCKSPORTS                                    7844       TOVAR
109        SPRINGFIELD NUCLEAR POWER PLANT

10 rows selected
```

Fixem-nos que, per assegurar l'aparició de tots els clients, cal utilitzar un outer-join, ja que altrament els clients que no tenen assignat representant no apareixerien.

Exemple 4 de ull-outer-join entre taules segons l'SQL-92

En l'esquema *empresa*, es volen mostrar tots els clients i tots els empleats relacionant cada client amb el seu representant (i, de retruc, cada empleat amb els seus clients).

La instrucció per assolir l'objectiu és aquesta:

```
select client_cod as "Client", nom as "Raó social", emp_no as "
    Codi", cognom as "Empleat"
from client left join emp on emp_no = repr_cod
union
select client_cod as "Client", nom as "Raó social", emp_no as "
    Codi", cognom as "Empleat"
from client right join emp on emp_no = repr_cod;
```

El resultat obtingut és aquest:

```
Client     Raó social                                    Codi       Empleat
---------- --------------------------------------------- ---------- ----------
100        JOCKSPORTS                                    7844       TOVAR
101        TKB SPORT SHOP                                7521       SALA
102        VOLLYRITE                                     7654       MARTIN
103        JUST TENNIS                                   7521       SALA
104        EVERY MOUNTAIN                                7499       ARROYO
105        K + T SPORTS                                  7844       TOVAR
106        SHAPE UP                                      7521       SALA
107        WOMEN SPORTS                                  7499       ARROYO
108        NORTH WOODS HEALTH AND FITNESS SUPPLY CENTER  7844       TOVAR
109        SPRINGFIELD NUCLEAR POWER PLANT
                                                         7369       SANCHEZ
                                                         7566       JIMENEZ
                                                         7698       NEGRO
                                                         7782       CEREZO
                                                         7788       GIL
                                                         7839       REY
                                                         7876       ALONSO
                                                         7900       JIMENO
                                                         7902       FERNANDEZ
                                                         7934       MUÑOZ

20 rows selected
```

El llenguatge SQL proporciona dues simplificacions en l'escriptura de les combinacions *join* que necessiten l'opció `on`, per als casos en què les columnes que cal combinar tinguin coincidència de noms:

- Si la combinació es vol efectuar per a totes les columnes que tinguin noms coincidents en les taules que s'han de combinar, disposem de les combinacions **natural join**. En aquest cas, la paraula **natural** davant el tipus de combinació *join* provoca que s'efectuï la combinació entre les taules per a totes les columnes que tenen coincidència de nom, sense haver d'indicar la condició de combinació. El resultat de les *natural join* proporciona una única columna per a les columnes de les taules combinades que tenen el mateix nom i, per tant, en cas d'haver de fer referència a aquesta columna en alguna clàusula de la sentència `SELECT`, no s'ha d'indicar el nom de la taula a la qual pertany, ja que pertany simultàniament a diferents taules.

- Si la combinació es vol efectuar per a algunes de les columnes que tinguin noms coincidents en les taules que cal combinar, disposem de les combinacions *join* amb l'opció `using (col1, col2 ...)`. En aquest cas, l'opció `using (col1, col2 ...)` provoca que s'efectuï la combinació entre les taules per a les columnes indicades, sense haver d'indicar la condició de combinació. Com en els *natural join*, també dóna com a resultat una única columna per a les columnes coincidents.

Exemple de simplificació d'una combinació inner join

La sentència següent, corresponent a l'esquema *empresa*, mostra els empleats (codi i cognom) juntament amb el codi i nom del departament al qual pertanyen.

```
select emp.emp_no as "Codi empleat", emp.cognom as "Empleat",
   emp.dept_no as "Codi departament", dnom as "Descripció"
from emp inner join dept on emp.dept_no = dept.dept_no;
```

o bé, el que és el mateix:

```
select emp.emp_no as "Codi empleat", emp.cognom as "Empleat",
   emp.dept_no as "Codi departament", dnom as "Descripció"
from emp join dept on emp.dept_no = dept.dept_no;
```

Com que en les taules `EMP` i `DEPT` hi ha coincidència de nom per a la columna de combinació deptno i no hi ha altres columnes amb noms coincidents, podem utilitzar un natural join/:

```
select emp.emp_no as "Codi empleat", emp.cognom as "Empleat",
   dept_no as "Codi departament", dnom as "Descripció"
from emp natural join dept;
```

Fixem-nos que, en visualitzar la columna deptno, en la clàusula select s'ha hagut de suprimir la referència a la taula a la qual pertany, ja que el natural join* només proporciona una de les dues columnes amb coincidència de noms. Però també s'hauria pogut utilitzar l'opció using:

```
select emp.emp_no as "Codi empleat", emp.cognom as "Empleat",
   dept_no as "Codi departament", dnom as "Descripció"
from emp inner join dept using (dept_no);
```

Així mateix, en tractar-se d'una combinació inner join, ens podem estalviar el mot `inner`, i tindríem el següent:

```
select emp.emp_no as "Codi empleat", emp.cognom as "Empleat",
   dept_no as "Codi departament", dnom as "Descripció"
from emp join dept using (dept_no);
```

Exemple de simplificació de combinacions left join i right join

Les sentències següents, corresponents a l'esquema *empresa*, mostren tots els departaments (codi i descripció) acompanyats del salari més alt dels seus empleats.

```
select d.dept_no as "Codi", dnom as "Departament",
   max(salari) as "Salari més alt"
from dept d left join emp e on d.dept_no = e.dept_no
group by d.dept_no, dnom
order by 1;

select d.dept_no as "Codi", dnom as "Departament",
   max(salari) as "Salari més alt"
from emp e right join dept d on e.dept_no = d.dept_no
group by d.dept_no, dnom
order by 1;
```

Atès que la columna de combinació té el mateix nom i no hi ha altres columnes amb coincidència de noms, hauríem pogut emprar un natural join:

```
select dept_no as "Codi", dnom as "Departament",
   max(salari) as "Salari més alt"
from dept natural left join emp e
group by dept_no, dnom
order by 1;

select dept_no as "Codi", dnom as "Departament",
   max(salari) as "Salari més alt"
from emp e natural right join dept d
group by dept_no, dnom
order by 1;
```

I també, utilitzant l'opció `using`:

```
select dept_no as "Codi", dnom as "Departament",
   max(salari) as "Salari més alt"
from dept left join emp e using (dept_no)
group by dept_no, dnom
order by 1;

select dept_no as "Codi", dnom as "Departament",
   max(salari) as "Salari més alt"
from emp e right join dept d using (dept_no)
group by dept_no, dnom
order by 1;
```

2.7 Subconsultes

De vegades, és necessari executar una sentència SELECT per aconseguir un resultat que cal utilitzar com a part de la condició de filtratge d'una altra sentència SELECT. El llenguatge SQL ens facilita efectuar aquest tipus d'operacions amb la utilització de les subconsultes.

Una **subconsulta** és una sentència SELECT que s'inclou en la clàusula `where` d'una altra sentència SELECT. La subconsulta es tanca entre parèntesis i no inclou el punt i coma finals.

Una subconsulta pot contenir, a la vegada, altres subconsultes.

Exemple de subconsulta que calcula un resultat a utilitzar en una clàusula where

En l'esquema *empresa*, es demana mostrar els empleats que tenen salari igual o superior al salari mitjà de l'empresa.

La instrucció per assolir l'objectiu és aquesta:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari"
from emp
where salari >= (select avg(salari) from emp)
order by 3 desc,1;
```

En certes situacions pot ser necessari accedir des de la subconsulta als valors de les columnes seleccionades en la consulta. El llenguatge SQL ho permet sense problemes i, en cas que els noms de les columnes coincideixin, es poden utilitzar àlies.

Els noms de columnes que apareixen en les clàusules d'una subconsulta s'intenten avaluar, en primer lloc, com a columnes de les taules definides en la clàusula `from` de la subconsulta, llevat que vagin acompanyades d'àlies que les identifiquin com a columnes d'una taula en la consulta contenidora.

Exemple de subconsulta que fa referència a columnes de la consulta contenidora

En l'esquema *empresa*, es demana mostrar els empleats de cada departament que tenen un salari menor que el salari mitjà del mateix departament.

La instrucció per assolir l'objectiu és aquesta:

```
select dept_no as "Dept.", emp_no as "Codi", cognom as "Empleat",

   salari as "Salari"
from emp e1
where salari >= (select avg(salari)
       from emp e2
       where dept_no=e1.dept_no
                )
order by 1, 4 desc,2;
```

Els valors retornats per les subconsultes s'utilitzen en les clàusules `where` com a part dreta d'operacions de comparacions en què intervenen els operadors:

```
=,!=,<,< =,>,> =,[not] in,%%<op>%% any i %%<op>%% all
```

Les subconsultes també es poden vincular a la consulta contenidora per la partícula **[not] exists**:

```
...
where [not] exists (subconsulta)
```

En aquest cas, la subconsulta acostuma a fer referència a valors de les taules de la consulta contenidora. S'anomenen **subconsultes sincronitzades**.

Les consultes que poden donar com a resultat un únic valor o cap poden actuar com a subconsultes en expressions en què el valor resultat es compara amb qualsevol operador de comparació.

Les consultes que poden donar com a resultat més d'un valor (encara que en execucions concretes només en donin un) mai no poden actuar com a subconsultes

en expressions en què els valors resultants es comparen amb l'operador =, ja que l'SGBDR no sabria amb quin dels resultats efectuar la comparació d'igualtat i es produiria un error.

Si cal aprofitar els resultats de més d'una columna de la subconsulta, aquesta es col·loca a la dreta de l'operació de comparació i en la part esquerra es col·loquen els valors que s'han de comparar, en el mateix ordre que els valors retornats per la subconsulta, separats per comes i tancats entre parèntesis:

```
...
where (valor1, valor2 ...) <op> (select col1, col2 ...)
```

Exemple d'utilització de l'operador = per comparar amb el resultat d'una subconsulta

En l'esquema *empresa*, es volen mostrar els empleats que tenen el mateix ofici que l'ofici que té l'empleat de cognom 'ALONSO'.

La instrucció per assolir l'objectiu sembla que podria ser aquesta:

```
select cognom as "Empleat"
from emp
where ofici = (select ofici
      from emp
      where upper(cognom)='ALONSO')
  and upper(cognom)!='ALONSO';
```

En aquesta sentència hem utilitzat l'operador = de manera errònia, ja que no podem estar segurs que no hi ha dos empleats amb el cognom 'ALONSO'. Com que només n'hi ha un, la sentència s'executa correctament, però en cas que n'hi hagués més d'un, fet que pot succeir en qualsevol moment, l'execució de la sentència provocaria l'error abans esmentat.

Per tant, hauríem de cercar un altre operador de comparació per evitar aquest problema:

```
select cognom as "Empleat"
from emp
where ofici in (select ofici
       from emp
       where upper(cognom)='ALONSO')
  and upper(cognom)!='ALONSO';
```

O també:

```
select cognom as "Empleat"
from emp e
where exists (select *
      from emp
      where upper(cognom)='ALONSO'
       and ofici=e.ofici
            )
  and upper(cognom)!='ALONSO';
```

Exemple d'utilització dels operadors ANY i EXISTS

En l'esquema *empresa*, es demana mostrar els noms i oficis dels empleats del departament 20 la feina dels quals coincideixi amb la d'algun empleat del departament de 'VENDES'.

La instrucció per assolir l'objectiu pot ser aquesta:

```
select cognom as "Empleat", ofici as "Ofici"
from emp
where dept_no=20
 and ofici =ANY (select ofici
        from emp
        where dept_no =ANY (select dept_no
```

```
                from dept
                where upper(dnom)='VENDES'
                )
        );
```

Aquesta instrucció està pensada perquè el resultat sigui correcte en cas que hi pugui haver diferents departaments amb nom 'VENDES'. En cas que la columna dnom taula DEPT tingués definida la restricció d'unicitat, també seria correcta la instrucció següent:

```
select cognom as "Empleat", ofici as "Ofici"
from emp
where dept_no=20
  and ofici =ANY (select ofici
        from emp
        where dept_no = (select dept_no
              from dept
              where upper(dnom)='VENDES'
               )
        );
```

Una altra manera de resoldre el mateix problema és amb la utilització de l'operador `EXISTS`:

```
select cognom as "Empleat", ofici as "Ofici"
from emp e
where dept_no=20
  and EXISTS (select *
      from emp, dept
      where emp.dept_no=dept.dept_no
       and upper(dnom)='VENDES'
       and ofici=e.ofici
      );
```

Exemple d'utilització de l'operador IN

En l'esquema *empresa*, es demana mostrar els empleats amb el mateix ofici i salari que 'JIMÉNEZ'.

La instrucció per assolir l'objectiu pot ser aquesta:

```
select emp_no "Codi", cognom "Empleat"
from emp
where (ofici,salari) IN (select ofici, salari
          from emp
          where upper(cognom)='JIMÉNEZ'
          )
  and upper(cognom)!='JIMÉNEZ';
```

Exemple de condició complexa de filtratge amb diverses subconsultes i operacions

Es demana, en l'esquema *empresa*, mostrar els empleats que efectuïn la mateixa feina que 'JIMÉNEZ' o que tinguin un salari igual o superior al de 'FERNÁNDEZ'.

```
select emp_no "Codi", cognom "Empleat"
from emp
where (ofici IN (select ofici
       from emp
       where upper(cognom)='JIMÉNEZ'
       )
   and upper(cognom)!='JIMÉNEZ'
   )
  or (salari>= (select salari
        from emp
        where upper(cognom)='FERNÁNDEZ'
       )
   and upper(cognom)!='FERNÁNDEZ'
   );
```

3. Annex 1. MySQL

MySQL és un sistema gestor de bases de dades relacionals (SGBDR) corporatiu de codi obert, molt utilitzat per donar suport a la gestió de les dades en aplicacions web, sovint juntament amb Apache i PHP.

LAMP, WAMP

MySQL és conegut perquè forma part dels paquets de desenvolupament web LAMP (Linux, Apache, MySQL, PHP) i WAMP (Windows, Apache, MySQL, PHP), enormement utilitzats per a la creació d'aplicacions web.

MySQL va néixer com a programari lliure, sota llicència GNU GP, amb aportacions i patrocinis. Actualment és propietat d'Oracle Corporation, que va comprar Sun Microsystems, propietari anterior de la marca.

Hi ha dues versions bàsiques, ben diferenciades, d'aquest programari:

- **La versió comercial**: MySQL Enterprise Edition
- **La versió lliure**: MySQL Community Server

A banda d'aquestes dues distribucions, també hi ha les versions MySQL Cluster i MySQL Cluster GCE que ofereixen sistemes específicament dissenyats per optimitzar bases de dades que hagin de donar servei a sistemes de temps real transaccionals en condicions d'alta demanda. La primera (MySQL Cluster) és de codi obert i la segona n'és la versió comercial (MySQL Cluster GCE).

Us podeu descarregar lliurament l'SGBD MySQL Community Server del seu web: http://www.mysql.com/.

Juntament amb el programari que integra estrictament l'SGBDR MySQL es pot descarregar MySQL Workbench, un frontal (*front-end*) que ofereix un entorn visual que facilita la creació, gestió, administració i explotació de les bases de dades del sistema. Aquesta eina també està disponible en versió comercial (*standard edition*) i lliure (*community edition*).

Frontals

Es coneixen amb el nom de *frontal* (*front-end*) les aplicacions gràfiques que complementen altres programaris, que habitualment no disposen d'interfície gràfica, i que, per tant, pretenen facilitar-ne l'administració i gestió.

També es pot entendre per *frontal* la part de l'aplicació que interacciona amb l'usuari (interfície de l'aplicació).

3.1 Instal·lació de MySQL i MySQL Workbench

Per instal·lar MySQL Community Server i el seu frontal MySQL Workbench, ho podeu fer amb el gestor de paquets. En el cas d'Ubuntu, podeu utilitzar Synaptic

(accessible a partir del submenú **Administració**, del menú **Sistema**).

També us podeu instal·lar les eines MySQL Query Browser i MySQL Administrator, i treballar-hi, eines que actualment estan disponibles dins de MySQL Workbench i que anteriorment s'oferien com a eines independents.

Per a Windows, us podeu descarregar del web de MySQL els instal·lables (.msi) de l'SGBDR i del Workbench i en uns pocs passos tenir-lo instal·lat en el vostre sistema. Vegeu les imatges que hi ha a continuació.

FIGURA 3.1. Instal·lació de MySQL i MySQL Workbench. Pas 1

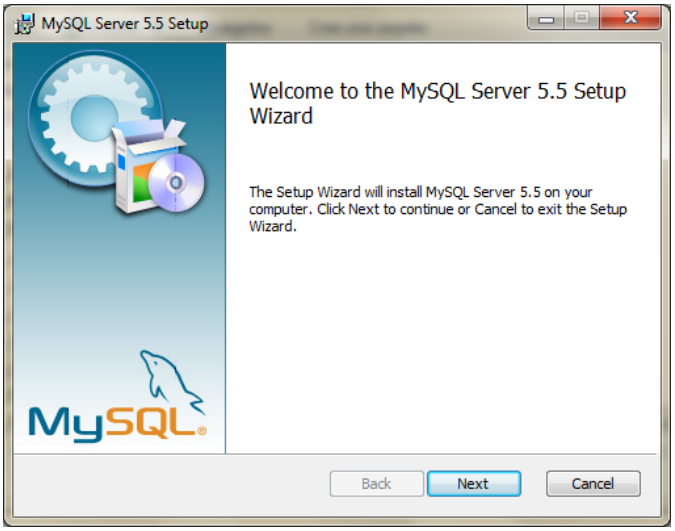


FIGURA 3.2. Instal·lació de MySQL i MySQL Workbench. Pas 2

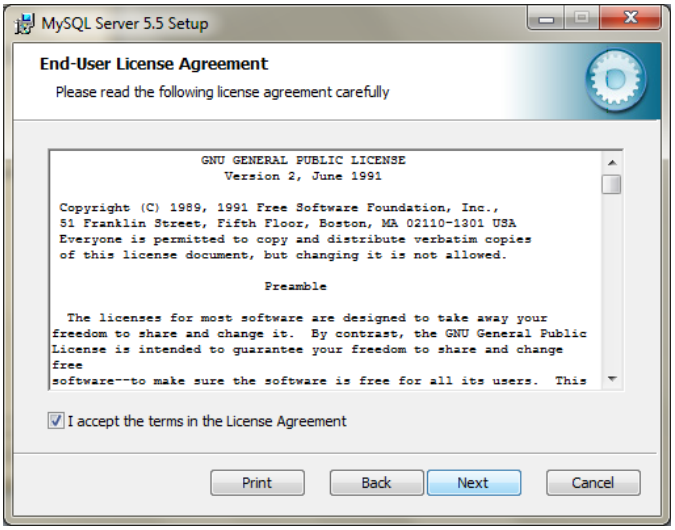


FIGURA 3.3. Instal·lació de MySQL i MySQL Workbench. Pas 3

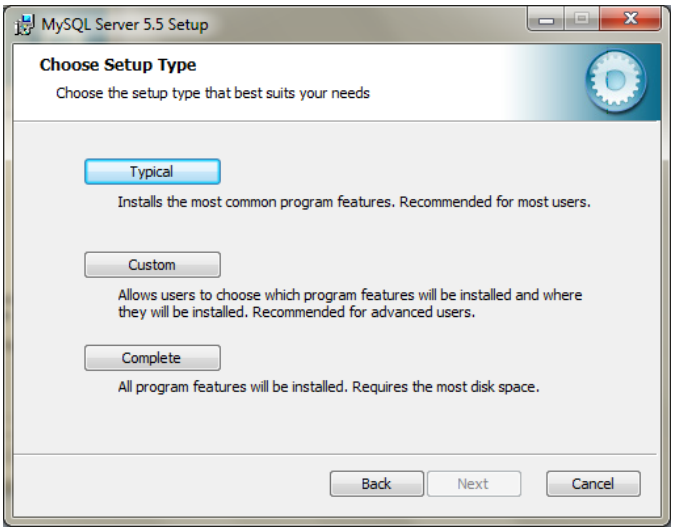


FIGURA 3.4. Instal·lació de MySQL i MySQL Workbench. Pas 4

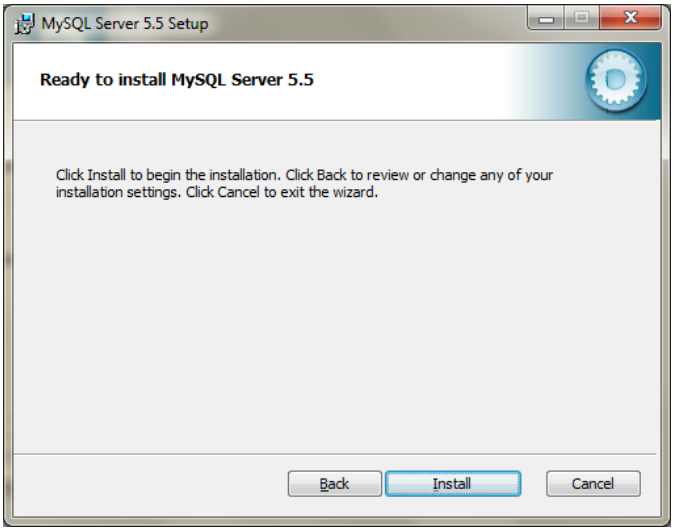


FIGURA 3.5. Instal·lació de MySQL i MySQL Workbench. Pas 5

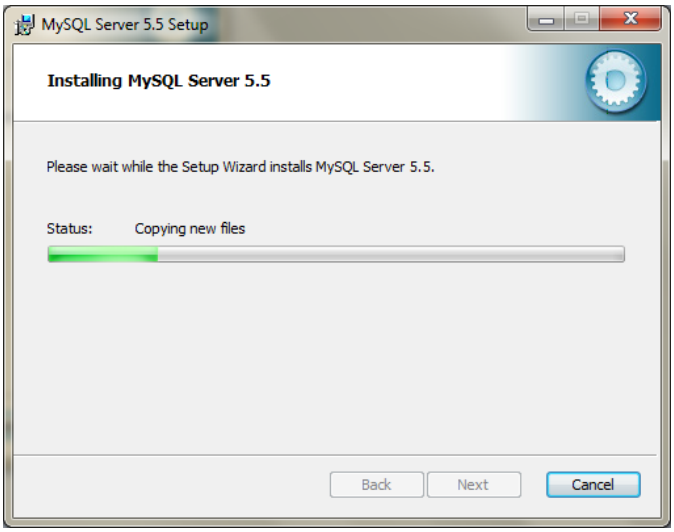


FIGURA 3.6. Instal·lació de MySQL i MySQL Workbench. Pas 6

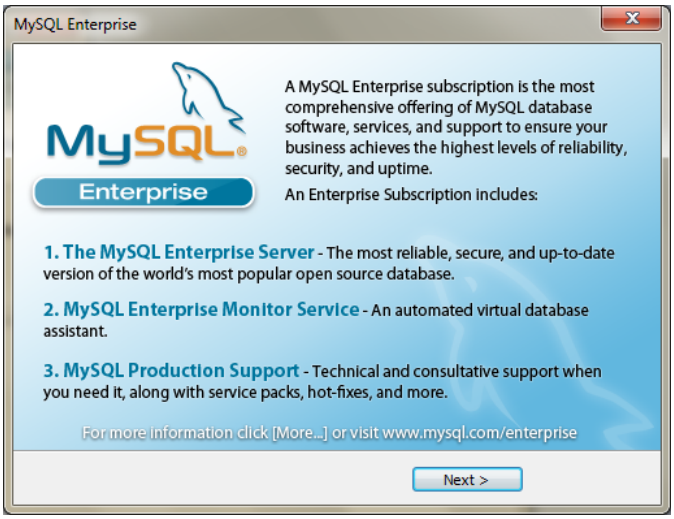


FIGURA 3.7. Instal·lació de MySQL i MySQL Workbench. Pas 7

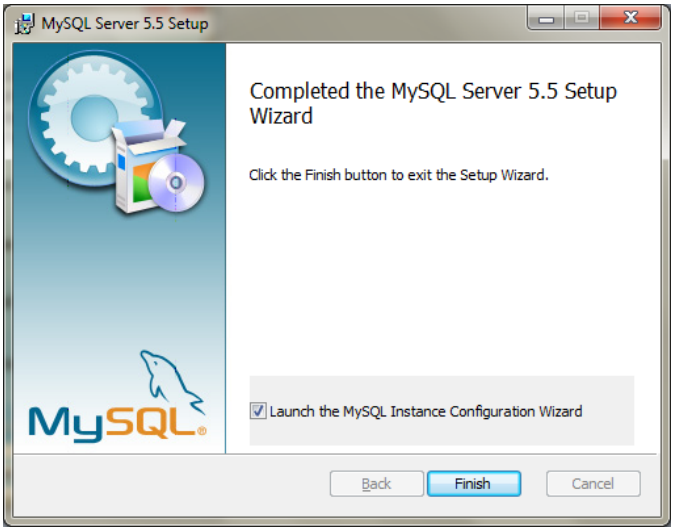


FIGURA 3.8. Instal·lació de MySQL i MySQL Workbench. Pas 8

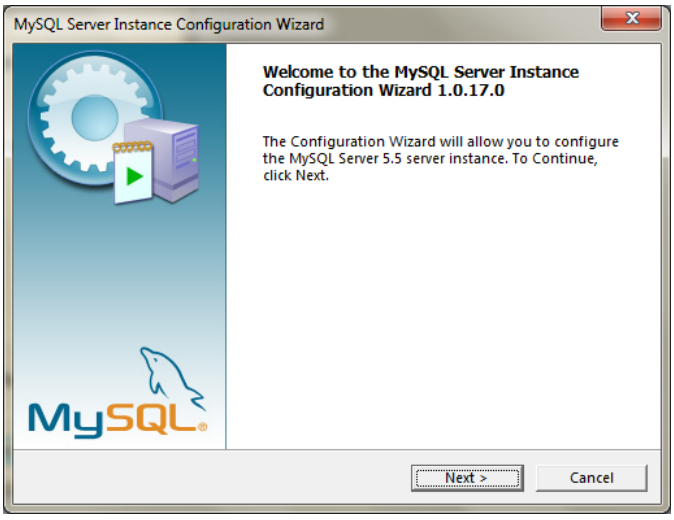


FIGURA 3.9. Instal·lació de MySQL i MySQL Workbench. Pas 9

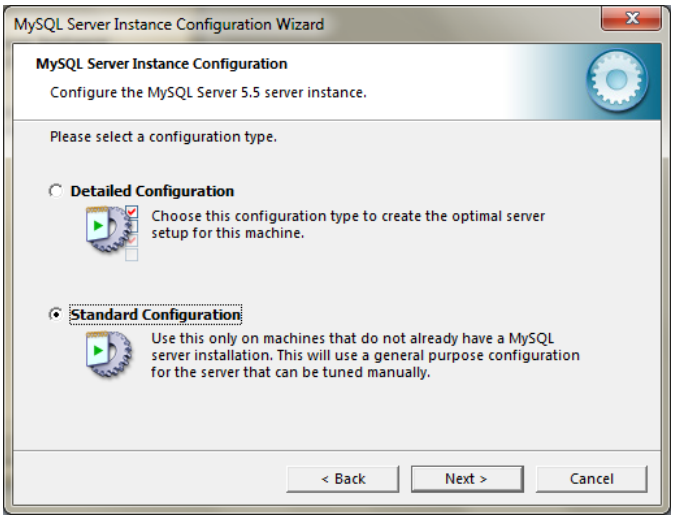


FIGURA 3.10. Instal·lació de MySQL i MySQL Workbench. Pas 10

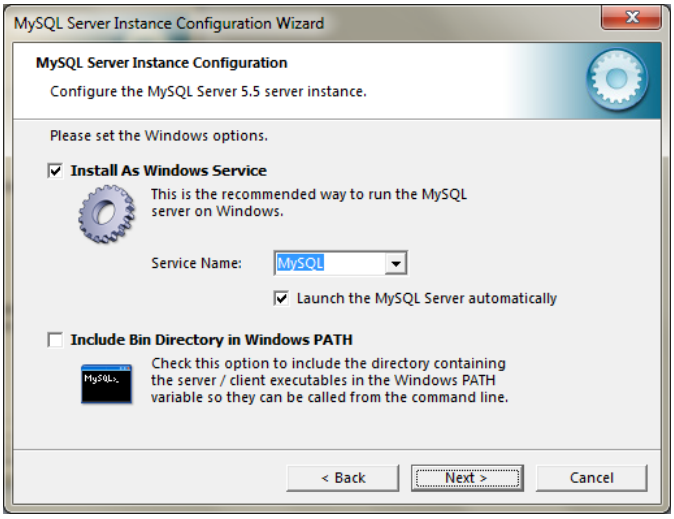


FIGURA 3.11. Instal·lació de MySQL i MySQL Workbench. Pas 11

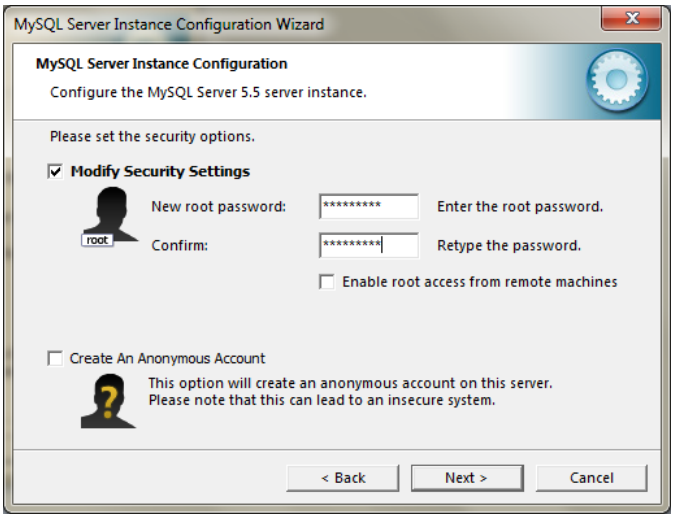


Figura 3.12. Instal·lació de MySQL i MySQL Workbench. Pas 12

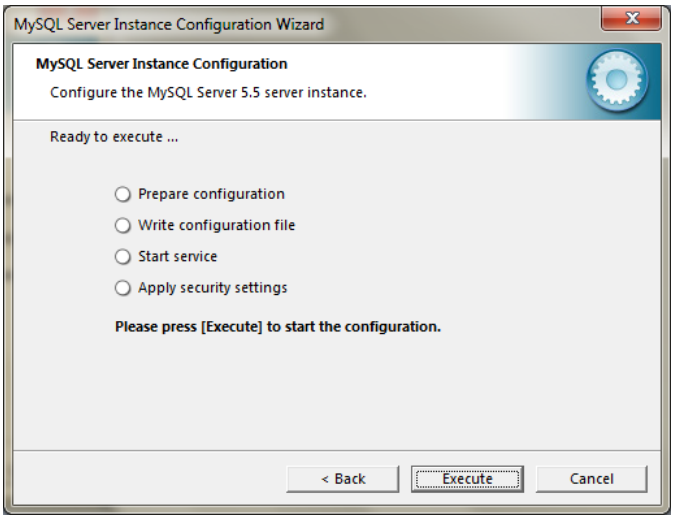


Figura 3.13. Instal·lació de MySQL i MySQL Workbench. Pas 13

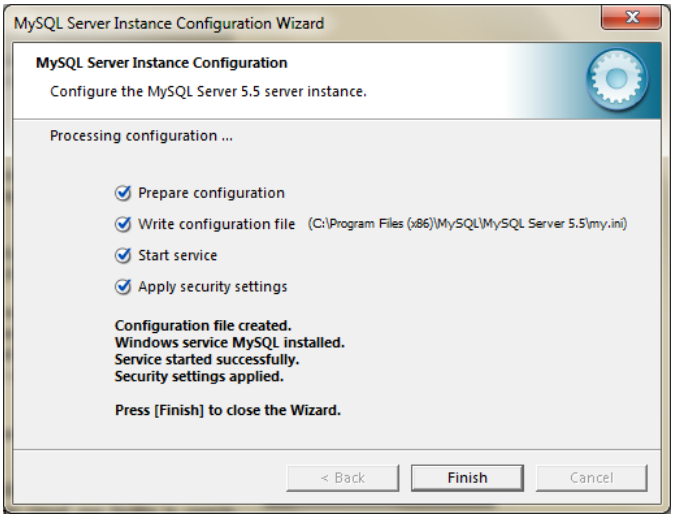


Figura 3.14. Instal·lació de MySQL i MySQL Workbench. Pas 14

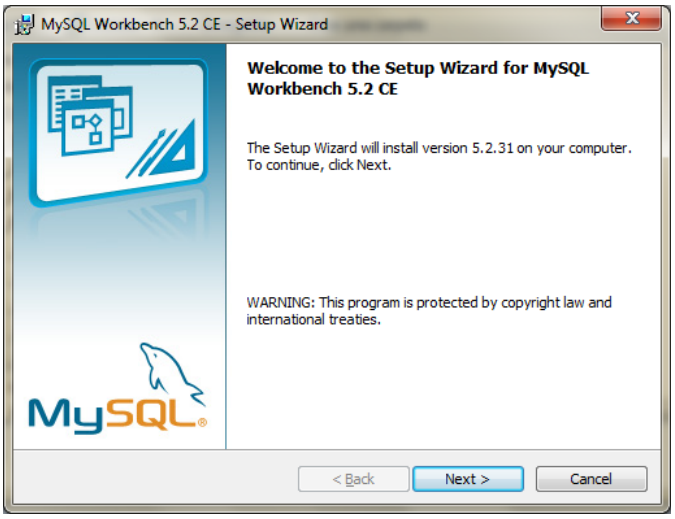


Figura 3.15. Instal·lació de MySQL i MySQL Workbench. Pas 15

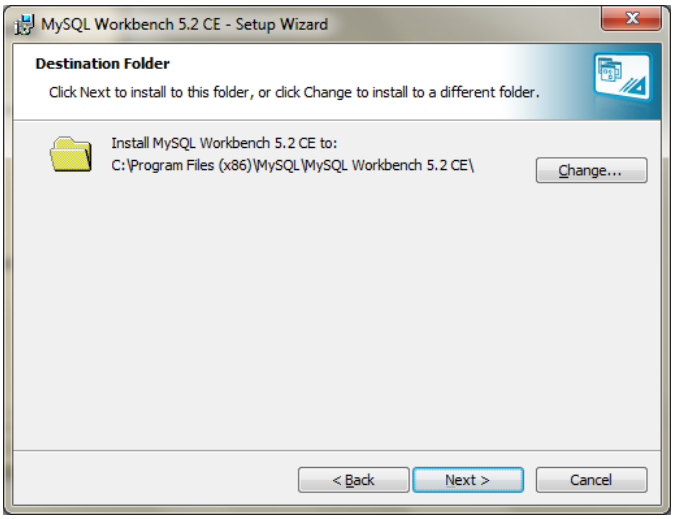


Figura 3.16. Instal·lació de MySQL i MySQL Workbench. Pas 16

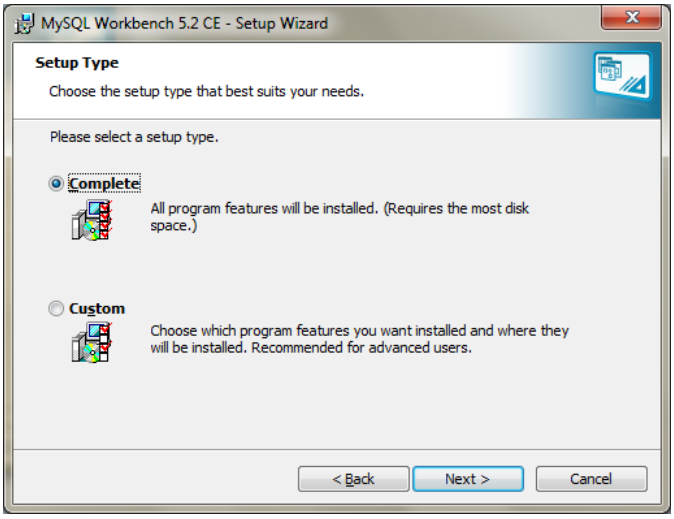


Figura 3.17. Instal·lació de MySQL i MySQL Workbench. Pas 17

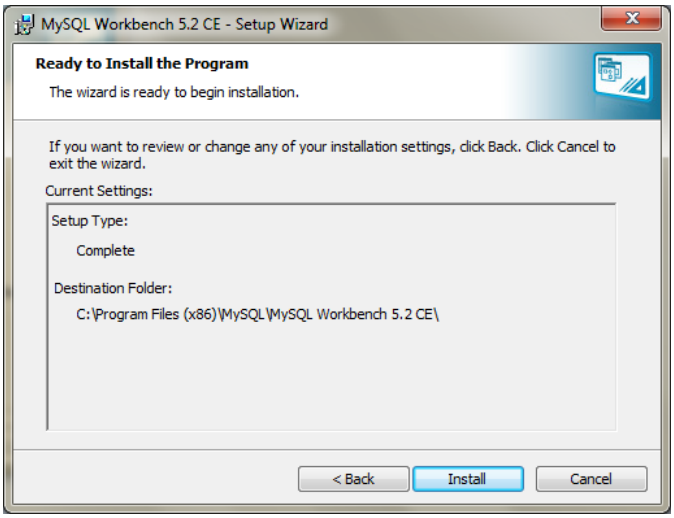


FIGURA 3.18. Instal·lació de MySQL i MySQL Workbench. Pas 18

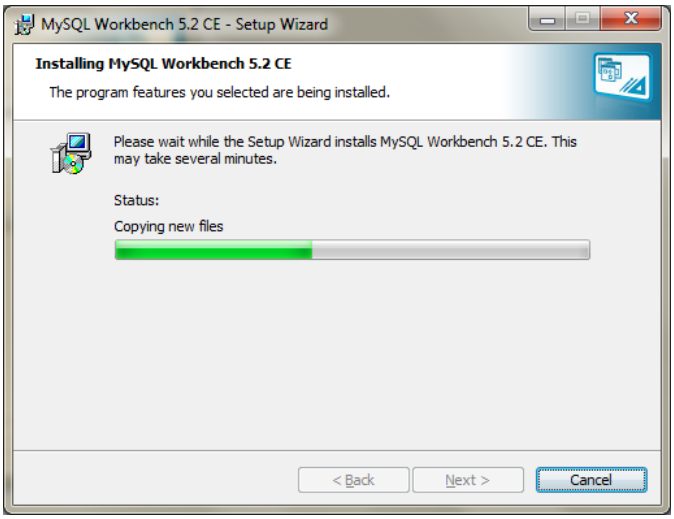


FIGURA 3.19. Instal·lació de MySQL i MySQL Workbench. Pas 19

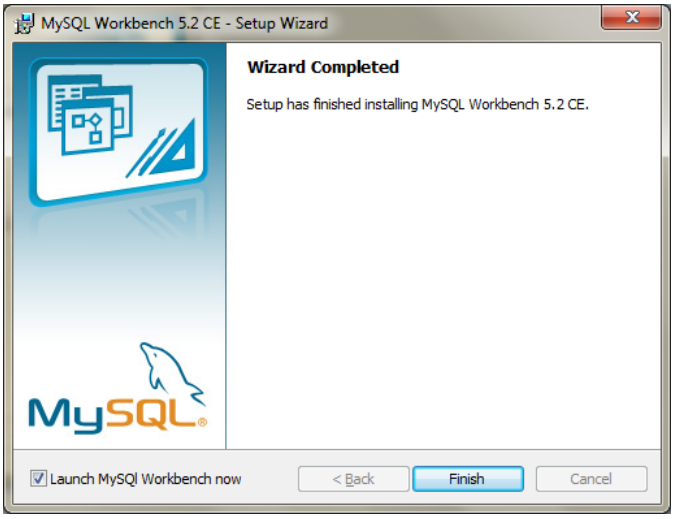
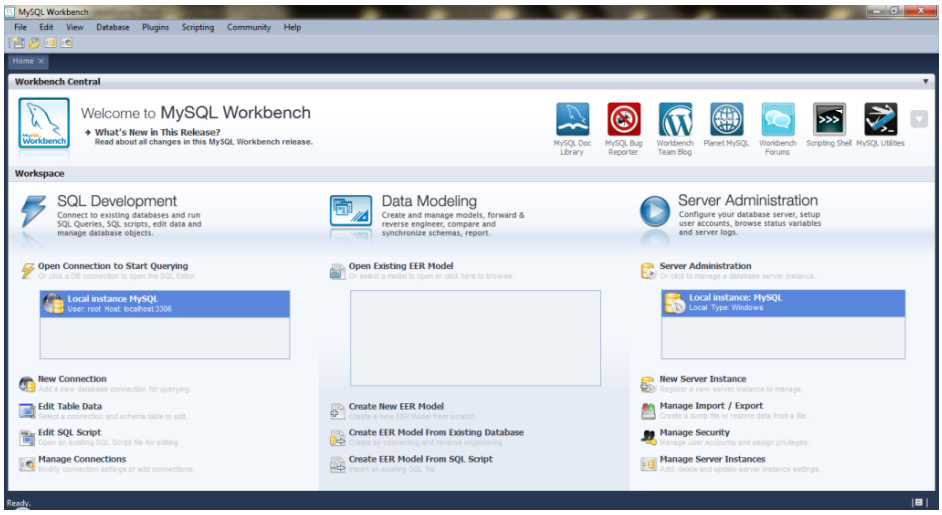


FIGURA 3.20. Instal·lació de MySQL i MySQL Workbench. Pas 20

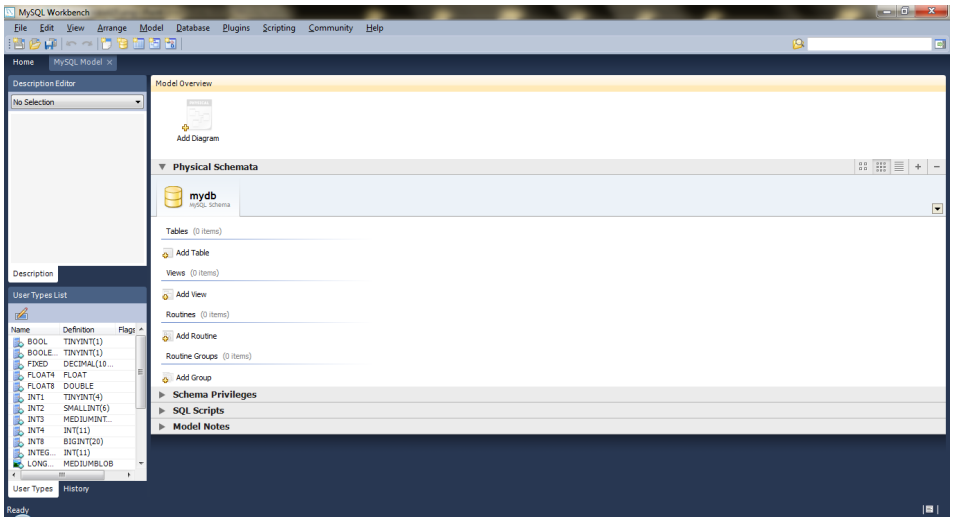


3.2 Primers passos en MySQL

Per començar a treballar podeu utilitzar el frontal que us heu instal·lat (Workbench).

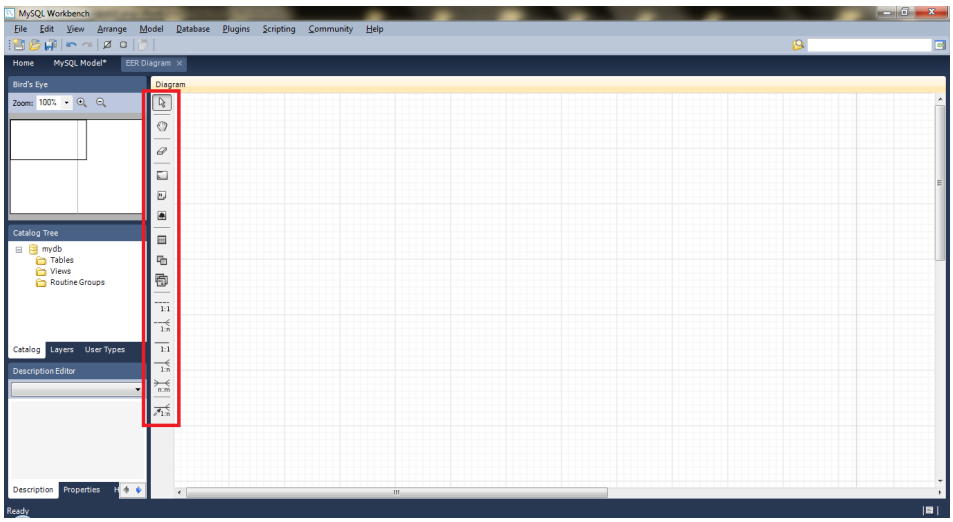
Per crear una base de dades visualment, podeu accedir a la secció central **Data Modeling** i seleccionar **Create New EER Model**.

Figura 3.21. Creació d'un model ER



Fent doble clic sobre l'opció **Add Diagram**, podem començar a treballar. També es poden crear les entitats, vistes i demés objectes de la base de dades fent doble clic sobre l'objecte corresponent. Es poden crear les entitats, dins del diagrama visual, gràcies a les icones que hi ha en la barra d'eines que, per defecte, es troba verticalment al costat esquerre del panell del diagrama.

Figura 3.22. Inserció d'objectes del model ER



Fent doble clic sobre un objecte, es permet visualitzar-ne les propietats. Les pestanyes que apareixen a la part inferior, permeten visualitzar i editar les diferents característiques relacionades amb cada objecte seleccionat del diagrama.

FIGURA 3.23. Propietats principals de les taules

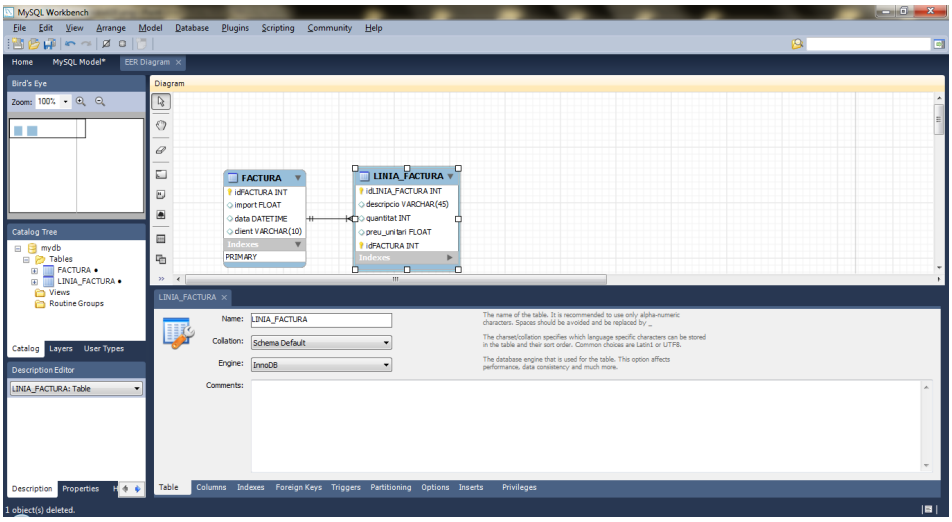


FIGURA 3.24. Propietats de les columnes de les taules

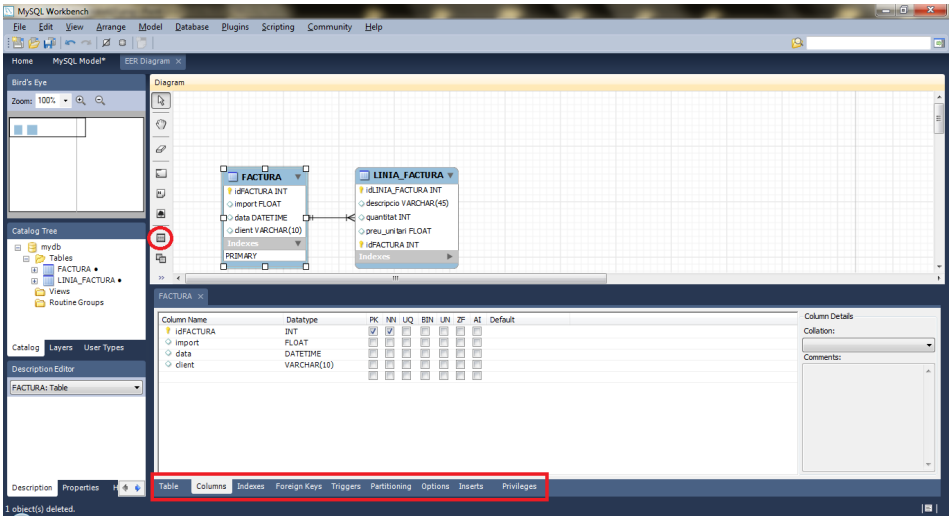
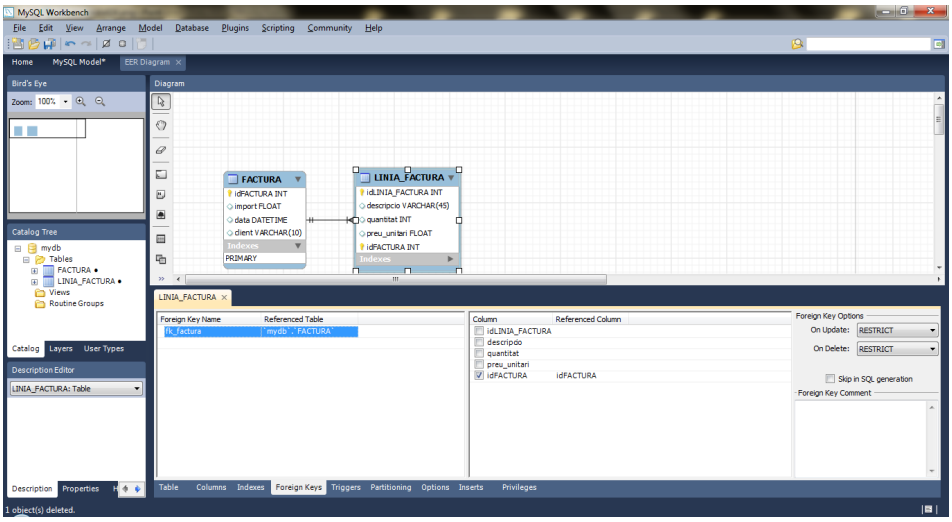


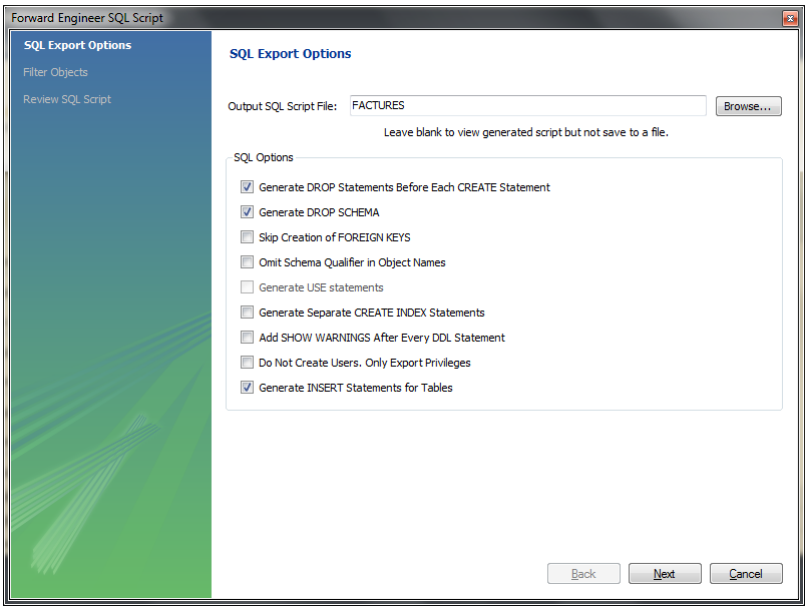
FIGURA 3.25. Claus foranes en les taules



Una vegada dissenyat un esquema de base de dades és útil exportar-lo en format script de SQL per tal de poder-lo carregar tantes vegades com calgui en un SGBD. També a tall de documentació de la BD i/o per seguretat. Per fer-ho, seleccionareu l'opció **Export**, del menú **File**. L'opció que escollireu per crear un script de creació de BD serà **Forward Engineer SQL CREATE Script**.

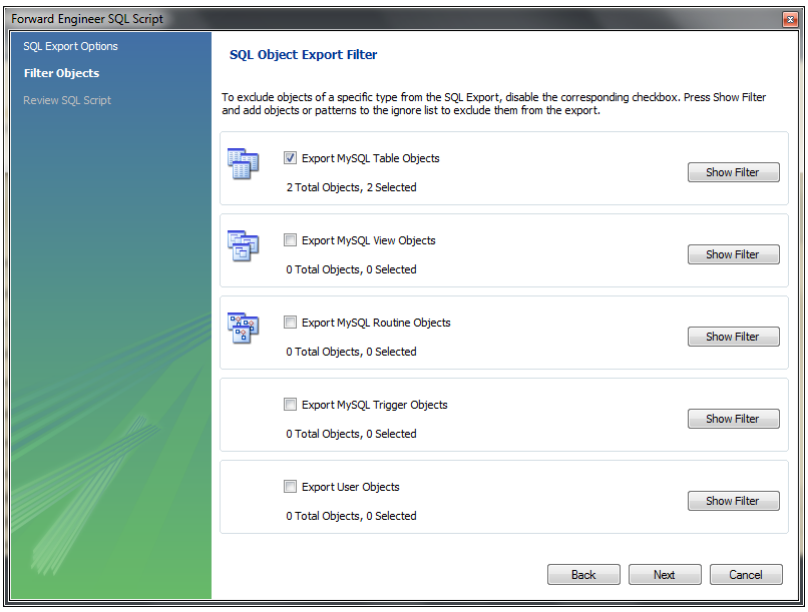
En primer lloc, caldrà donar un nom al fitxer que contindrà l'script i seleccionar les opcions de creació que es vulgui.

FIGURA 3.26. Exportació de BD a partir de la creació d'un script SQL



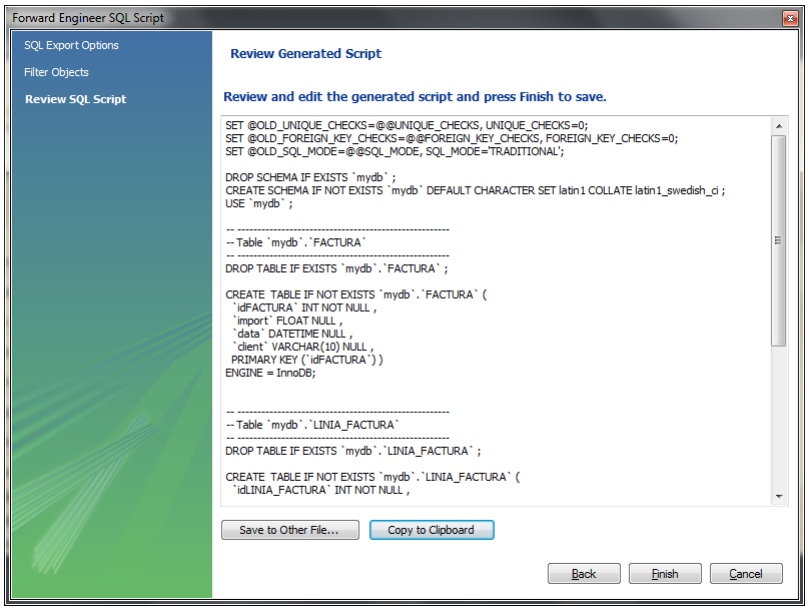
En segon lloc, cal seleccionar els objectes que cal exportar.

FIGURA 3.27. Creació d'un script SQL. Pas 2



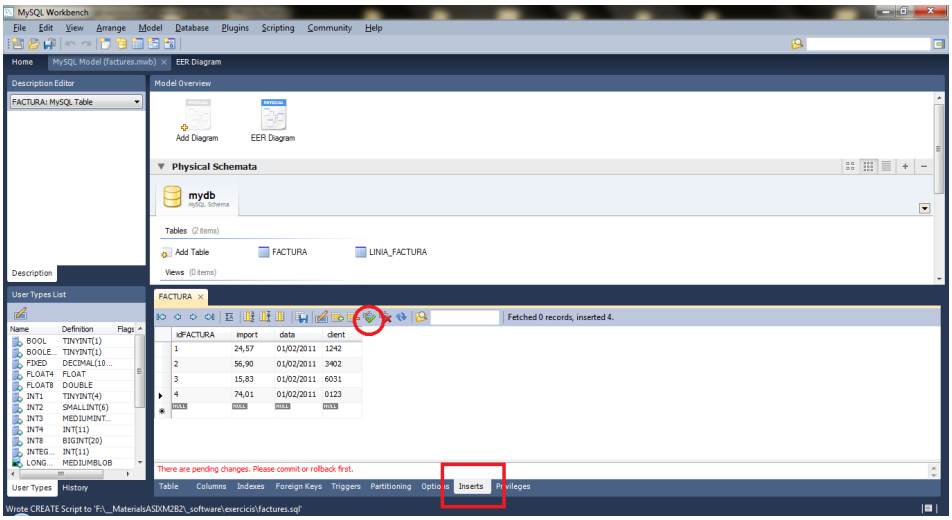
Finalment, es mostra l'script que es crearà i el procés finalitza.

FIGURA 3.28. Creació d'un script SQL. Pas 3



També podem introduir dades en les diferents taules de la base de dades de manera visual. Tan sols cal seleccionar la pestanya **Inserts** de les propietats de la taula en qüestió i introduir les dades volgudes, i prémer la icona **Apply changes to data** per emmagatzemar els canvis. En generar l'script de creació de la BD, podem seleccionar l'opció de generar les sentències corresponents per incloure aquestes dades en l'script.

FIGURA 3.29. Introducció de dades en les taules



3.3 Comencem a treballar amb MySQL

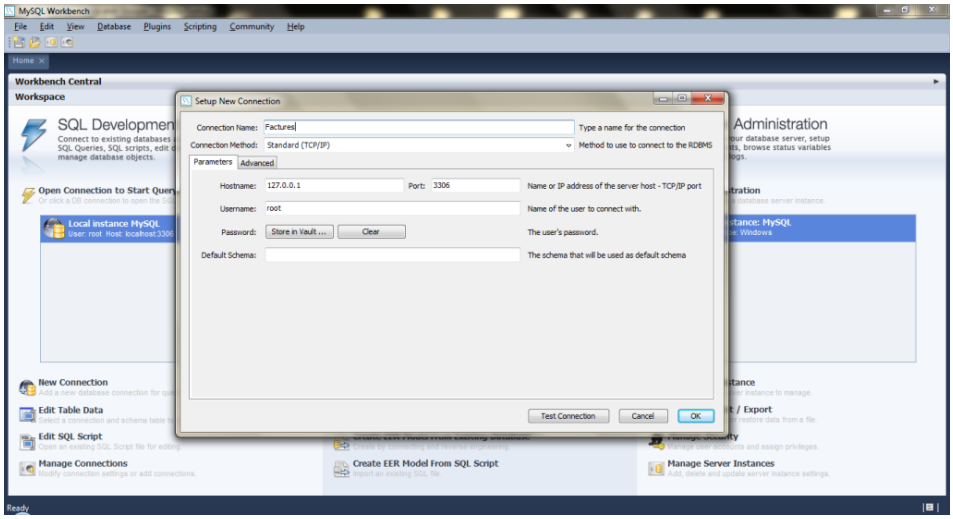
Partint d'un disseny de la base de dades, que s'ha creat amb l'ajuda de les eines de Data Modeling o perquè és determinat per una altra banda, mitjançant un script de SQL, per exemple, cal centrar-se en l'explotació de la base de dades.

L'explotació d'una base de dades consisteix bàsicament en la interacció amb la mateixa a partir de sentències SQL. Aquesta part es durà a terme mitjançant la secció SQL Development, de MySQL Workbench.

3.3.1 Nova connexió de BD

Podeu crear una nova connexió a la base de dades o, si treballem en local, utilitzar la que hi ha per defecte.

Figura 3.30. Nova connexió de BD



3.3.2 Nou esquema de BD

Per defecte, MySQL proporciona un esquema anomenat *mydb*, però podeu crear-ne un altre perquè contingui una BD nova.

Figura 3.31. Creació d'un esquema nou. Pas 1

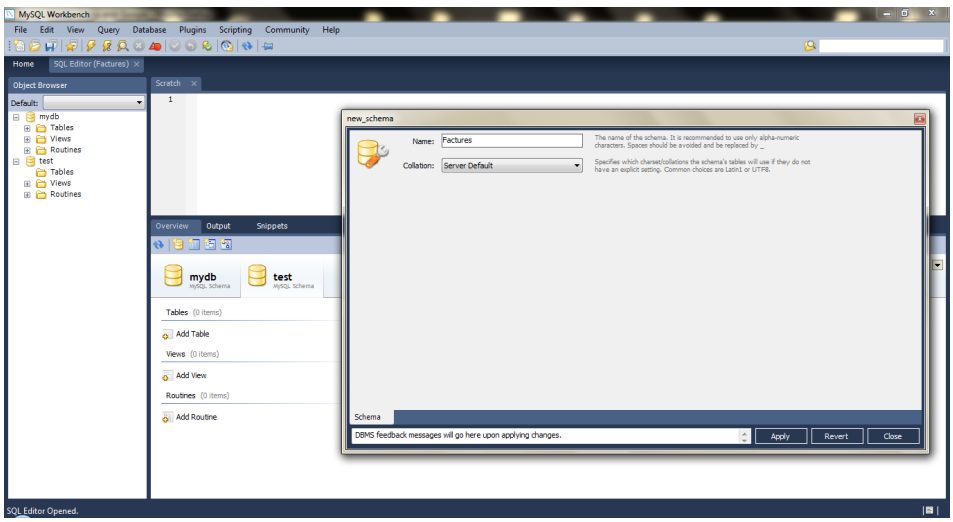


FIGURA 3.32. Creació d'un esquema nou. Pas 2

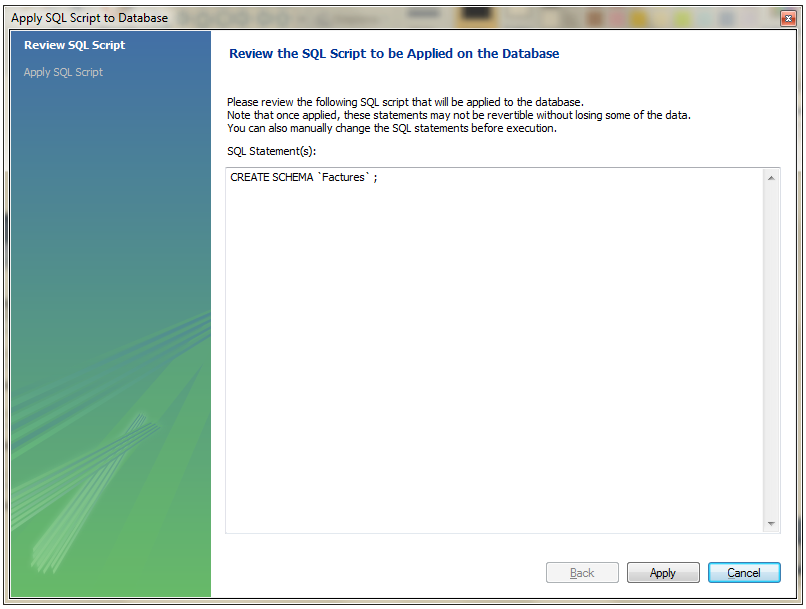
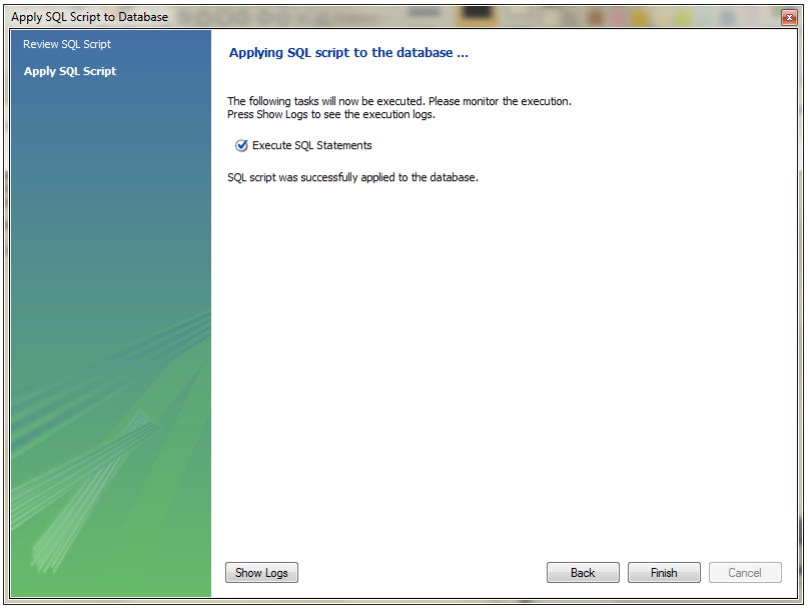


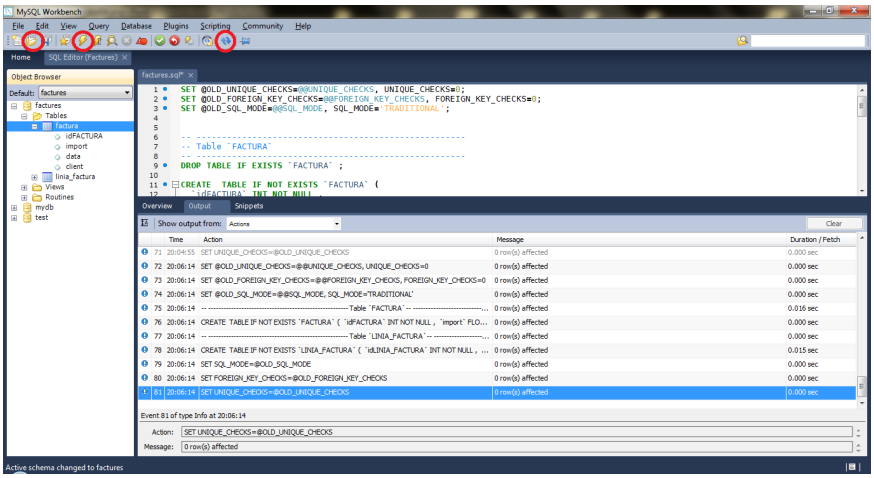
FIGURA 3.33. Creació d'un esquema nou. Pas 3



3.3.3 Importació d'una BD a partir d'un script SQL de creació

Ara es pot importar en l'esquema creat els objectes d'una base de dades a partir d'un script SQL de creació. Podeu fer-ho seleccionant, primer, que l'esquema per defecte sigui el nou que heu creat i, tot seguit, mitjançant la icona **Open** a SQL Script file, Execute SQL Script... i, finalment, fent un Refresh de l'esquema.

FIGURA 3.34. Importació d'un script SQL



3.3.4 Execució de sentències SQL

Una vegada es té una connexió a una BD, estem en disposició d'executar sentències SQL sobre ella.

FIGURA 3.35. Execució de sentències SQL. Inserció

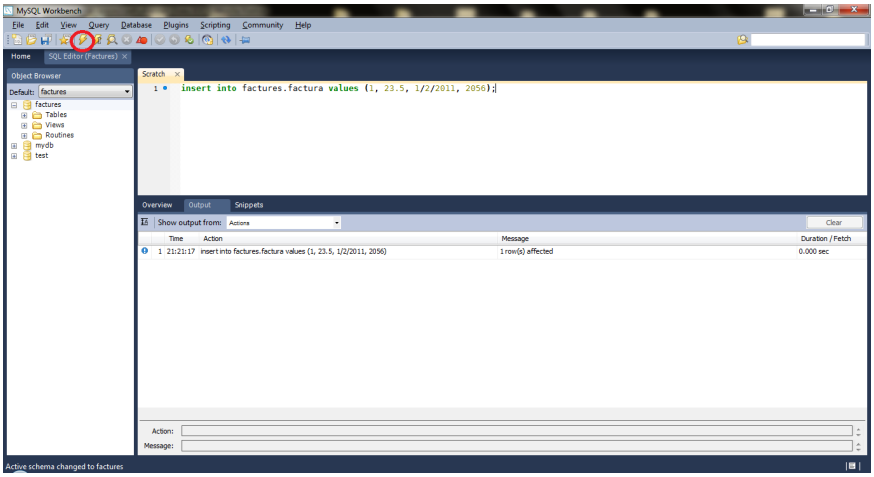
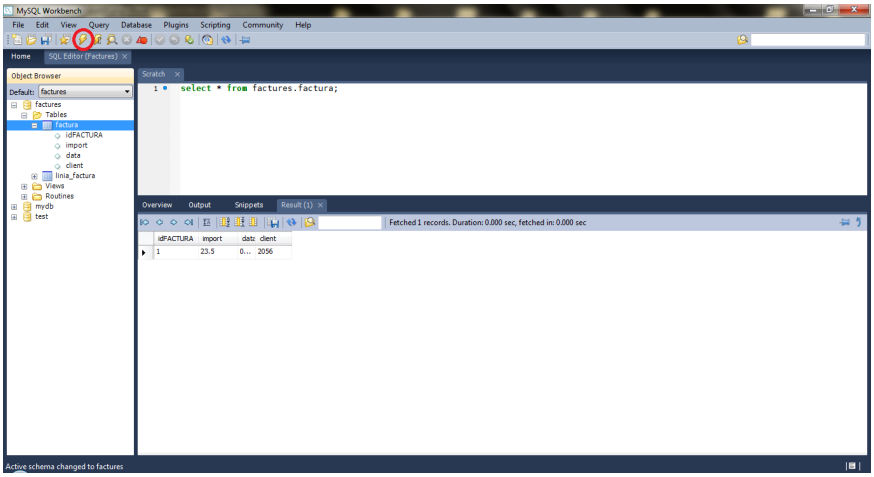


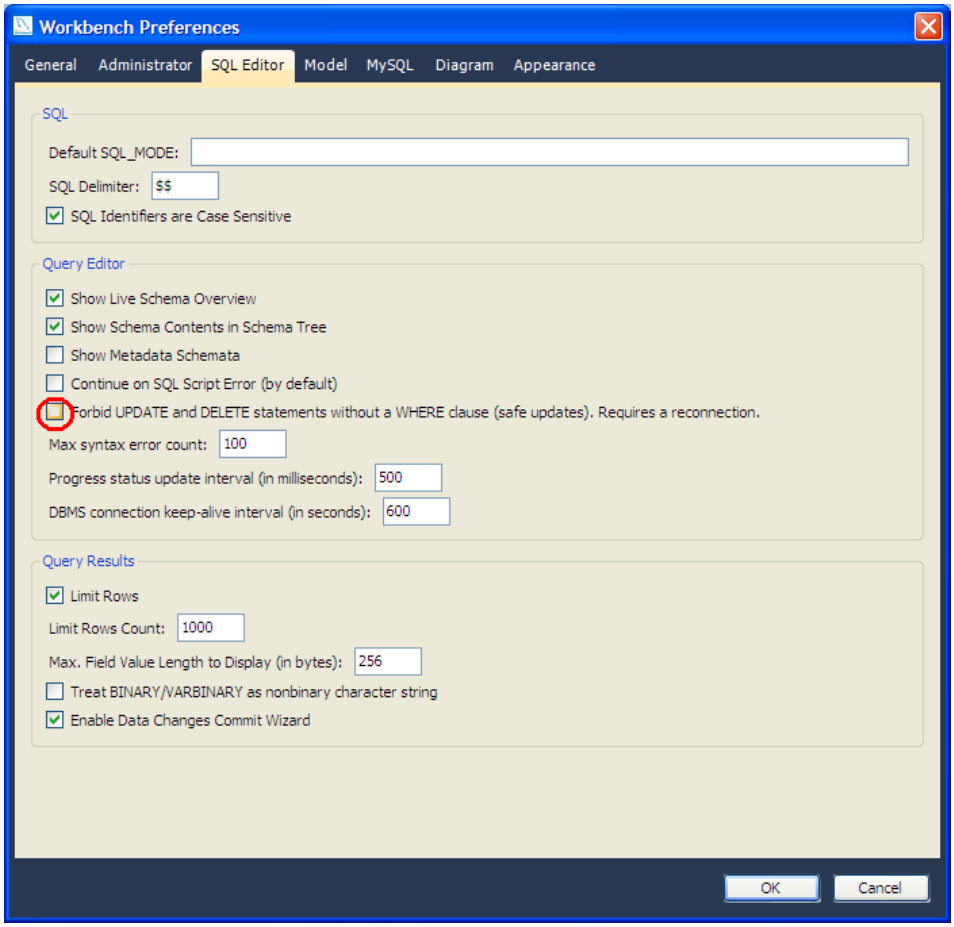
FIGURA 3.36. Execució de sentències SQL. Selecció



3.3.5 Resolució de problemes

En cas d'haver d'executar determinats tipus de sentències SQL, cal que estigui permesa l'execució d'actualització i esborrat amb clàusules **WHERE** complexes. Per aquest motiu, caldrà accedir a l'apartat **Edit**, opció de **Preferences...**, pestanya **SQL Editor** i deseleccionar l'opció indicada en la figura 3.37, i, a continuació, reiniciar la connexió a la BD.

FIGURA 3.37. Permetre actualització i esborrament amb clàusules WHERE



Llenguatge SQL per a la manipulació i definició de les dades. Control de transaccions i concurrència

Cristina Obiols Llopart
Adaptació de continguts: Isidre Guixà Miranda i Cristina Obiols Llopart

Índex

Introducció

Sobre les bases de dades no solament hi hem d'aplicar instruccions per tal d'extreure'n informació, sinó que és necessari poder manipular la informació enregistrada (afegint-ne de nova, eliminant-ne i modificant-ne la ja introduïda). Per això en aquesta unitat aprendrem les instruccions SQL per a la manipulació de dades d'una base de dades.

Així mateix, cal disposar d'algun mecanisme per definir taules, vistes, índexs i altres objectes que conformen la base de dades, i també per modificar-ne l'estructura si és necessari. I, per descomptat, també és molt important poder controlar l'accés a la informació que hi ha en la base de dades. El llenguatge SQL ens proporciona sentències per assolir tots aquests objectius.

En un SGBD en explotació, s'acostuma a encomanar a l'administrador de l'SGBD la definició de les estructures de dades. Però això no treu que tot informàtic –tant si és especialitzat en desenvolupament d'aplicacions informàtiques o en administració de sistemes informàtics– ha de conèixer les principals sentències que proporciona el llenguatge SQL per a la definició de les estructures de dades. Penseu que una persona que desenvolupi aplicacions ha de ser capaç de crear l'estructura de la base de dades (taules, vistes, índexs...).

Així, doncs, començarem coneixent les diverses possibilitats per manipular la informació, per continuar definint les estructures que permeten emmagatzemar les dades i acabar controlant el concepte de *transacció* i també les eines per controlar-les i per controlar l'accés concurrent a les dades.

Concretament, en l'apartat d'"Instruccions per a la manipulació de dades", aprendreu les instruccions per a afegir i eliminar files, modificar dades, reemplaçar files i treballar amb XML en MySQL.

En l'apartat "DDL" començareu coneixent els diversos motors d'emmagatzematge que ofereix MySQL per a tot seguit començar a veure les diferents instruccions per a crear i eliminar taules de la base de dades, així com modificar-ne l'estructura. La definició de vistes, la creació d'índexs, el canvi de noms de taules o l'eliminació de totes les files d'una taula també són accions que es poden dur a terme a través d'instruccions en MySQL que aprendrem en aquest apartat.

En l'apartat "Control de transaccions i concurrències" es defineix el concepte de transacció i de bloqueig i es descriu el procediment per a realitzar de forma segura la concurrència d'operacions sobre les dades d'una base de dades.

Per assolir un bon coneixement del llenguatge SQL, cal que aneu reproduint al vostre ordinador tots els exemples incorporats en el text, i també les activitats i els exercicis d'autoavaluació. I, per poder-ho fer, continuarem utilitzant l'SGBD MySQL i les eines adequades seguint les instruccions del material web.

Així mateix, per aprendre a aplicar amb agilitat les tècniques de disseny en el model relacional, les quals són molt teòriques, és imprescindible efectuar totes les activitats proposades i els exercicis d'autoavaluació del material web.

Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Consulta i modifica la informació emmagatzemada en una base de dades emprant assistents, eines gràfiques i el llenguatge de manipulació de dades.
    - Identifica eines i sentències per modificar el contingut de la base de dades.
    - Formula consultes per inserir, modificar i/o eliminar dades de la base de dades.
    - Insereix en una taula dades com a resultant de l'execució d'una consulta.
    - Identifica les transaccions i el seu funcionament.
    - Controla els canvis produïts per una transacció: parcialment o totalment.
    - Identifica els efectes de les diferents polítiques de bloqueig de registres.
    - Adopta mesures per mantenir la integritat i consistència de la informació.
    - Identifica les transaccions, concurrències i la recuperació d'errades.
2. Realitza el disseny físic de bases de dades utilitzant assistents, eines gràfiques i el llenguatge de definició de dades.
    - Identifica els tipus de llenguatges per definir i manipular dades sobre un SGBDR corporatiu de manera interactiva.
    - Identifica els elements de l'estructura d'una base de dades i els defineix emprant assistents, eines gràfiques i/o el llenguatge de definició de dades (DDL), a partir del disseny de la BBDD i dels requeriments d'usuari.
    - Empra assistents, eines gràfiques i el llenguatge de definició de dades per definir l'estructura d'una base de dades sobre un SGBDR corporatiu de manera interactiva i tenint en compte les regles sintàctiques.
    - Identifica les funcions, la sintaxi i les ordres bàsiques del llenguatge SQL per definir l'estructura d'una base de dades.
    - Defineix els índex en una bases de dades per tal de millorar el rendiment del sistema gestor de bases de dades.
    - Crea, modifica i elimina sinònims a taules i vistes de la BBDD.
    - Identifica i implanta les restriccions a les taules que estan reflectides en el disseny lògic.

1. Instruccions per a la manipulació de dades

Mitjançant la sentència `SELECT` podem consultar dades, però, com les podem manipular, definir i controlar?

El llenguatge SQL aporta un seguit d'instruccions amb les quals es poden realitzar les accions següents:

- La manipulació de les dades (instruccions LMD que ens han de permetre efectuar altes, baixes i modificacions).
- La definició de dades (instruccions LDD que ens han de permetre crear, modificar i eliminar les taules, els índexs i les vistes).
- El control de dades (instruccions LCD que ens han de permetre gestionar els usuaris i els seus privilegis).

Acrònims

Recordem els acrònims per als diferents apartats del llenguatge SQL: LC (llenguatge de consulta); LMD (llenguatge de manipulació de dades); LDD (llenguatge de definició de dades); LCD (llenguatge de control de dades).

El llenguatge SQL proporciona un conjunt d'instruccions, reduït però molt potent, per manipular les dades, dins el qual s'ha de distingir entre dos tipus d'instruccions:

- Les instruccions que permeten executar la manipulació de les dades, i que es redueixen a tres: `INSERT` per a la introducció de noves files, UPDATE per a la modificació de files, i `DELETE` per l'esborrament de files.
- Les instruccions per al control de transaccions, que han de permetre assegurar que un conjunt d'operacions de manipulació de dades s'executi amb èxit en la seva totalitat o, en cas de problema, s'avorti totalment o fins a un determinat punt en el temps.

Abans d'introduir-nos en l'estudi de les instruccions INSERT, UPDATE i DELETE, cal conèixer com l'SGBD gestiona les instruccions d'inserció, eliminació i modificació que hi puguem executar, ja que hi ha dues possibilitats de funcionament:

- Que quedin automàticament validades i no hi hagi possibilitat de tirar enrere. En aquest cas, els efectes de tota instrucció d'actualització de dades que tingui èxit són automàticament accessibles des de la resta de connexions de la base de dades.
- Que quedin en una cua d'instruccions, que permet tirar enrere. En aquest cas, es diu que les instruccions de la cua estan pendents de validació, i l'usuari ha d'executar, quan ho creu convenient, una instrucció per validar-les (anomenada *COMMIT*) o una instrucció per tirar enrere (anomenada *ROLLBACK*).

Aquest funcionament implica que els efectes de les instruccions pendents de validació no es veuen per la resta de connexions de la base de dades, però sí són accessibles des de la connexió on s'han efectuat. En executar la COMMIT, totes les connexions accedeixen als efectes de les instruccions validades. En cas d'executar ROLLBACK, les instruccions desapareixen de la cua i cap connexió (ni la pròpia ni la resta) no accedeix als efectes corresponents, és a dir, és com si mai haguessin existit.

Aquests possibles funcionaments formen part de la gestió de transaccions que proporciona l'SGBD i que cal estudiar amb més deteniment. A l'hora, però, d'executar instruccions INSERT, UPDATE i DELETE hem de conèixer el funcionament de l'SGBD per poder actuar en conseqüència.

Així, per exemple, un SGBD MySQL funciona amb validació automàtica després de cada instrucció d'actualització de dades llevat que s'indiqui el contrari i, en canvi, un SGBD Oracle funciona amb la cua d'instruccions pendents de confirmació o rebuig que ha d'indicar l'usuari.

En canvi, en MySQL, si es vol desactivar l'opció d'autocommit que hi ha per defecte, caldrà executar la instrucció següent:

```
1 SET autocommit=0;
```

1.1 Sentència INSERT

La sentència INSERT és la instrucció proporcionada pel llenguatge SQL per inserir noves files en les taules.

Admet dues sintaxis:

1. Els valors que s'han d'inserir s'expliciten en la mateixa instrucció en la clàusula `values`:

```
1 insert into <nom_taula> [(col1, col2...)]
2 values (val1, val2...);
```

2. Els valors que s'han d'inserir s'aconsegueixen per mitjà d'una sentència `SELECT`:

```
1 insert into <nom_taula> [(col1, col2...)]
2 select...;
```

En tot cas, es poden especificar les columnes de la taula què s'han d'emplenar i l'ordre en què se subministren els diferents valors. En cas que no s'especifiquin les columnes, l'SQL entén que els valors se subministren per a totes les columnes de la taula i, a més, en l'ordre en què estan definits en la taula.

La llista de valors de la clàusula `values` i la llista de resultats de la sentència SELECT han de coincidir en nombre, tipus i ordre amb la llista de columnes que s'han d'emplenar.

Exemple 1 de sentència INSERT

En l'esquema *empresa*, es demana inserir el departament 50 de nom 'INFORMÀTICA'.

La possible sentència per aconseguir l'objectiu és aquesta:

```
insert into dept (dept_no, dnom)
values (50, 'INFORMÀTICA');
```

Si executem una consulta per comprovar el contingut actual de la taula DEPT, trobarem la nova fila sense localitat assignada. L'SGBD ha permès deixar la localitat amb valor NULL perquè ho té permès així, com es pot veure en el descriptor de la taula DEPT:

```
SQL> desc dept;

Name                 Null     Type
-------------------- -------- ---------------------------------
DEPT_NO              NOT NULL NUMBER(2)
DNOM                 NOT NULL VARCHAR2(14)
LOC                           VARCHAR2(14)

3 rows selected
```

Exemple 2 de sentència INSERT

En l'esquema *sanitat*, es demana donar d'alta el doctor de codi 100 i nom 'BARRUFET D.'.

La solució sembla que podria ser aquesta:

```
insert into doctor (doctor_no, cognom)
values (100, 'BARRUFET D.');
```

En executar aquesta sentència, l'SGBDR dóna un error.

El cert és que la taula DOCTOR no admet valors nuls en la columna hospital_cod, ja que aquesta columna forma part de la clau primària. Mirem el descriptor de la taula DOCTOR:

```
SQL> desc doctor;

Name                 Null     Type
-------------------- -------- ---------------
HOSPITAL_COD         NOT NULL NUMBER(2)
DOCTOR_NO            NOT NULL NUMBER(3)
COGNOM               NOT NULL VARCHAR2(13)
ESPECIALITAT         NOT NULL VARCHAR2(16)

4 rows selected
```

A banda de la columna hospital_cod, també hauríem de donar un valor en la columna especialitat, ja que tampoc no admet valors nuls.

Recordem que, en el nostre esquema *sanitat*, la columna especialitat és una cadena que no té cap tipus de restricció definida ni és clau forana de cap taula en la qual hi hagi totes les especialitats possibles. Per tant, si volem saber quines especialitats hi ha per tal d'escriure la del doctor que volem inserir, idènticament a les ja introduïdes en cas que hi hagués algun doctor amb la mateixa especialitat del que hi volem inserir, fem el següent:

```
SQL> select distinct especialitat from doctor;

ESPECIALITAT

Urologia
```

```
Pediatria
Cardiologia
Neurologia
Ginecologia
Psiquiatria

6 rows selected
```

Suposem que el doctor 'BARRUFET D.' és psiquiatre. Com que ja hi ha algun doctor amb l'especialitat 'Psiquiatria', correspondria fer la inserció utilitzant la mateixa grafia per a l'especialitat. A més, suposem que volem donar d'alta el doctor a l'hospital 66.

```
insert into doctor (doctor_no, cognom, hospital_cod, especialitat
    )
values (100, 'BARRUFET D.', 66, 'Psiquiatria');
```

Aquesta vegada, l'SGBD també se'ns queixa amb un altre tipus d'error: ha fallat la referència a la clau forana.

L'error ens informa que una restricció d'integritat definida en la taula ha intentat ser violada i, per tant, la instrucció no ha finalitzat amb èxit. L'SGBD ens passa dues informacions perquè tinguem pistes d'on hi ha el problema:

- Ens dóna una descripció breu del problema (no es pot afegir una fila filla -*child row*-), la qual ens dóna a entendre que es tracta d'un error de clau forana, és a dir, que no existeix el codi en la taula referenciada.
- Ens diu la restricció que ha fallat (`sanitat.doctor, CONSTRAINT ... FOREIGN KEY (HOSPITA_COD) ...`).

L'SGBD té tota la raó. Recordem que la columna hospital_cod de la taula HOSPITAL és clau forana de la taula HOSPITAL. Això vol dir que qualsevol inserció en la taula DOCTOR ha de ser per a hospitals existents en la taula HOSPITAL, i això no passa amb l'hospital 66, com es pot veure en consultar els hospitals existents:

```
SQL> select * from hospital;

HOSPITAL_COD NOM        ADREÇA               TELÈFON  QTAT_LLITS
──────────── ────────── ──────────────────── ──────── ──────────
13           Provincial O Donell 50           964-4264 88
18           General    Atocha s/n            595-3111 63
22           La Paz     Castellana 1000        923-5411 162
45           San Carlos Ciudad Universitaria 597-1500 92

4 rows selected
```

Així, doncs, o ens hem equivocat d'hospital o hem de donar d'alta prèviament l'hospital 66. Suposem que és el segon cas i que, per tant, hem de donar d'alta l'hospital 66:

```
insert into hospital (hospital_cod, nom, adreca)
values (66,'General','De la font, 13');
```

L'SGBD ens accepta la instrucció. Fixem-nos que hem informat del codi d'hospital, del nom i de l'adreça. Mirem el descriptor de la taula HOSPITAL:

```
SQL> desc hospital;

Name               Null     Type
─────────────────── ──────── ──────────────────────────
HOSPITAL_COD       NOT NULL NUMBER(2)
NOM                NOT NULL VARCHAR2(10)
ADRECA        VARCHAR2(20)
TELEFON                     VARCHAR2(8)
QTAT_LLITS                  NUMBER(3)

5 rows selected
```

Hi veiem cinc camps, dels quals només els dos primers tenen marcada l'obligatorietat de valor. Per tant, no se'ns ha queixat perquè no hàgim indicat el telèfon de l'hospital ni la quantitat de llits que té l'hospital.

Comprovem la informació que ara hi ha en la taula `HOSPITAL`:

```
SQL> select * from hospital;

HOSPITAL_COD   NOM        ADREÇA               TELÈFON
    QTAT_LLITS
————————————— ———————————— ———————————————————— ————————
    ————————————
13             Provincial O Donell 50          964-4264 88
18             General    Atocha s/n           595-3111 63
22             La Paz     Castellana 1000      923-5411 162
45             San Carlos Ciudad Universitaria 597-1500 92
66             General    De la font, 13                0

5 rows selected
```

Sorpresa! Per al nou hospital, la columna telèfon no té valor (valor NULL), però la columna qtatllits té el valor 0. D'on ha sortit? Això es deu al fet que la columna qtatllits de la taula `HOSPITAL` té definit el valor per defecte (0) que l'SGBD utilitza per emplenar la columna qtat_llits quan es produeix una inserció en la taula sense indicar valor per a aquesta columna.

Ara sembla que ja hi podem inserir el nostre doctor 'BARRUFET D'.:

```
insert into doctor (doctor_no, cognom, hospital_cod, especialitat
    )
values (100, 'BARRUFET D.', 66, 'Psiquiatria');
```

No ens oblidem d'enregistrar els canvis amb la instrucció `COMMIT` o de fer `ROLLBACK`, si tenim l'autocommit desactivat.

Exemple 3 de sentència INSERT

Abans de començar, desactivarem l'autocommit que té configurat per defecte MySQL per poder practicar el `commit` i el `rollback`:

```
SET AUTOCOMMIT=0;
```

En l'esquema *empresa*, es vol inserir la instrucció identificada pel número 1.000, amb data d'ordre l'1 de setembre de 2000 i per al client 500.

Potser ens cal conèixer, en primer lloc, el descriptor de la taula `COMANDA`:

```
SQL> desc comanda;

Name                Null     Type
——————————————————— ———————— ——————————————————————————
COM_NUM             NOT NULL NUMBER(4)
COM_DATA                     DATE
COM_TIPUS                    VARCHAR2(1)
CLIENT_COD          NOT NULL NUMBER(6)
DATA_TRAMESA                 DATE
TOTAL                        NUMBER(8,2)

6 rows selected
```

Fixem-nos que tenim la informació corresponent a tots els camps obligatoris. Per tant, podem executar el següent:

```
insert into comanda (com_num, com_data, client_cod)
values (1000,'2000/09/01',500);
```

L'SGBD ens reporta l'error de restricció d'integritat sobre la clau forana. I, per descomptat, l'SGBD torna a tenir raó, ja que en l'esquema *empresa* la taula COMANDA té una restricció de clau forana en la columna client_cod.

Si consultem el contingut de la taula client, veurem que no hi ha cap client amb codi 500. Per això, l'SGBD ha donat un error. Suposem que era un error nostre i l'ordre corresponia al client 109 (que sí existeix en la taula CLIENT). Aquesta vegada la instrucció següent no ens dóna cap problema.

```
insert into comanda (com_num, com_data, client_cod)
values (1000,'2000/09/01',109);
```

Podem comprovar com ha quedat inserida l'ordre:

```
SQL> select * from comanda where com_num=1000;

COM_NUM    COM_DATA      COM_TIPUS CLIENT_COD DATA_TRAMESA  TOTAL
---------- ------------- --------- ---------- ------------- -----
1000       01/09/2000               109
```

Fem rollback per tirar enrere la inserció efectuada i així poder comprovar que també la podríem fer de diferents maneres. Recordem que no és obligatori indicar les columnes per a les quals s'introdueixen els valors. En aquest cas, l'SGBD espera totes les columnes de la taula en l'ordre en què estan definides en la taula. Així, doncs, podem fer el següent:

```
insert into comanda
values (1000,'2000/09/01',NULL,109,NULL,NULL);
```

Fem rollback per provar una altra possibilitat. Fixem-nos que l'SGBD també ens deixa introduir un preu total d'ordre qualsevol:

```
rollback;

insert into comanda
values (1000,DATE '2000-09-01',NULL,109,NULL,9999);

commit;
```

Disparadors

Un disparador és un conjunt d'instruccions que s'executen automàticament davant un esdeveniment determinat. Així, podem controlar que, en inserir, esborrar o modificar files de detall d'una ordre, l'import total de l'ordre s'actualitzi automàticament.

L'SGBDR ha acceptat aquesta sentència i hi ha inserit la fila corresponent. Però, hem introduït un import total que no es correspon amb la realitat, ja que no hi ha cap línia de detall. És a dir, el valor 9999 no és vàlid! Els SGBD proporcionen mecanismes (disparadors) per controlar aquests tipus d'incoherències de les dades.

Exemple 4 de sentència INSERT

En primer lloc, tornem a activar l'opció d'autocommit per tal que sigui més còmoda la feina.

```
SET AUTOCOMMIT=1;
```

Com a detall de l'ordre 1000 inserida en l'exemple anterior, en l'esquema *empresa* es volen inserir les mateixes línies que conté l'ordre 620.

En aquest cas, executarem una instrucció INSERT prenent com a valors que cal inserir els que ens dóna el resultat d'una sentència SELECT:

```
insert into detall
select 1000, detall_num, prod_num, preu_venda, quantitat, import
from detall
where com_num=620;
```

En aquesta instrucció, hem seleccionat les files de detall de l'ordre 620 i les hem inserit com a files de detall de l'ordre 1000. Hem de ser conscients que l'import total de l'ordre 1000 continua sent, però, incorrecte.

Com ja hem comentat, hem utilitzat una sentència SELECT per inserir valors en una taula. És una coincidència que totes dues sentències actuïn sobre la mateixa taula DETALL.

En no indicar, en la sentència INSERT, les columnes en què s'han d'inserir els valors, ha calgut construir la sentència SELECT de manera que les columnes de la clàusula `select` coincidissin, en ordre, amb les columnes de la taula en què s'ha d'efectuar la inserció. A més, com que per a totes les files de l'ordre 620 calia indicar 1000 com a número d'ordre, la clàusula `SELECT` ha incorporat la constant 1000 com a valor per a la primera columna.

1.2 Sentència UPDATE

La sentència `UPDATE` és la instrucció proporcionada pel llenguatge SQL per modificar files que hi ha en les taules.

La seva sintaxi és aquesta:

```
update <nom_taula>
set col1=val1, col2=val2, col3=val3...
[where <condició>];
```

La clàusula optativa `where` selecciona les files que s'han d'actualitzar. En cas d'inexistència, s'actualitzen totes les files de la taula.

La clàusula `set` indica les columnes que s'han d'actualitzar i el valor amb què s'actualitzen.

El valor d'actualització d'una columna pot ser el resultat obtingut per una sentència SELECT que recupera una única fila:

```
update <nom_taula>
set col1=(select exp1 from ... ),
set col2=(select exp2 from ... ),
set col3=val3,
...
[where <condició>];
```

En tals situacions, la sentència `SELECT` és una subconsulta de la sentència `UPDATE` que pot utilitzar valors de les columnes de la fila que s'està modificant en la sentència `UPDATE`.

Com que de vegades és possible que calgui actualitzar els valors de més d'una columna a partir de diferents resultats d'una mateixa sentència `SELECT`, no seria gens eficient executar diverses vegades la mateixa sentència `SELECT` per actualitzar més d'una columna. Per tant, la sentència `UPDATE` també admet la sintaxi següent:

```
update <nom_taula>
set (col1, col2)=(select exp1, exp2 from ... ),
set col3=val3,
...
[where <condició>];
```

Exemple 1 de sentència UPDATE

En l'esquema *empresa*, es vol modificar la localitat dels departaments de manera que quedin tots els caràcters amb minúscules.

La instrucció per resoldre la sol·licitud pot ser aquesta:

```
update dept
set loc = lower(loc);
```

Ara es vol modificar la localitat dels departaments de manera que quedin amb la inicial en majúscula i la resta de lletres amb minúscules.

La instrucció per resoldre la sol·licitud pot ser aquesta:

```
update dept
set loc=concat(upper(left(loc,1)),right(lower(loc),length(loc)-1)
    );
```

Exemple 2 de sentència UPDATE

En l'esquema *empresa*, es vol actualitzar l'import total real de la comanda 1000 a partir dels imports de les diferents línies de detall que formen la comanda.

```
update comanda c
set total = (select sum(import) from detall
     where com_num=c.com_num)
where com_num=1000;
```

Ara podem comprovar la correcció de la informació que hi ha en la base de dades sobre la comanda 1000:

```
SQL> select * from detall where com_num=1000;

COM_NUM    DETALL_NUM PROD_NUM   PREU_VENDA QUANTITAT  IMPORT
---------- ---------- ---------- ---------- ---------- ----------

1000       1          100860     35         10         350
1000       2          200376     2,4        1000       2400
1000       3          102130     3,4        500        1700

3 rows selected

SQL> select * from comanda where com_num=1000;

COM_NUM    COM_DATA   COM_TIPUS CLIENT_COD DATA_TRAMESA  TOTAL
---------- ---------- ---------- ---------- ------------- -----
1000       01/09/2000           109                      4450

1 rows selected
```

1.3 Sentència DELETE

La sentència DELETE és la instrucció proporcionada pel llenguatge SQL per esborrar files existents que hi ha en les taules.

La seva sintaxi és aquesta:

```
delete from <nom_taula>
[where <condició>];
```

La clàusula optativa `where` selecciona les files que s'han d'eliminar. Si no n'hi ha, s'eliminen totes les files de la taula.

Exemple de sentència DELETE

En l'esquema *empresa*, es vol eliminar la comanda 1000.

La instrucció sembla que podria ser aquesta:

```
delete from comanda
where com_num=1000;
```

En executar aquesta sentència, però, ens trobem amb un error que ens indica que no es pot eliminar una fila pare (*a parent row*).

El motiu és que la columna com_num de la taula DETALL és clau forana de la taula COMANDA, fet que impossibilita eliminar una capçalera de comanda si hi ha línies de detall corresponents. Aquestes s'eliminarien de manera automàtica si hi hagués definida l'eliminació en cascada, però no és el cas. Així, doncs, caldrà fer el següent:

```
delete from detall where com_num=1000;
delete from comanda where com_num=1000;
```

1.4 Sentència REPLACE

MySQL té una extensió del llenguatge SQL estàndard que permet inserir una nova fila que, en cas que la clau primària coincideixi amb una altra fila, prèviament sigui eliminada. Es tracta de la sentència REPLACE.

Hi ha tres possibles sintaxis per a la sentència **REPLACE**:

```
REPLACE [INTO] nom_taula [(columna1,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...

REPLACE [INTO] nom_taula
    SET columna1={expr | DEFAULT}, ...

REPLACE [INTO] nom_taula [(columna1,...)]
    SELECT ...
```

1.5 Sentència LOAD XML

LOAD XML permet llegir un fitxer en format xml i emmagatzemar les dades contingudes en una taula de la base de dades. La seva sintaxi és:

XML

XML és l'acrònim d'*extensible markup language* ('llenguatge d'etiquetatge extensible') un metallenguatge de marques que facilita l'organització de les dades en fitxers plans.

```
LOAD XML [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'nom_fitxer'
[REPLACE | IGNORE]
INTO TABLE [nom_base_dades.]nom_taula
[CHARACTER SET nom_charset]
[ROWS IDENTIFIED BY '<nom_tag>']
[IGNORE número [LINES | ROWS]]
[(columnes,...)]
[SET nom_columna = expressió,...]
```

2. DDL

A banda de les conegudes instruccions per consultar i modificar les dades, el llenguatge SQL aporta instruccions per definir les estructures en què s'emmagatzemen les dades. Així, per exemple, tenim instruccions per a la creació, eliminació i modificació de taules i índexs, i també instruccions per definir vistes.

En el MySQL, l'SQLServer i el PostgreSQL qualsevol instància de l'SGBD gestiona un conjunt de bases de dades o esquemes, anomenat *cluster database*, el qual pot tenir definit un conjunt d'usuaris amb els privilegis d'accés i gestió que corresponguin.

En el MySQL, l'SQLServer i el PostgreSQL, el llenguatge SQL proporciona una instrucció `CREATE DATABASE <nom_base_dades>` que permet crear, dins la instància, les diverses bases de dades. Aquesta instrucció CREATE DATABASE es pot considerar dins l'àmbit del llenguatge LDD.

Distinció entre àmbits LDD i LCD en el llenguatge SQL

Sovint, els àmbits LDD (llenguatge de definició de dades) i LCD (llenguatge per al control de les dades) es fonen en un únic àmbit i es parla únicament d'LDD.

La sentència CREATE SCHEMA en l'àmbit del llenguatge LDD està destinada a la creació d'un esquema en què es puguin definir taules, índexs, vistes, etc. En MySQL, CREATE SCHEMA i CREATE TABLE són sinònims.

Disposem, també, de la instrucció `USE <nom_base_dades>` per decidir la base de dades en què es treballarà (establiment de la base de dades de treball per defecte).

2.1 Regles i indicacions per anomenar objectes en MySQL

Dins de MySQL, a banda de taules, trobarem altres tipus d'objectes: índexs, columnes, àlies, vistes, procediments, etc.

Els noms dels objectes dins d'una base de dades, i les bases de dades mateixes, en MySQL actuen com a identificadors, i, com a tals no es podran repetir en un mateix àmbit. Per exemple, no podem tenir dues columnes d'una mateixa taula que s'anomenin igual, però sí en taules diferents.

> Els noms amb què anomenem els objectes dins d'un SGBD hauran de seguir unes regles sintàctiques que hem de conèixer.

En general, les taules i les bases de dades són *not case sensitive*, és a dir, que hi podem fer referència en majúscules o minúscules i no hi trobarem diferència, si el sistema operatiu sobre el qual estem treballant suporta *not case sensitive*.

Per exemple, en Windows podem executar indiferentment:

```
1 select * from emp;
```

O bé:

```
select * from EMP;
```

El que no acostumem a fer, però, és que dins d'una mateixa sentència ens referim a un mateix objecte en majúscules i en minúscules a la vegada:

```
select * from emp where EMP.EMP_NO=7499;
```

Els noms de columnes, índexs, procediments i disparadors (*triggers*), en canvi, sempre són *not case sensitive*.

Els noms dels objectes en MySQL admeten qualsevol tipus de caràcter, excepte / \ i .

De totes maneres, es recomana utilitzar caràcters alfabètics estrictament. Si el nom inclou caràcters especials és obligatori fer-hi referència entre cometes del tipus accent greu (`). Per exemple:

```
create table `ES UNA PROVA` (a int);
```

S'admeten també les cometes dobles (") si activem el mode `ANSI_QUOTES`:

```
SET sql_mode='ANSI_QUOTES';
create table "ES UNA ALTRA PROVA" (a int);
```

Qualsevol objecte pot ser referit utilitzant les cometes, encara que no calgui, com ara en l'exemple:

```
select * from `empresa`.`emp` where `emp`.`emp_no`=7499;
```

Tot i que no és recomanable, es poden anomenar objectes amb paraules reservades del mateix llenguatge com ara SELECT, INSERT, DATABASE, etc. Aquests noms, però, s'hauran de posar obligatòriament entre cometes.

La longitud màxima dels objectes de la base de dades és 64 caràcters, excepte per als àlies, que poden arribar a ser de 256.

Finalment, vegem algunes indicacions per anomenar objectes:

- **Utilitzar noms sencers, descriptius i pronunciables i, si no és factible, bones abreviatures**. En anomenar objectes, sospeseu l'objectiu d'aconseguir noms curts i fàcils d'utilitzar davant l'objectiu de tenir noms que siguin descriptius. En cas de dubte, escolliu el nom més descriptiu, ja que els objectes de la base de dades poden ser utilitzats per molta gent al llarg del temps.
- **Utilitzar regles d'assignació de noms que siguin coherents**. Així, per exemple, una regla podria consistir a començar amb gc_ tots els noms de les taules que formen part d'una gestió comercial.
- **Utilitzar el mateix nom per descriure la mateixa entitat o el mateix atribut en diferents taules**. Així, per exemple, quan un atribut d'una taula

és clau forana d'una altra taula, és molt convenient anomenar-lo amb el nom que té en la taula principal.

2.2 Comentaris en MySQL

El servidor MySQL suporta tres estils de comentaris:

- # fins al final de la línia.
- `-` `<espai en blanc>` fins al final de la línia.
- `/* fins a la propera seqüència */`. Aquests tipus de comentaris admeten diverses línies de comentari.

Exemples dels diferents tipus de comentaris són els següents:

```
SELECT 1+1; # Aquest és el primer tipus de comentari

SELECT 1+1; -- Aquest és el segon tipus de comentari

SELECT 1 /* Aquest és un tipus de comentari que es pot posar enmig de la línia
    */ + 1;

SELECT 1+
/*
Aquest és un
comentari
que es pot posar
en diverses línies*/
1;
```

2.3 Motors d'emmagatzematge en MySQL

MySQL suporta diferents tipus d'emmagatzematge de taules (motors d'emmagatzemament o *storage engines*, en anglès). I quan es crea una taula cal especificar en quin sistema dels possibles el volem crear.

Per defecte, MySQL a partir de la versió 5.5.5 crea les taules de tipus **InnoDB**, que és un sistema transaccional, és a dir, que suporta les característiques que fan que una base de dades pugui garantir que les dades es mantindran consistents.

Les propietats que garanteixen els sistemes transaccionals són les característiques anomenades `ACID`. *ACID* és l'acrònim anglès d'*atomicity, consistency, isolation, durability*:

- **Atomicitat**: es diu que un SGBD garanteix atomicitat si qualsevol transacció o bé finalitza correctament (*commit*), o bé no deixa cap rastre de la seva execució (*rollback*).

- **Consistència**: es parla de consistència quan la concurrència de diferents transaccions no pot produir resultats anòmals.
- **Aïllament (o isolament)**: cada transacció dins del sistema s'ha d'executar com si fos l'única que s'executa en aquell moment.
- **Definitivitat**: si es confirma una transacció, en un SGBD, el resultat d'aquesta ha de ser definitiu i no es pot perdre.

Només el motor **InnoDB** permet crear un sistema transaccional en MySQL. Els altres tipus d'emmagatzemament no són transaccionals i no ofereixen control d'integritat a les bases de dades creades.

Evidentment, aquest sistema (**InnoDB**) d'emmagatzematge és el que sovint interessarà utilitzar per a les bases de dades que creem, però hi pot haver casos en què sigui interessant considerar altres tipus de motors d'emmagatzematge. Per això, MySQL també ofereix altres sistemes com ara, per exemple:

- **MyISAM**: era el sistema per defecte abans de la versió 5.5.5 de MySQL. S'utilitza molt en aplicacions web i en aplicacions de magatzem de dades (*datawarehousing*).
- **Memory**: aquest sistema emmagatzema tot en memòria RAM i, per tant, s'utilitza per a sistemes que requereixin un accés molt ràpid a les dades.
- **Merge**: agrupa taules de tipus MyISAM per optimitzar llistes i cerques. Les taules que cal agrupar han de ser de semblants, és a dir, han de tenir el mateix nombre i tipus de columnes.

Per obtenir una llista dels motors d'emmagatzemament suportats per la versió MySQL que tingueu instal·lada, podeu executar l'ordre `SHOW ENGINES`.

2.4 Creació de taules

La sentència `CREATE TABLE` és la instrucció proporcionada pel llenguatge SQL per a la creació d'una taula.

Recordeu que els elements que es posen entre claudàtors (`[ ]`) són opcionals.

És una sentència que admet múltiples paràmetres, i la sintaxi completa es pot consultar en la documentació de l'SGBD que correspongui, però la sintaxi més simple i usual és aquesta:

```
1 create table [<nom_esquema>.]<nom_taula>
2 ( <nom_columna> <tipus_dada> [default <expressió>][<
      llista_restriccions_pera_a_la_columna>],
3 <nom_columna> <tipus_dada> [default <expressió>][<
      llista_restriccions_per_a_la_columna>],
4 ...
5 [<llista_restriccions_addicionals_per_a_una_o_vàries_columnes>]);
```

Fixem-nos que hi ha força elements que són optatius:

- Les parts obligatòries són el nom de la taula i, per cada columna, el nom i el tipus de dada.
- El nom de l'esquema en què es crea la taula és optatiu i, si no s'indica, la taula s'intenta crear dins l'esquema en què estem connectats.
- Cada columna té permès definir-hi un valor per defecte (opció default) a partir d'una expressió, el qual utilitzarà l'SGBD en les instruccions d'inserció quan no s'especifiqui un valor per a les columnes que tenen definit el valor per defecte. En MySQL el valor per defecte ha de ser constant, no pot ser, per exemple, una funció com ara `NOW()` ni una expressió com ara `CURRENT_DATE`.
- La definició de les restriccions per a una o més columnes també és optativa en el moment de procedir a la creació de la taula.

També és molt usual crear una taula a partir del resultat d'una consulta, amb la sintaxi següent:

```
1 create table [<nom_esquema>.]<nom_taula> [(<noms_dels_camps>]
2 as <sentència_select>;
```

En aquesta sentència, no es defineixen els tipus de camps que es corresponen amb els tipus de les columnes recuperades en la sentència `SELECT`. La definició dels noms dels camps és optativa; si no s'efectua, els noms de les columnes recuperades passen a ser els noms dels camps nous. Caldrà, però, afegir-hi les restriccions que corresponguin. La taula nova conté una còpia de les files resultants de la sentència `SELECT`.

A l'hora de definir taules, cal tenir en compte diversos conceptes:

- Els tipus de dades que l'SGBD possibilita.
- Les restriccions sobre els noms de taules i columnes.
- La integritat de les dades.

En l'apartat "Consultes de selecció simples" de la unitat "Llenguatge SQL. Consultes", es presenten àmpliament els tipus de dades més importants en l'SGBD MySQL.

L'SGBD MySQL proporciona diversos tipus de restriccions (*constraints* en la nomenclatura que s'ha d'utilitzar en els SGBD) o opcions de restricció per facilitar la integritat de les dades. En general, es poden definir en el moment de crear la taula, però també es poden alterar, afegir i eliminar amb posterioritat.

Cada restricció porta associat un nom (únic en tot l'esquema) que es pot especificar en el moment de crear la restricció. Si no s'especifica, l'SGBD n'assigna un per defecte.

Vegem, a continuació, els diferents tipus de restriccions:

Clau primària

Per definir la clau primària d'una taula, cal utilitzar la *constraint* `primary key`.

Si la clau primària és formada per una única columna, es pot especificar en la línia de definició de la columna corresponent, amb la sintaxi següent:

```
<columna> <tipus_dada> primary key
```

En canvi, si la clau primària és formada per més d'una columna, s'ha d'especificar obligatòriament en la zona final de restriccions sobre columnes de la taula, amb la sintaxi següent:

```
[constraint <nom_restricció>] primary key (col1,col2,...)
```

Les claus primàries que afecten una única columna també es poden especificar amb aquest segon procediment.

Obligatorietat de valor

Per definir l'obligatorietat de valor en una columna, cal utilitzar l'opció **not null**.

Aquesta restricció es pot indicar en la definició de la columna corresponent amb aquesta sintaxi:

```
<columna> <tipus_dada> [not null]
```

Per descomptat, no cal definir aquesta restricció sobre columnes que formen part de la clau primària, ja que formar part de la clau primària implica, automàticament, la impossibilitat de tenir valor nuls.

Unicitat de valor

Per definir la unicitat de valor en una columna, cal utilitzar la *constraint* **unique**.

Si la unicitat s'especifica per a una única columna, es pot assignar en la línia de definició de la columna corresponent, amb la sintaxi següent:

```
<columna> <tipus_dada> unique
```

En canvi, si la unicitat s'aplica sobre diverses columnes simultàniament, cal especificar-la obligatòriament en la zona final de restriccions sobre columnes de la taula, amb la sintaxi següent:

```
[constraint <nom_restricció>] unique (col1, col2...)
```

Aquest segon procediment també es pot emprar per aplicar la unicitat a una única columna.

Per descomptat, no cal definir aquesta restricció sobre un conjunt de columnes que formen part de la clau primària, ja que la clau primària implica, automàticament, la unicitat de valors.

Condicions de comprovació

Per definir condicions de comprovació en una columna, cal utilitzar l'opció `check (<condició>)`.

Aquesta restricció es pot indicar en la definició de la columna corresponent:

```
<columna> <tipus_dada> check (<condició>)
```

També es pot indicar en la zona final de restriccions sobre columnes de la taula, amb la sintaxi següent:

```
check (<condició>)
```

AUTO_INCREMENT

Podem definir les columnes numèriques amb l'opció `AUTO_INCREMENT`. Aquesta modificació permet que en inserir un valor null o no inserir valor explícitament en aquella columna definida d'aquesta manera, s'hi afegeixi un valor per defecte consistent en el més gran ja introduït incrementat en una unitat.

```
<columna> <tipus_dada> AUTO_INCREMENT
```

Comentaris de columnes

Podem afegir comentaris a les columnes de manera que puguin quedar guardats a la base de dades i ser consultats. La manera de fer-ho és afegir la paraula `COMMENT` seguida del text que calgui posar com a comentari entre cometes simples.

```
<columna> <tipus_dada> COMMENT 'comentari'
```

Integritat referencial

Per definir la integritat referencial, cal utilitzar la *constraint* `foreign key`.

Si la clau forana és formada per una única columna, es pot especificar en la línia de definició de la columna corresponent, amb la sintaxi següent:

```
<columna> <tipus_dada> [constraint <nom_restricció>] references <taula> [(
    columna)]
```

En canvi, si la clau forana és formada per més d'una columna, cal especificar-la obligatòriament en la zona final de restriccions sobre columnes de la taula, amb la sintaxi següent:

```
[constraint <nom_restricció>] foreign key (col1, col2...)
references <taula> [(col1, col2...)]
```

Les claus foranes que afecten una única columna també es poden especificar amb aquest segon procediment.

En qualsevol dels dos casos, es fa referència a la taula principal de la qual estem definint la clau forana, la qual cosa es fa amb l'opció `references <taula>`.

> En MySQL la integritat referencial només s'activa si es treballa sobre el motor **InnoDB** i s'utilitza la sintaxi de *constraint* de la zona de definició de restriccions del final i, a més, es defineix un índex per a les columnes implicades en la clau forana.

La sintaxi per a la integritat referencial activa en MySQL és la següent (si s'utilitzen altres sintaxis, l'SGBD les reconeix però no les valida):

```
index [<nom_index>] (col1, col2...)
[constraint <nom_restricció>] foreign key (col1, col2...) references <taula> (
    col1, col2...)
```

La sintaxi que hem presentat per tal de definir la integritat referencial no és completa. Ens falta tractar un tema fonamental: l'actuació que esperem de l'SGBD davant possibles eliminacions i actualitzacions de dades en la taula principal, quan hi ha files en altres taules que hi fan referència.

La *constraint* **foreign key** es pot definir acompanyada dels apartats següents:

- `on delete <acció>`, que defineix l'actuació automàtica de l'SGBD sobre les files de la nostra taula que es veuen afectades per una eliminació de les files a les quals fan referència.
- `on update <acció>`, que defineix l'actuació automàtica de l'SGBD sobre les files de la nostra taula que es veuen afectades per una actualització del valor al qual fan referència.

Per si no us ha quedat clar, pensem en les taules DEPT i EMP de l'esquema *empresa*. La taula EMP conté la columna dept_no, que és clau forana de la taula DEPT. Per tant, en la definició de la taula EMP hem de tenir definida una *constraint foreign key* en la columna dept_no fent referència a la taula DEPT. En definir

aquesta restricció de clau forana, el dissenyador de la base de dades va haver de prendre decisions respecte al següent:

- Com ha d'actuar l'SGBD davant l'intent d'eliminació d'un departament en la taula DEPT si hi ha files en la taula EMP que hi fan referència? Això es defineix en l'apartat `on delete <acció>`.
- Com ha d'actuar l'SGBD davant l'intent de modificació del codi d'un departament en la taula DEPT si hi ha files en la taula EMP que hi fan referència? Això es defineix en l'apartat `on update <acció>`.

En general, els SGBD ofereixen diverses possibilitats d'acció, però no sempre són les mateixes. Abans de conèixer aquestes possibilitats, també ens cal saber que alguns SGBD permeten diferir la comprovació de les restriccions de clau forana fins a la finalització de la transacció, en lloc d'efectuar la comprovació -i actuar en conseqüència- després de cada instrucció. Quan això és factible, la definició de la *constraint* va acompanyada del mot `deferrable` o `not deferrable`. L'actuació per defecte acostuma a ser no diferir la comprovació.

Per tant, la sintaxi de la restricció de clau forana es veu clarament ampliada. Si s'efectua en el moment de definir la columna, tenim el següent:

```
1 [constraint <nom_restricció>] foreign key (col1, col2, ...) references <taula>
      (col1,
2 col2, ...) [on delete <acció>] [on update <acció>]
```

Les opcions que ens podem arribar a trobar en referència a l'acció que acompanyi els apartats **on update** i **on delete** són aquestes: RESTRICT | CASCADE | SET NULL | NO ACTION

- `NO ACTION` o `RESTRICT`: són sinònims. És l'opció per defecte i no permet l'eliminació o actualització de dades en la taula principal.
- `CASCADE`: quan s'actualitza o elimina la fila pare, les files relacionades (filles) també s'actualitzen o eliminen automàticament.
- `SET NULL`: quan s'actualitza o elimina la fila pare, les files relacionades (filles) s'actualitzen a NULL. Cal haver-les definit de manera que admetin valors nuls, és clar.
- `SET DEFAULT`: quan s'actualitza o elimina la fila pare, les files relacionades (filles) s'actualitzen al valor per defecte. MySQL suporta la sintaxi, però no actua, davant d'aquesta opció.

L'opció `CASCADE` és molt perillosa en utilitzar-la amb `on delete`. Pensem què passaria, en l'esquema *empresa*, si algú decidís eliminar un departament de la taula DEPT i la clau forana en la taula EMP fos definida amb `on delete cascade`: tots els empleats del departament serien, immediatament, eliminats de la taula EMP.

De vegades, però, és molt útil acompanyant `on delete`. Pensem en la relació d'integritat entre les taules COMANDA i DETALL de l'esquema *empresa*. La

taula DETALL conté la columna com_num, que és clau forana de la taula COMANDA. En aquest cas, pot tenir molt de sentit tenir definida la clau forana amb `on delete cascade`, ja que l'eliminació d'una ordre provocarà l'eliminació automàtica de les seves línies de detall.

A diferència de la caució en la utilització de l'opció `cascade` per a les actuacions `on delete`, s'acostuma a utilitzar molt per a les actuacions `on update`.

La taula 2.1 mostra les opcions proporcionades per alguns SGBD actuals.

TAULA 2.1. Opcions de la restricció foreign key proporcionades per alguns SGBD actuals

SGBD	on update	on delete	Diferir actuació	no action	restrict	cascade	set null	set default
Oracle	No	Sí	No	Sí	No	Sí	Sí	No
MySQL	Sí	Sí	No	Sí	Sí	Sí	Sí	Sí
PostgreSQL	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
SQLServer 2005	Sí	Sí	No	Sí	No	Sí	Sí	Sí
MS-Access 2003	Sí	Sí	No	Sí	No	Sí	3	No

Exemple 1 de creació de taules. Taules de l'esquema empresa

```
CREATE TABLE IF NOT EXISTS empresa.DEPT (
 DEPT_NO  TINYINT (2) UNSIGNED,
 DNOM     VARCHAR(14) NOT NULL UNIQUE,
 LOC      VARCHAR(14),
 PRIMARY KEY (DEPT_NO) ) ;


CREATE TABLE IF NOT EXISTS empresa.EMP (
 EMP_NO    SMALLINT (4) UNSIGNED,
 COGNOM    VARCHAR (10) NOT NULL,
 OFICI     VARCHAR (10),
 CAP       SMALLINT (4) UNSIGNED,
 DATA_ALTA DATE,
 SALARI    INT UNSIGNED,
 COMISSIO  INT UNSIGNED,
 DEPT_NO   TINYINT (2) UNSIGNED NOT NULL,
 PRIMARY KEY (EMP_NO),
 INDEX IDX_EMP_CAP (CAP),
 INDEX IDX_EMP_DEPT_NO (DEPT_NO),
 FOREIGN KEY (DEPT_NO) REFERENCES empresa.DEPT(DEPT_NO)) ;
```

En primer lloc, cal destacar l'opció `IF NOT EXISTS`, opció molt utilitzada a l'hora de crear bases de dades per tal d'evitar errors en cas que la taula ja existeixi prèviament.

D'altra banda, cal destacar que la taula es defineix amb el nom de l'esquema `empresa`. Això no és necessari si prèviament es defineix aquest esquema com a esquema per defecte. Això es pot fer amb la sentència `USE`:

```
USE empresa;
```

En les dues definicions de taula, la clau primària s'ha definit en la part inferior de la sentència, no pas en la mateixa definició de la columna, malgrat que es tractava de claus primàries que només tenien una única columna.

Fixeu-vos com s'han definit dos índexs per a les columnes `CAP` i `DEPT_NO` per tal de crear *a posteriori* les claus foranes corresponents. En el moment de la creació es crea la clau

forana que fa referència de EMP a DEPT. No es crea, en canvi, la que fa referència de EMP a la mateixa taula i que serveix per referir-se al cap d'un empleat. El motiu és que si es definís aquesta clau forana en el moment de la creació de la taula, no hi podríem inserir valors fàcilment, ja que no tindria el valor de referència prèviament introduït a la taula. Per exemple, si hi volem inserir l'empleat número (EMP_NO) 7369 que té com a empleat cap el 7902 i aquest no és encara a la taula, no el podríem inserir perquè no existeix el 7902 encara a la taula. Una possible solució a aquest problema que s'anomena *interbloqueig* o *deadlock*, en anglès, és definir la clau forana *a posteriori* de les insercions. O bé, si l'SGBD ho permet, desactivar la **foreign key** abans d'inserir-la i tornar-la a activar en acabar. Així, doncs, després de les insercions caldrà modificar la taula i afegir-hi la restricció de clau forana.

Observeu, també, que s'ha utilitzat el modificador de tipus UNSIGNED per tal de definir els camps que necessàriament són considerats positius. De manera que si es fa una inserció del tipus següent, l'SGBD ens retornarà un error:

```
insert into EMP values (100, 'Rodríguez', 'Venedor', NULL,
     sysdate, -5000, NULL, 10)
```

Fixem-nos, també, en la importància de l'ordre en què es defineixen les taules, ja que no seria possible definir la integritat referencial en la columna dept_no de la taula EMP sobre la taula DEPT si aquesta encara no fos creada.

```
CREATE TABLE IF NOT EXISTS empresa.CLIENT (
 CLIENT_COD          INT(6) UNSIGNED PRIMARY KEY,
 NOM                 VARCHAR (45) NOT NULL,
 ADREÇA              VARCHAR (40) NOT NULL,
 CIUTAT              VARCHAR (30) NOT NULL,
 ESTAT               VARCHAR (2),
 CODI_POSTAL         VARCHAR (9) NOT NULL,
 AREA                SMALLINT(3),
 TELEFON             VARCHAR (9),
 REPR_COD            SMALLINT(4) UNSIGNED,
 LIMIT_CREDIT        DECIMAL(9,2) UNSIGNED,
 OBSERVACIONS        TEXT,
 INDEX IDX_CLIENT_REPR_COD (REPR_COD),
 FOREIGN KEY (REPR_COD) REFERENCES empresa.EMP(EMP_NO));
```

En aquesta taula, en canvi, la clau primària s'ha definit en la mateixa definició de la columna.

```
CREATE TABLE IF NOT EXISTS empresa.PRODUCTE (
 PROD_NUM     INT (6) UNSIGNED PRIMARY KEY,
 DESCRIPCIO   VARCHAR (30) NOT NULL  UNIQUE);

CREATE TABLE IF NOT EXISTS empresa.COMANDA(
 COM_NUM             SMALLINT(4) UNSIGNED PRIMARY KEY,
 COM_DATA            DATE,
 COM_TIPUS           CHAR (1) CHECK (COM_TIPUS IN ('A','B','C')),
 CLIENT_COD          INT (6) UNSIGNED NOT NULL,
 DATA_TRAMESA        DATE,
 TOTAL               DECIMAL(8,2) UNSIGNED,
 INDEX IDX_COMANDA_CLIENT_COD (CLIENT_COD),
 FOREIGN KEY (CLIENT_COD) REFERENCES empresa.CLIENT(CLIENT_COD) );


CREATE TABLE IF NOT EXISTS empresa.DETALL  (
 COM_NUM             SMALLINT(4) UNSIGNED,
 DETALL_NUM          SMALLINT(4) UNSIGNED,
 PROD_NUM            INT(6) UNSIGNED NOT NULL,
 PREU_VENDA          DECIMAL(8,2) UNSIGNED,
 QUANTITAT           INT (8),
 IMPORT              DECIMAL(8,2),
 CONSTRAINT DETALL_PK PRIMARY KEY (COM_NUM,DETALL_NUM),
 INDEX IDX_DETAL_COM_NUM (COM_NUM),
 INDEX IDX_PROD_NUM (PROD_NUM),
 FOREIGN KEY (COM_NUM) REFERENCES empresa.COMANDA(COM_NUM),
 FOREIGN KEY (PROD_NUM) REFERENCES empresa.PRODUCTE(PROD_NUM));
```

Exemple 2 de creació de taules. Taules de l'esquema sanitat

```
CREATE TABLE IF NOT EXISTS sanitat.HOSPITAL (
 HOSPITAL_COD   TINYINT (2) PRIMARY KEY,
 NOM            VARCHAR(10) NOT NULL,
 ADREÇA         VARCHAR(20),
 TELEFON        VARCHAR(8),
 QTAT_LLITS     SMALLINT(3) UNSIGNED DEFAULT 0 );



CREATE TABLE IF NOT EXISTS sanitat.SALA (
 HOSPITAL_COD  TINYINT (2),
 SALA_COD      TINYINT (2),
 NOM           VARCHAR(20) NOT NULL,
 QTAT_LLITS    SMALLINT(3) UNSIGNED DEFAULT 0,
 CONSTRAINT SALA_PK PRIMARY KEY (HOSPITAL_COD, SALA_COD),
 INDEX IDX_SALA_HOSPITAL_COD (HOSPITAL_COD),
 FOREIGN KEY (HOSPITAL_COD) REFERENCES sanitat.HOSPITAL(
     HOSPITAL_COD)) ;
```

Recordem que la taula es defineix amb el nom de l'esquema `sanitat`, però que això no és necessari si prèviament es defineix aquest esquema com a esquema per defecte:

```
USE sanitat;
```

La definició de la taula `SALA` necessita declarar la constraint `PRIMARY KEY` al final de la definició de la taula, ja que és formada per més d'un camp. En casos com aquest, aquesta és l'única opció i no és factible definir la constraint `PRIMARY KEY` al costat de cada columna, ja que una taula només admet una definició de constraint `PRIMARY KEY`.

```
CREATE TABLE IF NOT EXISTS sanitat.PLANTILLA (
 HOSPITAL_COD  TINYINT (2),
 SALA_COD      TINYINT (2),
 EMPLEAT_NO    SMALLINT(4) NOT NULL,
 COGNOM        VARCHAR (15) NOT NULL,
 FUNCIO        VARCHAR (10),
 TORN          VARCHAR (1) CHECK (TORN IN ('M','T','N')),
 SALARI        INT (10),
 CONSTRAINT PLANTILLA_PK PRIMARY KEY (HOSPITAL_COD, SALA_COD,
     EMPLEAT_NO),
 INDEX IDX_PLANTILLA_HOSP_SALA (HOSPITAL_COD, SALA_COD),
 FOREIGN KEY (HOSPITAL_COD, SALA_COD) REFERENCES sanitat.SALA (
     HOSPITAL_COD, SALA_COD)) ;
```

La definició de la taula `PLANTILLA` necessita declarar les restriccions `PRIMARY KEY` i `FOREIGN KEY` al final de la definició de la taula perquè ambdues fan referència a una combinació de columnes.

```
CREATE TABLE IF NOT EXISTS sanitat.MALALT (
 INSCRIPCIO    INT (5) PRIMARY KEY,
 COGNOM        VARCHAR (15) NOT NULL,
 ADREÇA        VARCHAR (20),
 DATA_NAIX     DATE,
 SEXE          CHAR (1) NOT NULL  CHECK (SEXE = 'H' OR SEXE = 'D'),
 NSS           CHAR(9)) ;

CREATE TABLE IF NOT EXISTS sanitat.INGRESSOS (
 INSCRIPCIO    INT (5) PRIMARY KEY,
 HOSPITAL_COD  TINYINT (2) NOT NULL,
 SALA_COD      TINYINT (2) NOT NULL,
 LLIT          SMALLINT(4) UNSIGNED,
 INDEX IDX_INGRESSOS_INSCRIPCIO (INSCRIPCIO),
 INDEX IDX_INGRESSOS_HOSP_SALA (HOSPITAL_COD, SALA_COD),
 FOREIGN KEY (INSCRIPCIO) REFERENCES sanitat.MALALT(INSCRIPCIO),
 FOREIGN KEY (HOSPITAL_COD, SALA_COD) REFERENCES sanitat.SALA (HOSPITAL_COD,
     SALA_COD));

```

```
19
20 CREATE TABLE IF NOT EXISTS sanitat.DOCTOR (
21  HOSPITAL_COD  TINYINT (2),
22  DOCTOR_NO     SMALLINT(3),
23  COGNOM        VARCHAR(13) NOT NULL,
24  ESPECIALITAT  VARCHAR(16) NOT NULL,
25  CONSTRAINT DOCTOR_PK PRIMARY KEY (HOSPITAL_COD, DOCTOR_NO),
26  INDEX IDX_DOCTOR_HOSP (HOSPITAL_COD),
27  FOREIGN KEY (HOSPITAL_COD) REFERENCES sanitat.HOSPITAL(HOSPITAL_COD)) ;
```

2.5 Eliminació de taules

La sentència `DROP TABLE` és la instrucció proporcionada pel llenguatge SQL per a l'eliminació (dades i definició) d'una taula.

La sintaxi de la sentència `DROP TABLE` és aquesta:

```
1 drop table [<nom_esquema>.]<nom_taula> [if exists];
```

L'opció `if exists` es pot especificar per tal d'evitar un error en cas que la taula no existeixi.

També es poden afegir les opcions `cascade` o `restrict` que en alguns SGBD fan que s'eliminin totes les definicions de restriccions d'altres taules que fan referència a la taula que es vol eliminar abans de fer-ho, o que s'impedeixi l'eliminació, respectivament. Sense l'opció `cascade` la taula que és referenciada per altres taules (a nivell de definició, independentment que hi hagi o no, en un moment determinat, files referenciades), l'SGBDR no l'elimina.

En MySQL, però, no es tenen efecte les opcions `cascade` o `restrict` i sempre cal eliminar les taules referides per tal de poder eliminar la taula referenciada.

Exemple d'eliminació de taules

Suposem que volem eliminar la taula DEPT de l'esquema *empresa*.

L'execució de la sentència següent és errònia:

```
1 drop table dept;
```

L'SGBD informa que hi ha taules que hi fan referència i que, per tant, no es pot eliminar. I és lògic, ja que la taula `DEPT` està referenciada per la taula `EMP`.

Si de veritat es vol aconseguir eliminar la taula `DEPT` i provocar que totes les taules que hi fan referència eliminin la definició corresponent de clau forana, caldrà eliminar EMP prèviament. I, abans que aquesta, les altres taules que fan referència a aquesta altra. De forma que l'ordre per eliminar les taules sol ser l'ordre invers en què les hem creades.

```
1 use empresa;
2 drop table detall;
```

```
drop table comanda;
drop table producte;
drop table client;
drop table emp;
drop table dept;
```

2.6 Modificació de l'estructura de les taules

De vegades, cal fer modificacions en l'estructura de les taules (afegir-hi o eliminar-ne columnes, afegir-hi o eliminar-ne restriccions, modificar els tipus de dades...).

> La sentència `ALTER TABLE` és la instrucció proporcionada pel llenguatge SQL per modificar l'estructura d'una taula.

La seva sintaxi és aquesta:

```
alter [IGNORE] table [<nom_esquema>.]<nom_taula>
<clàusules_de_modificació_de_taula>;
```

És a dir, una sentència `alter table` pot contenir diferents clàusules (com a mínim una) que modifiquin l'estructura de la taula. Hi ha clàusules de modificació de taula que poden anar acompanyades, en una mateixa sentència `alter table`, per altres clàusules de modificació, mentre que n'hi ha que han d'anar soles.

Cal tenir present que, per efectuar una modificació, l'SGBD no hauria de trobar cap incongruència entre la modificació que s'ha d'efectuar i les dades que ja hi ha en la taula. No tots els SGBD actuen de la mateixa manera davant aquestes situacions.

Així, l'SGBD MySQL, per defecte, no permet especificar la restricció d'obligatorietat (not null) a una columna que ja conté valors nuls (cosa lògica, no?) ni tampoc disminuir l'amplada d'una columna de tipus `varchar` a una amplada inferior a l'amplada màxima dels valors continguts en la columna.

En canvi, però, si s'activa l'opció `ignore`, la modificació especificada s'intenta fer, encara que calgui truncar o modificar dades de la taula ja existent. Per exemple, si intentem modificar la característica `not null` de la columna telèfon de la taula hospital, de l'esquema sanitat, atès que hi ha un valor nul, la sentència següent no s'executarà:

```
alter table hospital
  modify telefon varchar(8) not null;
```

I el resultat de l'execució d'aquesta modificació serà un missatge tipus `Error: Data truncated for column telefon at row 5`.

En canvi, si utilitzem l'opció `ignore` podem executar la sentència següent que ens permetrà modificar l'opció `not null` de telèfon de manera que posarà un *string*

buit en lloc de valor nul en les columnes que no compleixin la condició:

```
alter ignore table hospital
  modify telefon varchar(8) not null;
```

De manera similar, si volem disminuir la mida de la columna adreça de la taula hospital:

```
alter table hospital
  modiy adreca varchar(7);
```

Aquest codi mostrarà un error similar a `Error: Data truncated for column adreca at row 1` i no s'executarà la sentència. En canvi, si hi afegim l'opció `ignore`, el resultat serà la modificació de l'estructura de la taula i el truncament dels valors de les columnes afectades.

```
alter ignore table hospital
  modiy adreca varchar(7);
```

Vegem, a continuació, les diferents possibilitats d'alteració de taula, tenint en compte que en MySQL s'admeten diversos tipus d'alteracions en una mateixa clàusula d'`alter table`, separades per coma:

1. Per afegir una columna

```
ADD [COLUMN] nom_columna definició_columna [FIRST | ALFTER nom_columna ]
```

O bé, si cal definir-ne unes quantes de noves:

```
ADD [COLUMN] (nom_columna definició_columna,...)
```

2. Per eliminar una columna

```
DROP [COLUMN] <nom_columna>
```

3. Per modificar l'estructura d'una columna

```
MODIFY [COLUMN] nom_columna definició_columna [FIRST | AFTER col_name]
```

O bé:

```
CHANGE [COLUMN] nom_columna_antic nom_columna_nou definició_columna
        [FIRST|AFTER nom_columna]
```

4. Per afegir restriccions

```
ADD [CONSTRAINT <nom_restricció>] <restricció>
```

Concretament, les restriccions que es poden afegir en MySQL són les següents:

```
ADD [CONSTRAINT [símbol]] PRIMARY KEY [tipus_index] (nom_columna_index,...) [
    opcions_index] ...

```

```
ADD [CONSTRAINT [símbol]] UNIQUE [INDEX|KEY] [nom_index] [tipus_index] (
    nom_columna_index,...) [opcions_index] ...

ADD [CONSTRAINT [símbol]] FOREIGN KEY [nom] (nom_columna1,...) REFERENCES taula
     (columna1, ....)
```

5. Per eliminar restriccions

```
DROP PRIMARY KEY

DROP {INDEX|KEY} nom_index

DROP FOREIGN KEY nom
```

6. Per afegir índexs

```
ADD {INDEX|KEY} [nom_index]
        [tipus_index] (nom_columna,...) [opcions_index] ...
```

7. Per habilitar o deshabilitar els índexs

```
DISABLE KEYS

ENABLE KEYS
```

8. Per reanomenar una taula

```
RENAME [TO] nom_nou_taula
```

9. Per reordenar les files d'una taula

```
ORDER BY nom_columna1 [, nom_columna2] ...
```

10. Per canviar o eliminar el valor per defecte d'una columna

```
ALTER [COLUMN] nom_columna {SET DEFAULT literal | DROP DEFAULT}
```

Exemple 1 de modificació de l'estructura d'una taula

Recordem l'estructura de la taula DEPT de l'esquema *empresa*:

```
SQL> desc DEPT;
Name          Null     Type
_____________ ________ _____________
DEPT_NO       NOT NULL TINYINT(2)
DNOM          NOT NULL VARCHAR(14)
LOC                    VARCHAR(14)
```

Es vol modificar l'estructura de la taula DEPT de l'esquema empresa de manera que passi el següent:

- La columna loc passi a ser obligatòria.
- Afegim una columna numèrica de nom numEmps destinada a contenir el nombre d'empleats del departament.
- Eliminem l'obligatorietat de la columna nom.
- Ampliem l'amplada de la columna dnom a vint caràcters.

Ho podem aconseguir fent el següent:

```
alter table dept
modify loc varchar(14) not null,
add numEmps number(2) unsigned,
modify dnom varchar(20);
```

Exemple 2 de modificació de l'estructura d'una taula, per problemes de deadlock

Per tal de crear l'estructura de les taules `DEPT` i `EMP` de l'empresa es creen les taules següents i s'hi afegeixen les files amb els valors introduint les sentències següents:

```
CREATE TABLE IF NOT EXISTS empresa.DEPT (
 DEPT_NO  TINYINT (2) UNSIGNED,
 DNOM     VARCHAR(14) NOT NULL UNIQUE,
 LOC      VARCHAR(14),
 PRIMARY KEY (DEPT_NO) ) ;


INSERT INTO empresa.DEPT VALUES (10, 'COMPTABILITAT', 'SEVILLA');
INSERT INTO empresa.DEPT VALUES (20, 'INVESTIGACIÓ', 'MADRID');
INSERT INTO empresa.DEPT VALUES (30, 'VENDES', 'BARCELONA');
INSERT INTO empresa.DEPT VALUES (40, 'PRODUCCIÓ', 'BILBAO');


CREATE TABLE IF NOT EXISTS empresa.EMP (
 EMP_NO    SMALLINT (4) UNSIGNED,
 COGNOM    VARCHAR (10) NOT NULL,
 OFICI     VARCHAR (10),
 CAP       SMALLINT (4) UNSIGNED,
 DATA_ALTA DATE,
 SALARI    INT UNSIGNED,
 COMISSIO  INT UNSIGNED,
 DEPT_NO   TINYINT (2) UNSIGNED NOT NULL,
 PRIMARY KEY (EMP_NO),
 INDEX IDX_EMP_CAP (CAP),
 INDEX IDX_EMP_DEPT_NO (DEPT_NO),
 FOREIGN KEY (DEPT_NO) REFERENCES empresa.DEPT(DEPT_NO)) ;


INSERT INTO empresa.EMP VALUES (7369,'SÁNCHEZ','EMPLEAT',7902, '1980-12-17',
    104000,NULL,20);
INSERT INTO empresa.EMP VALUES (7499,'ARROYO','VENEDOR',7698, '1980-02-20',
    208000,39000,30);
INSERT INTO empresa.EMP VALUES (7521,'SALA','VENEDOR',7698, '1981-02-22',
    162500,65000,30);
INSERT INTO empresa.EMP VALUES (7566,'JIMÉNEZ','DIRECTOR',7839, '1981-04-02',
    386750,NULL,20);
INSERT INTO empresa.EMP VALUES (7654,'MARTÍN','VENEDOR',7698, '1981-09-29',
    162500,182000,30);
INSERT INTO empresa.EMP VALUES (7698,'NEGRO','DIRECTOR',7839, '1981-05-01',
    370500,NULL,30);
INSERT INTO empresa.EMP VALUES (7782,'CEREZO','DIRECTOR',7839, '1981-06-09',
    318500,NULL,10);
INSERT INTO empresa.EMP VALUES (7788,'GIL','ANALISTA',7566, '1981-11-09',
    390000,NULL,20);
INSERT INTO empresa.EMP VALUES (7839,'REY','PRESIDENT',NULL, '1981-11-17',
    650000,NULL,10);
INSERT INTO empresa.EMP VALUES (7844,'TOVAR','VENEDOR',7698, '1981-09-08',
    195000,0,30);
INSERT INTO empresa.EMP VALUES (7876,'ALONSO','EMPLEAT',7788, '1981-09-23',
    143000,NULL,20);
INSERT INTO empresa.EMP VALUES (7900,'JIMENO','EMPLEAT',7698, '1981-12-03',
    123500,NULL,30);
INSERT INTO empresa.EMP VALUES (7902,'FERNÁNDEZ','ANALISTA',7566, '1981-12-03',
     390000,NULL,20);
INSERT INTO empresa.EMP VALUES (7934,'MUÑOZ','EMPLEAT',7782, '1982-01-23',
    169000,NULL,10);
```

Per tal d'afegir la restricció que la columna cap fa referència a un empleat, de la mateixa taula, cal afegir-hi la restricció següent:

```
ALTER TABLE empresa.EMP
ADD FOREIGN KEY (CAP) REFERENCES EMP(EMP_NO);
```

Fixeu-vos que no es podria haver definit aquesta restricció abans d'inserir els valors en les files perquè ja la primera fila inserida ja no compliria la restricció que el codi del seu cap fos prèviament inserit en la taula.

2.7 Índexs per a taules

Índex tipus B-tree i hash

Els índexs B-tree són una organització de les dades en forma d'arbre, de manera que buscar un valor d'una dada resulti més ràpid que buscar-la dins d'una estructura lineal en què s'hagi de buscar des de l'inici fins al final passant per tots els valors.

Els índexs tipus hash tenen com a objectiu accedir directament a un valor concret mitjançant una funció anomenada funció de hash. Per tant, buscar un valor és molt ràpid.

Els SGBD utilitzen índexs per accedir de manera més ràpida a les dades. Quan cal accedir a un valor d'una columna en què no hi ha definit cap índex, l'SGBD ha de consultar tots els valors de totes les columnes des de la primera fins a l'última. Això resulta molt costós en temps i, com més files té la taula en qüestió, més lenta és l'operació. En canvi, si tenim definit un índex en la columna de cerca, l'operació d'accedir a un valor concret resulta molt més ràpid, perquè no cal accedir a tots els valors de totes les files per trobar el que es busca.

MySQL utilitza índexs per facilitar l'accés a columnes que són `PRIMARY KEY` o `UNIQUE` i sol emmagatzemar els índexs utilitzant el tipus d'índex B-tree. Per a les taules emmagatzemades en MEMORY s'utilitzen, però, índexs de tipus HASH.

És lògic crear índexs per facilitar l'accés per a les columnes que necessitin accessos ràpids o molt freqüents. L'administrador de l'SGBD té, entre les seves tasques, avaluar els accessos que s'efectuen a la base de dades i decidir, si escau, l'establiment d'índexs nous. Però també és tasca de l'analista i/o dissenyador de la base de dades dissenyar els índexs adequats per a les diferents taules, ja que és la persona que ha ideat la taula pensant en les necessitats de gestió que tindran els usuaris.

La sentència CREATE INDEX és la instrucció proporcionada pel llenguatge SQL per a la creació d'índexs.

La seva sintaxi simple és aquesta:

```
create index [<nom_esquema>.]<nom_índex>
on <nom_taula> (col1 [asc|desc], col2 [asc|desc], ...);
```

Tot i que la creació d'un índex té associades moltes opcions, que en MySQL poden ser les següents:

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX nom_index
  [USING {BTREE | HASH}
  ON nom_taula (columna1 [longitud][asc|desc], ....)
  [opcions_administració];
```

En MySQL podem definir índexs que mantinguin valors no repetits, especificant la clàusula `UNIQUE`. També podem indicar que indexin tenint en compte el camp sencer d'una columna de tipus `TEXT`, si utilitzem un motor d'emmagatzemament de tipus MyISAM. MySQL suporta índexs sobre els tipus de dades geomètriques que suporta (`SPATIAL`).

Les opcions `USING BTREE` i `USING HASH` permeten forçar la creació d'un índex d'un tipus o un altre (índex tipus B-tree o tipus hash).

La modificació `ASC` o `DESC` sobre cada columna, però, és suportada sintàcticament donant suport a l'estàndard SQL però no té efecte: tots els índexs en MySQL són ascendents.

La sentència `DROP INDEX` és la instrucció proporcionada pel llenguatge SQL per a l'eliminació d'índexs.

La seva sintaxi és aquesta:

```
drop index [<nom_esquema>.]<nom_índex> on <nom_taula>;
```

Exemple 1 de creació d'índexs. Taules de l'esquema empresa

El dissenyador de les taules de l'esquema *empresa* va considerar oportú crear els índexs següents:

```
-- Per tenir els empleats indexats pel cognom:

create index EMP_COGNOM on EMP (COGNOM);

-- Per tenir els empleats indexats pel departament al qual estan assignats:

create index EMP_DEPT_NO_EMP on EMP (DEPT_NO,EMP_NO);

-- Per tenir els clients indexats pel nom:

create index CLIENT_NOM on CLIENT (NOM);

-- Per tenir els clients indexats pel representant (+ codi de client):

create index CLIENT_REPR_CLI on CLIENT (REPR_COD, CLIENT_COD);

-- Per tenir les comandes indexades per la seva data (+ número de comanda):

create index COMANDA_DATA_NUM on COMANDA (COM_DATA, COM_NUM);

-- Per tenir les comandes indexades per la data de tramesa (+ número de comanda
    ):

create index COMANDA_DATA_TRAMESA on COMANDA (DATA_TRAMESA);

-- Per tenir les línies de detall indexades per producte (+ comanda + número de
     línia):

create index DETALL_PROD_COM_DET on DETALL (PROD_NUM,COM_NUM,DETALL_NUM);
```

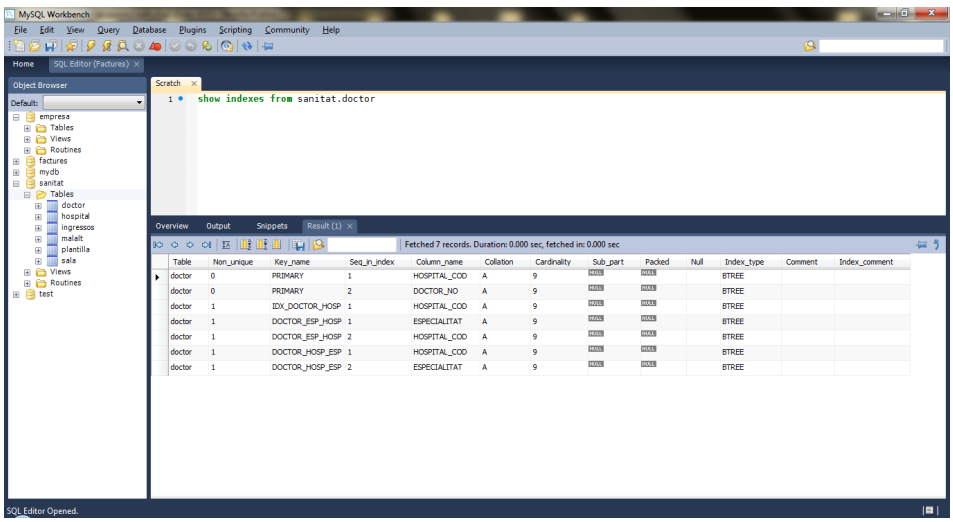
Tots aquests índexs s'afegeixen als existents a causa de les restriccions de clau primària i d'unicitat.

Exemple 2 de creació d'índexs. Taules de l'esquema sanitat

El dissenyador de les taules de l'esquema *sanitat* va considerar oportú crear els índexs següents:

```
-- Per tenir els hospitals indexats pel nom:

CREATE INDEX HOSPITAL_NOM ON HOSPITAL (NOM);

-- Per tenir les sales indexades pel nom dins cada hospital:

CREATE INDEX SALA_HOSP_NOM ON SALA (HOSPITAL_COD, NOM);

-- Per tenir la plantilla indexada per cognom dins cada hospital:

CREATE INDEX PLANTILLA_HOSP_COGNOM ON PLANTILLA (HOSPITAL_COD, COGNOM);

-- Per tenir la plantilla indexada per la funció dins cada hospital:

CREATE INDEX PLANTILLA_HOSP_FUNCIO ON PLANTILLA (HOSPITAL_COD, FUNCIO);

-- Per tenir la plantilla indexada per la funció (entre tots els hospitals
    sales):

CREATE INDEX PLANTILLA_FUNCIO_HOSP_SALA ON PLANTILLA (FUNCIO, HOSPITAL_COD,
    SALA_COD);

-- Per tenir els malalts indexats per data de naixement i cognom:

CREATE INDEX MALALT_NAIX_COGNOM ON MALALT (DATA_NAIX, COGNOM);

-- Per tenir els malalts indexats per cognom i data de naixement:

CREATE INDEX MALALT_COGNOM_NAIX ON MALALT (COGNOM, DATA_NAIX);

-- Per tenir els ingressats indexats per hospital sala:

CREATE INDEX INGRESSOS_HOSP_SALA ON INGRESSOS (HOSPITAL_COD, SALA_COD);

-- Per tenir els doctors indexats per la seva especialitat (entre tots els
    hospitals):

CREATE INDEX DOCTOR_ESP_HOSP ON DOCTOR (ESPECIALITAT, HOSPITAL_COD);

-- Per tenir els doctors indexats per la seva especialitat dins cada hospital:

CREATE INDEX DOCTOR_HOSP_ESP ON DOCTOR (HOSPITAL_COD, ESPECIALITAT);
```

FIGURA 2.1. Índexos de la taula 'doctor' mostrats a través de Workbench de MySQL



Tots aquests índexs s'afegeixen als existents a causa de les restriccions de clau primària i d'unicitat. Per visualitzar tots els índexs existents sobre una taula concreta podem utilitzar l'ordre:

```
show indexes from [nom_esquema.]nom_taula
```

En el cas de la taula doctor de l'esquema sanitat, podem observar el resultat visualitzat en l'eina Workbench de MySQL, en la figura 2.1.

2.8 Definició de vistes

Les vistes es corresponen amb els diferents tipus de consultes que proporciona l'SGBDR MS-Access.

Una **vista** és una taula virtual per mitjà de la qual es pot veure i, en alguns casos canviar, informació d'una o més taules.

Una vista té una estructura semblant a una taula: files i columnes. Mai no conté dades, sinó una sentència `SELECT` que permet accedir a les dades que es volen presentar per mitjà de la vista. La gestió de vistes és semblant a la gestió de taules.

La sentència `CREATE VIEW` és la instrucció proporcionada pel llenguatge SQL per a la creació de vistes.

La seva sintaxi és aquesta:

```
create [or replace] view [<nom_esquema>.]<nom_vista> [(col1, col2...)]
as <sentència_select>
[with [cascaded| local] check option];
```

Com observareu, aquesta sentència és similar a la sentència per crear una taula a partir del resultat d'una consulta. La definició dels noms dels camps és optativa; si no s'efectua, els noms de les columnes recuperades passen a ser els noms dels camps nous. La sentència SELECT es pot basar en altres taules i/o vistes.

L'opció `with check option` indica a l'SGBD que les sentències INSERT i UPDATE que es puguin executar sobre la vista han de verificar les condicions de la clàusula `where` de la vista.

L'opció `or replace` en la creació de la vista permet modificar una vista existent amb una nova definició. Cal tenir en compte que aquesta és l'única via per modificar una vista sense eliminar-la i tornar-la a crear.

La sentència `DROP VIEW` és la instrucció proporcionada pel llenguatge SQL per a l'eliminació de vistes.

La seva sintaxi és aquesta, que permet eliminar una o diverses vistes:

```
drop view [<nom_esquema>.]<nom_vista> [,[<nom_esquema>.]<nom_vista>] ;
```

La sentència `ALTER VIEW` és la instrucció proporcionada per modificar vistes. La seva sintaxi és aquesta:

```
alter view <nom_vista> [(columna1, ....)]
  as <sentència_select>;
```

Exemple 1 de creació de vistes

En l'esquema *empresa*, es demana una vista que mostri totes les dades dels empleats acompanyades del nom del departament al qual pertanyen.

La sentència pot ser la següent:

```
create view EMPD
as select emp_no, cognom, ofici, cap, data_alta, salari, comissio
    , e.dept_no, dnom
  from emp e, dept d
  where e.dept_no = d.dept_no;
```

Una vegada creada la vista, es pot utilitzar com si fos una taula, com a mínim per executar-hi sentències `SELECT`:

```
SQL> select * from empd;

EMP_NO  COGNOM     OFICI      CAP  DATA_ALTA   SALARI  COMISSIÓ   DEPT_NO  DNOM
_______ _________ __________ ____ ___________ _______ __________ ________
    ______________
7369    SÁNCHEZ    EMPLEAT    7902 17/12/1980  104000             20
    INVESTIGACIÓ
7499    ARROYO     VENEDOR    7698 20/02/1980  208000  39000      30
    VENDES
7521    SALA       VENEDOR    7698 22/02/1981  162500  65000      30
    VENDES
7566    JIMÉNEZ    DIRECTOR   7839 02/04/1981  386750             20
    INVESTIGACIÓ
7654    MARTÍN     VENEDOR    7698 29/09/1981  162500  182000     30
    VENDES
7698    NEGRO      DIRECTOR   7839 01/05/1981  370500             30
    VENDES
7782    CEREZO     DIRECTOR   7839 09/06/1981  318500             10
    COMPTABILITAT
7788    GIL        ANALISTA   7566 09/11/1981  390000             20
    INVESTIGACIÓ
7839    REY        PRESIDENT       17/11/1981  650000             10
    COMPTABILITAT
7844    TOVAR      VENEDOR    7698 08/09/1981  195000  0          30
    VENDES
7876    ALONSO     EMPLEAT    7788 23/09/1981  143000             20
    INVESTIGACIÓ
7900    JIMENO     EMPLEAT    7698 03/12/1981  123500             30
    VENDES
7902    FERNÁNDEZ  ANALISTA   7566 03/12/1981  390000             20
    INVESTIGACIÓ
7934    MUÑOZ      EMPLEAT    7782 23/01/1982  169000             10
    COMPTABILITAT
```

Exemple 2 de creació de vistes

En l'esquema *empresa*, es demana una vista per visualitzar els departaments de codi parell.

La sentència pot ser aquesta:

```
create view DEPT_PARELL
as select * from DEPT where mod(dept_no,2) = 0;
```

2.8.1 Operacions d'actualització sobre vistes en MySQL

Les operacions d'actualització (`INSERT`, `DELETE` i `UPDATE`) són, per als diversos SGBD, un tema conflictiu, ja que les vistes es basen en sentències `SELECT` en què poden intervenir moltes o poques taules i, fins i tot, altres vistes, i per tant cal decidir a quina d'aquestes taules i/o vistes correspon l'operació d'actualització sol·licitada.

Per a cada SGBD, caldrà conèixer molt bé les operacions d'actualització que permet sobre les vistes.

Cal destacar que les vistes en MySQL poden ser actualitzables o no actualitzables: les vistes en MySQL són actualitzables, és a dir, admeten operacions `UPDATE`, `DELETE` o `INSERT` com si es tractés d'una taula. Altrament són vistes no actualitzables.

Les vistes actualitzables han de tenir relacions un a un entre les files de la vista i les files de les taules a què fan referència. Així, doncs, hi ha clàusules i expressions que fan que les vistes en MySQL siguin no actualitzables, per exemple:

- Funcions d'agregació (SUM( ), MIN( ), MAX( ), COUNT( ), etc.)
- `DISTINCT`
- `GROUP BY`
- `HAVING`
- `UNION`
- Subconsultes en la sentència `select`
- Alguns tipus de `join`
- Altres vistes no actualitzables en la clàusula `from`
- Subconsultes en la sentència `where` que facin referència a taules de la clàusula `FROM`

Una vista que tingui diverses columnes calculades no és inserible, però sí que es poden actualitzar les columnes que contenen dades no calculades.

Exemple d'actualització en una vista

Recordem una de les vistes creades sobre l'esquema *empresa*:

```
create view EMPD
as select emp_no, cognom, ofici, cap, data_alta, salari, comissio
    , e.dept_no, dnom
  from emp e, dept d
  where e.dept_no = d.dept_no;
```

Si volem modificar la comissió d'un empleat concret (`emp_no=7782`) mitjançant la vista `EMPD` ho podem fer tal com segueix:

```
update empd set comissio=10000 where emp_no=7782;
```

Amb el resultat esperat, que s'haurà canviat la comissió de l'empleat, també, en la taula `EMP`.

Si volem canviar, però, el nom de departament d'aquest empleat (emp_no=7782) i ho fem amb la instrucció següent:

```
update empd set dnom='ASSESSORIA COMPTABLE' where emp_no=7782;
```

El resultat serà que també s'hauran canviat els noms dels departament de comptabilitat dels companys del mateix departament, ja que, efectivament, s'ha canviat el nom del departament dins de la taula de `DEPT`, i potser aquest no és el resultat que esperàvem.

Exemple d'eliminació i inserció en una vista

Recordem la vista EMPD creada sobre l'esquema *empresa*:

```
create view EMPD
as select emp_no, cognom, ofici, cap, data_alta, salari, comissio
    , e.dept_no, dnom
  from emp e, dept d
  where e.dept_no = d.dept_no;
```

Si intentem executar una sentència `DELETE` sobre la vista `EMPD`, el sistema no ho permetrà, en tractar-se d'una vista que conté una join i, per tant, dades de dues taules diferents.

```
delete from empd where emp_no=7782;
```

Podem executar `INSERT` sobre la vista EMPD i tampoc no ho podrem fer.

```
insert into empd values(7777,'PLAZA','VENEDOR',7698,'1984-05-01'
    ,200000,NULL,10,NULL);
```

Exemple d'operacions d'actualització sobre vistes

Recordem la vista `DEPT_PARELL` creada sobre l'esquema *empresa*:

```
create view DEPT_PARELL
as select * from DEPT where mod(dept_no,2) = 0;
```

Ara efectuarem algunes insercions, alguns esborraments i algunes modificacions de departaments parells per mitjà de la vista `DEPT_PARELL`.

```
insert into DEPT_PARELL values (60,'INFORMÀTICA', 'BARCELONA');
```

Aquesta instrucció provoca la inserció d'una fila sense cap problema en la taula `DEPT`.

```
insert into DEPT_PARELL values (55,'MAGATZEM','LLEIDA');
```

Aquesta instrucció provoca la inserció d'una fila sense cap problema, però aquesta inserció no es produiria si la vista hagués estat creada amb l'opció with check option, ja que en aquesta situació els departaments inserits en la taula `DEPT` per mitjà de la vista `DEPTPARELL` haurien de verificar la clàusula `where*` de la definició de la vista.

Podem comprovar que si fem aquesta modificació, ens dóna un error en la inserció d'un codi no parell:

```
create or replace view DEPT_PARELL
  as select * from DEPT where mod(dept_no,2) = 0 with check
       option;

insert into DEPT_PARELL values (65,'MAGATZEM2','GIRONA');
```

La instrucció següent provoca error perquè s'ha definit l'opció `with check` option, ja que en aquesta situació el departament 50 ha estat seleccionat però no ha pogut canviar a 51 perquè no compleix la clàusula where de la vista. Si tornem a evitar l'opció with check option en la definició de la vista, l'actualització del departament 50 (seleccionable per la vista, en ser parell) cap a departament 51 no donarà cap error i farà el canvi de codi volgut:

```
create or replace view DEPT_PARELL
  as select * from DEPT where mod(dept_no,2) = 0;

update DEPT_PARELL set dept_no = dept_no+1 where dept_no = 50;
```

```
delete from DEPT_PARELL where dept_no IN (50, 55);
```

Aquesta instrucció no esborra cap departament, ja que el 50 no existeix (l'hem canviat a 51), i el 55 existeix però no és seleccionable per mitjà de la vista ja que no és parell.

2.9 Sentència RENAME

La sentència `RENAME` és la instrucció proporcionada pel llenguatge SQL per modificar el nom d'una o diverses taules del sistema.

La seva sintaxi és aquesta:

```
rename <nom_actual> to <nou_nom> [, <nom_actual2> to <nou_nom2>, ....];
```

2.10 Sentència TRUNCATE

La sentència `TRUNCATE` és la instrucció proporcionada pel llenguatge SQL per eliminar totes les files d'una taula.

La seva sintaxi és aquesta:

```
truncate [table] <nom_taula>;
```

`TRUNCATE` és similar a `delete` de totes les files (sense clàusula `where`). Funciona, però, eliminant la taula (`DROP TABLE`) i tornant-la a crear (`CREATE TABLE`).

2.11 Creació, actualització i eliminació d'esquemes o bases de dades en MySQL

> Recordem que en MySQL un `SCHEMA` és sinònim de `DATABASE` i que consisteix en una agrupació lògica d'objectes de base de dades (taules, vistes, procediments, etc.).

Per crear una base de dades o un esquema es pot utilitzar la sintaxi bàsica següent:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] nom_bd;
```

Per modificar una base de dades o un esquema es pot utilitzar la sintaxi següent, que permet canviar el nom del directori en què està mapada la base de dades o el conjunt de caràcters:

```
ALTER {DATABASE | SCHEMA} nom_bd
  { UPGRADE DATA DIRECTORY NAME
  | [DEFAULT] CHARACTER SET [=] nom_charset
  | [DEFAULT] COLLATE [=] nom_collation } ;
```

Per eliminar una base de dades o un esquema es pot utilitzar la sintaxi bàsica següent:

```
DROP {DATABASE | SCHEMA} [IF NOT EXISTS] nom_bd;
```

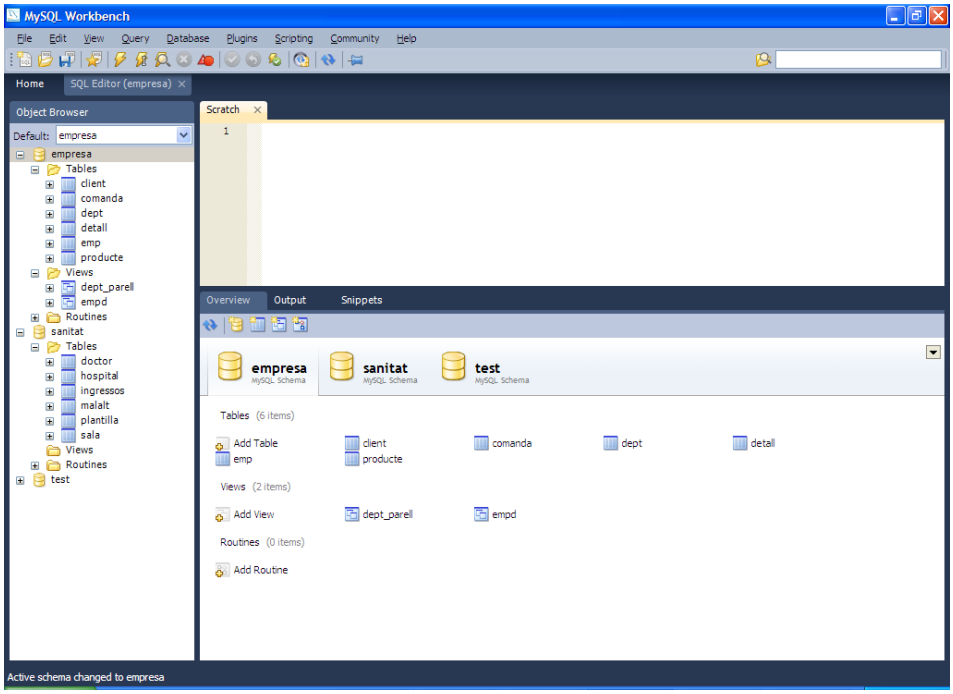
2.12 Com es poden conèixer els objectes definits en un esquema de MySQL

Una vegada que sabem definir taules, vistes, índexs o esquemes, i com modificar, en alguns casos, les definicions existents ens sorgeix un problema: com podem accedir de manera ràpida als objectes existents?

> L'eina **MySQL Workbench** és una eina gràfica que permet, entre altres coses, veure els objectes definits dins del SGBD MySQL i explorar les bases de dades que integra.

Es pot veure en la figura 2.2 l'eina MySQL Workbench en l'apartat SQL Development com es pot navegar pels diferents objectes de les bases de dades que gestiona MySQL.

FIGURA 2.2. MySQL Workbench: eina gràfica de MySQL. Navegació pels diferents objectes de la base de dades



És important saber, també, que l'SGBD MySQL ens proporciona un conjunt de taules (que formen el diccionari de dades de l'SGBD) que permeten accedir a les definicions existents. N'hi ha moltes, però ens interessa conèixer les de la taula 2.2. Totes incorporen una gran quantitat de columnes, la qual cosa fa necessari esbrinar-ne l'estructura, per mitjà de la instrucció `desc`, abans d'intentar trobar-hi una informació.

TAULA 2.2. Vistes de l'SGBD MySQL que proporcionen informació sobre els objectes definits en l'esquema

Taula	Contingut	Exemple d'ús
information_schema.schemata	Informació sobre les bases de dades de l'SGBD.	select * from information_schema.schemata;
information_schema.tables	Informació sobre les taules de les diferents bases de dades de MySQL.	select * from information_schema.TABLES where table_schema='sanitat';
information_schema.columns	Informació sobre columnes de les taules de les diferents bases de dades de MySQL.	SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT FROM INFORMATION_SCHEMA.COLUMNS WHERE table_name = 'doctor' AND table_schema = 'sanitat';
information_schema.table_constraints	Informació sobre les restriccions de les taules de la base de dades.	SELECT * from information_schema.TABLE_CONSTRAINTS where table_schema='sanitat' ;
information_schema.views	Informació sobre les vistes de les diferents bases de dades de MySQL.	select * from information_schema.views where table_schema='empresa';
information_schema.referential_constraints	Informació sobre les claus foranes de les taules de la base de dades.	select * from information_schema.REFERENTIAL_CONSTRAINTS;

Hi ha formes abreujades, però, de mostrar la informació d'aquestes taules del diccionari; per exemple, per mostrar les taules o les columnes de les taules, podem utilitzar aquestes formes simplificades:

```
SHOW TABLES

SHOW COLUMNS
  FROM nom_taula
  [FROM nom_base_dades]
```

3. Control de transaccions i concurrències

Una **transacció** és una seqüència d'instruccions SQL que l'SGBD gestiona com una unitat. Les sentències `COMMIT` i `ROLLBACK` permeten indicar un fi de transacció.

Exemple de transacció: operació al caixer automàtic

Una transacció típica és una operació en un caixer automàtic, per exemple: si anem a un caixer automàtic a treure diners d'un compte bancari esperem que si l'operació acaba bé (i obtenim els diners extrets) es reflecteixi aquesta operació en el compte, i, en canvi, si hi ha hagut algun error i el sistema no ens ha pogut donar els diners, esperem que no es reflecteixi aquesta extracció en el nostre compte bancari. L'operació, doncs, cal que es consideri com una unitat i acabi bé (*commit*), però, que si acaba malament (*rollback*) tot quedi com estava inicialment abans de començar.

Transaccions en MySQL

Les transaccions en MySQL només tenen sentit sota el motor d'emmagatzemament InnoDB, que és l'únic motor transaccional de MySQL. Recordeu que els altres sistemes d'emmagatzemament són no transaccionals i, per tant, cada instrucció que s'executa és independent i funciona, sempre, de manera autocommitiva.

Una transacció habitualment comença en la primera sentència SQL que es produeix després d'establir connexió en la base de dades, després d'una sentència COMMIT o després d'una sentència ROLLBACK.

Una transacció finalitza amb la sentència COMMIT, amb la sentència ROLLBACK o amb la desconnexió (intencionada o no) de la base de dades.

Els canvis realitzats en la base de dades en el transcurs d'una transacció només són visibles per a l'usuari que els executa. En executar una COMMIT, els canvis realitzats en la base de dades passen a ser permanents i, per tant, visibles per a tots els usuaris.

Si una transacció finalitza amb ROLLBACK, es desfan tots els canvis realitzats en la base de dades per les sentències de la transacció.

Recordem que MySQL té l'autocommit definit per defecte, de manera que s'efectua un **COMMIT** automàtic després de cada sentència SQL de manipulació de dades. Per desactivar-lo cal executar:

```
1 set autocommit=0;
```

Per tal de tornar a activar el sistema d'autocommit:

```
1 set autocommit=1;
```

Cal tenir en compte que una transacció només té sentit si no està definit l'autocommit.

El funcionament de transaccions no és el mateix en tots els SGBD i, per tant, caldrà esbrinar el tipus de gestió que proporciona abans de voler-hi treballar.

3.1 Sentència START TRANSACTION en MySQL

`START TRANSACTION` defineix explícitament l'inici d'una transacció. Per tant, el codi que hi hagi entre `start transaction` i `commit` o `rollback` formarà la transacció.

Iniciar una transacció implica un bloqueig de taules (`LOCK TABLES`), així com la finalització de la transacció provoca el desbloqueig de les taules (`UNLOCK TABLES`).

En MySQL `start transaction` és sinònim de `begin` i, també, de `begin work`. I la sintaxi per a `start transaction` és la següent:

```
1 START TRANSACTION [WITH CONSISTENT SNAPSHOT] | BEGIN [WORK]
```

L'opció `WITH CONSISTENT SNAPSHOT` inicia una transacció que permet lectures consistents de les dades.

3.2 Sentències COMMIT i ROLLBACK en MySQL

`COMMIT` defineix explícitament la finalització esperada d'una transacció. La sentència `ROLLBACK` defineix la finalització errònia d'una transacció.

La sintaxi de `commit` i `rollback` en MySQL és la següent:

```
1 COMMIT [WORK] [AND [NO] CHAIN | [NO] RELEASE]
2 ROLLBACK [WORK] [AND [NO] CHAIN | [NO] RELEASE]
```

L'opció `AND CHAIN` provoca l'inici d'una nova transacció que començarà tot just acabi l'actual.

L'opció `REALEASE` causarà la desconnexió de la sessió actual.

Quan es fa un `rollback` és possible que el sistema processi de manera lenta les operacions, ja que un `rollback` és una instrucció lenta. Si es vol visualitzar el conjunt de processos que s'executen es pot executar l'ordre `SHOW PROCESSLIST` i visualitzar els processos que s'estan *desfent* a causa del `rollback`.

L'SGBDR *MySQL* realitza una `COMMIT` implícita abans d'executar qualsevol sentència LDD (llenguatge de definició de dades) o `LCD` (llenguatge de control de dades), o en executar una desconnexió que no hagi estat precedida d'un error. Per tant, no té sentit incloure aquest tipus de sentències dintre de les transaccions.

3.3 Sentències SAVEPOINT i ROLLBACK TO SAVEPOINT en MySQL

Hi ha la possibilitat de marcar punts de control (`savepoints`) enmig d'una transacció, de manera que si s'efectua ROLLBACK aquest pugui ser total (tota la transacció) o fins a un dels punts de control de la transacció.

La instrucció `SAVEPOINT` permet crear punts de control. La seva sintaxi és aquesta:

```
savepoint <nom_punt_control>;
```

La sentència `ROLLBACK` per desfer els canvis fins a un determinat punt de control té aquesta sintaxi:

```
rollback [work] to  [savepoint] <nom_punt_control>;
```

Si en una transacció es crea un punt de control amb el mateix nom que un punt de control que ja existeix, aquest queda substituït pel nou.

Si es vol eliminar el punt de control sense executar un `commit` ni un `rollback` podem executar la instrucció següent:

```
release savepoint <nom_punt_control>
```

Exemple d'utilització de punts de control

Considerem la situació següent:

```
SQL> instrucció_A;
SQL> savepoint PB;
SQL> instrucció_B;
SQL> savepoint PC;
SQL> instrucció C;
SQL> instrucció_consulta_1;
SQL> rollback to PC;
SQL> instrucció_consulta_2;
SQL> rollback; o commit;
```

La instrucció de consulta 1 veu els canvis efectuats per les instruccions A, B i C, però el `ROLLBACK TO PC` desfà els canvis produïts des del punt de control `PC`, per la qual cosa la instrucció de consulta 2 només veu els canvis efectuats per les instruccions A i B (els canvis per C han desaparegut), i el darrer `ROLLBACK` desfà tots els canvis efectuats per A i B, mentre que el darrer COMMIT els deixaria com a permanents.

3.4 Sentències LOCK TABLES i UNLOCK TABLES

Per tal de prevenir la modificació de certes taules i vistes en alguns moments, quan es requereix accés exclusiu a les mateixes, en sessions paral·leles (o concurrents), és possible bloquejar l'accés a les taules.

Concurrència

La concurrència en l'execució de processos implica l'execució simultània de diverses accions, cosa que habitualment té com a conseqüència l'accés simultani a unes dades comunes, que caldrà tenir en compte a l'hora de dissenyar els processos individuals a fi d'evitar inconsistència en les dades.

El bloqueig de taules (`LOCK TABLES`) protegeix contra accessos inapropiats de lectures o escriptures d'altres sessions.

La sintaxi per bloquejar algunes taules o vistes, i impedir que altres accessos puguin canviar simultàniament les dades, és la següent:

```
lock tables <nom_taula1> [[as] <alies1>] read | write
  [, <nom_taula1> [[as] <alies1>] read | write ] ...
```

L'opció `read` permet llegir sobre la taula, però no escriure-hi. L'opció `write` permet que la sessió que executa el bloqueig pugui escriure sobre la taula, però la resta de sessions només la puguin llegir, fins que acabi el bloqueig.

De vegades, els bloquejos s'utilitzen per simular transaccions (en motors d'emmagatzemament que no siguin transaccionals, per exemple) o bé per aconseguir accés més ràpid a l'hora d'actualitzar les taules.

Quan s'executa `lock tables` es fa un `commit` implícit, per tant, si hi havia alguna transacció oberta, aquesta acaba. Si acaba la connexió (normalment o anormalment) abans de desbloquejar les taules, automàticament es desbloquegen les taules.

És possible, també, en MySQL bloquejar totes les taules de totes les bases de dades de l'SGBD, per fer, per exemple, còpies de seguretat. La sentència que ho permet és `FLUSH TABLES WITH READ LOCK`.

Per desbloquejar les taules (totes les que estiguessin bloquejades) cal executar la sentència `UNLOCK TABLES`.

3.4.1 Funcionament dels bloquejos

Quan es crea un bloqueig per accedir a una taula, dins d'aquesta zona de bloqueig no es pot accedir a altres taules (a excepció de les taules del diccionari de l'SGBD -`information_schema`-) fins que no finalitzi el bloqueig. Per exemple:

```
mysql> LOCK TABLES t1 READ;
mysql> SELECT COUNT(*) FROM t1;
+-----------+
| COUNT(*) |
+-----------+
|         3 |
+-----------+
mysql> SELECT COUNT(*) FROM t2;
ERROR 1100 (HY000): Table 't2' was not locked with LOCK TABLES
```

No es pot accedir més d'una vegada a la taula bloquejada. Si es necessita accedir dos cops a la mateixa taula, cal definir un àlies per al segon accés a l'hora de fer el bloqueig.

```
mysql> LOCK TABLE t WRITE, t AS t1 READ;
mysql> INSERT INTO t SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> INSERT INTO t SELECT * FROM t AS t1;
```

Si es bloqueja una taula especificant-ne un àlies, cal fer-hi referència amb aquest àlies. Provoca un error accedir-hi directament amb el seu nom:

```
mysql> LOCK TABLE t AS myalias READ;
mysql> SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> SELECT * FROM t AS myalias;
```

Si es vol accedir a una taula (bloquejada) amb un àlies, cal definir l'àlies en el moment d'establir el bloqueig:

```
mysql> LOCK TABLE t READ;
mysql> SELECT * FROM t AS myalias;
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES
```

3.5 Sentència SET TRANSACTION

MySQL permet configurar el tipus de transacció amb la sentència `set transaction`.

La sintaxi per configurar les transaccions és:

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
  { READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }
```

Aquesta sentència permet definir determinats paràmetres per a la transacció en curs o per a la transacció següent que s'obrirà. Les característiques que es defineixen poden afectar globalment (`GLOBAL`) o bé afectar la sessió en curs (`SESSION`).

- `READ UNCOMMITTED`: es permet accedir a les dades de les taules, encara que no s'hagi fet un `commit`. Per tant, és possible accedir a dades no consistents (*dirty read*).
- `READ COMMITTED`: només es permet accedir a dades que s'hagin acceptat (**commit**).
- `REPEATABLE READ` (opció per defecte): permet accedir a les dades de manera consistent dins de les transaccions, de manera que totes les lectures de les dades, dins d'una transacció de tipus `REPEATABLE READ`, permetran obtenir les dades com a l'inici de la transacció, encara que ja haguessin canviat.
- `SERIALIZABLE`: permet accedir a les dades de manera consistent en qualsevol lectura de les dades, encara que no ens trobem dins d'una transacció.

Gestió d'usuaris

Joan Anton Pérez Braña

Índex

Introducció

Si volem utilitzar un SGBD per a accedir a la informació continguda en una base de dades, el primer que caldrà comprovar és quines autoritzacions tenim sobre aquelles dades; d'això s'encarrega el component de seguretat de l'SGBD.

Aquest component cada dia esdevé més important, atès que ara tots els ordinadors estan interconnectats i, per tant, qualsevol persona podria esdevenir usuari d'una base de dades. En moltes organitzacions la informació és un actiu intangible i de naturalesa sensible, i per això cal saber quins són les obligacions legals que tenim.

En aquesta unitat formativa plantejarem els components lògics de control sobre les estructures de dades prèviament definides. La definició de cadascun d'aquests components es fa mitjançant sentències SQL, que ja hem estudiat quasi completament.

Les vistes havien estat durant molt de temps un simple mecanisme de simplificació de consultes, però actualment tenen una importància cabdal en diferents àrees: el disseny extern, el gestor de dades (Data Warehouse), la informàtica distribuïda. Veurem els mecanismes que incorpora el SGBD PostgreSQL que permeten de fer actualitzable qualsevol vista mitjançant la definició de regles.

També caldrà tenir en compte que en la definició d'aquests components hi poden haver divergències entre el que diu la darrera versió de l'SQL estàndard. En el nostre cas, estudiarem aquestes característiques amb el SGBD PostgreSQL, de manera que en acabar l'estudi d'aquesta unitat es pugui definir correctament en el sistema mencionat cadascun dels diferents components lògics de dades i de control.

Com a última consideració cal tenir en compte que, per aprendre els conceptes que apareixen en la unitat i aplicar amb agilitat les tècniques esmentades, serà imprescindible implementar els exemples ilustratius, efectuar totes les activitats proposades i els exercicis d'autoavaluació.

Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Implanta mètodes de control d'accés utlitzant assistents, eines gràfiques i comandes del llenguatge del sistema gestor de bases de dades corporatiu.

- Coneix la normativa vigent sobre la protecció de dades.
- Identifica els diferents tipus d'usuaris d'una organització, per tal identificar els privilegis.
- Crea, modifica i elimina comptes d'usuaris; assignant privilegis sobre la base de dades i els seus objectes, garantint el compliment dels requisits de seguretat.
- Agrupa i desagrupa privilegis, per tal d'assignar i eliminar privilegis a usuaris, garantint el compliment dels requisits de seguretat.
- Agrupa i desagrupa grups de privilegis a usuaris, garantint el compliment dels requisits de seguretat.
- Assigna i desassigna rols a usuaris.
- Crea vistes personalitzades per a cada tipus d'usuari de la base de dades.

1. Gestió d'usuaris i privilegis

En un sistema informàtic les dades constitueixen un recurs valuós que ha d'estar controlat i gestionat estrictament.

Entenem per *seguretat d'un sistema* el conjunt de mecanismes de protecció enfront d'accessos no autoritzats, ja siguin intencionats o accidentals.

A més, si la informació fa referència a persones i s'emmagatzemen dades de naturalesa sensible ens caldrà saber quines són les obligacions legals que tenim.

1.1 Introducció als problemes de seguretat en les bases de dades

Quan utilitzem un sistema gestor de bases de dades (SGBD) per accedir a la informació emmagatzemada en una base de dades, primerament cal comprovar quines autoritzacions tenim sobre aquelles dades; d'això s'encarrega el component de seguretat de l'SGBD. Aquest component cada dia esdevé més important, ja que avui dia tots els ordinadors estan interconnectats i, per tant, qualsevol persona podria esdevenir un usuari potencial d'una base de dades.

En un sistema d'informació, les diferents aplicacions i usuaris de l'organització fan servir un únic conjunt de dades, anomenat *base de dades corporativa*, amb l'SGDB. D'una banda, això resol problemes de redundància, inconsistència i independència entre les dades i els programes i, de l'altra, fa que la seguretat esdevingui un dels problemes més importants en aquests entorns.

En moltes organitzacions la informació és un actiu intangible i de naturalesa sensible, i per això cal saber quines són les obligacions legals que tenim.

1.1.1 Conceptes associats a la seguretat

La paraula *seguretat* incorpora diferents conceptes. Els més importants són aquests:

1. **Confidencialitat:** cal protegir l'ús de la informació per part de persones no autoritzades. Això implica que un usuari només ha de poder accedir a la informació per a la qual té autorització i que a partir d'aquesta informació no podrà inferir altra informació que es consideri secreta.
2. **Integritat:** la informació s'ha de protegir de modificacions no autoritzades; això també inclou tant la inserció de dades falses com la destrucció de dades.

3. **Disponibilitat:** la informació ha d'estar disponible en el moment que li faci falta a l'usuari.

1.1.2 Amenaces i violacions del sistema

Per aconseguir seguretat en un entorn de base de dades és necessari identificar les amenaces a la quals pot estar subjecta i triar les polítiques i els mecanismes per evitar-les.

Definirem el concepte **amenaça** com tot aquell agent hostil que, de manera casual o intencionada i utilitzant una tècnica especialitzada, pot revelar o modificar la informació gestionada pel sistema.

Com s'ha esmentat anteriorment, les violacions sobre una base de dades consisteixen en lectures, modificacions o esborraments incorrectes de les dades. Les conseqüències d'aquestes violacions es poden agrupar en tres categories:

1. **Lectura inadequada d'informació**. Causat per la lectura de dades per part d'usuaris no autoritzats mitjançant un accés intencionat o accidental. S'inclouen les violacions del secret derivades de les deduccions d'informació que es considera secreta.
2. **Modificació impròpia de les dades**. Correspon a totes les violacions de la integritat de les dades per tractaments o modificacions fraudulentes d'aquestes. Les modificacions impròpies no involucren necessàriament lectures no autoritzades, ja que les dades es poden falsificar sense ser llegides.
3. **Denegació de serveis**. Correspon a accions que puguin impedir que els usuaris accedeixin a les dades o utilitzin els recursos que tenen assignats.

Les amenaces a la seguretat es poden classificar d'acord amb la manera en què poden ocórrer.

1. **Amenaces no fraudulentes.** Les amenaces no fraudulentes són accidents casuals, entre els quals es poden distingir els següents:
    - *Desastres naturals o accidentals*: normalment són accidents que danyen el maquinari del sistema, com per exemple aquells produïts per terratrèmols, inundacions o foc.
    - *Errors del sistema*: corresponen a tots aquells errors accidentals en el maquinari o en el programari que poden conduir a accessos no autoritzats.
    - *Errors humans*: corresponen a aquelles errades involuntàries derivades de l'acció dels usuaris en introduir dades o utilitzar aplicacions que treballen sobre aquestes.

2. **Amenaces fraudulentes.** Aquestes amenaces generen violacions intencionades i són causades per dos tipus d'usuaris diferents:

   - *Usuaris autoritzats* que abusen dels seus privilegis.
   - *Agents hostils o usuaris impropis* que executen accions de vandalisme sobre el programari o el maquinari del sistema o també lectures o escriptures de dades.

1.1.3 Nivells de seguretat

Com hem esmentat, la seguretat de les bases de dades es refereix a la protecció enfront d'accessos malintencionats. No és possible una protecció absoluta de la base de dades contra aquest mal ús, però es pot incrementar suficientment el cost per a qui el comet per dissuadir-lo en la major part, si no en la totalitat, de tenir accés a la base de dades sense l'autorització adequada. Per protegir la base de dades s'han d'adoptar mesures a diferents nivells:

En diferent documentació veureu que s'utilitza l'expressió anglesa *database management system* (DBMS) per fer referència als sistemes gestors de bases de dades.

- **Sistema gestor de base de dades:** pot ser que alguns usuaris de la base de dades solament tinguin accés a una part limitada de la base de dades. Pot ser que altres usuaris tant sols tinguin autorització per fer consultes però que no puguin modificar les dades. És responsabilitat de l'administrador de l'SGBD que no es violin aquestes restriccions d'autorització.
- **Sistema operatiu:** independentment del nivell de seguretat assolit en l'SGBD la debilitat de la seguretat del sistema operatiu pot servir com a mitjà per a accessos no autoritzats a la base de dades.
- **Xarxa:** atès que gairebé tots els sistemes de bases de dades permeten l'accés remot mitjançant terminals o xarxes, la seguretat en el nivell de programari de la xarxa és tan important com la seguretat física, tant a Internet com en les xarxes privades de les empreses.
- **Físic:** els llocs on estan ubicats els sistemes d'informació cal que estiguin adequadament protegits contra l'entrada d'intrusos.
- **Humà:** els usuaris han d'estar degudament autoritzats per reduir la possibilitat que algun doni accés a intrusos a canvi de suborns o d'altres favors.

1.1.4 Mecanismes bàsics de seguretat emprats en l'SGBD

Els sistemes d'informació i les dades que s'emmagatzemen i es processen són recursos molt valuosos que cal protegir. Els mecanismes emprats per protegir les dades enfront d'amenaces intencionades o accidentals van des dels controls físics fins a procediments administratius.

Identificació i autenticació

La primera acció que cal fer per assolir la seguretat d'un sistema d'informació és la capacitat de verificar la identitat dels usuaris. Aquest procés està format per dues parts:

- **Identificació**: implica la manera en què l'usuari proporciona la seva identitat al sistema (veure qui és). Segons els requisits operacionals, una identitat pot descriure un individu, més d'un individu, o un o més individus només durant un període de temps.

- **Autenticació**: és la manera en què un individu estableix la validesa de la seva identitat (verificar que l'usuari és qui diu que és).

Mentre que les identitats poden ser públiques, la informació d'autenticació es desa en secret, i això proporciona el recurs pel qual es prova que és realment qui diu que és.

Les contrasenyes són el mecanisme clàssic d'autenticació. La seguretat d'aquest mecanisme depèn de la capacitat de mantenir-les en secret. L'administrador de l'SGBD s'encarregarà d'emprar l'algorisme de xifratge més adequat per a cada cas.

Les targetes, ja siguin amb banda magnètica o amb microxip incorporat, donen un sistema de seguretat més gran. En aquests casos la contrasenya proporcionada ha de coincidir amb la que hi ha emmagatzemada a la targeta i a més alguna informació de la targeta ha de coincidir amb alguna informació emmagatzemada a l'ordinador.

Val la pena esmentar que avui dia es tendeixen a utilitzar sistemes biomètrics com poden ser empremtes dactilars, veu, iris o d'altres patrons que es poden considerar únics.

Control d'accés

Són mecanismes que asseguren que els usuaris accedeixen només als llocs als quals estan autoritzats amb l'objectiu de poder fer exclusivament allò per al qual tenen permís.

> Definim *control d'accés* com el conjunt de funcions de l'SGBD per assegurar que els accessos al sistema estan d'acord amb les regles establertes per la política de protecció fixada pel model de negoci.

Així doncs, direm que el control d'accés controla la interacció (lectura, escriptura, modificació i esborrament) entre els subjectes (usuaris i processos) i els objectes als quals accedeixen (taules, esquemes, funcions, altres usuaris, etc.).

El control d'accés es pot considerar format per dos components:

1. *Polítiques d'accés*: defineixen els principis pels quals s'autoritza un usuari o es denega l'accés específic a un objecte de la base de dades.

2. *Mecanismes de seguretat*: formats per tots aquells procediments que s'apliquen a les consultes amb l'objectiu que els usuaris compleixin els principis anteriors.

Les diferents polítiques d'accés es poden classificar en control d'accés obligatori i control d'accés discrecional.

Control d'accés discrecional basat en privilegis

El control d'accés discrecional (DAC) es basa en la identitat dels usuaris o grups d'usuaris per autoritzar o restringir l'accés als diferents objectes de la base de dades. El control discrecional és el mecanisme més comú en els sistemes d'informació actuals.

DAC

DAC correspon a les sigles de *discretionary access controls*, i és el mecanisme més emprat en els SGBD actuals.

Papers

Un paper fa referència al conjunt d'autoritzacions o restriccions que té un usuari o grups d'usuaris en la base de dades.

Podem representar l'estructura de control d'accés discrecional amb una taula (taula 1.1) en què veiem que les interseccions entre files i columnes indiquen els drets de cada usuari o grup d'usuaris sobre cada objecte.

TAULA 1.1. Estructura de control d'accés discrecional

	Objecte 1	**Objecte 2**	**...**	**Objecte *n***
Paper 1				Autoritzacions o restriccions dels usuaris del **paper 1** sobre l'**objecte *n***
Paper 2	Autoritzacions o restriccions dels usuaris del **paper 2** sobre l'**objecte 1**			
...				
Paper *n*		Autoritzacions o restriccions dels usuaris del **paper *n*** sobre l'**objecte 2**		

Els objectes als quals fa referència aquesta taula corresponen a objectes de la base de dades, com poden ser taules, esquemes, funcions, altres usuaris, etc.

Control d'accés obligatori per la seguretat multinivell

El control d'accés obligatori (MAC) s'acostuma a fer servir en aquelles bases de dades en les quals les dades tenen una estructura de classificació molt rígida i estàtica, com per exemple, les bases de dades militars i governamentals. Sovint aquest control d'accés es pot combinar amb el descrit anteriorment.

A continuació farem una pinzellada sobre com es fa aquest tipus de control d'accés.

Les polítiques de control d'accés obligatori es basen en la idea que cada dada té un nivell de classificació pel que fa a la seva seguretat. Les classes de seguretat usuals són:

- secret màxim (TS: *top secret*)
- secret (S)
- confidencial (C)
- no classificat (U: *unclassified*)

en què TS correspon al nivell més alt i U al més baix.

El model que se sol emprar s'anomena *Bell-LaPadula* i assigna a cada subjecte (usuari, grup d'usuaris o programa) i a cada objecte (taula, registres, atribut, etc.) una de les classificacions de seguretat descrites anteriorment. Ens referirem a la classificació del subjecte com a *classif(S)* i a la classificació de l'objecte com a *classif(O)*. Així doncs, les restriccions d'accés es basen en el següent:

- Un subjecte pot veure un objecte si i solament si $classif(S) \geq classif(O)$
- Un subjecte pot modificar un objecte si el seu nivell d'acreditació és igual que el nivell de classificació de l'objecte, és a dir, si $classif(S) = classif(O)$

Integritat i consistència

Són mecanismes perquè la base de dades resti sempre en un estat que compleixi totes les regles de negoci del model de dades, encara que es produeixin canvis.

Per assolir aquest objectiu el dissenyador de la base de dades ha hagut d'establir les regles d'integritat referencial i altres restriccions perquè en qualsevol cas els canvis indeguts tinguin el menor efecte. Cal tenir en compte l'estat dels atributs derivats que sovint s'utilitzen en una base de dades per assolir millores en el rendiment.

Auditoria

L'auditoria correspon a un conjunt de mecanismes per saber qui ha fet què, és a dir, portar un registre de qui fa tots els canvis i consultes a la base de dades. Més que un mecanisme de seguretat és un mecanisme per detectar el culpable.

S'utilitza per als casos següents:

- La investigació d'una activitat sospitosa.
- El monitoratge d'activitats específiques de la base de dades.

El sistema d'auditoria ha de permetre diferents formes d'utilització:

- Auditar sentències. L'auditoria indicarà quan i qui ha utilitzat un tipus de sentència correcta. Per exemple, auditar totes les insercions o esborraments.
- Auditar objectes. El sistema auditarà cada vegada que es faci una operació sobre un objecte determinat.

- Auditar sentències sobre objectes, una versió combinada de les dues anteriors.
- Auditar usuaris o grups.

La informació que s'acostuma a emmagatzemar quan es fa una tasca d'auditoria és el nom de l'usuari, l'identificador de la sessió, l'identificador del terminal, el nom de l'objecte al qual s'ha accedit, l'operació executada o intentada, el codi complet de l'operació, la data i l'hora.

1.1.5 El paper de l'administrador de l'SGBD en la seguretat de les bases de dades

Una de les principals raons d'emprar un SGBD és tenir un control centralitzat tant de les dades com dels accessos que fan els usuaris. La persona que fa el control central sobre el sistema s'anomena *administrador de la base de dades*. Pel que fa a la seguretat, les funcions de l'administrador de la base de dades inclouen:

1. **Definició de l'esquema.** L'administrador crea l'esquema original de la base de dades escrivint un conjunt d'instruccions de definició de dades.
2. **Definició de l'estructura i del mètode d'accés.** Referent al programari client emprat i les diferents activitats relacionades amb l'emmagatzematge i recuperació utilitzant diferents estàndards.
3. **Modificació de l'esquema i l'organització física.** Els administradors de la base de dades fan canvis en l'esquema i l'organització física per reflectir les necessitats canviants dins de l'organització, o per fer alteracions en l'organització física per millorar-ne el rendiment.
4. **Concessió d'autorització per a l'accés a les dades.** La concessió de diferents tipus d'autorització permet a l'administrador de la base de dades determinar a quines parts de la base de dades pot accedir cada usuari: la informació d'autorització es manté en una estructura de l'esquema especial que el sistema de base de dades consulta quan s'intenta fer l'accés a les dades.
5. **Manteniment rutinari.** Alguns exemples d'activitats rutinàries de manteniment de l'administrador són:
    - Còpia de seguretat periòdica de la base de dades, sobre cinta o sobre servidors remots per prevenir la pèrdua de dades a causa de desastres naturals.
    - Assegurar-se que hi ha prou espai lliure al disc per a les operacions habituals i incrementar-lo en cas que sigui necessari.
    - Supervisar les tasques que s'executen a la base de dades i assegurar-se que el rendiment no es degrada per tasques molt costoses iniciades per alguns usuaris.

1.2 L'SGBD PostgreSQL

PostgreSQL és un gestor de bases de dades relacional orientat a objectes molt conegut i usat en entorns de programari lliure perquè compleix els estàndards SQL92 i SQL99, i també pel conjunt de funcionalitats avançades que suporta, cosa que el situa al mateix nivell o a un de millor que molts SGBD comercials.

L'origen del PostgreSQL se situa en el gestor de bases de dades POSTGRES, desenvolupat a la Universitat de Berkeley, i que es va abandonar en favor del PostgreSQL a partir de 1994. Aleshores ja tenia prestacions que el feien únic en el mercat i que altres gestors de bases de dades comercials han anat afegint durant aquest temps.

El PostgreSQL es distribueix sota llicència BSD, la qual cosa en permet l'ús, la redistribució i la modificació amb l'única restricció de mantenir el copyright del programari dels seus autors, en concret el PostgreSQL Global Development Group i la Universitat de Califòrnia.

El PostgreSQL pot funcionar en múltiples plataformes: Linux, FreeBSD, Solaris, Mac OS X i Windows.

1.2.1 Procés d'instal·lació del PostgreSQL

El PostgreSQL està disponible per a la majoria de distribucions de GNU/Linux. La instal·lació és tan senzilla com executar l'instal·lador de paquets corresponent.

En Debian, el procediment següent instal·la el servidor i el client, respectivament:

```
# apt-get install postgresql

# apt-get install postgresql-client
```

En distribucions basades en RPM, els noms dels paquets són una mica diferents:

```
# rpm -Uvh postgresql-server

# rpm -Uvh postgresql
```

Una vegada instal·lat, s'escriurà un *script* d'inici que permet llençar i aturar el servei PostgreSQL; d'aquesta manera, per iniciar el servei, haurem d'executar l'ordre següent:

```
# /etc/init.d/postgresql start
```

Anàlogament per aturar el servei cal fer:

```
# /etc/init.d/postgresql stop
```

1.2.2 L'usuari postgres

En acabar la instal·lació, en el sistema operatiu s'haurà creat l'usuari *postgres*, i en PostgreSQL s'haurà creat un usuari amb el mateix nom. En aquests moments, aquest és l'únic usuari existent en la base de dades i ara, doncs, serà l'únic que podrà crear noves bases de dades i nous usuaris.

Normalment, a l'usuari *postgres* del sistema operatiu no se li permetrà l'accés des de l'intèrpret d'ordres ni tindrà contrasenya assignada, excepte en el cas que en el procés d'instal·lació ens hagi sol·licitat la seva paraula de pas, per la qual cosa ens haurem de convertir en l'usuari *root*, per després convertir-nos en l'usuari *postgres* i fer tasques en nom seu:

```
ioc@localhost:~$ su


Password:

# su – postgres

postgres@localhost:~$
```

L'usuari *postgres* pot crear noves bases de dades des de l'intèrpret d'ordres utilitzant l'ordre **createdb**. En aquest cas, li indiquem que l'usuari propietari de la base de dades serà l'usuari *postgres*:

```
postgres@localhost:~$ createdb demo -–owner=postgres

create database
```

De manera anàloga podem emprar l'ordre **dropdb** per eliminar una base de dades.

1.2.3 El client psql

Per connectar-se amb un servidor, es requereix, òbviament, un programa client. Amb la distribució de PostgreSQL s'inclou un client, *psql*, fàcil d'utilitzar, que permet la introducció interactiva d'ordres en mode text. Abans d'intentar connectar-nos amb el servidor, ens hem d'assegurar que està funcionant i que admet connexions, locals (l'SGBD s'està executant a la mateixa màquina que intenta la connexió) o remotes.

El pas següent és conèixer el nom d'una base de dades resident en el servidor.

L'ordre següent permet conèixer les bases de dades residents en el servidor:

```
ioc@localhost:~$ psql –l

List of databases

Name | Owner | Encoding

```

```
-----------+----------+-----------

demo | postgres | SQL_ASCII

template0 | postgres | SQL_ASCII

template1 | postgres | SQL_ASCII

(3 rows)

~$
```

Per fer una connexió, es requereixen les dades següents:

- Servidor. Si no s'especifica, s'utilitza `localhost`.
- Usuari. Si no s'especifica, s'utilitza el nom d'usuari Unix que executa el *psql*.
- Base de dades.

Exemples de l'ús del *psql* per connectar-se amb un servidor de bases de dades:

```
ioc@localhost:~$ psql -d demo

ioc@localhost:~$ psql demo
```

Les dues formes anteriors executen `psql` amb la base de dades `demo`:

```
~$ psql -d demo -U nom_usuari

~$ psql demo nom_usuari

~$ psql -h nom_servidor.org -U nom_usuari -d nom_basedades
```

A partir del fragment anterior, el client *psql* mostrarà una cosa similar al següent:

```
Welcome to psql, the PostgreSQL interactive terminal.

Type: \copyright for distribution terms

\h for help with SQL commands

\? for help on internal slash commands

\g or terminate with semicolon to execute query

\q to quit

demo=#
```

El símbol **#** significa que el *psql* està llest per llegir l'entrada de l'usuari. Les sentències SQL s'envien directament al servidor per interpretar-les, les ordres internes tenen la forma **\ordre** i ofereixen opcions que no estan incloses en SQL i són interpretades internament pel *psql*.

Per acabar la sessió de *psql*, utilitzem l'ordre **\q** o podem polsar **Ctrl-D**.

Podeu veure els indicadors d'estat del *psql* a la taula 1.2:

Taula 1.2. Indicadors d'estat del //psql//

Indicador	Significat
=#	Espera una nova sentència
-#	La sentència encara no s'ha acabat amb ; o \g
"#	Hi ha una cadena en cometes dobles que no s'ha tancat
'#	Hi ha una cadena en cometes simples que no s'ha tancat
(#	Hi ha un parèntesi que no s'ha tancat

Introducció de sentències

Les sentències SQL que escriguem en el client hauran d'acabar amb ; o bé amb \g:

```
demo=# select user;

current_user

--------------

postgres

(1 row)

demo=#
```

Quan una ordre ocupa més d'una línia, l'indicador canvia de forma i va assenyalant l'element que encara no s'ha completat:

```
demo=# select

demo-# user\g

current_user

--------------

postgres

(1 row)

demo=#
```

La memòria intermèdia en el psql

El client *psql* emmagatzema la sentència fins que se li dóna l'ordre d'enviar-la a l'SGBD. Per visualitzar el contingut de la memòria intermèdia (*buffer*) on ha emmagatzemat la sentència, disposem de l'ordre **\p**:

```
demo=# SELECT

demo-# 2 * 10 + 2

demo-# \p
```

```
SELECT

2 * 10 + 2

demo-# \g

?column?

----------

22

(1 row)

demo=#
```

El client també disposa d'una ordre **\r** que permet esborrar completament la memòria intermèdia per començar de nou amb la sentència:

```
demo=# select 'Hola Mon'\r

Query buffer reset (cleared).

demo=#
```

Ordres de consulta d'informació

El client *psql* ofereix diverses alternatives per obtenir informació sobre l'estructura de la nostra base de dades. En la taula 1.3 es mostren algunes ordres de molta utilitat:

Taula 1.3. Ordres de consulta d'informació

Ordre	Descripció
\l	Fa una llista de les bases de dades
\d	Descriu les taules de la base de dades en ús
\ds	Fa una llista de les seqüències
\di	Fa una llista dels índexs
\dv	Fa una llista de les vistes
\dp \z	Fa una llista dels privilegis sobre les taules
\da	Fa una llista de les funcions d'agregats
\df	Fa una llista de les funcions
\g arxiu	Executa les ordres d'arxiu
\H	Canvia el mode de sortida HTML
\! ordre	Executa una ordre del sistema operatiu

1.2.4 El client gràfic pgAdmin III

El màxim exponent de client gràfic de PostgreSQL és el programari *pgAdmin3*, que té llicència "*Artist License*", aprovada per l'FSF.

En el *pgAdmin3* podem treballar amb gairebé tots els objectes de la base de dades, examinar-ne les propietats i fer tasques administratives.

Una característica interessant del pgAdmin3 és que, cada vegada que fem alguna modificació en un objecte, escriu la sentència o sentències SQL corresponents, cosa que fa que, a més d'una eina molt útil, sigui alhora didàctica.

El *pgAdmin3* també incorpora funcionalitats per fer consultes, examinar-ne l'execució (com l'ordre *explain*) i treballar amb les dades.

Totes aquestes característiques fan del *pgAdmin3* l'única eina gràfica que realment necessitarem per treballar amb PostgreSQL, tant des del punt de vista de l'usuari com de l'administrador.

Hi ha altres eines gràfiques que tenen prestacions semblants al PgAdmin3, com l'SQuirreL.

Evidentment, les accions que podem fer en cada moment dependran dels permisos de l'usuari amb què ens connectem a la base de dades.

1.3 Gestió d'usuaris

Conceptualment, els usuaris de la base de dades estan totalment separats dels usuaris del sistema d'explotació.

Per crear un usuari des d'un client de PostgreSQL s'utilitza l'ordre SQL `CREATE USER`:

Cal diferenciar entre `CREATE USER`, que és una instrucció SQL, i `createuser`, que és una sentència que es pot executar des de l'intèrpret de comades un cop s'ha instal·lat el Postgres.

```
1 CREATE USER nom_usuari [ [ WITH ] opcions [ ... ] ];
```

També es poden crear usuaris des de l'intèrpret de comandes, o shell del sistema, amb la instrucció `createuser`.

Les opcions poden ser:

```
1  SYSID ID_usuari
2
3  CREATEDB | NOCREATEDB
4
5  CREATEUSER | NOCREATEUSER
6
7  IN GROUP nom_grup [, ...]
8
9  [ ENCRYPTED | UNENCRYPTED ]
10
11 PASSWORD 'password'
12
13 VALID UNTIL 'abstime'
```

abstime vol dir 'vàlid fins a'. La clàusula posa un temps absolut després del qual la contrasenya de l'usuari ja no és vàlida. Si aquesta clàusula s'omet la contrasenya serà vàlida per sempre.

I per donar de baixa un usuari s'utilitza DROP USER:

```
DROP USER nom_usuari;
```

Cal diferenciar entre DROP USER, que és una instrucció SQL i dropuser, que és una sentència que es pot executar des de l'intèrpret de comades un cop s'ha instal·lat el Postgres.

També es poden eliminar usuaris des de l'intèrpret d'ordres amb la instrucció dropuser.

Un usuari d'una base de dades pot tenir una sèrie d'atributs que defineixen els seus privilegis i la interacció amb el sistema d'autenticació del client. Són els atributs CREATEUSER o CREATEDB, que li confereixen el permís de crear nous usuaris o crear bases de dades, respectivament.

Els atributs dels usuaris es poden modificar amb l'ordre ALTER USER:

```
ALTER USER nom_usuari [ [ WITH ] opcions [ ... ] ]
```

I les opcions poden ser:

```
CREATEDB | NOCREATEDB

| CREATEUSER | NOCREATEUSER

| [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'

| VALID UNTIL 'abstime''
```

alguns exemples són:

```
ALTER USER nom RENAME TO nou_nom
```

Amb SET canvia la configuració de sessió per defecte d'un usuari per una configuració determinada.

```
ALTER USER nom SET paràmetre { TO | = } { valor | DEFAULT }
```

Amb RESET es restaura la configuració per defecte.

```
ALTER USER nom RESET paràmetre
```

1.4 Autoritzacions: grups i papers

A més de poder donar permisos d'utilització dels diferents recursos del sistema de manera individual a cada usuari, el nostre SGBD disposa de diverses eines que permeten:

- Donar privilegis a un determinat paper al qual s'assignaran els usuaris (tots els usuaris que exerceixin aquest paper heretaran els privilegis i permisos d'aquest).
- Gestionar diversos grups preestablerts de l'SGBD.

Trobareu SGBD que només implementen una de les dues eines i en troba-reu que les implementen totes dues. Ara aprendrem les diferències entre totes dues eines i com utilitzar-les:

1) Rols . Un paper és una forma d'agrupar diferents permisos. Es tracta de pensar en les tasques que han de fer una sèrie d'usuaris i agrupar les que són comunes dins d'un paper. Una vegada establert i definit aquest paper, pot ser assignat als usuaris que facin aquestes tasques. Ens ajudarà a simplificar la gestió de la seguretat dins del nostre sistema.

Exemple de creació de paper

```
1 CREATE role ADMINISTRADOR ;
2 GRANT select,insert,update, delete ON empleats TO ADMINISTRADOR ;
3 GRANT select,insert,update, delete ON projectes TO ADMINISTRADOR
      ;
4 GRANT ADMINISTRADOR TO usuari_amb_permisos ;
```

Aquí creem un paper ''ADMINISTRADOR'', al qual donem certs permisos en dues taules diferents "empleats" i "projectes". Finalment, assignem a l'usuari "usuari_amb_permisos" el paper que acabem de crear.

2) Grups . Els grups tenen una filosofia molt similar als papers. Ara els grups ja vénen predefinits pel sistema, l'únic que podem fer és assignar usuaris als grups que ja tenen uns certs permisos establerts i no modificables. Els grups més comuns que podem trobar als SGBD més comercialitzats són:

- **Administrador de sistema**. És l'usuari que té accés a totes les bases de dades i disposa de tots els recursos. És el nivell més alt i més poderós de tot el sistema.
- **Administrador de les bases de dades**. És l'usuari que té accés a tots els recursos d'una base de dades específica. Pot fer modificacions en tots els objectes de la base de dades específica.
- **Administrador de seguretat**. Té el poder de donar o restringir l'accés a qualsevol usuari dintre de l'SGBD.
- **Operacions de control**. És el grup d'usuaris que té permès fer les còpies de seguretat o les restauracions del sistema.

1.4.1 Grups de PostgreSQL

En el PostgreSQL també tenim la possibilitat de crear grups amb l'ordre `CREATE GROUP`, modificar-los amb `ALTER GROUP` i esborrar-los amb `DROP GROUP`.

No obstant això, cal esmentar que actualment PostgreSQL difereix de l'SQL estàndard, ja que aquest utilitza `CREATE ROLE`, per crear grups d'usuaris.

1.4.2 Els papers

El PostgreSQL 9.0 administra els permisos d'accés a la base de dades d'accés utilitzant el concepte dels papers.

Un paper pot ser entès com un usuari de base de dades, o un grup d'usuaris, depenent de com s'estableixi aquest paper. Un paper pot ser propietari dels objectes de la base de dades (per exemple, taules) i pot assignar privilegis dels objectes dels quals és propietari a d'altres papers per controlar qui té accés a aquests objectes. A més, és possible la concessió de la pertinença d'un paper a un altre paper, la qual cosa permet a un membre del paper utilitzar els privilegis assignats a un altre paper. Podem veure, doncs, que permet implementar el concepte d'herència de privilegis.

El concepte dels papers aglutina els conceptes d'*usuaris* i *grups*. En les versions de PostgreSQL anteriors a la 8.1, els usuaris i grups corresponien a diferents tipus d'entitats, però en aquesta versió només hi ha papers. Així doncs, qualsevol paper pot actuar com un usuari, grup, o totes dues coses.

Creació i eliminació de papers

Els papers d'una base de dades estan conceptualment completament separats dels usuaris del sistema operatiu. En la pràctica, podria ser convenient mantenir-hi una correspondència, però això no és necessari. Els papers d'una base de dades són globals quan fem una instal·lació en clúster d'una bases de dades; per tant, no són exclusivament locals per a cada instància de la base de dades individual dins del clúster.

Per crear un paper cal utilitzar l'ordre SQL `CREATE ROLE`:

```
1 CREATE ROLE nom_del_paper;
```

en què *nom_del_paper* segueix les regles dels identificadors de SQL.

Per eliminar un paper existent, utilitzeu l'ordre anàloga `DROP ROLE`:

```
1 DROP ROLE nom_del_paper;
```

Per a més comoditat, els programes incorporats en el sistema **createuser** i **dropuser** ens proporcionen un substitutiu d'aquestes ordres SQL que ens permeten fer la crida corresponent des de l'intèrpret d'ordres:

```
1 createuser nom_del_paper
2
3 dropuser nom_del_paper
```

Per determinar el conjunt de papers existents, cal examinar el catàleg del sistema; en concret, la taula *pg_roles*; per exemple:

```sql
SELECT rolname FROM pg_roles;
```

La metainstrucció del programa *psql* **\du** també és útil per veure la llista de papers existents.

Per tal d'arrencar el sistema de base de dades, el nou sistema inicialitzat sempre conté una funció d'identificació predefinida. Aquest paper és sempre un *root*, i per defecte (si no és alterat quan s'executa `initdb`) tindrà el mateix nom que l'usuari del sistema operatiu que inicialitza el clúster de base de dades. Habitualment, aquest paper serà anomenat *postgres*. Per tal de crear més papers primerament cal connectar-se com aquest paper inicial.

Cada connexió amb el servidor de base de dades es fa mitjançant el nom d'algun paper en particular, i aquest paper determina els privilegis d'accés inicial.

El nom del paper que s'utilitza per a una connexió de base de dades en particular s'indica al programari client que inicia la sol·licitud de connexió d'una manera específica. Per exemple, el programa *psql* utilitza l'ordre `-U` per indicar el paper amb què ens connectarem a l'inici de la sessió.

Moltes aplicacions assumeixen el nom de l'usuari actual del sistema operatiu per defecte (aquí inclourem **createuser** i **psql**). Per tant, en aquest casos sol ser convenient mantenir una correspondència entre els noms dels papers i els usuaris del sistema operatiu.

Atès que un paper és una funció d'identitat i determina el conjunt de privilegis a disposició d'un client connectat, és important configurar acuradament privilegis quan treballem en un entorn multiusuari.

Atributs dels papers

Un *paper de base de dades* pot tenir una sèrie d'atributs que defineixen els seus privilegis i li permeten interactuar amb el sistema d'autenticació del client.

- *Privilegi `LOGIN`*: solament els papers que tenen atribut d'inici de sessió, `LOGIN`, poden ser utilitzats com a nom de paper inicial d'una connexió de base de dades. Un paper amb l'atribut `LOGIN` pot ser considerat com un "usuari de la base de dades". Per crear un paper amb el privilegi d'inici de sessió, podeu utilitzar indistintament :

```sql
CREATE ROLE nom_del_paper LOGIN;
-- o
CREATE USER nom_del_paper;
```

- *Estatus de superusuari*: un superusuari de base de dades supera qualsevol comprovació de permisos. Aquest és un privilegi perillós i no ha de ser utilitzat amb descuit; el millor és fer un usuari que faci la major part del seu treball amb un paper que no sigui de superusuari. Per crear un nou superusuari, utilitzeu:

```
CREATE ROLE nom_del_paper SUPERUSER
```

- *Creació de noves bases de dades*: a un paper se li pot atorgar explícitament el permís per crear bases de dades (a excepció de superusuaris, ja que passen per alt tots els controls de permís, i per tant ja adquireixen aquest privilegi.

```
CREATE ROLE nom_del_paper CREATEDB
```

- *Creació de nous papers*: a un paper li podem donar explícitament permisos per crear nous papers. Un paper amb el privilegi `CREATEROLE` pot modificar i eliminar altres papers, i també atorgar o revocar la pertinença d'un usuari o paper a un altre paper. No obstant això, per crear, modificar, treure o canviar la pertinença a un paper de superusuari es requereix que qui faci aquest canvi també sigui un superusuari.

```
CREATE ROLE nom_del_paper CREATEROLE
```

- *Contrasenya:* les contrasenyes són només útils si el mètode d'autenticació del client requereix que l'usuari proporcioni una contrasenya quan es connecta a la base de dades. El mètodes d'autenticació MD5 permeten fer un bon ús de les contrasenyes. Les contrasenyes de bases de dades són independents de les contrasenyes del sistema operatiu.

```
CREATE ROLE nom_del_paper PASSWORD 'la_contrasenya'
```

Superusuari

És una bona pràctica crear un paper que tingui els privilegis `CREATEDB` i `CREATEROLE`, però que aquest no sigui un superusuari, i després utilitzar aquest paper per a totes les tasques de bases de dades i d'altres papers. Aquest enfocament evita els perills d'operar com un superusuari per a les tasques que realment no ho requereixen.

Membres d'un paper

Els membres d'un paper poden utilitzar els privilegis de la funció de dues maneres.

En primer lloc, qualsevol membre d'un grup pot fer de manera explícita `SET ROLE` per convertir-se de manera temporal a aquell nou grup. En aquest estat, la sessió de base de dades té accés als privilegis de la funció de grup en lloc del paper assignat originalment a l'inici de sessió, i qualsevol objecte de base de dades creat es considerarà propietat del grup no és el paper d'inici de sessió.

En segon lloc, els membres d'un paper que poden heretar (`INHERIT`) poden fer ús dels privilegis dels papers dels quals són membres de manera automàtica, incloent-hi els privilegis heretats pels papers.

A tall d'exemple, suposem que hem fet:

```
CREATE ROLE joan LOGIN INHERIT;
CREATE ROLE admin NOINHERIT;
CREATE ROLE gestor NOINHERIT;
GRANT admin TO joan;
GRANT gestor TO admin;
```

Immediatament després de connectar-nos a la base de dades amb el paper de *joan* podrem fer ús dels privilegis concedits directament a *joan*, a més dels privilegis concedits a *admin*, perquè *joan* "hereta" els privilegis d'*admin*. En canvi *admin* no hereta els privilegis de *gestor*, i en conseqüència tampoc *joan*.

Si després fem:

```
SET ROLE admin;
```

La nostra sessió tindria ús exclusiu dels privilegis concedits a *admin*, i però no dels que hem concedit a *joan*.

Si ara fem:

```
SET ROLE gestor;
```

La sessió tindrà ús exclusiu dels privilegis concedits a *gestor*, però cap dels que es concedeixen a *joan* ni a *admin*.

El conjunt de privilegis originals es pot restaurar amb qualsevol acció d'aquestes:

```
SET ROLE joan;
SET ROLE NONE;
RESET ROLE;
```

1.5 Privilegis i permisos

Quan es crea un objecte aquest és assignat a un propietari. El propietari normalment és el mateix usuari que ha executat la comanda de creació.

Per a la majoria dels objectes, l'estat inicial és aquell en què el propietari (o un superusuari) pot fer alguna cosa amb aquest objecte. Per tal de deixar a altres usuaris utilitzar l'objecte, cal atorgar-li privilegis.

Existeixen diferents privilegis: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `RULE`, `REFERENCES`, `TRIGGER`, `CREATE`, `TEMPORARY`, `EXECUTE` i `USAGE`.

Per exemple si el privilegi és `RULE` o `TRIGGER` vol dir que l'usuari pot crear regles o triggers en la taula especificada.

Per tal d'assignar privilegis s'utilitza la comanda `GRANT`.

On `PUBLIC` significa que els drets són donats a tots els usuaris. Inclòs aquells que es puguin crear posteriorment.

`WITH GRANT OPTION` significa que el que té aquest privilegi el pot transferir a altres usuaris.

`ALL PRIVILEGES` significa que li dóna tots els drets disponibles de l'objecte de cop.

Per eliminar els drets d'un usuari o grups d'usuaris s'utilitza `REVOKE`.

Un objecte pot ser assignat a un nou usuari amb la comanda `ALTER`.

1.5.1 Tipus de privilegis

Els privilegis es poden donar sobre diversos recursos a diferents usuaris del sistema gestor de bases de dades. Els recursos més comuns són:

- Connexió a la base de dades
- Taules: qui hi pot accedir i les modificar.
- Objectes de la base de dades: qui pot crear/esborrar els objectes que formen part de la base de dades.
- Sistema: qui pot efectuar accions de sistema en l'SGBD.
- Programa: qui pot crear, modificar i usar programes de la base de dades.
- Programes emmagatzemats: qui pot executar funcions i procediments específics.

Privilegis sobre bases de dades

Quan es crea una base de dades el propietari té tots els privilegis sobre aquesta. La base de dades serà inaccessible a altres usuaris, excepte l'usuari *postgres*, fins que el propietari els autoritzi privilegis.

En el PostgreSQL hi ha tres tipus de privilegis sobre bases de dades.

`CONNECT`: permet a l'usuari connectar-se a la base de dades especificada. Aquest privilegi es comprova en l'inici de connexió (a més de comprovar que no s'infringeix cap de les restriccions imposades en l'arxiu de configuració *pg_hba.conf*).

`CREATE`: permet crear nous esquemes a la base de dades.

`TEMPORAL`: permet crear taules temporals durant l'ús de la base de dades especificada.

```
1 GRANT { { CREATE | CONNECT | TEMPORARY | TEMP } [,...] | ALL [ PRIVILEGES ] }
2       ON DATABASE database_name [, ...]
3      TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

Per altra banda, cal recordar que una base de dades pot estar formada per diferents esquemes.

Teniu més informació sobre la gestió d'esquemes en la secció "Annexos" del web del mòdul.

Així doncs, es poden atorgar privilegis sobre aquests. Aquests són:

`USAGE`: pot fer servir els elements d'un determinat esquema d'una base de dades a la qual tingui accés.

CREATE: pot crear objectes dins de l'esquema de la base de dades a la qual té accés.

```
GRANT { { CREATE | USAGE } [,...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

Privilegis de taules

Els privilegis que es poden donar sobre les taules estan relacionats amb totes les accions que es poden fer sobre les taules i vistes, que són:

- SELECT: permet seleccionar dades d'una vista i taula donades.
- INSERT: permet inserir dades a una vista/taula.
- UPDATE: permet actualitzar dades d'una taula o vista.
- DELETE: permet esborrar dades d'una taula donada.
- ALL: permet fer les accions anteriors sobre una taula/vista en concret.
- REFERENCES: permet referenciar mitjançant restriccions de clau forana a una taula de la qual l'usuari no és propietari.

```
GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( column [, ...] )
    [,...] | ALL [ PRIVILEGES ] ( column [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

Exemple de gestió de privilegis de taules:

```
GRANT UPDATE ON NOTES TO SECRETARIA1
```

Permet a l'usuari "secretaria1" modificar el contingut dels registres de la taula "notes". No podrà crear un registre nou, però sí que podrà modificar les dades enregistrades.

```
GRANT DELETE ON MATRICULA TO ADMINISTRATIU4
```

Permet a l'usuari "administratiu4" esborrar els registres de la taula "matrícula".

Privilegis d'objectes de bases de dades

Els objectes d'una base de dades estan formats per totes les estructures que es poden crear, que són:

- bases de dades
- espai per a taules (*tablespace*)
- les taules
- índexs
- *triggers* (disparadors)

Qui tingui permís sobre els objectes de la base de dades podrà crear estructures de la base de dades.

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER }
    [,...] | ALL [ PRIVILEGES ] }
ON { [ TABLE ] table_name [, ...]
        | ALL TABLES IN SCHEMA schema_name [, ...] }
TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

Especificar privilegis sobre espais de taules:

```
GRANT { CREATE | ALL [ PRIVILEGES ] }
    ON TABLESPACE tablespace_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

Especificar privilegis sobre seqüències:

```
GRANT { { USAGE | SELECT | UPDATE }
    [,...] | ALL [ PRIVILEGES ] }
    ON { SEQUENCE sequence_name [, ...]
         | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

Generalment només el DBA tindrà aquest privilegi, ja que, si l'estén a més usuaris, serà difícil controlar el creixement de la base de dades.

Exemple de gestió de privilegis d'objectes de bases de dades:

```
GRANT CREATE table,
 CREATE index
TO usuari456,
 Usuari_excepcional;
```

En aquest exemple podem veure com es dóna permís als usuaris *usuari456* i *usuari_excepcional* per poder crear taules i índexs.

Privilegis de sistema

Els privilegis de sistema estan relacionats amb totes les gestions que es poden portar a terme respecte al sistema gestor, que són:

- arxivar arxius `LOG`,
- reiniciar o apagar el servidor de bases de dades,
- tasques de monitorització,
- etc.

Privilegis sobre programes i procediments

Els privilegis sobre programes i procediments donen el privilegi `EXECUTE` als usuaris que hagin d'executar algun programa o procediment emmagatzemat en l'SGBD.

```
GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON { FUNCTION function_name ( [ [ argmode ] [ arg_name ] arg_type [, ...] ]
         ) [, ...]
        | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

Exemple de gestió de privilegis sobre programes i procediments:

```
GRANT EXECUTE ON procediment
TO user456;
```

Aquí donem permís a l'usuari "user456" perquè pugui executar el programa "procediment".

Privilegis per a tothom

Els privilegis per a tothom és un tipus de privilegi que utilitzarem quan haguem de donar permís sobre un cert recurs a tots els usuaris de l'SGBD. Val a dir que aquest tipus d'assignació de permisos és molt còmode però també és molt perillós. Heu d'anar molt en compte quan el feu servir, ja que una vegada fet públic pot ser molt complicat tornar a tenir un control absolut del recurs.

Intentarem sempre evitar clàusules en què apareixen les dues instruccions següents: `PUBLIC` i `WITH GRANT OPTION`.

Exemple de gestió de privilegis per a tothom:

```
GRANT DELETE ON Llibres TO PUBLIC;
```

Aquí acabem de fer que tot usuari de l'SGBD pugui en qualsevol moment esborrar registres de la taula llibres.

1.5.2 Retirar privilegis

Per retirar els privilegis concedits anteriorment, tenim la sentència `REVOKE`. Es tracta de formar les mateixes ordres que quan donem un permís, però ara canviarem la paraula `GRANT` per `REVOKE`. Si un objecte és eliminat de la base de dades, automàticament també es perden els privilegis sobre l'objecte.

Exemple de retirada de privilegis:

```
REVOKE UPDATE on Llibres (isbn) FROM usuari;
```

En aquest cas traiem el permís a l'usuari "usuari" perquè pugui modificar l'atribut "isbn" de la taula "Llibres".

Heu d'estar molt atents quan retireu privilegis si aquests han estat atorgats amb `WITH GRANT OPTION`, ja que la retirada d'un privilegi a un usuari que hagi donat privilegis a altres usuaris implica que tots ells perdin el permís per utilitzar el recurs. Es coneix com a **retirada de permís en cascada**. Així doncs, eviteu donar privilegis amb l'opció `WITH GRANT OPTION`.

S'ha de fer una última consideració respecte al fet de crear exclusions en grups d'usuaris a l'hora de donar o treure permisos. Imagineu que voleu donar privilegis a tots els usuaris de l'SGBD excepte a un (o uns quants). Una manera de fer-ho seria:

```
1 GRANT DELETE on Llibres to PUBLIC;
2 REVOKE DELETE on Llibres from usuari;
```

Heu de tenir present que aquest tipus d'accions no són permeses en tots els SGBD. Així doncs, haureu d'esbrinar consultant el manual del sistema gestor si és una forma viable de fer exclusions de grups d'usuaris o bé haureu de buscar formes alternatives de fer aquest tipus d'accions.

1.6 Legislació sobre protecció de dades

La protecció de les dades de caràcter personal ha pres darrerament una gran rellevància. Les persones es mostren cada dia més curoses amb les seves dades i són més conscients de la protecció de què ha de gaudir la seva informació personal.

La situació actual és producte, d'una banda, de la normativa en matèria de protecció de dades i, de l'altra, de l'activitat creixent de l'**Agència Espanyola de Protecció de Dades**, organisme autònom encarregat d'assegurar el compliment de la legislació vigent (i fruit de la mateixa legislació).

Per a més informació sobre l'Agència Espanyola de Protecció de Dades, consulteu la secció "Adreces d'interès" del web.

Agències autonòmiques

A data d'avui no totes les comunitats autònomes han creat les seves agències de protecció de dades. Catalunya sí que en té: és l'Agència Catalana de Protecció de Dades, consulteu la secció "Adreces d'interès" del web.

Veurem a continuació com han anat evolucionant les lleis; la primera en aparèixer va ser la **Llei Orgànica 15/1999, de 13 de desembre, de protecció de dades de caràcter personal (LOPD)**. Aquesta norma tenia per objecte garantir i protegir, en relació amb el tractament de dades personals, les llibertats públiques i els drets fonamentals de les persones físiques, i en especial el seu honor, intimitat i privacitat. La LOPD va crear els anomenats drets ARCO:

- Dret d'**Accés**: Reconeix als ciutadans la potestat de defensar la seva privacitat controlant per si mateixos l'ús que es fa de les seves dades personals.
- Drets de **Rectificació** : La LOPD també regula els drets de rectificació i cancel·lació: quan les dades personals d'un ciutadà resulten ser incompletes, inexactes, excessives o inadequades aquest pot requerir al responsable del fitxer la seva rectificació o cancel·lació.
- Dret de **Cancel·lació**: El ciutadà pot exigir al responsable del fitxer la supressió de dades que consideri inadequades o excessives.
- Dret d'**Oposició**: Consisteix en el dret dels titulars de les dades per dirigir-se al responsable del fitxer perquè deixi de tractar les seves dades sense el seu consentiment per a fins de publicitat o prospecció comercial.

Posteriorment, amb el desenvolupament i popularització d'Internet i l'aparició de comerços online va aparèixer al 2002 la llei de serveis de la societat de la informació i comerç electrònic, coneguda per les seves sigles com LSSI.

Al 2003 apareix la llei de la firma electrònica per regular els certificats digitals i donar validesa jurídica a aquesta firma. Al 2003 també s'aprova el Reglament que desenvolupa la llei de protecció de dades de caràcter personal de 1999. El 2007 s'aprova la llei de conservació de dades a les comunicacions electròniques i a les xarxes públiques de comunicacions.

El 27 d'abril de 2016 s'aprova el **el Reglament General de Protecció de dades (RGPD)**, que no va entrar en vigor fins al Maig del 2018, per donar un marc Europeu. Aquest reglament, entre altres coses, amplia els drets ARCO.

El 5 de desembre de 2018 s'aprova la llei orgànica 3/2018, **Protecció de Dades Personals i Garanties dels Drets Digitals** (LOPDGD), que adapta l'RGPD a la normativa espanyola. Amb LOPDGD i l'RGPD es deroga l'antiga LOPD.

A continuació teniu un llistat d'aquestes lleis :

- Llei Orgànica 15/1999, de 13 de desembre, de protecció de dades de caràcter personal (LOPD).
- Llei 34/2002, d'11 de juliol, de serveis de la societat de la informació i comerç electrònic (LSSICE) o, habitualment (LSSI).
- Llei 59/2003, de 19 de desembre, de firma electrònica.
- Llei Orgànica 15/2003, de 25 de novembre, per la qual es modifica la Llei Orgànica 10/1995, de 23 de novembre, del Codi Penal.
- Reial Decret 1720/2007, de 21 de desembre, pel que s'aprova el Reglament de desenvolupament de la Llei Orgànica 15/1999, de 13 de desembre, de protecció de dades de caràcter personal.
- Llei 25/2007, de 18 d'octubre, de conservació de dades relatives a las comunicacions electròniques i a les xarxes públiques de comunicacions.
- Llei Orgànica 5/2010, de 22 de juny, per la qual es modifica la Llei Orgànica. 10/1995, de 23 de novembre, del Codi Penal.
- Reglament General de Protecció de dades (RGPD) del 27 d'Abril de 2016.
- Llei orgànica 3/2018 Protecció de Dades Personals i Garanties dels Drets Digitals (LOPDGD) del 5 de desembre de 2018.

Per dur a terme una tasca professional de qualitat és molt important (fins i tot ens atreviríem a dir que imprescindible) conèixer la normativa espanyola aplicable a la protecció de dades de caràcter personal.

1.6.1 El Reglament General de Protecció de dades (RGPD)

Reviseu el subapartat "El Codi Penal i les conductes il·lícites vinculades a la informàtica", d'aquesta mateixa unitat.

Aquest reglament és una norma d'àmbit europeu que protegeix les dades personals de tots els residents a la Unió Europea i garanteix el flux de dades entre els països de la Unió Europea. Per tant, els països necessiten **integrar** aquest reglament a les seves legislacions.

Aquest reglament estableix l'obligació de les organitzacions d'adoptar mesures destinades a garantir la protecció d'aquestes dades que afecten sistemes informàtics, fitxers, suports d'emmagatzematge, demanar el consentiment per usar les dades de caràcter personal i procediments operatius. Aquestes mesures han d'adoptar-les totes les organitzacions que operen amb residents a la Unió Europea, encara que no hi tinguin la seva seu.

Els fitxers que han de satisfer mesures de seguretat no són tan sols aquells als quals es pot accedir a Internet, sinó tots els que continguin dades personals.

En el Capítol 7 d'aquest reglament es crea el Comitè Europeu de protecció de dades per supervisar el Reglament i la seva aplicació als diferents països d'Europa. En el Capítol 11, *Disposicions finals*, s'estableix com a màxim el 25 de maig del 2020 per fer una primera avaluació i revisió del reglament per tal d'anar-lo actualitzant als nous temps. Posteriorment, aquesta revisió es repetirà cada 4 anys.

L'RGPD és aplicable a qualsevol informació sobre persones físiques identificades o identificables (nom i cognoms, edat, sexe, dades d'identificació fiscal, estat civil, professió, domicili, dades biomètriques...) enregistrada en qualsevol suport físic (inclòs el paper), que en permeti el tractament manual o automatitzat i ús posterior pel sector públic o privat. Traspassat a l'àmbit de les empreses, s'ha d'interpretar que l'RGPD és aplicable a qualsevol organització que manipuli o arxivi fitxers, tant en paper com en suport magnètic, que continguin informació o dades de caràcter personal, tant dels seus treballadors com dels seus clients o proveïdors (persones físiques), la qual cosa obliga les empreses, institucions, professionals i, en general, totes les persones jurídiques o físiques que operin amb fitxers de dades de caràcter personal, al compliment d'una sèrie d'obligacions legals. Cal tenir present, però, que al considerand 18, diu: "El reglament no s'aplica al tractament de dades de caràcter personal dut a terme per una persona física en el curs d'una activitat exclusivament personal o domèstica, és a dir sense cap connexió amb una activitat professional o comercial".

Què és una dada de caràcter personal?

Segons el Reglament General de Protecció de dades (RGPD), una dada de caràcter personal és "qualsevol informació sobre una persona física identificada o identificable (l'interessat)".

Per **tractament** s'entén "qualsevol operació o conjunt d'operacions realitzades sobre dades personals o conjunts de dades personals, ja sigui per procediments automatitzats o no, com la recollida, el registre, l'organització, l'estructuració, la conservació, l'adaptació o la modificació, l'extracció, la consulta, la utilització, la comunicació per transmissió, difusió o qualsevol altra forma d'habilitació d'accés, acarament o interconnexió, limitació, supressió o destrucció".

1.6.2 Objectiu del reglament i principis bàsics de l'RGPD

El parlament Europeu i el Consell de la Unió Europea, a partir del Tractat de funcionament de la Unió Europea, en concret de l'article 16, i d'una proposta de la Comissió Europea, van enviar una proposta del text legislatiu als parlaments nacionals, per posteriorment elaborar dos dictàmens. L'RGPD considera que la protecció del tractament de les dades personals és un dret fonamental, tal i com està a la Carta dels Drets Fonamentals de la Unió Europea a l'article 8, que estableix que qualsevol persona té dret a la protecció de les dades de caràcter personal que l'afecten. Pel que fa al tractament de les dades personals s'han de respectar les llibertats i els drets fonamentals, especialment el dret a la protecció de les dades de caràcter personal, sigui quina sigui la seva nacionalitat o residència.

L'**objectiu de l'RGPD** és, doncs, garantir i protegir la privacitat i la intimitat de les persones físiques. Tal i com queda clar a l'article 1 del RGPD on s'explica l'objecte d'aquest, engloba tres objectes:

1. Establir les normes relatives a la protecció de les persones físiques pel que fa al tractament de les dades personals i les normes relatives a la lliure circulació d'aquestes dades.
2. Protegir els drets i les llibertats fonamentals de les persones físiques i el seu dret a la protecció de les dades personals.
3. Evitar restriccions a la lliure circulació de les dades personals a la Unió Europea originades per les necessitats de protecció de dades.

L'RGPD canvia alguns articles de la LOPD i afegeix noves obligacions per a les empreses.

Els canvis més importants de l'RGPD respecte la LOPD són:

- El principi de **responsabilitat proactiva**. El nou Reglament indica que el responsable del tractament ha d'aplicar mesures apropiades per poder demostrar que el tractament és conforme al Reglament, tal i com aparèix a l'article 5. Les organitzacions han d'analitzar quines dades tracten i amb quines finalitats ho fan i han de mirar quins tipus d'operacions de tractament realitzen per tal d'aplicar les mesures que preveu l'RGPD. Aquestes mesures han de ser les adequades per complir amb el Reglament. També han de poder demostrar el compliment del Reglament davant de tercers. Aquest principi exigeix que el responsable del tractament ha de tenir una actitud proactiva, davant de tots els tractaments de dades que realitzi.
- El principi de l'**enfocament de risc**. El nou Reglament indica que s'ha de tenir en compte el risc per als drets i les llibertats de les persones. Així, algunes de les mesures només s'han d'aplicar quan hi hagi un alt risc per als drets i les llibertats. Les mesures previstes per l'RGPD s'han d'adaptar a les característiques de les organitzacions. El que pot ser bo per a una

organització no necessàriament ho ha de ser per a una altra. No és el mateix una organització que utilitza dades de milions de persones, amb tractaments que contenen informació personal sensible o volums importants de dades sobre cada persona, que una petita empresa amb poques dades i que treballa amb dades no sensibles.

A més, manté (ampliats en alguns casos) els següents principis ja recollits a la LOPD:

- **Principi de qualitat de les dades**: les dades de caràcter personal només es poden recollir per al seu tractament i sotmetre's a aquest tractament quan siguin adequades, pertinents i no excessives amb relació a l'àmbit i les finalitats determinades, explícites i legítimes per a les quals s'hagin obtingut. L'RGPD exigeix reduir al mínim necessari tant el tractament de les dades com les persones autoritzades a accedir a aquestes dades.
- **Finalitat expressa**: les dades de caràcter personal objecte de tractament no poden ser usades per a finalitats que no siguin compatibles amb aquelles per a les quals s'han recollit. Es consideren compatibles, tanmateix, el tractament posterior d'aquestes dades amb finalitats històriques, estadístiques o científiques.
- **Necessitat de consentiment de la persona afectada**: el tractament de les dades requereix el consentiment de la persona afectada.
- **Actualitat de les dades**: les dades personals que s'incorporin en un fitxer han de respondre a una situació actual.
- **Principi d'exactitud**: les dades personals han de ser susceptibles de modificació i de rectificació des del moment en què se'n coneix la modificació.
- **Deure d'informació a la persona afectada**: les persones interessades a les quals se sol·licitin dades de caràcter personal hauran de ser advertides prèviament de manera expressa, precisa i inequívoca:
  - Que les seves dades seran incloses en un fitxer, de la finalitat de la recollida i dels destinataris de la informació.
  - De l'obligatorietat o voluntarietat de donar aquestes dades.
  - De les conseqüències que porten aparellades l'obtenció de les dades o de la negativa a subministrar-les.
  - De la possibilitat d'exercir els **drets d'accés, rectificació, cancel·lació i oposició** (drets ARCO).
  - De la identificació i de l'adreça de la persona encarregada de dur a terme el tractament del fitxer o, si escau, del seu representant, perquè els afectats puguin exercir els seus drets.

A l'RGPD alguns d'aquests drets s'han ampliat:

- El dret de cancel·lació ha passat a denominar-se dret de supressió i té un aspecte molt comentat però adreçat essencialment als navegadors d'internet i xarxes socials: **el dret a l'oblit**.
- El dret al consentiment: L'RGPD requereix que l'interessat presti el consentiment mitjançant una declaració inequívoca o una acció afirmativa clara. Als efectes del nou Reglament, les caselles ja marcades, el consentiment tàcit o la inacció no constitueixen un consentiment vàlid. Igualment, perquè les dades estiguin especialment protegides, és necessari donar el consentiment exprés i per escrit.

També s'han incorporat dos nous drets: limitació del tractament i portabilitat.

- El dret a la limitació del tractament amplia el dret del consentiment; és el dret de l'usuari a posar limitacions als tractaments sobre les seves dades.
- El dret a la portabilitat de les dades inclou, per una banda, que la informació com a resposta al dret d'accés s'ha de proporcionar de manera completa i en format compatible d'ús corrent i, per una altra, que ha de poder-se transmetre a petició de l'interessat en aquest format directament a una altra organització (per exemple, si canviem de proveïdor).

Cancel·lació i bloqueig de dades

És el procediment en virtut del qual el responsable cessa en l'ús de les dades. La cancel·lació implicarà el bloqueig de les dades, que consisteix a identificar-les i reservar-les per impedir-ne el tractament, excepte per posar-les a disposició de les administracions públiques, jutges i tribunals per atendre les possibles responsabilitats nascudes del tractament, i només durant el termini de prescripció de les responsabilitats esmentades. Transcorregut aquest termini, caldrà eliminar efectivament les dades.

És precís informar a les persones afectades per l'ús de les seves dades dels ítems que es llisten a continuació, per tal que puguin exercir pròpiament els drets anteriors:

- La base jurídica del tractament.
- Interessos legítims que es volen assolir.
- Necessitat de donar un consentiment. Aquest s'ha de donar amb un acte afirmatiu clar, específic, informat i inequívoc. Pot realitzar-se en paper o a través de mitjans electrònics.
- Termini de conservació de les dades. Quan aquest venci, el responsable del tractament n'ha de limitar el tractament a través de mitjans tècnics com impedir-hi l'acces als usuaris, trasllat temporal de les dades afectades a un altre sistema de tractament o retirada temporal d'un lloc d'Internet de les dades afectades.
- Dades de contacte amb el delegat de protecció de dades (si n'hi ha).
- Existència del dret a reclamar a una autoritat de control. Això és important, ja que també existeix, en cas de tractament inadequat o negligent, el dret a obtenir una reparació, i si escau una indemnització per part del perjudicat.

- Existència de decisions automatitzades o l'elaboració de perfils (si n'hi ha). L'interessat té dret a oposar-se a que les dades personals que l'afecten siguin objecte d'un tractament, inclosa l'elaboració de perfils. El responsable del tractament ha de deixar de tractar aquestes dades personals, tret que acrediti motius legítims imperiosos per al tractament que prevalguin sobre els interessos, els drets i les llibertats de l'interessat, o per a la formulació, l'exercici o la defensa de reclamacions. L'interessat també té dret a no ser objecte de decisions basades exclusivament en un tractament automatitzat.

- Dret a la informació de l'afectat davant canvis en les seves dades: Si hi ha un canvi de les dades s'ha d'informar del canvi a l'afectat, per tal de que les verifiqui i conegui el canvi.

- Si es transmetran les dades a tercers. Cal tenir present que només s'han de fer transferències de dades personals que es tracten o que es tractaran quan es transfereixin a un tercer país o a una organització internacional si, sens perjudici de la resta de disposicions del RGPD, el responsable i l'encarregat del tractament compleixen les condicions adecuades, incloses les relatives a les transferències posteriors de dades personals des del tercer país o organització internacional a un altre tercer país o una altra organització internacional.

La informació proporcionada en tot moment ha de ser clara i fàcilment intel·ligible: No s'ha de posar lletra petita, ni usar paraules ambígües ni frases complicades o difícils d'entendre.

La LOPDGD tracta, a més, dels drets que s'apliquen al cas de menors i de dades de persones difuntes.

1.6.3 Obligacions de les empreses i els implicats en els tractaments

La necessitat de proporcionar als usuaris els drets recollits per l'RGPD, deriva en una sèrie d'obligacions per a les empreses i persones responsables i encarregades d'efectuar els tractaments, com són:

- Proporcionar procediments senzills per exercitar els drets.

- Disposar de formularis conformes amb l'RGPD i la LOPDGD per informar als usuaris i perquè aquests exerceixin els seus drets.

- Pseudonimització de les dades i les bases de dades.

- Protecció de dades des del disseny i per defecte (article 25 RGPD); això implica tenir en compte les mesures de seguretat abans de l'inici del tractament i quan aquest s'està duent a terme).

- Tenir un registre de les activitats del tractament.

- Poder demostrar davant l'autoritat que es segueix la llei si s'és sol·licitat per aquesta.
- Notificar les violacions de seguretat.

D'altra banda, no és obligatori registrar a l'autoritat de control els fitxers amb dades personals que té l'organització, com passava amb l'anterior LOPD.

Altres obligacions recollides a l'RGPD són:

- En el Capítol 4 apareix l'obligació de xifrar les dades personals, a més de guardar-les amb pseudònims (pseudonimització) per tal de que sigui més difícil d'identificar de qui són les dades.
- En aquest mateix capítol, a l'article 42, s'assenyala que els organismes es podran certificar de forma voluntària.

1.6.4 Notificació de violacions de seguretat

L'article 33 de l'RGPD, *Notificació d'una violació de la seguretat de les dades personals a l'autoritat de control*, diu que el responsable ha de notificar a l'autoritat de control la violació de seguretat, sense dilació indeguda i, si és possible, en un termini màxim de 72 hores i de conformitat amb l'article 55, tret que sigui improbable que constitueixi un risc per als drets i les llibertats de les persones.

Quan sigui probable que la violació comporti un alt risc per als drets de les persones interessades, el responsable l'ha de comunicar a les persones afectades sense dilacions indegudes i en un llenguatge clar i senzill tal i com diu l'article 34, tret que:

- El responsable hagi adoptat mesures de protecció adequades, com ara que les dades no siguin intel·ligibles per a persones no autoritzades.
- El responsable hagi aplicat mesures posteriors que garanteixen que ja no hi ha la probabilitat que es concreti l'alt risc.
- Suposi un esforç desproporcionat. En aquest cas, cal optar per una comunicació pública o una mesura semblant.

La notificació de la fallada a les autoritats dins de les 72 hores següents a partir del moment al qual el responsable n'ha tingut constància pot ser objecte d'interpretacions variades. Normalment, es considera que se'n té constància quan hi ha certesa i coneixement suficient de les circumstàncies. La mera sospita no obliga a notificar ja que, en aquests casos, no és possible conèixer suficientment l'abast del succés.

Ara bé, si sospitem que el problema pot tenir un gran impacte, és recomanable contactar amb l'autoritat de supervisió.

En cas que no sigui possible realitzar la notificació dins el termini de 72 hores, pot fer-se més tard, però cal justificar-hi les causes del retard.

L'RGPD estableix el contingut mínim de la notificació. Aquests contenen elements com:

- La naturalesa de la violació.
- Les categories de dades i d'interessats afectats.
- Les mesures adoptades pel responsable per a solucionar la fallada i, si és el cas, les mesures aplicades per pal·liar els possibles efectes negatius sobre les persones interessades.

La informació també es pot proporcionar de forma escalonada, quan no es pugui fer completament al mateix moment de la notificació.

Finalment, el responsable del tractament ha de documentar qualsevol violació de la seguretat de les dades personals, inclosos els fets que hi estan relacionats, els seus efectes i les mesures correctores que s'han adoptat.

1.6.5 El responsable, l'encarregat del tractament i el delegat de protecció de dades (DPD)

L'RGPD introdueix les figures del responsable del tractament de dades, de l'encarregat del tractament i del delegat de protecció de dades.

El capítol IV de l'RGPD tracta del responsable, de l'encarregat del tractament i del delegat de protecció de dades.

Hi pot haver representants dels responsables i/o dels encarregats del tractament quan aquests no estan establerts a la Unió, però entra dins de l'àmbit del Reglament, segons recull l'article 3, apartat 2. En aquests casos, el responsable o l'encarregat del tractament ha de designar per escrit un representant a la Unió.

El responsable del tractament

El responsable del tractament o responsable és la persona física o jurídica, autoritat pública, servei o qualsevol altre organisme que, sol o juntament amb d'altres, determina les finalitats i els mitjans del tractament. El responsable ho és i ha de poder demostrar (*accountability*) que les dades personals siguin:

- Adequades, pertinents i limitades al que és necessari en relació amb les finalitats per a les quals es tracten (minimització de dades).

- Conservades de manera que permetin identificar els interessats durant un període no superior al necessari per a les finalitats del tractament de dades personals.

- Exactes. Això implica que, quan sigui precís, s'hauran d'actualitzar. Cal adoptar les mesures raonables perquè es suprimeixin o es rectifiquin les dades personals que siguin inexactes amb les finalitats per a les quals es tracten ("exactitud");

- Tractades de manera lícita, lleial i transparent en relació amb l'interessat (licitud, lleialtat i transparència).

- Recollides amb finalitats determinades, explícites i legítimes; posteriorment no s'han de tractar de manera incompatible amb aquestes finalitats. D'acord amb l'article 89, el tractament posterior de les dades personals amb finalitats d'arxiu en interès públic, amb finalitats de recerca científica i històrica o amb finalitats estadístiques no es considera incompatible amb les finalitats inicials (limitació de la finalitat).

- Tractades de manera que se'n garanteixi una seguretat adequada, inclosa la protecció contra el tractament no autoritzat o il·lícit i contra la seva pèrdua, destrucció o dany accidental, mitjançant l'aplicació de les mesures tècniques o organitzatives adequades ("integritat i confidencialitat"), fent copies de seguretat...

Així, per exemple, el responsable del tractament serà qui haurà de decidir si les dades recollides inicialment amb el consentiment del client continuen essent vàlides per a una altra finalitat o no ho són i s'ha de tornar a demanar el consentiment al client. El responsable del tractament ha de prendre les mesures oportunes per facilitar a l'interessat tota la informació que indiquen els articles 13 (*Informació que cal facilitar quan les dades personals s'obtenen de l'interessat*) i 14 (*Informació que cal facilitar quan les dades personals no s'han obtingut de l'interessat*).

El responsable del tractament ha de facilitar a l'interessat l'exercici dels seus drets, en virtut dels articles 15 a 22.

L'encarregat del tractament

L'article 28 del RGPD tracta de l'**encarregat del tractament** o **encarregat**. L'encarregat és la persona física o jurídica, autoritat pública, servei o qualsevol altre organisme que tracta dades personals per compte del responsable del tractament. L'encarregat és únic i el nomena el responsable del tractament de les dades. L'encarregat del tractament pot, però, contractar a altres encarregats de tractament de dades amb el consentiment per escrit del responsable del tractament de dades. El tractament efectuat per l'encarregat s'ha de regir per un contracte o per un altre acte jurídic conforme al dret de la Unió o dels estats membres. Aquest contracte ha de vincular l'encarregat respecte del responsable i ha d'establir l'objecte, la durada, la naturalesa i la finalitat del tractament, així com el tipus de dades personals

i categories d'interessats i les obligacions i els drets del responsable. Aquest contracte o acte jurídic ha d'estipular, en particular, que l'encarregat:

- Tracta les dades personals únicament seguint instruccions documentades del responsable.
- Garanteix que les persones autoritzades per tractar dades personals s'han compromès a respectar-ne la confidencialitat o estan subjectes a una obligació de confidencialitat de naturalesa estatutària.
- Respecta les condicions establertes als apartats 2 i 4, per recórrer a un altre encarregat del tractament.
- Pren totes les mesures necessàries, de conformitat amb l'article 32.
- Assisteix el responsable sempre que sigui possible, d'acord amb la naturalesa del tractament i mitjançant les mesures tècniques i organitzatives adequades perquè pugui complir amb l'obligació de respondre les sol·licituds que tinguin per exercici dels drets dels interessats.
- Ajuda el responsable a garantir el compliment de les obligacions.
- A elecció del responsable, ha de suprimir o retornar totes les dades personals, una vegada finalitzada la prestació dels serveis de tractament, i suprimir les còpies existents, tret que sigui necessari conservar les dades personals en virtut del dret de la Unió o dels estats membres.
- Ha de posar a disposició del responsable tota la informació necessària per demostrar que compleix les obligacions assenyalades en aquest article 28 de l'RGPD. Així mateix, ha de permetre i contribuir a la realització d'auditories, incloses inspeccions, per part del responsable o d'un altre auditor autoritzat pel responsable.

El delegat de protecció de dades (DPD)

El Reglament, a l'article 37, introdueix la figura del **Delegat de Protecció de Dades (DPD)** i especifica quan és necessari nomenar-lo.

El Delegat de Protecció de Dades pot formar part de la plantilla del responsable o l'encarregat o bé actuar en el marc d'un contracte de serveis.

El delegat de protecció de dades és nomenat pel responsable i l'encarregat del tractament i se l'ha de nomenar quan es alguna d'aquestes condicions:

- El tractament l'efectua una autoritat o un organisme públic, tret dels tribunals que actuen en l'exercici de la seva funció judicial.
- Les activitats principals del responsable o de l'encarregat consisteixen en operacions de tractament que requereixen una observació habitual i sistemàtica a gran escala.

- Les activitats principals del responsable o de l'encarregat consisteixen en el tractament a gran escala de categories especials de dades personals i de les dades relatives a condemnes i infraccions.

El delegat de protecció de dades s'ha de designar atenent a les seves qualitats professionals i als coneixements especialitzats del dret, a la pràctica en matèria de protecció de dades i a la capacitat per exercir les funcions esmentades a l'article 39, que principalment són:

- Assessorar respecte de l'avaluació d'impacte relativa a la protecció de dades.
- Actuar com a punt de contacte de l'autoritat de control per a qüestions relatives al tractament.
- Cooperar amb l'autoritat de control.
- Informar i assessorar el responsable o l'encarregat i els treballadors sobre les obligacions que imposa la normativa de protecció de dades.
- Supervisar que es compleix l'RGPD i la resta de legislació relativa a la protecció de dades.

Això no vol dir que el DPD hagi de tenir una titulació específica, però, tenint en compte que entre les funcions del DPD s'inclou l'assessorament al responsable o l'encarregat en tot el referent a la normativa sobre protecció de dades, els coneixements jurídics en la matèria són sens dubte necessaris; també cal que compti amb coneixements aliens a l'àmbit estrictament jurídic, com per exemple en matèria de tecnologia aplicada al tractament de dades o en relació amb l'àmbit d'activitat de l'organització en la qual exerceix la seva tasca.

Altres coses a tenir en compte són:

- Un grup empresarial pot nomenar un únic delegat de protecció de dades, sempre que sigui fàcilment accessible des de cada establiment.
- Si el responsable o l'encarregat del tractament és una autoritat o un organisme públic, tret de jutjats i tribunals, es pot tenir un únic delegat de protecció de dades per diversos organismes.
- La posició del DPD a les organitzacions ha de complir els requisits que l'RGPD estableix expressament. Entre aquests requisits hi ha la total autonomia en l'exercici de les seves funcions, la necessitat que es relacioni amb el nivell superior de la direcció o l'obligació que el responsable o l'encarregat li facilitin tots els recursos necessaris per desenvolupar la seva activitat.

Els sistemes informàtics encarregats del tractament i del manteniment de dades gestionen sovint dades de caràcter personal. Quan ens trobem en aquesta situació, hem de complir l'RGPD i la resta de legislació de protecció de dades. Com que el tractament es fa en fitxers de l'empresa, la llei ens diu que hem d'adoptar les mesures necessàries per garantir la seguretat de les dades personals.

1.6.6 Dades personals

El concepte de *dada de caràcter personal* genera força confusions. Per determinar què és realment, ens hem de fixar en l'RGPD, que el defineix com "qualsevol informació sobre una persona física identificada o identificable, com ara un nom, un número d'identificació, dades de localització, un identificador en línia o un o diversos elements propis de la identitat física, fisiològica, genètica, psíquica, econòmica, cultural o social d'aquesta persona".

Així, doncs, quan parlem de *dada personal* ens referim a qualsevol informació relativa a una persona concreta. Les dades personals ens identifiquen com a individus i caracteritzen les nostres activitats en la societat, tant públiques com privades. El fet que diguem que les dades són de caràcter personal no vol dir que només tinguin protecció les vinculades a la vida privada o íntima de la persona, sinó que són dades protegides totes les que ens identifiquen o que en combinar-les permeten la nostra identificació.

Només les dades de persones físiques, i no les dades de persones jurídiques, com empreses, societats..., són dades de caràcter personal.

Tenen la consideració de dades personals:

- Nom i cognoms, data de naixement.
- Número de telèfon, adreça postal i electrònica.
- Dades biomètriques (empremtes, iris, dades genètiques, imatge, raça, veu...).
- Dades sanitàries (malalties, avortaments, cirurgia estètica...).
- Orientació sexual.
- Ideologia, creences religioses, afiliació sindical, estat civil...
- Dades econòmiques: bancàries, solvència, compres.
- Consums (aigua, gas, electricitat, telèfon...), subscripcions premsa...
- Dades judicials (antecedents penals).

Dades personals

Dades com el correu electrònic o dades biomètriques també són dades personals, ja que permeten identificar la persona. L'Agència de Protecció de Dades fins i tot considera la IP (Informe 327/2003) una dada personal.

Dades personals sensibles

No totes les dades personals són igual d'importants. Algunes s'anomenen **sensibles** a causa de la seva transcendència per a la nostra intimitat i a la necessitat d'evitar que siguin usades per discriminar-nos. No es tracta de preservar la nostra intimitat, sinó d'evitar perjudicis per l'ús que es pugui fer d'aquestes dades.

Les dades sensibles reben una protecció més alta que la resta.

Tenen la consideració de **dades sensibles** les que es refereixen a la nostra raça, opinions polítiques, a les conviccions religioses, a les afiliacions a partits polítics o a sindicats, a la nostra salut o orientació sexual, genètiques, biomètriques.

1.6.7 Infraccions i sancions de l'RGPD

L'incompliment d'una normativa legal pot comportar sancions. En el cas de l'RGPD, el règim de responsabilitat previst és de caràcter **administratiu** (menys greu que el penal i que no pot representar sancions privatives de llibertat). L'import de les sancions varia segons els drets personals afectats, volum de dades efectuats, els beneficis obtinguts, el grau d'intencionalitat i qualsevol altra circumstància que l'agència estimi oportuna.

Una diferència amb l'antiga LOPD és que no hi ha tipus de sancions (lleus, greus, molt greus). A l'article 83.2 especifica que les multes aniran en funció de la infracció. Les multes administratives poden arribar a ser d'entre 10 i 20 milions d'euros, o entre el 2 i el 4% del volum de negoci anual global. Per determinar la quantitat de les sancions es mirarà el cas particular tenint en compte:

- La naturalesa, gravetat i la durada de la infracció, estudiant la naturalesa, abast o propósit de la mateixa, així com el nombre d'interesats afectats i el nivell dels danys i perjudicis que hagin sofert.
- La intencionalidat o negligència en la infracció.
- Qualsevol mesura presa pel responsable o encarregat del tractament per solucionar i reduir els danys soferts pels interessats.
- El grau de responsabilitat de l'encarregat del tractament de les dades, segons les mesures aplicades per protegir la informació.
- Totes les infraccions anteriors dels responsables o encarregats del tractament.
- El grau de cooperació amb l'autoritat de control amb la finalitat de solucionar la infracció i mitigar els possibles efectes adversos de la infracció.
- Les categories de les dades de caràcter personal afectades per la infracció.
- La forma amb que l'autoritat de control va tenir coneixement de la infracció, en concret si el responsable o l'encarregat va notificar la infracció i en quina mesura.
- Que el responsable o l'encarregat ja hagin estat sancionats, amb advertència del compliment de les mesures.
- L'adhesió a codis de conducta o a mecanismes de certificació aprovats segons l'articulat del propi RGPD.
- Qualsevol altre factor agravant o atenuant aplicable a les circumstàncies del cas, com als beneficis financers obtinguts o a les pèrdues evitades, directa o indirectament, amb la infracció.

Exemple d'infracció i multa amb la nova llei

Donar les dades a una empresa de serveis, sense haver firmat el corresponent acord, amb les mesures de seguretat necessàries establertes per l'RGPD, que amb la LOPD era castigat fins a 300.000€, pasarà a ser multat fins a 10 milions d'euros o un 2% del volum de negoció total anual de l'any anterior.

2. Vistes i regles

2.1 Concepte de vista

Sovint per obtenir dades de diferents taules cal construir una sentència `SELECT` complexa i si en un altre moment hem de fer la mateixa consulta, cal construir de nou la sentència `SELECT`.

També cal considerar que seria molt còmode obtenir les dades d'una consulta complexa a partir d'una senzilla consulta `SELECT`.

Taula lògica o virtual

De fet, quan emprem els termes *taula lògica* o *taula virtual* ens referim al concepte de relació inherent a l'àlgebra relacional

Una vista és una *taula lògica* que permet accedir a la informació d'una o diverses taules per mitjà d'una consulta predefinida. No conté informació per si mateixa sinó que aquesta està basada en informació d'altres taules.

Com ja sabem, una vista és una *taula virtual* per mitjà de la qual es pot veure, i en alguns casos canviar, informació d'una o més taules. Una vista té una estructura semblant a una taula: files i columnes. Mai no conté dades, sinó una sentència `SELECT` que permet accedir a les dades que es volen presentar per mitjà de la vista. Podem dir que la gestió de vistes és semblant a la gestió de taules, ja que en tots dos casos en el fons parlem de relacions.

Fer un ús adient de les vistes és un aspecte clau per assolir un bon disseny d'una base de dades, ja que ens permet ocultar els detalls de les taules i mantenir la visió de l'usuari independent de l'evolució que pugui anar tenint l'estructura de taules.

La sentència `CREATE VIEW` és la instrucció proporcionada pel llenguatge SQL per a la creació de vistes.

Les vistes en PostgreSQL s'implementen utilitzant el sistema de regles (*rules*).

Aquest sistema és molt poderós i es pot emprar per a procediments, vistes i disparadors.

El sistema de regles en PostgreSQL consisteix a modificar les consultes d'acord amb regles emmagatzemades com a part de la base de dades. Aquestes consultes modificades són passades, en ordre, a l'optimitzador, posteriorment al planificador i finalment a l'executor. Això el diferencia d'altres SGBD, en què s'implementen els sistemes de regles com a procediments i disparadors emmagatzemats.

2.1.1 Creació de vistes

Per crear una vista s'utilitza `CREATE VIEW` amb la sintaxi següent:

```
1 CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW name [ ( column_name [, ...] )
      ]
2     AS query
```

Però com que les vistes s'implementen utilitzant el sistema de regles (*rules*), no hi ha diferència entre les sentències anteriors i la següent:

```
CREATE TABLE nom_vista (llista d'atributs de nom_taula);


CREATE RULE nom_rule AS ON SELECT TO nom_vista DO INSTEAD SELECT * FROM
    nom_taula;
```

Ja que això és el que fa internament la instrucció `CREATE VIEW`. Veiem que crea una relació anomenada `nom_vista` amb la definició dels atributs corresponents i posteriorment estableix una regla que permet visualitzar les ocurrències de `nom_taula` per mitjà de `nom_vista`, ja que totes dues relacions, en tenir els mateixos atributs, són compatibles. Cal tenir en compte que en aquest cas com a "nom_rule" caldria emprar el nom `_RETURN`.

Això té alguns efectes, i un és que la informació sobre una vista en el sistema de catàlegs de PostgreSQL és exactament la mateixa que per a una taula.

Les opcions opcionals en l'SQL estàndard de `CREATE VIEW` són les següents:

```
CREATE VIEW nom_vista [(columna [, ...])]

AS query

[WITH [CASCADE | LOCAL ] CHECK OPTION ]
```

- `CHECK OPTION`. Cal utilitzar aquesta opció amb les vistes actualitzables. Totes les ordres `INSERT` i `UPDATE` sobre la vista es controlen per assegurar que les dades són conformes a la condició definida en la vista. Si no, l'actualització o inserció és rebutjada.

- `LOCAL`. Per controlar la integritat de la vista.

- `CASCADE`. Per controlar la integritat sobre aquesta vista i totes les vistes dependents.

Si no s'especifica ni `LOCAL` ni `CASCADE` s'assumeix `CASCADE` per defecte.

Quan es fa servir el *psql*, es pot veure una llista de les vistes de la base de dades utilitzant la metainstrucció **\dv**:

També es pot visualitzar la definició d'una vista emprant la metainstrucció **\d nom_vista** :

2.1.2 Modificació de vistes

`ALTER VIEW` permet fer diversos canvis auxiliars referents a les propietats d'una vista. Si voleu modificar la definició de consulta de la vista, caldrà utilitzar `CREATE`

`OR REPLACE ALTER VIEW`. Com a mínim cal ser propietari de la vista per emprar una sentència `ALTER VIEW`. El superusuari pot fer els canvis sobre qualsevol vista.

Anem a veure'n uns quants exemples d'ús.

Establir un valor per defecte en una columna

El sistema de regles de PostgreSQL el veurem més endavant en aquesta mateixa unitat formativa.

`SET DEFAULT` és la manera d'establir el valor per defecte per a una columna. Un valor per defecte associat amb una columna de la vista s'insereix en les instruccions `INSERT` sobre la vista abans d'aplicar una regla (*rule*) `ON INSERT`, sempre que l'operació `INSERT` no especifiqui un valor per a la columna.

```
1 ALTER VIEW nom_vista ALTER [ COLUMN ] column SET DEFAULT expression
```

Eliminar un valor per defecte en una columna

`DROP DEFAULT` és la manera d'eliminar el valor per defecte per a una columna de la vista.

```
1 ALTER VIEW nom_vista ALTER [ COLUMN ] column DROP DEFAULT
```

Canvi de propietari d'una vista

Per modificar el propietari d'una vista, també s'ha de ser membre directe o indirecte de la regla nova que s'estableix i aquest nou paper ha de tenir permís `CREATE` en l'esquema de la vista.

```
1 ALTER VIEW nom_vista OWNER TO nou_propietari
```

Reanomenar una vista

```
1 ALTER VIEW nom_vista RENAME TO nou_nom_vista
```

Canvi de l'esquema al qual correspon la vista

Per canviar una vista a un altre esquema el propietari de la vista cal que tingui el privilegi `CREATE` en el nou esquema.

```
1 ALTER VIEW nom_vista SET SCHEMA nou_esquema
```

2.1.3 Eliminació de vistes

Per eliminar una vista podem utilitzar la mateixa ordre que l'SQL estàndard, `DROP VIEW`. Cal ser el propietari de la vista per eliminar-la.

La sintaxi de `DROP VIEW` és la següent:

```
DROP VIEW nom_vista [,...] [CASCADE | RESTRICT ]
```

Amb l'opció `CASCADE` s'esborren automàticament els objectes dependents de la vista, com poden ser altres vistes.

Amb `RESTRICT` no s'esborra la vista si hi ha objectes que en depenen. És l'opció per defecte.

2.2 Vistes del sistema

PostgreSQL disposa d'algunes vistes ja confeccionades. Algunes vistes del sistema tenen accés a les consultes més utilitzades en els catàlegs del sistema. Altres donen accés a l'estat del servidor intern.

Algunes de les principals vistes que hi ha disponibles són les següents:

- **pg_indexes** índexs
- **pg_locks** bloquejos
- **pg_rules** regles
- **pg_settings** paràmetres
- **pg_stats** estadístiques
- **pg_tables** taules
- **pg_user** usuaris
- **pg_views** vistes

Qualsevol de les vistes anteriors utilitza altres vistes també ja definides.

2.3 Avantatges i desavantatges en l'ús de les vistes

Ja sabem, llavors, quina és la definició de vista, i podeu imaginar també que aquest model de representació de les dades té els seus avantatges i desavantatges; a continuació veurem quins són els beneficis i problemes d'utilitzar vistes en un model de base de dades relacional.

2.3.1 Avantatges en l'ús de les vistes

- **Seguretat**: les vistes poden proporcionar un nivell addicional de seguretat. Per exemple, en la taula d'empleats, cada responsable de departament només tindrà accés a la informació dels seus empleats.

- **Simplicitat**: les vistes permeten ocultar la complexitat de les dades. Una base de dades es compon de moltes taules. La informació de dues o més taules es pot recuperar utilitzant una combinació de dues o més taules (relacional), i aquestes combinacions poden arribar a ser molt confuses. Creant una vista com a resultat de la combinació es pot ocultar la complexitat a l'usuari.

- **Organització**: les vistes ajuden a mantenir un noms de la base de dades per accedir a consultes complexes.

- **Exactitud de les dades demanades**: permeten accedir a un subconjunt de dades específiques, i ometen dades i informació innecessària i irrellevant per a l'usuari.

- **Amplia les perspectives de la base de dades**: proporciona diversos models d'informació basats en les mateixes dades, i els enfoca envers diferents usuaris amb necessitats específiques. Mostrar la informació des de diferents angles ens ajuda a crear ambients de treball i operació d'acord amb els objectius de l'empresa. S'ha d'avaluar el perfil i els requisits d'informació dels usuaris destinataris de la vista.

- **Transparència en les modificacions**: l'usuari final no es veurà afectat pel disseny o alteracions que es fan en l'esquema conceptual de la base de dades. Si el sistema requereix una modificació en el seu funcionament intern, es podran afectar diverses estructures que proveeixen l'acompliment d'aquest, i es pretén que els usuaris finals ho no adverteixin com a alteracions.

2.3.2 Possibles desavantatges en l'ús de les vistes

Encara que l'ús de vistes implica molts avantatges, i molt profitosos tots, també comporta una sèrie de desavantatges que cal considerar a l'hora de dissenyar una base de dades relacional. Aquests desavantatge tenen a veure amb les limitacions del motor de base de dades que s'utilitzarà. Per això en cada implementació d'un SGBD relacional veurem diferents restriccions en aquest aspecte.

En el SGBD que ens ocupa, PostgreSQL, les vistes no són actualitzables; és a dir, si bé és cert que són tractades com a taules, no és possible fer `INSERT`, `DELETE` ni `UPDATE` sobre les vistes. Aquest desavantatge és una característica particular del PostgreSQL, atès que aquesta qualitat sí que està disponible en altres motors de bases de dades com ORACLE, Informix i SQL Server, però cal notar que el PostgreSQL cobreix aquesta mancança en les vistes amb la creació de regles (`CREATE RULE`) que permeten omplir el buit deixat per la vista i ens permeten controlar quin tipus de modificacions podem fer per mitjà de la vista seguint les regles del negoci per al qual fa servei la base de dades.

2.4 Vistes actualitzables

2.4.1 Restriccions de les vistes actualitzables

En la majoria d'SGBD hi ha una sèrie de restriccions que cal considerar en l'esborrament, l'actualització i la inserció de vistes en una taula per mitjà d'una vista:

Esborrament de files per mitjà d'una vista: per esborrar files d'una taula per mitjà d'una vista, aquesta s'ha de crear:

- amb files d'una sola taula
- sense utilitzar la clàusula `GROUP BY` ni `DISTINCT`
- sense usar funcions de grup ni referències a pseudocolumnes

Actualització de files per mitjà d'una vista: per actualitzar files en una taula per mitjà d'una vista, aquesta ha d'estar definida segons les restriccions anteriors i, a més, cap de les columnes que es volen actualitzar no s'ha d'haver definit com una expressió.

Inserció de files per mitjà d'una vista. per inserir files en una taula per mitjà d'una vista s'han de tenir en compte totes les restriccions anteriors, i a més totes les columnes obligatòries de la taula associada han d'estar presents en la vista.

2.4.2 El sistema de regles emprat en el PostgreSQL

Situació

Tenim una taula de clients i una taula de línies telefòniques. Volem tenir una vista des de la qual es vegin totes les línies i els camps dels clients a qui pertanyen.

També volem crear nous clients i línies, i a més poder modificar les dades tant de la línia com del client, emprant aquesta vista.

```
CREATE SEQUENCE linies_id_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 1
  CACHE 1;

CREATE TABLE clients
(
  client_id serial NOT NULL,
  nom character varying(100),
  CONSTRAINT "PK_clients" PRIMARY KEY (client_id)
)
WITH (
  OIDS=FALSE
);


CREATE TABLE linies
(
  client_id integer,
  numero character(9),
  linia_id integer NOT NULL DEFAULT nextval('linies_id_seq'::regclass),
  CONSTRAINT "PK_linia" PRIMARY KEY (linia_id),
  CONSTRAINT "FK_linies_clients" FOREIGN KEY (client_id)
      REFERENCES clients (client_id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
  OIDS=FALSE
);
```

La vista

```
CREATE VIEW clients_linies AS

SELECT c.client_id, linia_id, nom, numero

FROM clients c, linies l

WHERE c.client_id = l.client_id;
```

Ara verifiquem el funcionament de la vista:

```
SELECT * FROM clients_linies;
```

Les regles d'inserció

La regla següent ens permetrà inserir un client nou i la seva línia telefònica per mitjà de la vista. Observem que la condició d'inserció indica que l'identificador del client sigui *null*, ja que inserirem aquest valor emprant el nou valor de la seqüència definit per defecte en la definició de la taula corresponent. Com a segona acció inserim a la taula *línies* el mateix valor de la seqüència més el número de telèfon. Per tant, com a conseqüència d'aquesta regla quan inserim per mitjà de la vista un nou client amb el seu número de línia de telèfon tan sols cal emprar aquests dos valors: el nom i el telèfon.

```
CREATE RULE ins_clients_linies_nou AS

 ON INSERT TO clients_linies

WHERE NEW.client_id IS NULL

DO INSTEAD

(

INSERT INTO clients (nom)

VALUES (NEW.nom)

;

INSERT INTO linies (client_id, numero)

VALUES (currval('clients_client_id_seq'),NEW.numero);

 );
```

Definim ara la regla corresponent a la inserció d'un número de telèfon nou per a un client existent. En tot cas suposem que el valor de l'identificador del client que utilitzem existeix a la taula *clients* per mantenir la integritat referencial. Així doncs, tan sols necessitarem emprar dos valors en la inserció: l'identificador del client i el nou número de telèfon.

```
CREATE RULE ins_client_linia_existent AS

ON INSERT TO clients_linies

WHERE NEW.client_id IS NOT NULL

DO INSTEAD
INSERT INTO linies (client_id, numero)

VALUES (NEW.client_id, NEW.numero);
```

Ara definim una última regla incondicional per als casos d'inserció i així ens assegurem que en altres casos no definits no faci res.

```
CREATE RULE ins_client_linia_nothing AS

ON INSERT TO clients_linies

DO INSTEAD NOTHING;
```

Amb posterioritat a les insercions següents podrem, mitjançant un `SELECT` tant de la vista com de les taules implicades, hem de verificar la correctesa de les regles.

Primerament inserim un client nou amb la seva línia i així verifiquem el funcionament de la regla **ins_clients_linies_nou**

```
INSERT INTO clients_linies (nom, numero) VALUES ('Pau Pi', '234-4567');
```

A continuació inserim una línia nova per a un client existent i així verifiquem el funcionament de la regla **ins_client_linia_existent**

```
INSERT INTO clients_linies (client_id, numero) VALUES (3, '987-1233');
```

Les regles d'actualització

Regla que ens permet actualitzar el nom del client per mitjà de la vista

```
CREATE RULE upd_clients_linies_client AS

ON UPDATE TO clients_linies

 WHERE NEW.client_id IS NOT NULL

DO INSTEAD

UPDATE clients

SET nom= NEW.nom

WHERE client_id = NEW.client_id;
```

Regla que ens permet actualitzar el número de la línia de telèfon per mitjà de la vista

```
CREATE RULE upd_clients_linies_linia AS

ON UPDATE TO clients_linies

WHERE NEW.linia_id IS NOT NULL

DO INSTEAD

UPDATE linies

SET numero = NEW.numero

WHERE linia_id = NEW.linia_id;
```

Regla incondicional

```
CREATE RULE upd_clients_linies_nothing AS

ON UPDATE TO clients_linies

DO INSTEAD NOTHING;
```

Fem l'actualització seguint la definició de la regla **upd_clients_linies_client**

```
UPDATE clients_linies SET nom = 'Josep Pons' WHERE client_id = 3;
```

Fem l'actualització seguint la definició de la regla **upd_clients_linies_linia**

```
UPDATE clients_linies SET numero = '1-800-8888' WHERE linia_id = 4;
```

També podem fer actualitzacions emprant les dues regles **upd_clients_linies_client** i **upd_clients_linies_linia** alhora

```
UPDATE clients_linies
SET numero = '1-800-7777', nom = 'Maria Bassas'
WHERE client_id = 3 AND linia_id = 6;
```

2.4.3 Traducció de consultes sobre vistes

La informació sobre una vista en el sistema de catàlegs de PostgreSQL és la mateixa que per a una taula. D'aquesta manera, per als traductors de *queries*, no hi ha diferència entre una taula i una vista, ja que són el mateix: relacions.

El sistema de regles incorpora les definicions de les vistes en l'arbre de traducció original (*querytree*). La implementació del sistema de regles és una tècnica anomenada *reescriptura de la consulta*. El sistema de reescriptura és un mòdul que hi ha entre l'etapa del traductor i el planificador/optimitzador.

El sistema de reescriptura de la consulta processa l'arbre tornat per l'etapa de traducció, que representa una consulta de l'usuari, i si existeix una regla que calgui aplicar a la consulta, reescriu l'arbre d'una manera alternativa.

El *querytree* és la representació interna d'una consulta en la qual se separen i agrupen els components en forma d'arbre.

Components d'un *querytree*:

- La instrucció: `SELECT`, `UPDATE`, `INSERT` o `DELETE`.
- La *range table* (abast de la taula): inclou les relacions que utilitza.
- La *result relation* (relació resultant): un índex a la *range table* on hi haurà els resultats. Generalment `SELECT` no l'inclou.
- La *target list* (llista d'etiquetes): és la llista d'elements entre el `SELECT` i el `FROM` en una instrucció de `SELECT`, la llista de files afectades en `INSERT` i `UPDATE`. No s'utilitza en `DELETE`.
- La qualificació correspon al `WHERE` i indica si cal actualitzar o no una fila.
- El *join tree* (arbre del `JOIN`), combina parts del `FROM` i el `WHERE` per descriure l'estructura del `JOIN`.
- *others* (la resta), altres clàusules com `ORDER BY`.

Els beneficis d'implementar les vistes amb el sistema de regles són que l'optimització té tota la informació sobre quines taules han de ser revisades, les relacions entre aquestes taules, les qualificacions restrictives a partir de la definició de les vistes i les qualificacions de la *query* original, tot en un únic arbre de traducció. I aquesta és també la situació quan la *query* original és una JOIN entre vistes.

L'optimitzador cal que decideixi quina és la millor ruta per executar la *query*. Com més informació tingui l'optimitzador, millor serà la decisió. I la manera com s'implementa el sistema de regles de PostgreSQL assegura que tota la informació sobre la *query* és utilitzable.

Per comprendre com treballa el sistema de regles, és necessari conèixer quan s'invoca i quines són les seves entrades i els seus resultats.

El sistema de regles se situa entre el traductor de la *query* i l'optimitzador. Agafa la sortida del traductor, un *querytree*, i les regles de reescriptura del catàleg `pg_rewrite`, que són també *querytree*, amb alguna informació extra, i crea cap o molts *querytree* com a resultat. D'aquesta manera, l'entrada i la sortida són sempre tal com el traductor mateix les podria haver produït i tot apareix representable com una instrucció SQL.

Aquests *querytree* són visibles quan arrenquem el motor de PostgreSQL amb nivell de depuració 4 i teclegem *queries* en l'interfície d'usuari interactiva. Les accions de les regles emmagatzemades en el catàleg de sistema `pg_rewrite` estan emmagatzemades també com a *querytree*. No estan formades com la sortida del *debug*, però contenen exactament la mateixa informació.

Les representacions d'SQL de *querytree* són suficients per entendre el sistema de regles.

A continuació mostrarem un exemple de com es poden implementar vistes utilitzant el sistema de regles.

El sistema de reescriptura fa els passos següents:

- Pren la consulta donada per la part d'acció de la regla.
- Adapta la llista objectiu per recollir el nombre i ordre dels atributs donats en la consulta d'usuari.
- Afegeix la qualificació donada en la clàusula `WHERE` de la consulta de l'usuari a la qualificació de la consulta donada en la part de l'acció de la regla.

D'acord amb això la consulta de l'usuari serà reescrita de la manera següent:

La reescriptura es fa en la representació interna de la consulta de l'usuari tornada per l'etapa de traducció però la nova estructura de dades representarà la consulta anterior.

Programació de bases de dades

Joan Anton Pérez Braña

Índex

Introducció

En aquests moments coneixem la potència del llenguatge SQL per efectuar consultes complexes i actualitzacions (insercions, modificacions i eliminacions) en les bases de dades. L'avantatge d'aquestes instruccions, que constitueixen el que s'anomena *llenguatge SQL autosuficient*, és que permeten un accés directe a la base de dades.

La possibilitat d'accedir directament a la base de dades per gestionar les dades no elimina la necessitat de continuar desenvolupant programes per gestionar les dades emmagatzemades. En l'actualitat, hi ha diverses tècniques que ens permeten desenvolupar programes i subprogrames per automatitzar tasques de gestió de dades en les bases de dades.

Així, ens trobem que els principals SGBD proporcionen uns llenguatges de tercera generació anomenats *extensions procedimentals del llenguatge SQL*, que permeten dissenyar petits programes que s'han d'executar dins de guions i dissenyar subprogrames (funcions i accions) que s'emmagatzemen dins la base de dades i es poden executar des de múltiples entorns.

PostgreSQL és un gestor de bases de dades relacional orientat a objectes (ORDBMS en les sigles en anglès) molt conegut i usat en entorns de programari lliure perquè compleix els estàndards SQL92 i SQL99, i també pel conjunt de funcionalitats avançades que suporta, cosa que el situa al mateix nivell o en un nivell millor que molts SGBD comercials.

La versió de PostgreSQL que s'ha utilitzat durant la redacció d'aquest material, i en els exemples, és la 9.0, l'última versió estable en aquest moment.

En l'apartat "El dialecte SQL de PostgreSQL" coneixereu, en primer lloc, les característiques pròpies del dialecte SQL de PostgreSQL i el conjunt de funcions predefinides que aporta. Veurem també la gestió de la concurrència en les transaccions que fa PostgreSQL. I si ens trobem amb la necessitat d'agrupar una seqüència d'instruccions SQL per aconseguir un resultat determinat i és possible que ens interessi poder repetir diverses vegades, podrem construir guions.

En l'apartat "PL/PgSQL: extensió procedimental del llenguatge SQL" ens endinsem en el coneixement de l'extensió procedimental del llenguatge SQL que aporta PostgreSQL (en concret en PL/PgSQL) i, com és normal en l'estudi de qualsevol llenguatge de programació, estudiarem l'estructura dels programes, els tipus de dades i les estructures de control.

En els apartats "Cursors i control d'errors" i "Disparadors" aprofundireu en la utilització del llenguatge PL/PgSQL per escriure codi que queda emmagatzemat en la base de dades i desenvolupar funcions utilitzant cursors i gestionant errors; també veureu la funcionalitat i la implementació de disparadors.

Com a última consideració cal tenir en compte que, per aprendre a aplicar amb agilitat les tècniques de desenvolupament esmentades, serà imprescindible implementar els exemples il·lustratius, fer totes les activitats proposades i els exercicis d'autoavaluació.

Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Desenvolupa procediments emmagatzemats avaluant i utilitzant les sentències del llenguatge incorporat en el sistema gestor de base de dades corporatiu.

- Identifica les eines disponibles en el sistema gestor de bases de dades per a editar guions.
- Defineix guions per automatitzar tasques que gestionen la base de dades.
- Identifica els tipus de dades, identificadors, variables i constants.
- Utilitza estructures de control de flux i llibreries de funcions.
- Desenvolupa procediments i funcions d'usuari
- Gestiona els possibles errors dels procediments i funcions i controla les transaccions.
- Gestiona els possibles errors dels procediments i funcions i controla les transaccions.
- Utilitza cursors per manipular les dades d'una base de dades.
- Utilitza les funcions incorporades en el sistema gestor de bases de dades.
- Desenvolupa disparadors.

1. El dialecte SQL de PostgreSQL

PostgreSQL és un sistema de bases de dades relacionals (RDBMS). Això significa que és un sistema de gestió de dades en què aquestes estan emmagatzemades en relacions. Com coneixem, el terme *relació* és el terme matemàtic que emprem per designar una taula. La idea d'emmagatzemar dades en les taules és molt comuna avui dia i fins i tot pot semblar una cosa òbvia, però com sabem, hi ha altres models.

Cada taula és un conjunt de files. Cada fila d'una taula donada té el mateix conjunt d'atributs, representats sota els noms de cada columna o camp, i cada columna és d'un tipus de dades específic. Mentre que les columnes tenen un ordre fix en cada fila, és important recordar que l'SQL no garanteix l'ordre de les files de la taula, encara que poden ser ordenades de manera explícita per ser visualitzades.

Les taules s'agrupen en diferents esquemes i aquests conformen les bases de dades. Una col·lecció de bases de dades gestionades per una instància de servidor PostgreSQL constitueix un conjunt de bases de dades (clúster).

1.1 Tipus de dades

Una taula d'una base de dades relacional és molt similar a una taula en paper: es compon de files i columnes. El nombre i ordre de les columnes és fix, i cada columna té un nom. El nombre de files és variable, ja que reflecteix la quantitat de dades emmagatzemades en un moment donat. Quan una taula es llegeix, les files es mostren sense cap ordre, llevat que es demani expressament un criteri d'ordenació. Això és una conseqüència del model matemàtic subjacent en SQL, el model relacional.

Cada columna té un tipus de dades. El tipus de dades limita el conjunt de valors possibles que es poden assignar a una columna i assigna la semàntica de les dades emmagatzemades a la columna perquè puguin ser utilitzades en diferents càlculs. Per exemple, en una columna declarada com de tipus numèric no s'accepten cadenes de text arbitràries, i les dades emmagatzemades en una columna d'aquest tipus es poden utilitzar per a càlculs matemàtics. D'altra banda, una columna declarada com de tipus cadena de caràcters accepta gairebé qualsevol tipus de dades, però no es presta a càlculs matemàtics, encara que sí a altres operacions, com la concatenació de cadenes de caràcters.

El PostgreSQL inclou un conjunt important de tipus de dades que s'adapten a moltes aplicacions. Els usuaris també poden definir els seus tipus de dades propis. La majoria dels tipus predefinits de dades tenen noms i semàntica bastant òbvia. Alguns dels tipus de dades utilitzats amb freqüència són **integer** per a nombres enters, **numeric** per als nombres fraccionaris, **text** per a cadenes de caràcters, **date**

per a les dates, **time** per a valors de temps del dia, i **timestamp** per a valors que contenen la data i l'hora.

1.1.1 Tipus lògics

El PostgreSQL incorpora el tipus lògic **boolean**, també anomenat **bool**. Ocupa un byte d'espai d'emmagatzemament i pot emmagatzemar els valors *fals* i *veritable* (taula 1.1).

TAULA 1.1. Valors de tipus booleà

Valor	Nom
Fals	false, 'f', 'n', 'no', 0
Vertader	true, 't', 'y', 'yes', 1

El PostgreSQL suporta els operadors lògics següents: **and**, **or** i **not**.

Encara que els operadors de comparació s'apliquen sobre pràcticament tots els tipus de dades proporcionades pel PostgreSQL, atès que el seu resultat és un valor lògic, en descriurem el comportament en la taula 1.2.

TAULA 1.2. Operadors de comparació

Operador	Descripció
>	Major que
<	Menor que
<=	Menor o igual que
>=	Major o igual que
<>	Diferent de
!=	Diferent de

1.1.2 Tipus numèrics

El PostgreSQL disposa dels tipus enters **smallint**, **int** i **bigint**, que es comporten com ho fan els enters en molts llenguatges de programació.

Espai dels valors de tipus numeric

Sens dubte, aquest avantatge té un cost, i els valors de tipus **numeric** ocupen un espai d'emmagatzemament considerablement gran i les operacions s'executen sobre aquests molt lentament. Per tant, no és aconsellable utilitzar el tipus **numeric** si no es necessita una alta precisió o es prioritza la velocitat de processament.

Els nombres amb punt flotant, dels tipus **real** i **double precision**, emmagatzemen quantitats amb decimals. Una característica dels nombres de punt flotant és que perden exactitud a mesura que creixen o decreixen els valors.

Encara que aquesta pèrdua d'exactitud no sol tenir importància en la majoria de vegades, el PostgreSQL inclou el tipus **numeric**, que permet emmagatzemar quantitats molt grans o molt petites sense pèrdua d'informació. Vegeu la taula 1.3.

Taula 1.3. Tipus numèrics

Nom	Mida	Altres noms	Comentari
smallint	2 bytes	int2	
int	4 bytes	int4, integer	
bigint	8 bytes	int8	
numeric(p,e)	11 + (p/2)		'p' és la precisió, 'e' és l'escala
real	4 bytes	float, float4	
double precision	8 bytes	float8	
serial			No és un tipus, és un enter autoincrementable

CREATE

Si es declaren diverses columnes amb *serial* en una taula, es crearà una seqüència i un índex per a cada una.

La declaració **serial** és un cas especial, ja que no es tracta d'un nou tipus. Quan s'utilitza com a nom de tipus d'una columna, aquesta prendrà automàticament valors consecutius en cada registre nou.

Exemple d'una taula que defineix la columna *foli* com a tipus **serial**.

```
create table Factura(
foli serial,
client varchar(30),
suma real
);
```

El PostgreSQL respondria aquesta instrucció amb dos missatges:

- En el primer avisa que s'ha creat una seqüència de nom `factura_foli_seq`:

```
NOTICE: CREATE TABLE will create implicit sequence '
    factura_foli_seq' for SERIAL column '
```

- En el segon avisa de la creació d'un índex únic en la taula utilitzant la columna *foli*:

```
NOTICE: CREATE TABLE / UNIQUE will create implicit index '
    factura_foli_key' for table 'factura'
```

Operadors numèrics

Teniu més informació sobre la gestió de seqüències en la secció "Annexos" del web del mòdul.

El PostgreSQL ofereix un conjunt d'operadors numèrics predefinits, que presentem en la taula 1.4.
A continuació tenim alguns exemples de l'ús d'aquests operadors:

```
select |/ 9;
select 43 % 5;
select !! 7;
select 7!;
```

1.1.3 Tipus de caràcters

Els valors de cadena de PostgreSQL es delimiten per cometes simples.

TAULA 1.4. Operadors numèrics predefinits de PostgreSQL

Símbol	Operador
+	Addició
-	Resta
*	Multiplicació
/	Divisió
%	Mòdul
^	Exponenciació
|/	Arrel quadrada
||/	Arrel cúbica
!	Factorial
!!	Factorial com a operador fix
@	Valor absolut
&	AND binari
|	OR binari
#	XOR binari
~	Negació binària
<<	Corriment binari a l'esquerra
>>	Corriment binari a la dreta

```
demo=# select 'Hola món';
?column?
------------
Hola món
(1 row)
```

Es pot incloure una cometa simple dins d'una cadena amb **\'** o **'' ‘**:

```
demo=# select 'ell va dir: Hola';
?column?
-----------------
ell va dir: Hola
(1 row)
```

Caràcters especials

Les cadenes poden contenir caràcters especials amb les anomenades *seqüències d'escapament*, que s'inicien amb el caràcter '\':

TAULA 1.5. Seqüències d'escapament

Caràcter	Descripció
\n	nova línia
\r	retorn
\t	tabulador
\b	retrocés
\f	canvi de pàgina
\\	el caràcter \

Les cometes dobles delimiten identificadors que contenen caràcters especials.

Les seqüències d'escapament se substitueixen pel caràcter corresponent:

```
demo=# select 'Això està en \n dues línies';
?column?
--------------------------
Això està en
dues línies
(1 row)
```

El PostgreSQL ofereix els tipus següents per a cadenes de caràcters (taula 1.6):

Taula 1.6. Tipus de cadena de caràcters

Tipus	Altres noms	Descripció
char(n)	character(n)	Reserva *n* espais per emmagatzemar la cadena
varchar(n)	character varying(n)	Utilitza els espais necessaris per emmagatzemar una cadena més petita o igual que *n*
text		Emmagatzema cadenes de qualsevol magnitud

Operadors amb cadenes de caràcters

En la taula 1.7 es descriuen els operadors per a cadenes de caràcters.

Taula 1.7. Operador per a cadenes de caràcters

Operador	Descripció	Distingeix majúscules i minúscules?
||	Concatenació	-
~	Correspondència amb expressió regular	Sí
~*	Correspondència amb expressió regular	No
!~	No correspondència amb expressió regular	Sí
!~*	No correspondència amb expressió regular	-

Sobre les cadenes també podem utilitzar els operadors de comparació que ja coneixem.

En aquest cas, el resultat de la comparació "més petit que" és *fals*:

```
demo=# select 'HOLA' < 'hola';
?column?
----------
f
(1 row)
```

1.1.4 Dates i hores

En la taula 1.8 es mostren els tipus de dades referents al temps que ofereix el PostgreSQL.

TAULA 1.8. Dades referents al temps

Tipus de dada	Unitats	Mida	Descripció	Precisió
date	dia-mes-any	4 bytes	Data	Dia
time	hrs:min:seg:micro	4 bytes	Hora	Microsegon
timestamp	dia-mes-any	8 bytes	Data més hora	Microsegon

Hi ha un tipus de dada **timez** que inclou les dades del tipus *time* i, a més, la zona horària.

El tipus de dades **date** emmagatzema el dia, mes i any d'una data donada i es mostra per omissió amb el format següent: *YYYY-MM-DD*

```
demo=# create table Persona ( naixement date );
CREATE
demo=# insert into persona values ( '2004-05-22' );
INSERT 17397 1
demo=# select * from persona;
naixement
------------
2004-05-22
(1 row)
```

Per canviar el format de presentació, hem de modificar la variable d'entorn **datestyle**:

```
demo=# SHOW DATESTYLE;
NOTICE: DateStyle is ISO with US (NonEuropean) conventions
SHOW VARIABLE
demo=# SET DATESTYLE TO 'SQL, EUROPEAN';
SET VARIABLE
demo=# SHOW DATESTYLE;
NOTICE:  DateStyle is SQL with European conventions
SHOW VARIABLE
demo=# RESET DATESTYLE;
RESET VARIABLE
demo=# SHOW DATESTYLE;
NOTICE:  DateStyle is ISO with US (NonEuropean) conventions
SHOW VARIABLE
```

1.1.5 Matrius

Definició

Les matrius no compleixen la primera forma normal de Cood, per la qual cosa molts els consideren inacceptables en el model relacional.

El tipus de dades *array* és una de les característiques especials de PostgreSQL, i permet l'emmagatzemament de més d'un valor del mateix tipus en la mateixa columna.

```
create table Estudiant (
nom varchar(30),
```

```
parcials int [3]
);
```

La columna *parcials* accepta tres qualificacions dels estudiants.

Les matrius, igual que qualsevol columna quan no s'especifica el contrari, accepten valors nuls. Els valors de la matriu s'escriuen sempre entre claus.

```
demo=# insert into Estudiant values ( 'Josep' );
INSERT 17416 1
demo=# insert into Estudiant values ( 'Joan' , '{90,95,97}' );
INSERT 17417 1
```

També és possible assignar un sol valor de la matriu:

```
demo=# insert into Estudiant( nom, parcials[2]) values ( 'Pere' , '90');
INSERT 17418 1
demo=# select * from Estudiant ;
nom | parcials
——————+——————
Josep |
Joan | {90,95,97}
Pere | [2:2]={90}
(3 rows)
```

Per seleccionar un valor d'una matriu en una consulta s'especifica entre claudàtors la cel·la que es visualitzarà:

```
demo=# select nom, parcials[3] from Estudiant;
nom | parcials
——————— +————————
Josep |
Joan | 97
Pere |
(3 rows)
```

Només en Joan té qualificació en el tercer parcial.

En molts llenguatges de programació, les matrius s'implementen amb longitud fixa; PostgreSQL en permet augmentar la mida dinàmicament.

La columna *parcials* del registre *Pau* inclou quatre cel·les i només l'última té valor.

```
demo=# insert into Estudiant( nom, parcials[4]) values ( 'Pau' , '70');
INSERT 17419 1
demo=# select * from Estudiant;

nom | parcials
———————— +————————
Josep |
Joan | {90,95,97}
Pere | [2:2]={90}
Pau | [4:4]={70}
(4 rows)
```

Mitjançant la funció **array_dims()** podem conèixer les dimensions d'una matriu:

```
demo=# select nom, array_dims(parcials) from Estudiant;
nom | array_dims
———————— +————————
Josep |
```

```
Joan | [1:3]
Pere | [2:2]
Pau | [4:4]
(4 rows)
```

1.2 Funcions

Una funció és una agrupació de sentències que s'executa com una unitat. Són molt útils quan cal fer sovint manipulacions automatitzades de taules. Aquestes s'emmagatzemen en la base de dades.

En el PostgreSQL una funció i un procediment emmagatzemat és exactament el mateix. La diferència és més conceptual que concreta.

Les funcions accepten uns valors d'entrada, fan alguna consulta o manipulació sobre aquests, i tornen un valor de sortida.

El PostgreSQL proporciona tres tipus de funcions:

- **Funcions de llenguatge de consultes**, escrites en SQL: aquestes funcions executen una llista arbitrària de consultes SQL, i tornen els resultats de la darrera consulta de la llista. Poden ser:
  1. *Funcions sobre tipus base*: no té arguments i sols retorna un tipus base, com per exemple un *int4*.
  2. *Funcions sobre tipus compostos*: en especificar funcions amb arguments de tipus compostos també cal especificar els atributs d'aquests arguments.
- **Funcions de llenguatge procedural**, escrites, per exemple en PL/PgSQL.
- **Funcions de llenguatge de programació**, escrites en un llenguatge de programació compilat, com per exemple en C.

Un exemple d'una funció SQL sobre tipus base pot ser la següent:

```
CREATE FUNCTION suma(int4, int4) RETURNS int4
AS $$
SELECT $1 + $2;
$$LANGUAGE SQL;
```

Si fem la consulta d'aquesta funció passant dos nombres com a paràmetres, com per exemple el 3 i el 7:

```
SELECT suma(3,7) AS resultat;

resultat
————————
10
```

Després veurem detalladament la sintaxi de la creació de les funcions, però ja podem avançar que `$n` significa l'ordre dels paràmetres, `$1` el primer paràmetre, `$2` el segon, etc.

> Les funcions en PostgreSQL es poden escriure en diferents llenguatges, com per exemple en C, SQL i PL/PgSQL.
> Es creen amb `CREATE FUNCTION` i s'eliminen amb `DROP FUNCTION`.

El PostgreSQL disposa de diverses funcions predefinides que es poden consultar amb l'ordre **\df** de **psql**, des del terminal.

Podeu veure exemples de funcions predefinides en la secció "Annexos" del web del mòdul.

La consulta ens informa de l'esquema a què pertany, el nom de la funció, el tipus de dades de sortida i el tipus de dades dels arguments.

També és pot utilitzar per veure aquesta informació d'una funció especifica, com per exemple per a `UPPER`. Farem: **\df UPPER** i ens informa en concret d'aquesta funció:

```
\df upper
Listado de funciones
Schema | Nombre | Tipo de dato de salida | Tipos de datos de argumentos
------------+--------+------------------------+------------------------------
pg_catalog | upper | text | text
```

Provem la funció i veiem que accepta una sèrie de caràcters, els converteix en majúscules i torna la nova sèrie:

```
SELECT UPPER('abcdef');
ABCDEF
```

1.3 Transaccions i bloquejos

Fins ara, hem evitat qualsevol discussió en profunditat sobre els aspectes multiusuari del PostgreSQL, i simplement hem indicat la visió idealitzada que, com qualsevol base de dades relacional amb bones prestacions, el PostgreSQL oculta els detalls de suport a múltiples usuaris concurrents.

El PostgreSQL proporciona un servidor de base de dades ràpid i eficient que ofereix un servei als seus clients com si tots els usuaris simultanis hi tinguessin accés exclusiu. No obstant això, la realitat és que el PostgreSQL, encara que és molt capaç, no pot fer màgia, i l'aïllament de cada usuari de tots els altres requereix un treball de fons.

En aquest apartat, tindrem en compte dos aspectes importants que han de suportar els SGBD per a múltiples usuaris: les transaccions i el bloqueig.

Les transaccions permeten recopilar una sèrie de canvis discrets en la base de dades en una unitat de treball única.

El bloqueig evita conflictes quan diferents usuaris volen fer canvis en la base de dades al mateix temps.

Per tant, tractarem els temes següents:

- Què constitueix una transacció
- Els beneficis de les transaccions en una base de dades d'un sol usuari
- Transaccions amb múltiples usuaris
- Bloqueig de taula i fila.

1.3.1 Concepte de transacció

Com hem esmentat anteriorment, en una situació ideal com la que hem estat suposant fins ara, s'han enregistrat els canvis en la base de dades mitjançant accions declaratives simples.

No obstant això, en aplicacions del món real, aviat arriba un punt en el qual s'han de fer diversos canvis en una base de dades que no es poden expressar en una sola sentència d'SQL.

Tot i que no es fan en una sola declaració, nosaltres necessitem que tots els canvis que es produeixin per actualitzar la base de dades es facin correctament. Si ocorre un problema amb qualsevol part del grup dels canvis, llavors cal que cap dels canvis fets a la base de dades no sigui enregistrat com a definitiu. En altres paraules, cal fer una sola unitat de treball indivisible, en la qual s'executin diverses instruccions SQL per executar-les completament, ja sigui amb totes les declaracions SQL executades amb èxit o sense l'execució de cap d'aquestes.

Exemple d'unitat de treball indivisible en la qual s'executen diverses instruccions

L'exemple clàssic és el procés de transferència de diners entre dos comptes d'un banc, que poden estar representats en les diferents taules d'una base de dades, de manera que a un compte se li carrega una quantitat de diners i en l'altre s'ingressen. Cap banc no pot romandre en el negoci si de tant en tant desapareixen diners en algunes operacions, tan comunes com pot ser una transferència entre comptes (en cas de fer-se la primera operació i fallar la segona).

En bases de dades basades en ANSI SQL, i PostgreSQL ho és, dur a terme aquesta tasca *de tot o res* s'aconsegueix amb les transaccions.

Una transacció és una unitat lògica de treball que no ha de ser dividida.

Agrupació dels canvis fets en les dades en unitats lògiques

Què s'entén per *una unitat lògica de treball*?

És simplement un conjunt de canvis lògics de la base de dades, en el qual es produeixen tots els canvis o cap d'aquests, igual que l'exemple anterior de la transferència de diners entre comptes. En PostgreSQL, aquests canvis són controlats per quatre sentències clau:

- `START` o `BEGIN` inicia una transacció.
- `SAVEPOINT` ***nom_punt_de_salvaguarda*** demana al servidor que recordi l'estat actual de la transacció. Aquesta declaració només es pot utilitzar després d'un `BEGIN` i abans d'una `COMMIT` o `ROLLBACK`, és a dir, mentre que una transacció s'està fent.
- `COMMIT` diu que tots els elements de la transacció s'han completat (accions sobre les dades dins de la base de dades), i el nou estat ha de ser persistent i accessible a totes les transaccions simultànies i posteriors.
- `ROLLBACK` **[TO *nom_punt_de_salvaguarda*]** diu que la transacció hagi de ser abandonada, i que es cancel·lin tots els canvis fets en les dades de transaccions d'SQL. La base de dades ha d'aparèixer en tots els usuaris, com si cap dels canvis s'hagués produït després del `BEGIN` anterior, i la transacció es tanca. En la versió alternativa, amb l'addició de la clàusula `TO`, es permet revertir a un *punt_de_ salvaguarda*, i no es completa una transacció.

Accés multiusuari simultani a les dades

Un segon aspecte de les transaccions és que tota transacció a la base de dades està aïllada d'altres transaccions que tenen lloc en la base de dades al mateix temps. Idealment cada transacció es comporta com si no tingués accés exclusiu a la base de dades. Malauradament, com veurem més endavant quan ens fixem en les transaccions amb múltiples usuaris, la possibilitat real d'aconseguir un bon rendiment significa que cal prendre compromisos amb freqüència.

Vegem un exemple diferent de quan una operació és necessària.

Exemple de reserva d'un bitllet d'avió en línia

Suposeu que esteu tractant de reservar un bitllet d'avió en línia. Comproveu el vol que voleu i descobriu un bitllet disponible.

Encara no ho sabem, però és l'últim bitllet en aquest vol. Mentre esteu escrivint les dades de la targeta de crèdit, un altre client amb un compte especial en l'aerolínia fa la comanda per al bitllet. Nosaltres encara no hem pagat el bitllet i l'altra persona ha vist un seient lliure i l'ha reservat mentre estem escrivint les dades de la targeta de crèdit. Ara confirmem la compra del bitllet, i ja que el sistema sabia que hi havia un seient disponible quan es va iniciar la transacció, de manera incorrecta assumeix que un seient està disponible, i es fa el pagament amb la targeta. (Per descomptat, les línies aèries tenen sistemes més sofisticats d'evitar aquest tipus bàsic d'errors de reserva de bitllets, però aquest exemple serveix per il·lustrar el principi.)

El codi executat per la sol·licitud de reserva pot ser com aquest:

1. *Comprovar si hi ha seients disponibles.*
2. *Si és així, oferir seient al client.*
3. *Si el client accepta l'oferta, preguntarà pel nombre de targeta de crèdit.*
4. *Autoritzar les transaccions de targetes de crèdit amb el banc.*
5. *Dèbit a la targeta.*
6. *Assignar seients.*
7. *Reduir el nombre de places lliures disponibles segons la quantitat comprada.*

La seqüència de les dues accions concurrents la veiem a la taula 1.9.

TAULA 1.9. Seqüència de dues accions que tenen lloc en la base de dades al mateix temps

Client A	Client B	Seients disponibles
Comprova si hi ha seients disponibles		1
	Comprova si hi ha seients disponibles	1
Si és així, s'ofereix seient al client		1
	Si és així, s'ofereix seient al client	1
Si el client accepta l'oferta se li pregunta si fa servir targeta de crèdit o té un compte de la companyia		1
	Si el client accepta l'oferta se li pregunta si fa servir targeta de crèdit o té un compte de la companyia	1
Donem el codi de la targeta de crèdit	Donem el compte de client	1
Demana autorització de transacció en el banc		1
	Verifica si el compte és vàlid	1
	Actualitza el compte amb la nova transacció	1
Ho carrega al compte del banc	Assigna seient	1
Assigna seient	Actualitza el nombre de seients disponibles	0
Actualitza el nombre de seients disponibles		-1

Com podem resoldre aquest problema pel que fa a la reserva de bitllets?

Podem millorar el codi verificant si el seient estava disponible tan bon punt com ens disposem a carregar els diners, però encara que reduïm l'interval de temps el risc continua existint.

Podríem anar a l'extrem oposat per resoldre el problema, i permetre que una sola persona tingui accés al sistema de tiquets de reserva en qualsevol moment, però el rendiment seria terrible i els clients se n'anirien a un altre lloc.

Pel que fa a l'aplicació, el que tenim és una secció crítica de codi, una petita secció de codi que necessita accés exclusiu a algunes de les dades. Podríem escriure la nostra aplicació utilitzant un semàfor, o una tècnica similar, per administrar l'accés

a la secció crítica de codi. Per a això seria necessari que totes les aplicacions d'accés a la base de dades haguessin d'utilitzar el mateix semàfor. En lloc d'escriure la lògica de l'aplicació, és més fàcil emprar l'SGBD per resoldre el problema.

Pel que fa a la base de dades, el que tenim aquí és una transacció, un conjunt de manipulacions de dades de comprovació de la disponibilitat de places per mitjà de fer el dèbit del compte o targeta i l'assignació del seient, la qual cosa ha de passar com una sola unitat de treball.

Regles ACID

ACID és un acrònim d'ús freqüent per descriure les quatre propietats que ha de tenir una transacció:

- **Atòmica** *(atomic)*: una transacció, tot i que és un grup d'accions individuals sobre la base de dades, ha d'ocórrer com una sola unitat. Una transacció ha de passar exactament una vegada, sense subconjunts i sense la repetició involuntària de cap acció. En el nostre exemple la banca, el moviment de diners, ha de ser atòmic. El dèbit d'un compte i el crèdit dels altres dos han de passar com si fossin una sola acció, tot i que les sentències SQL són consecutives.

- **Consistent** *(consistent)*: al final d'una transacció, el sistema ha de ser deixat en un estat coherent. En el nostre exemple de la banca, al final d'una transacció tots els comptes han de reflectir amb precisió els crèdits i dèbits produïts.

- **Aïllada** *(isolated)*: això significa que cada transacció, sense importar quantes transaccions hi hagi en aquell moment en progrés en una base de dades, ha d'aparèixer com a independent de totes les altres transaccions. En el nostre exemple d'avió de reserva, les transaccions de processament de dos clients simultanis que es comporten com si cada un tingués l'ús exclusiu de la base de dades. En la pràctica, sabem que això no pot ser veritat si volem tenir un rendiment raonable sobre la base de dades multiusuari, i de fet resulta ser un dels punts que cal tenir en compte des d'un punt de vista pràctic en aplicacions en el món real i pot ser un obstacle molt important perquè la nostra base de dades tingui un comportament ideal.

- **Durable** *(durable)*: una vegada que una transacció s'hagi completat, ha de romandre completada. Una vegada que els diners han estat transferits amb èxit entre els comptes, han de romandre transferits, fins i tot si falla l'alimentació i la màquina que executa la base de dades té un poder sense control cap avall. En el PostgreSQL, com en la majoria de bases de dades relacionals, això s'aconsegueix utilitzant un fitxer de registre de transaccions, tal com es descriu a continuació. La durabilitat de la transacció passa sense intervenció de l'usuari.

El registre de transaccions

Els arxius de registre de transaccions s'utilitzen internament a la base de dades per assegurar-se que una transacció perdura.

La manera com treballa l'arxiu de registre de transaccions és molt senzilla. Quan s'executa una transacció, no solament s'escriuen els canvis a la base de dades, sinó també en un registre. Quan es completa una transacció, s'escriu un marcador per dir que la transacció ha acabat, i les dades del fitxer de registre es veuen obligades a emmagatzemar-se permanentment, de manera segura, encara que es bloquegi el servidor de base de dades.

Si el servidor de bases de dades per alguna raó cau, enmig d'una transacció, i a continuació, es torna a arrencar el servidor, aquest ha de ser capaç de garantir de manera automàtica que les transaccions fetes es reflecteixin correctament en la base de dades (per mitjà d'operacions a termini en el registre de transaccions, però no en la base de dades). En cap cas no hi ha hagut cap canvi en les transaccions que encara estaven en curs quan el servidor va deixar de donar servei.

El registre de transaccions que manté PostgreSQL no solament és el registre de tots els canvis que s'estan fent en la base de dades, sinó que també registra la manera de revertir. Òbviament, aquest arxiu es pot fer ràpidament molt gran. Una vegada que s'executa una sentència `COMMIT` per a una transacció, llavors el PostgreSQL sap que ja no és necessari emmagatzemar la informació sobre "com es desfà", ja que el canvi de base de dades ara és irrevocable, si més no per a la base de dades (l'aplicació podria executar codi addicional per revertir els canvis).

Podeu trobar més detalls sobre el funcionament de `WAL` en la documentació del PostgreSQL.

El PostgreSQL en realitat utilitza una tècnica en què les dades s'escriuen en el registre de transaccions abans que s'escriguin al disc per a les taules, perquè sap que una vegada que les dades s'escriuen en el fitxer de registre, es pot recuperar l'estat previst de les dades de la taula a partir del registre, encara que el sistema falli abans que els arxius de dades reals hagin estat actualitzats. Això es diu *escriptura anticipada de registre* (`WAL`).

1.3.2 Dinàmica de transaccions

Abans d'examinar els aspectes més complexos de les transaccions i com es comporten amb múltiples usuaris concurrents de la base de dades, hem de veure com es comporten amb un sol usuari. Fins i tot d'aquesta manera més aviat simplista de treball, hi ha avantatges reals per a l'ús de transaccions.

El gran avantatge de les transaccions és que permeten executar diverses instruccions SQL, i després, en una etapa posterior, permeten desfer la feina que han fet, si així ho decideixen, com es mostra a la figura 1.1, figura 1.2 i figura 1.3. D'altra banda, si un dels seus estats d'SQL falla, pot desfer la feina que han fet de nou al punt predeterminat.

FIGURA 1.1. Funcionament d'una acció "COMMIT"

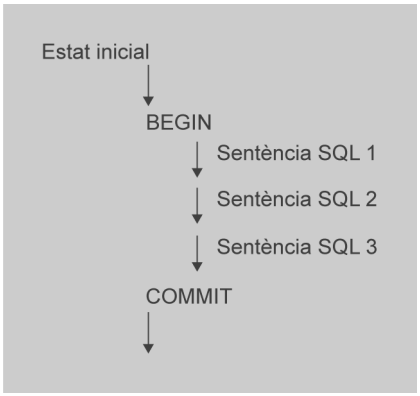


FIGURA 1.2. Funcionament d'una acció "ROLLBACK"

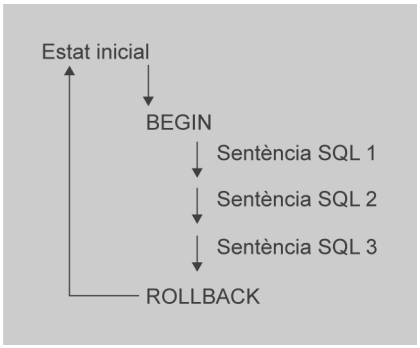
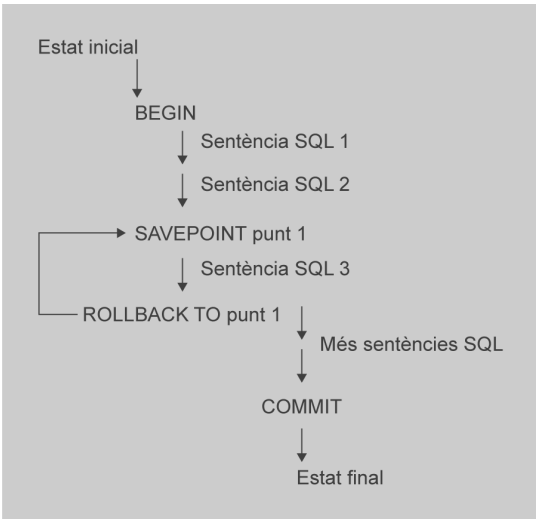


FIGURA 1.3. Funcionament d'una acció "SAVEPOINT"



Mitjançant una transacció, l'aplicació no s'ha de preocupar per si s'han fet els canvis en l'emmagatzematge a la base de dades ni de la manera de desfer. Simplement pot demanar al motor de la base de dades per desfer un lot de canvis alhora.

1.3.3 Nivells d'aïllament

El nivell d'aïllament d'una transacció determina quines dades podem veure de la transacció quan altres transaccions estan funcionant en el mateix moment.

Mentre es consulta una base de dades, cada transacció veu una imatge de les dades, és a dir, una versió de la base de dades, sense tenir en compte l'estat actual de les dades que hi ha per sota. Així s'evita que la transacció vegi dades inconsistents produïdes per l'actualització d'una altra transacció concurrent, i proporciona aïllament transaccional per a cada sessió de la base de dades.

L'estàndard SQL defineix quatre nivells d'aïllament d'una transacció tenint en compte tres fenòmens dels quals hem de ser previnguts quan es fan transaccions concurrents.

Aquests fenòmens no desitjats són:

- **Lectura bruta** *(dirty read)*: una transacció llegeix dades escrites per una transacció concurrent no confirmada.
- **Lectura no repetida** *(nonrepeatable read)*: una transacció torna a llegir dades que prèviament ha llegit i troba que les dades han estat modificades per una altra transacció (que va ser confirmada des de la lectura inicial de la primera).
- **Lectura fantasma** *(phantom read)*: una transacció torna a executar una consulta que retorna un conjunt de files que satisfan una condició de cerca i troba que el conjunt de files que satisfà la condició ha canviat pel fet que s'ha comès recentment una altra transacció.

Per assolir això es defineixen els nivells d'aïllament següents:

- `READ COMMITTED`, quan una transacció només pot veure els canvis confirmats abans que ella comenci. És el valor per defecte.
- `SERIALIZABLE`, quan totes les instruccions de la transacció en curs poden veure sols els canvis confirmats abans que la primera consulta o la primera instrucció de modificació de dades s'hagi executat en aquesta transacció.

L'SQL estàndard defineix dos nivells addicionals, `READ UNCOMMITTED` i `REPEATABLE READ`.

En el PostgreSQL, sols tenim els dos primers nivells d'aïllament, ja que `READ UNCOMMITTED` és tractat com `READ COMMITTED` i `REPEATABLE READ` és tractat com `SERIALIZABLE`.

El PostgreSQL utilitza el que s'anomena *aïllament transaccional* i *regles de resolució de conflictes* per resoldre operacions concurrents, i té dos nivells d'aïllament: serialitzable i lectura confirmada (`SERIALIZABLE` i `READ COMMITTED`).

- En el *nivell serialitzable* (`SERIALIZABLE`) es pren una instantània al començament de la transacció. Es fixa una vista de la lectura de la base de dades durant la transacció.

- En el *nivell de lectura confirmada* (`READ COMMITTED`) es pren una nova instantània al començament de cada consulta. Així, la vista de la base de dades és estable durant la iteració d'una consulta, però pot canviar durant les altres consultes que es facin dins d'una transacció.

Com veurem a la taula 1.10, l'estàndard ANSI defineix quins nivells d'aïllament diferents d'una base de dades es poden utilitzar per fer front a la possibilitat de tipus de fenòmens indesitjables, *dirty reads*, *unrepeatable reads* i *phantom reads*.

Taula 1.10. Nivells d'aïllament i possibilitat del succés d'un comportament indesitjat

Definició del nivell d'aïllament	Dirty Read	Unrepeatable Read	Phantom Read
Read Uncommitted	Possible	Possible	Possible
Read Committed	No possible	Possible	Possible
Repeatable Read	No possible	No Possible	Possible
Serializable	No possible	No possible	No possible

1.3.4 Sentències SQL implicades en la gestió de transaccions

Inici d'una transacció

Cal remarcar que en el PostgreSQL la sentència `START TRANSACTION` té la mateixa funció que `BEGIN`, i com hem vist, serveixen per iniciar una transacció.

La sintaxi és així:

```
BEGIN [ WORK | TRANSACTION ] [ mode_transacció [, ...] ]
```

O també així:

```
START TRANSACTION [ mode_transacció [, ...] ]
```

En tots dos casos el `mode_transacció` pot ser un dels següents:

```
ISOLATION LEVEL
{ SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED }
READ WRITE | READ ONLY
```

Canvi de nivell d'aïllament

Si cal es pot canviar el nivell d'aïllament utilitzant la sentència següent:

```
ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED
| READ UNCOMMITTED }
READ WRITE | READ ONLY
```

La sentència `SET TRANSACTION` afecta només la transacció actual i n'inicialitza les característiques.

Si es vol canviar el nivell d'aïllament per a la sessió es pot utilitzar la sentència `SET SESSION`:

```
SET SESSION CHARACTERISTICS AS

TRANSACTION ISOLATION LEVEL

{ SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED }

READ WRITE | READ ONLY
```

En el PostgreSQL, les transaccions per defecte funcionen amb el nivell d'aïllament `READ COMMITTED`.

El nivell d'aïllament de la transacció no es pot modificar després que la primera consulta o la primera instrucció de modificació de dades (`SELECT`, `INSERT`, `DELETE`, `UPDATE`, `FETCH` o `COPY`) d'una transacció hagi estat executada.

Finalització d'una transacció

Per donar per finalitzat una transacció s'utilitza indistintament el `COMMIT` i l'`END`.

L'ordre `END` és un sinònim de `COMMIT`. La sintaxi és la següent:

```
END [WORK | TRANSACTION]
```

La sintaxi de `COMMIT` és:

```
COMMIT [WORK | TRANSACTION]
```

Avortar una transacció

Per avortar la transacció s'utilitza `ROLLBACK`:

```
ROLLBACK [WORK | TRANSACTION]
```

En tots els casos `WORK` | `TRANSACTION` són opcionals i poden ser ignorats o utilitzar-los per fer el nostre SQL més llegible.

1.3.5 Bloquejos

La majoria de bases de dades de les aplicacions de transaccions, en particular, en aïllar les transaccions d'usuari diferents l'una de l'altra, utilitza els bloquejos per restringir l'accés a les dades d'altres usuaris.

De manera simplista, hi ha dos tipus de bloquejos:

- El **bloqueig compartit**, que permet a altres usuaris llegir, però no actualitzar les dades.

- El **bloqueig exclusiu**, que impedeix altres transaccions, fins i tot la lectura de les dades.

Per exemple, el servidor bloqueja les files que estan essent modificades per una transacció fins que es completi la transacció, i llavors els bloquejos (`LOCK`) són alliberats. Tot això es fa automàticament, en general sense que els usuaris de la base de dades siguin conscients del bloqueig que està duent a terme.

La mecànica real i les estratègies necessàries per al bloqueig són molt complexes, i s'utilitzen diferents tipus de `LOCK`, depenent de les circumstàncies.

La documentació del PostgreSQL descriu vuit tipus diferents de combinacions de bloqueig. El PostgreSQL també implementa un mecanisme inusual per a l'aïllament de les transaccions utilitzant un model multiversió, cosa que redueix els conflictes entre els bloquejos i en millora significativament el rendiment en comparació d'altres règims.

Afortunadament, els usuaris de la base de dades, en general s'han de preocupar pel que fa als bloquejos només en dues circumstàncies: evitar abraçades mortals (i la recuperació d'aquestes) i el bloqueig explícit generat per una aplicació.

Control de concurrència multiversió

El PostgreSQL manté la consistència de les dades en un model multiversió: control de la concurrència multiversió (MVCC). Aquesta és una tècnica avançada per millorar les prestacions d'una base de dades en un entorn multiusuari que implementa el PostgreSQL des de la versió 6.5 del juny de 1999.

L'MVCC és la tecnologia utilitzada per evitar bloquejos innecessaris. Si alguna vegada hem utilitzat algun SGBD amb capacitats SQL, com ara MySQL, probablement hem pogut notar que hi ha vegades en què una lectura ha d'esperar per accedir a la informació de la base de dades. L'espera està provocada per usuaris que estan escrivint en aquesta base de dades. Resumint, el lector està bloquejat pels escriptors que estan actualitzant els registres.

Mitjançant l'ús d'MVCC, el PostgreSQL evita aquest problema per complet. L'MVCC està considerat millor que el bloqueig en l'àmbit de fila perquè un lector mai no és bloquejat per un escriptor. En comptes d'això, el PostgreSQL manté un registre de totes les transaccions fetes pels usuaris de la base de dades. El PostgreSQL és capaç de manipular els registres sense necessitat que els usuaris hagin d'esperar que els registres estiguin disponibles.

Abraçades mortals

Què passa quan dues aplicacions diferents intenten canviar les mateixes dades al mateix temps?

És fàcil de veure: només cal posar en marxa dues sessions de *psql* i tractar de canviar la mateixa fila en les dues transaccions (taula 1.11).

TAULA 1.11. Exemple de canvi de dades en dues sessions diferents

Sessió 1	Sessió 2
Actualitza fila 10	
Actualitza fila 11	Actualitza fila 10
	Actualitza fila 11

En aquest punt, les dues sessions estan bloquejades, ja que cada una està esperant que l'altra faci `COMMIT`.

Aquest comportament és la clau per entendre per què el valor per defecte que assigna el PostgreSQL al mode d'aïllament d'una transació és `READ COMMITED`.

Cal un compromís *(trade-off)* entre la concurrència, el rendiment i minimitzar el nombre de bloquejos per una banda, i la consistència i el comportament ideal per l'altra. A mesura que augmenta el nivell d'aïllament, el rendiment de la base de dades multiusuari es degrada.

A mesura que intentem ajustar el comportament de la base de dades com a ideal, observem que el nombre de bloquejos necessaris augmenta, la concurrència entre els diferents usuaris disminueix, i també les caigudes de rendiment en general. Es tracta d'un lamentable però inevitable *trade-off*.

En general, si dues sessions d'usuari intenten accedir a la mateixa fila, no hi ha un impacte real sobre els usuaris, tret que el segon usuari ha d'esperar l'accés del primer usuari per completar el seu. Una situació molt més greu és quan dues sessions es fan una abraçada mortal entre si.

Bloqueig explícit

De tant en tant, és possible que el bloqueig automàtic que ofereix PostgreSQL no sigui suficient per a les nostres necessitats. En aquest cas, pot ser que hàgim de bloquejar de manera explícita algunes files o potser tota la taula.

És possible bloquejar files o només les taules dins d'una transacció.

Un cop finalitzi la transacció, ja sigui amb un `COMMIT` o `ROLLBACK`, tots els bloquejos adquirits durant l'operació s'alliberaran automàticament.

No hi ha manera d'alliberar bloquejos de manera explícita durant una transacció, per la senzilla raó que el fet d'alliberar el bloqueig en una fila que es canvia durant una operació pot permetre a una altra aplicació fer el canvi, cosa que impediria fer correctament l'acció d'una ordre `ROLLBACK` i poder desfer el canvi inicial.

Bloquejos a escala de fila

Aquest tipus de bloquejos es produeixen quan s'actualitzen camps interns d'una fila (o s'esborren o es marquen per ser actualitzats). El PostgreSQL no reté en memòria cap informació sobre les files modificades.

La necessitat més comuna és la de bloquejar un nombre de files abans de fer-hi canvis. Això pot ser útil per evitar abraçades mortals. Mitjançant el bloqueig "per endavant" podem preveure quin és el conjunt de files que hauran de canviar i assegurar-nos que no tindrem cap conflicte amb altres transaccions.

Per bloquejar un conjunt de files, simplement emetem una instrucció `SELECT` *query* `FOR UPDATE`, com en aquest exemple:

```
1 BEGIN
2 SELECT customer_id FROM customer WHERE town = 'Nicetown' FOR
3 UPDATE;
```

I ens retorna tantes files com resultin d'aquesta consulta O també:

```
BEGIN
SELECT 1 FROM customer WHERE town = 'Nicetown' FOR
UPDATE;
```

I això no ens retorna cap fila, ja que possiblement les hem previstes acuradament i de moment no ens interessa conèixer-ne el valor, amb la qual cosa minimitzem la quantitat de retorn de dades.

En aquest instant, hi podria haver dues files amb la variable *customer_id* amb valors 3 i 6. Si nosaltres les volguéssim actualitzar en una sessió *psql* concurrent:

```
sessio2=> BEGIN;
BEGIN
sessio2 => UPDATE customer SET phone = '023 3376' WHERE customer_id = 2;
UPDATE 1
sessio2 => UPDATE customer SET phone = '023 3267' WHERE customer_id = 3;
```

Ara veiem que aquesta segona sessió roman bloquejada fins que premem Ctrl+C per avortar-la o la primera sessió faci un `COMMIT` o un `ROLLBACK`.

Cal tenir en compte que `SELECT FOR UPDATE` modificarà les files seleccionades marcant-les de tal manera que no puguin ser escrites en disc per part d'altres transaccions.

Els bloquejos a escala de fila no afecten les dades consultades. Aquests s'utilitzen només per bloquejar escriptures a la mateixa fila.

Bloquejos a escala de taula

El bloqueig de taula és altament recomanable per assegurar l'aïllament en la transacció.

El PostgreSQL ofereix diferents tipus de bloqueig per controlar l'accés concurrent a les dades d'una taula.

La sintaxi del bloqueig de taules és la següent:

```
LOCK [ TABLE ] table_name
LOCK [ TABLE ] table_name IN [ ROW | ACCESS ] { SHARE | EXCLUSIVE } MODE
LOCK [ TABLE ] table_name IN SHARE ROW EXCLUSIVE MODE
```

Generalment si volem blocar una taula podem emprar la sintaxi més simple:

```
LOCK TABLE table_name
```

Que seria el mateix com fer:

```
LOCK TABLE table_name ACCESS EXCLUSIVE MODE
```

Totes les formes de bloqueig (excepte el tipus de bloqueig `ACCESS SHARE`) adquirides en una transacció es mantenen fins al final d'aquesta.

Dues transaccions no poden conservar bloquejos de tipus en conflicte sobre una mateixa taula en el mateix moment. No obstant això, una transacció no entra mai en conflicte amb ella mateixa.

1.4 Guions

De vegades pot interessar agrupar en seqüència diferents instruccions SQL que cal executar repetidament i, per a aquests casos, els SGBD acostumen a oferir la possibilitat de crear guions que agrupen les diverses sentències, de manera seqüencial.

Els guions no són altra cosa que la seqüència ordenada d'instruccions SQL, que sol proporcionar el següent:

- Establiment de variables d'entorn.
- Connexió com a superusuari.
- Eliminació de l'usuari o esquema corresponent si ja existia.
- Creació d'un usuari o esquema corresponent.
- Concessió de privilegis al nou usuari.
- Connexió com a nou usuari.
- Creació de taules i índexs amb inserció de dades.

1.4.1 Creació i execució de guions

Podem recollir un grup de sentències de *psql* (SQL i intern) en un arxiu i l'utilitzem com un simple *script*.

Us podeu descarregar un exemple de guions a:

http://goo.gl/yu352

La instrucció **\i** interna llegirà un conjunt d'ordres *psql* des d'un arxiu.

Aquesta característica és especialment útil per crear i omplir taules.

Creem un *script* amb un editor i donem l'extensió *.sql* a l'arxiu per una qüestió de convenció, i executem la instrucció interna *\i* :

```
1 sessio1=#\i sample.sql
2 CREATE TABLE
3 CREATE TABLE
4 ...
5 sessio1=#
```

A més de ser interactiu, el *psql* pot processar guions (ordres per lots emmagatzemades en un arxiu del sistema operatiu) mitjançant la sintaxi següent:

```
$ psql demo -f demo.psql
```

Encara que l'ordre següent també funciona en el mateix sentit, no és recomanable usar-lo perquè d'aquesta manera el *psql* no mostra informació de depuració important, com els nombres de línia on es localitzen els errors, si n'hi ha:

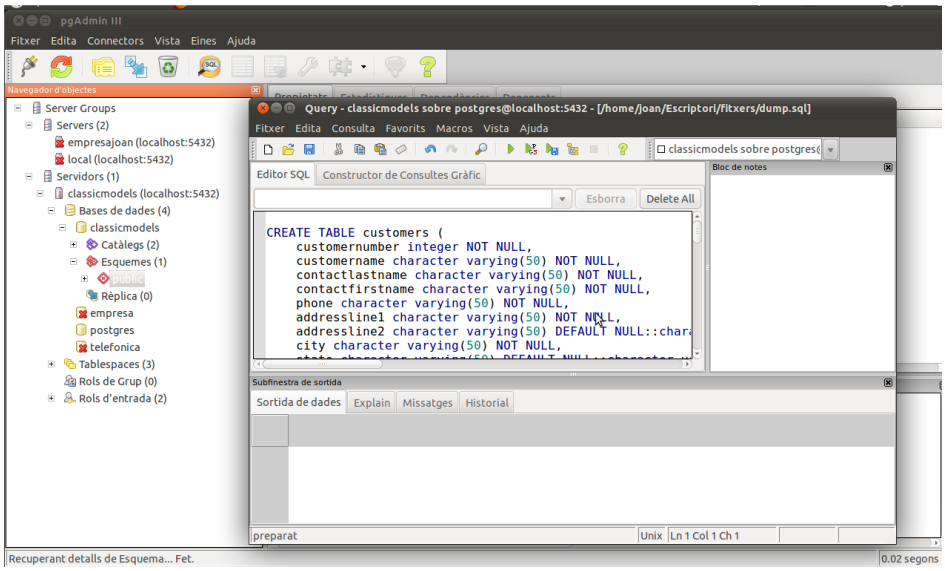
```
$ psql demo < demo.psql
```

Es pot sol·licitar l'execució d'una sola ordre i acabar immediatament mitjançant la manera següent:

```
$ psql -d demo -c "ordre sql"
```

El PgAdmin 3 ofereix una interfície gràfica per a l'edició i execució de guions.

FIGURA 1.4. Interfície gràfica per a l'edició i execució de guions



1.4.2 Formats de sortida

L'intèrpret *psql* mateix ens proporciona mecanismes per emmagatzemar en fitxer el resultat de les sentències:

- Especificant el fitxer destinació directament en finalitzar una sentència:

```
demo=# select user \g /tmp/a.txt
```

Podem veure que hem emmagatzemat el resultat en el fitxer */tmp/a.txt*.

- Mitjançant una canonada enviem la sortida a una ordre Unix:

```
demo=# select user \g | cat > /tmp/b.txt
```

- Mitjançant l'ordre \o es pot indicar on ha d'anar la sortida de les sentències SQL que s'executin d'ara endavant:

```
demo=# \o /tmp/sentències.txt
```

En l'ordre \o s'ha d'especificar un fitxer o bé una ordre que anirà rebent els resultats mitjançant una canonada.

- Quan es vulgui tornar a la sortida estàndard STDOUT, simplement es donarà l'ordre \o sense cap paràmetre.

```
demo=# select user;

demo=# select 1+1+4;
'
demo=# \o

demo=# select 1+1+4;


?column?

----------

6

(1 row)

demo=#
```

- Es pot especificar el format de sortida dels resultats d'una sentència. Per defecte, el *psql* els mostra en forma tabular mitjançant text. Per canviar-ho, s'ha de modificar el valor de la variable interna `format` mitjançant l'ordre \pset.

Vegem, en primer lloc, l'especificació del format de sortida:

```
demo=# \pset format html

Output format is html.


demo=# select user;

<table border="1">

<tr>

<th align="center">current_user</th>

</tr>

<tr valign="top">

<td align="left">postgres</td>

</tr>
'
</table>
```

```
23
24 <p>(1 row)<br />
25
26 </p>
27
28 demo=#
```

En haver especificat que es vol la sortida en HTML, la podríem redirigir a un fitxer (ja hem vist com s'ha de fer) i generar un arxiu HTML que permetés veure el resultat de la consulta mitjançant un navegador web convencional.

Hi ha altres formats de sortida, com *aligned*, *unaligned*, *html* i *latex*. Per defecte, el *psql* mostra el resultat en format *aligned*.

Tenim també multitud de variables per ajustar els separadors entre columnes, el nombre de registres per pàgina, el separador entre registres, el títol de la pàgina HTML, etc.

```
1  demo=# \pset format unaligned
2
3  Output format is unaligned.
4
5  demo=# \pset fieldsep ','
6
7  Field separator is ",".
8
9
10 demo=# select user, 1+2+3 as resultat;
11
12 current_user,resultat
13
14 postgres,6
15
16
17 (1 row)
18
19 demo=#
```

Amb aquesta configuració, i dirigint la sortida a un fitxer, generaríem un fitxer CSV preparat per ser llegit en un full de càlcul o un altre programa d'importació de dades.

2. PL/PgSQL: extensió procedimental del llenguatge SQL

Els usuaris avantatjats d'SGBD no en tenen prou amb la gestió de dades que proporciona el llenguatge SQL, ja que moltes vegades interessarà automatitzar processos repetitius o prendre decisions de gestió de dades en funció del contingut de les dades mateixes i, per aconseguir-ho, cal disposar d'una extensió procedimental al llenguatge SQL, extensió que proporciona PostgreSQL amb el llenguatge PL/PgSQL.

Com en tot llenguatge procedimental, el domini del llenguatge PL/PgSQL implica el domini del següent: estructura del programa, tipus de dades, estructures de control, interacció amb l'usuari, interacció amb el llenguatge SQL i tractament d'errors.

Els SGBD relacionals proporcionen el llenguatge SQL per executar diferents tipus de tasques en les bases de dades, i s'acostuma a distingir, dins el llenguatge SQL, quatre subconjunts segons el tipus de tasca:

- llenguatge SQL-LC per a la consulta de dades (sentència `SELECT`)
- llenguatge SQL-LMD per a la manipulació de dades (sentències `INSERT`, `UPDATE` i `DELETE`)
- llenguatge SQL-LDD per a la definició de dades (sentències `CREATE`, `ALTER` i `DROP` aplicades a taules, índexs i altres estructures de dades)
- llenguatge SQL-LCD per al control de dades (sentències `GRANT` i `REVOKE`).

Les instruccions proporcionades pel llenguatge SQL, unes més complicades que altres, es poden posar en execució en una consola de l'SGBD de manera similar a les ordres de consola per a un sistema operatiu.

Exemples d'instruccions SQL executades des d'una consola:

```
1 select emp_no, cognom, ofici from emp;
```

Però, molt sovint, ens trobarem amb la necessitat d'executar un seguit d'instruccions i poder prendre decisions en funció dels seus resultats.

Això no és factible amb la utilització de guions creats com a seqüències ordenades d'instruccions. Ens cal poder utilitzar nocions bàsiques de programació estructurada i modular (definició de variables, utilització d'estructures condicionals i iteratives, i disseny de subprogrames) i, per aquest motiu, els SGBD acostumen a proporcionar una extensió procedimental per al seu llenguatge SQL.

L'extensió procedimental d'un llenguatge SQL és un llenguatge de tercera generació que permet dissenyar programes per ser executats dins la base de dades, i que inclouen sentències SQL.

Així, cada SGBD aporta la seva extensió procedimental, que acostuma a tenir un nom. Presentem alguns llenguatges procedimentals proporcionats per diferents SGBD relacionals:

- Oracle: PL/SQL
- SQLServer: Transact-SQL
- MySQL: Incorpora l'extensió procedimental a partir de les versions 5.0 (any 2006)

PL/PgSQL és un llenguatge procedimental per a sistemes de bases de dades PostgreSQL. PL/PgSQL va ser creat amb els objectius següents:

- crear funcions i procediments disparadors,
- afegir estructures de control al llenguatge SQL,
- millorar càlculs complexos,
- permetre l'herència de tipus definits per l'usuari, funcions i operadors,
- estar emmagatzemat i executat per l'SGBD,
- ser fàcil d'emprar.

En el PostgreSQL 9.0 i els posteriors, PL/PgSQL s'instal·la per defecte. No obstant això, és un mòdul que pot ser carregat, per això per raons de seguretat els administradors el poden eliminar.

2.1 Avantatges d'emprar PL/PgSQL

SQL és el llenguatge del PostgreSQL i de moltes altres bases de dades relacionals emprat com a llenguatge de consulta. És portable i fàcil d'aprendre. Però cal tenir en compte que cada sentència SQL s'executa de manera independent en el servidor de la base de dades. Això implica que una aplicació client ha d'enviar cada consulta al servidor de la base de dades, esperar que sigui processada i rebre i processar els resultats, fer els càlculs convenients i enviar les consultes següents al servidor segons siguin els valors dels càlculs fets. Això fa que els processos de comunicació entre el client i el servidor provoquin una sobrecàrrega de la xarxa en cas que el client i el servidor estiguin en màquines diferents.

Amb PL/PgSQL es pot agrupar un bloc de càlculs i una sèrie de consultes en el servidor, i per tant té el potencial d'un llenguatge procedimental i la facilitat d'ús d'SQL, però amb un considerable estalvi de comunicació client/servidor.

Això pot resultar en un augment considerable en el rendiment en comparació d'una aplicació que no utilitza les funcions emmagatzemades.

També, amb PL/PgSQL es poden emprar tots els tipus de dades, operadors i funcions d'SQL.

Les funcions escrites en PL/PgSQL poden acceptar com a arguments qualsevol dada escalar o matriu de dades que estiguin suportats per l'SGBD i poden retornar un resultat de qualsevol d'aquests tipus. També podem acceptar o retornar un tipus de dada compost (*row type*) especificat pel nom. És també possible declarar una funció PL/PgSQL que retorni un registre, la qual cosa significa que el resultat és un tipus de fila en què les seves columnes han estat determinades per l'especificació en la crida de la consulta.

Les funcions PL/PgSQL poden ser declarades per acceptar i retornar qualsevol element polimòrfic, *anyarray*, *anynonarray* i *anyenum*.

2.2 Estructura de PL/PgSQL

PL/PgSQL és un llenguatge estructurat en blocs. El codi de la definició d'una funció es considera un bloc.

Un bloc es defineix de la manera següent:

```
[ <<etiqueta>> ]
[ DECLARE
<declaracions de constants i variables>]
BEGIN
<sentències executables>;
[EXCEPTION
<declaració d'excepcions>;]
END [ etiqueta ];
```

Fixem-nos que consta de tres zones clarament diferenciades, de les quals una és obligatòria: la zona `BEGIN ... END` (zona d'execució).

La zona `DECLARE` conté, com en la majoria de llenguatges de programació, les constants i les variables que hem d'utilitzar en el programa.

La zona `EXCEPTION` conté les instruccions per tractar els errors d'accés a la base de dades que es puguin produir en el programa.

Clàusula "BEGIN"

Una errada comuna és escriure un punt i coma després de `BEGIN`. Això és incorrecte i generarà un error de sintaxi.

Cal saber el següent:

- Tota instrucció en PL/SQL finalitza en punt i coma.
- Cada declaració i cada sentència dins d'un bloc es finalitza amb un punt i coma `;`. Un bloc que aparegui dins d'un altre bloc cal que tingui un punt i coma `;` després de posar `END`, com es mostra en l'exemple anterior, encara que aquest `END` final que finalitza el cos d'una funció no requereix un punt i coma .
- Una etiqueta és necessària solament si volem identificar el bloc per emprar-lo en una sentència `EXIT`, o per qualificar els noms de les variables decla-

rades en el bloc. Si utilitzem una etiqueta després d'un `END`, és necessari marcar l'etiqueta a l'inici del bloc.

- Totes les paraules clau són insensibles a ser escrites en majúscules o minúscules (*case-insensitive*).
- Els comentaris es codifiquen de la mateixa manera en PL/PgSQL que en SQL. Un doble guió (–) inicia un comentari que finalitza en la mateixa línia. Si volem comentar un bloc de línies aquest comença amb `/*` i finalitza amb `*/`.

2.3 Creació de funcions

`CREATE FUNCTION` està definit en SQL estàndard a partir d'SQL:1999. La sentència `CREATE FUNCTION` requereix, com a mínim:

- Un nom per a la funció.
- El nombre d'arguments (paràmetres).
- El tipus de cada argument.
- El tipus de retorn de la funció.
- L'acció (el programa com a tal).
- El llenguatge que utilitza.

Estructura d'una sentència de creació d'una funció:

```
CREATE [ OR REPLACE ] FUNCTION nom ( [ [ arg_nom ] arg_tipus [,
...] ] )

RETURNS ret_tipus
{ LANGUAGE lleng_nom
| IMMUTABLE | STABLE | VOLATILE
| CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
| [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY
DEFINER
| AS 'definició'
| AS 'obj_arxiu', 'link_simbol'
} ...
[ WITH ( atribut [, ...] ) ]
```

Descripció:

- La sentència `CREATE [ OR REPLACE ] FUNCTION` pot crear una funció nova o reemplaçar una funció ja existent. Cal tenir en compte que hi pot haver altres funcions amb el mateix nom però amb un tipus d'arguments diferents (sobrecàrrega). Per això amb `CREATE [ OR REPLACE ] FUNCTION` no es pot canviar el nom o els tipus d'argument d'una funció, ni el tipus de retorn d'una funció existent. Per fer-ho caldrà suprimir-la i tornar-la a crear.

- **arg_nom** i **arg_tipus** són el nom i el tipus d'un argument. El tipus d'una columna s'utilitza escrivint `nom_taula.nom_columna%TYPE`. `%TYPE` proporciona el tipus de dades d'una columna d'una taula. Utilitzar aquesta funcionalitat ens pot ajudar a mantenir una funció independentment de les modificacions que es facin en la definició d'una taula.
- **ret_tipus** és el tipus de retorn de la funció.
- **lleng_nom** és el nom del llenguatge en què la funció està implementada.

Si ens cal afegir un nou llenguatge a la base de dades de PostgreSQL es pot fer amb `CREATELANG`, sempre que siguin llenguatges proporcionats en la distribució de PostgreSQL:

```
CREATELANG [opcions connexió...] lleng_nom [dbnom]
CREATELANG [opcions connexió...] --llista | -l dbnom
```

També es pot utilitzar `CREATE LANGUAGE`:

`CREATE LANGUAGE` és una extensió de PostgreSQL i no hi ha una sentència en SQL estàndard.

```
CREATE [ TRUSTED ] [ PROCEDURAL ] LANGUAGE nom
HANDLER call_handler [ VALIDATOR valfunció ]
```

Encara que es recomana utilitzar `CREATELANG`, perquè fa un cert nombre de comprovacions i és molt més fàcil d'utilitzar.

Si fem una consulta dels llenguatges que vénen en la darrera versió del PostgreSQL, la 8.0, fent un `SELECT` a la taula **pg_language**:

```
SELECT * FROM pg_language;
```

Observem que en l'execució obtenim els llenguatges habilitats en el servidor PostgreSQL. Si veiem que PL/PgSQL no hi és caldrà fer:

```
CREATE LANGUAGE plpgsql;
```

Un exemple de funció que reculli alguns dels conceptes esmentats seria la següent:

```
CREATE FUNCTION unafuncio() RETURNS integer AS $$
<< altrebloc >>
DECLARE
quantitat integer := 30;
BEGIN
RAISE NOTICE 'El valor de la variable quantitat correspon a  %', quantitat;
-- mostra  30
quantitat:= 50;

/*
Això és un comentari multilínia
Creem un subbloc
*/
DECLARE
quantitat integer := 80;
BEGIN
RAISE NOTICE ' El valor de la variable quantitat correspon a %', quantitat;
-- mostra 80
RAISE NOTICE ' El valor de la quantitat anterior correspon a %', altrebloc.
    quantitat;
-- mostra 50
```

```
END;
RAISE NOTICE 'la quantitat val aquí %', quantitat;
-- mostra 50
RETURN quantitat;
END;
$$ LANGUAGE plpgsql;
```

2.4 Declaracions de variables

Les variables emprades en un bloc han de ser declarades en la secció de declaracions del bloc.

> Les úniques excepcions permeses pel que fa a l'obligatorietat de declaració de variables corresponen a les variables controladores de les voltes que es fan en un bucle. Normalment aquestes són automàticament declarades com una variable entera (*integer*) i d'una manera semblant totes aquelles que iteren sobre un cursor que es declararà automàticament com una variable de registre.

Les variables PL/PgSQL poden tenir qualsevol tipus de dades SQL, com per exemple *integer*, *varchar* i *char*.

Aquí tenim alguns exemples de declaració de variables:

```
user_id integer;
quantity numeric(5);
url varchar;
myrow tablename%ROWTYPE;
myfield tablename.columnname%TYPE;
arow RECORD;
```

La sintaxi general d'una declaració de variable és:

```
name [ CONSTANT ] type [ NOT NULL ] [ { DEFAULT | := } expression ];
```

La clàusula `DEFAULT`, si és necessària, especifica el valor inicial assignat a la variable quan s'executa el bloc. Si la clàusula `DEFAULT` no s'utilitza llavors la variable és inicialitzada amb el valor `SQL NULL`.

L'opció `CONSTANT` preveu que el valor de la variable romangui constant dins del bloc.

Si especifiquem `NOT NULL`, l'assignació d'un valor nul provoca en error en temps d'execució. Totes les variables declarades com a `NOT NULL` han de tenir especificat un valor per defecte no nul.

Valor per defecte d'una variable

El valor d'una variable per defecte s'avalua i s'assigna a la variable cada vegada que s'entra al bloc (no solament una vegada en cada crida). Així, per exemple, l'assignació de `now()` a una variable de tipus *timestamp* fa que la variable emmagatzemi el valor del temps actual

en el moment de fer la crida a la funció, no el moment en què la funció va ser compilada prèviament.

Exemples:

```
quantity integer DEFAULT 32;
url varchar := 'http://mysite.com';
user_id CONSTANT integer := 10;
```

2.4.1 ALIAS

Podem declarar un àlies per a qualsevol variable, no solament per als paràmetres. La sintaxi SQL és:

```
newname ALIAS FOR oldname;
```

El principal ús pràctic d'això és assignar un nom diferent per a les variables amb els noms predeterminats, com ara `NEW` o `OLD` dins d'un procediment lligat a un `TRIGGER`.

Exemples:

```
DECLARE
anterior ALIAS FOR old;
actualitzat ALIAS FOR new;
```

A partir del moment que s'ha creat un `ALIAS` hi ha dues maneres d'anomenar el mateix objecte, i un ús no restringit pot generar confusions. És millor emprar-lo amb el propòsit de sobreescriure noms predeterminats.

2.4.2 Declaració de variables a partir de tipus de dades preexistents

El PL/PgSQL ofereix la possibilitat de declarar variables i constants basades en tipus de dades ja existents, de diferents maneres:

- Copiant tipus preexistents
- Tipus fila (*row types*)
- Tipus de registre (*record types*)

Copiant tipus preexistents

A partir d'altres dades (variables però no constants) o columnes de taules o vistes de la base de dades.

`%TYPE` proporciona el tipus de dada d'una variable o columna d'una taula.

Per exemple, tenim una columna anomenada *user_id* a la taula *users*.

Per declarar una variable amb el mateix tipus de dada que *users.user_id* cal escriure:

```
user_id users.user_id%TYPE;
```

Aquesta possibilitat dóna una gran potència a la programació en PL/PgSQL, ja que –per a les variables destinades a contenir valors de les taules o vistes de la base de dades– modificacions posteriors en la definició de les taules o vistes poden implicar, únicament, la nova compilació dels programes PL/PgSQL afectats, sense haver-ne de canviar el codi font. Amb aquest tipus de declaració es redueix el manteniment dels programes.

La sintaxi d'aquesta declaració és la següent:

```
<nom_variable> [constant] <nom_variable_PL/PgSQL>%type [not null] [:= <expressi
    ó>];
```

o

```
<nom_variable> [constant] <esquema.taula.columna>%type [not null] [:= <expressi
    ó>];
```

Exemples de declaració de variables i constants a partir de columnes de taules:

```
declare
v_dept_no constant dept.dept_no%type;
aux_dept_no v_dept_no%type;
```

`%TYPE` és particularment valuós en funcions polimòrfiques, ja que els tipus de dades necessàries per a les variables internes poden canviar d'una crida a la següent. Es poden crear variables adequades de l'aplicació de `%TYPE` als arguments de la funció o marcadors de posició de resultat.

Tipus fila (row types)

A partir d'una fila sencera d'una taula o vista. Això és possible perquè el PL/PgSQL també ofereix la possibilitat de declarar variables tuple. En aquesta situació, els camps de la variable tenen els mateixos noms i tipus que les columnes de la taula o vista.

La sintaxi d'aquesta declaració és la següent:

```
<nom_variable> <esquema.taula>%rowtype;
```

Exemples de declaració de tuples a partir de files de taules o vistes:

```
declare
r_dept dept%rowtype;
r_emp emp%rowtype;

name table_name%ROWTYPE;
name composite_type_name;
```

Una variable d'un tipus de compost s'anomena *variable de fila* (fila o de tipus variable). Aquestes variables poden contenir una fila sencera d'un resultat de la consulta `SELECT` o `FOR`, i el conjunt de columnes que retorni la consulta cal que coincideixi amb les del tipus declarat de la variable.

Les variables de tipus fila poden contenir una fila sencera corresponent al resultat d'una consulta `SELECT`, sempre que el conjunt de columnes de la consulta coincideixi amb el tipus declarat de la variable.

Als diferents camps del valor de la fila s'accedeix utilitzant la notació amb punt, com per exemple *rowvar.field*.

Una variable de fila es pot declarar que té el mateix tipus que les files d'una taula o vista existent, mitjançant l'ús de la notació `table_name%ROWTYPE`, o pot declarar donar el nom d'un tipus de compost. Com que cada taula té un tipus compost associat del mateix nom, en realitat no importa en PostgreSQL si escriviu `%ROWTYPE` o no. Però la forma amb `%ROWTYPE` és més portable.

Els paràmetres a una funció poden ser de tipus compost (files completes de taula). En aquest cas, l'identificador corresponent `$n` serà una variable de fila i els camps s'hi poden seleccionar com, per exemple, `$1.user_id`.

Només les columnes definides per l'usuari d'una fila de la taula són accessibles en una variable de tipus fila, no l'OID o columnes d'un altre sistema.

Els camps del tipus de fila hereten la mida del camp de la taula o la precisió dels tipus de dades com `char(n)`.

En aquest exemple *taula1* i *taula2* són les taules existents que tenen almenys els camps esmentats:

```
CREATE FUNCTION concatena_camps(t_row taula1) RETURNS text AS $$
DECLARE
t2_row taula2%ROWTYPE;
BEGIN
SELECT * INTO t2_row FROM taula2 WHERE ... ;
RETURN t_row.f1 || t2_row.f3 || t_row.f5 || t2_row.f7;
END;
$$ LANGUAGE plpgsql;

SELECT concatena_camps(t.*) FROM taula1 t WHERE ... ;
```

Tipus de registre (record types)

Les variables de registre són similars a les variables de tipus de fila, però no tenen una estructura predefinida.

```
name RECORD;
```

Es prenen en l'estructura de la fila actual i s'assignen en un `SELECT` o en un `FOR`. La subestructura d'una variable de registre es pot canviar cada vegada que s'ha assignat.

Una conseqüència d'això és que fins que no s'assigni valor a una variable de registre, no té infraestructura, i qualsevol intent d'accedir a un camp genera un error en temps d'execució.

El registre no és un tipus de dades real, només un marcador de posició. També cal adonar-se que quan una funció PL/PgSQL es declara per retornar un tipus de registre, aquest no és el mateix concepte que una variable de registre, tot i que aquesta funció podria utilitzar una variable de registre per mantenir el seu resultat. En tots dos casos l'estructura de la fila actual es desconeix quan la funció està escrita, però per a un registre de retorn d'una funció l'estructura real es determina quan la consulta s'ha analitzat, mentre que una variable de registre pot canviar la seva estructura de files en temps d'execució.

2.5 Paràmetres d'una funció

Els paràmetres que passem a les funcions són anomenats amb els identificadors `$1`, `$2`, etc. Opcionalment podem declarar àlies dels `$n` paràmetres per facilitar-ne la llegibilitat. Es pot emprar tant l'àlies com l'identificador numèric per fer referència al valor del paràmetre.

Hi ha dues maneres de crear un àlies. La millor manera és donar un nom al paràmetre en la sentència `CREATE FUNCTION`; per exemple:

```
1 CREATE FUNCTION sales_tax(subtotal real) RETURNS real AS $$
2 BEGIN
3 RETURN subtotal * 0.06;
4 END;
5 $$ LANGUAGE plpgsql;
```

L'altra forma, la qual és l'única permesa en versions anteriors al PostgreSQL 8.0, és declarar un àlies explícitament en l'apartat de declaracions:

```
1 name ALIAS FOR $n;
```

Variable subtotal

Aquests dos exemples no són del tot equivalents. En el primer cas, `subtotal` pot ser referenciat com `sales_tax.subtotal`, però en el segon cas no es podia. Si haguéssim unit una etiqueta per al bloc intern, `subtotal` es podria, d'altra manera, qualificar amb aquesta etiqueta.

L'exemple anterior amb aquest estil seria com això que ve a continuació:

```
1 CREATE FUNCTION sales_tax(real) RETURNS real AS $$
2 DECLARE
3 subtotal ALIAS FOR $1;
4 BEGIN
5 RETURN subtotal * 0.06;
6 END;
7 $$ LANGUAGE plpgsql;
```

Alguns exemples més:

```
1 CREATE FUNCTION instr(varchar, integer)
2 RETURNS integer AS $$
3 DECLARE
```

```
v_string ALIAS FOR $1;
index ALIAS FOR $2;
BEGIN
-- fem els càlculs corresponents emprant  v_string i index aquí '
END;
$$ LANGUAGE plpgsql;


CREATE FUNCTION concat_selected_fields(in_t sometablename)
RETURNS text AS $$
BEGIN
RETURN in_t.f1 || in_t.f3 || in_t.f5 || in_t.f7;
END;
$$ LANGUAGE plpgsql;
```

Quan una funció PL/PgSQL es declara amb paràmetres de sortida, als paràmetres de sortida donem `$n` noms i àlies opcionals de la mateixa manera que als paràmetres d'entrada normal. Un paràmetre de sortida és en realitat una variable que s'inicialitza, ha de ser assignat durant l'execució de la funció. El valor final del paràmetre és el que es retorna.

L'exemple anterior, d'impostos sobre vendes, també es podria fer d'aquesta manera:

```
CREATE FUNCTION sales_tax(subtotal real, OUT tax real) AS $$
BEGIN
tax := subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;
```

Tingueu en compte s'ha omès `RETURNS real` –que podríem haver inclòs, però seria redundant.

Els paràmetres de sortida són més útils, ja que es fa retorn de valors múltiples. Un exemple trivial és el següent:

```
CREATE FUNCTION sum_n_product(x int, y int, OUT sum int, OUT prod int) AS $$
BEGIN
sum := x + y;
prod := x * y;
END;
$$ LANGUAGE plpgsql;
```

Això crea efectivament un tipus de registre anònim dels resultats de la funció. Si s'utilitzés la clàusula `RETURN`, caldria escriure `RETURN RECORD`.

Una altra manera de declarar una funció PL/PgSQL és amb `RETURNS TABLE`; per exemple:

```
CREATE FUNCTION extended_sales(p_itemno int)
RETURNS TABLE(quantity int, total numeric) AS $$
BEGIN
RETURN QUERY SELECT quantity, quantity * price FROM sales
WHERE itemno = p_itemno;
END;
$$ LANGUAGE plpgsql;
```

Això és exactament equivalent a declarar un o més paràmetres de sortida (`OUT` *parameters*) i especificar `RETURNS SETOF sometype`.

2.6 Sobrecàrrega de funcions i funcions polimòrfiques

Cal considerar dos aspectes quant a la implementació de funcions:

- La sobrecàrrega de funcions
- Les funcions polimòrfiques

2.6.1 Sobrecàrrega de funcions

El PostgreSQL considera funcions diferents si tenen noms diferents, si tenen un nombre diferent de paràmetres, o si els seus paràmetres són de tipus diferent. Considerem la funció següent:

```
1 CREATE FUNCTION suma_un (int4) RETURNS int4 as $$
2 BEGIN
3 return $1 + 1;
4 end;
5 $$ language plpgsql;
```

Aquesta funció s'executarà adequadament quan passem un paràmetre de tipus enter, però en canvi donarà un error quan el paràmetre sigui de coma flotant.

Podem crear altres funcions *suma_un* que puguin tractar amb diferents tipus. Per exemple:

```
1 CREATE FUNCTION suma_un (float8) RETURNS float8 as $$
2 BEGIN
3 return $1 + 1;
4 end;
5 $$ language plpgsql;
```

I veurem que ara accepta qualsevol de les execucions següents:

```
1 SELECT suma_un(3);
2 SELECT suma_un(3.1);
```

Per veure el codi d'aquestes funcions podem executar:

```
1 SELECT prosrc FROM pg_proc WHERE proname = 'suma_un';
```

2.6.2 Funcions polimòrfiques

Quan declarem el tipus de retorn d'una funció PL/PgSQL de tipus polimòrfic (*anyelement*, *anyarray*, *anynonarray*, o *anyenum*), es crea un paràmetre especial `$0`. Aquest tipus de dada correspon al tipus actual de retorn de la funció. `$0`

s'inicialitza com a *null* i pot ser modificat per la funció, i per això poden ser usats per sostenir el valor de retorn, si volem, però no és necessari. `$0` també pot tenir un àlies.

Per exemple, aquesta funció es pot utilitzar en qualsevol tipus de dades que tingui un operador:

```
CREATE FUNCTION add_three_values(v1 anyelement, v2 anyelement, v3 anyelement)
RETURNS anyelement AS $$
DECLARE
result ALIAS FOR $0;
BEGIN
result := v1 + v2 + v3;
RETURN result;
END;
$$ LANGUAGE plpgsql;
```

Tindrem el mateix efecte mitjançant la declaració d'un o més paràmetres de sortida com a tipus polimòrfic. En aquest cas el paràmetre especial `$0` no s'utilitza; els paràmetres de sortida s'utilitzen per al mateix propòsit.

Per exemple:

```
CREATE FUNCTION add_three_values(v1 anyelement, v2 anyelement, v3 anyelement,
OUT sum anyelement)
AS $$
BEGIN
sum := v1 + v2 + v3;
END;
$$ LANGUAGE plpgsql;
```

2.7 Assignacions

L'assignació d'un valor a una variable PL/PgSQL s'escriu de la manera següent :

```
nom_variable := expressio;
```

Com es va explicar anteriorment, l'expressió d'aquesta declaració s'avalua per mitjà d'una ordre SQL `SELECT` enviada al motor de base de dades principal. L'expressió ha de cedir un sol valor (possiblement un valor de la fila, si la variable és una fila o una variable de registre). La variable de destinació pot ser una variable simple (opcionalment qualificada amb el nom del bloc), un camp d'una fila o una variable de registre, o un element d'una matriu que és una variable simple o un camp.

Exemples:

```
tax := subtotal * 0.06;
my_record.user_id := 20;
```

2.8 Estructures condicionals

Mitjançant les sentències `IF` i `CASE` podem executar sentències alternatives basades en certes condicions.

PL/PgSQL te tres formes de `IF`:

- `IF ... THEN`
- `IF ... THEN ... ELSE`
- `IF ... THEN ... ELSIF ... THEN ... ELSE`

I dues formes de `CASE`:

- `CASE ... WHEN ... THEN ... ELSE ... END CASE`
- `CASE WHEN ... THEN ... ELSE ... END CASE`

2.8.1 Alternativa simple (IF-THEN)

Si la condició es compleix s'executen les instruccions que segueixen la clàusula `THEN`.

```
1 if <condició> then
2 <conjunt_de_sentències_si_la_condició_s'avalua_com_a_certa>;
3 end if;
```

2.8.2 Alternativa doble (IF ... THEN ... ELSE)

Si la condició es compleix s'executen les instruccions que segueixen la clàusula `THEN`. En cas contrari, s'executaran les instruccions que segueixen la clàusula `ELSE`.

```
1 if <condició> then
2 <conjunt_de_sentències_si_la_condició_s'avalua_com_a_certa>;
3 else
4 <conjunt_de_sentències_si_la_condició_s'avalua_com_a_falsa>;
5 end if;
```

2.8.3 Alternativa múltiple (IF ... THEN ... ELSIF ... THEN ... ELSE)

Avalua, començant des de l'inici, cada condició, fins que en troba alguna que es compleixi; en aquest cas executarà les instruccions que segueixin la clàusula `THEN` corresponent.

```
if <condició1> then
<conjunt_de_sentències_si_condició1_s'avalua_com_a_certa>;
elsif <condició2> then
<conjunt_de_sentències_si_condició2_s'avalua_com_a_certa>;
elsif
...
else
<conjunt_de_sentències_si_última_cond_s'avalua_com_a_falsa>;
end if;
```

La clàusula `ELSE` és opcional, per al cas que no s'hagi complert cap de les condicions anteriors.

Aquí tenim un exemple:

```
IF number = 0 THEN
result := 'zero';
ELSIF number > 0 THEN
result := 'positiu';
ELSIF number < 0 THEN
result := 'negatiu';
ELSE
-- umm, ara l'única possibilitat seria que el nombre fos null
result := 'NULL';
END IF;
```

La paraula clau `ELSIF` també es pot escriure com `ELSE IF`.

Un exemple alternatiu d'ús seria l'exemple següent:

```
IF demo_row.sex = 'm' THEN
pretty_sex := 'home';
ELSE
IF demo_row.sex = 'f' THEN
pretty_sex := 'dona';
END IF;
END IF;
```

2.8.4 CASE simple

El tipus més simple de `CASE` proporciona una execució condicional basada en la igualtat dels operands. L'expressió de cerca és avaluada un cop i comparada successivament en cada expressió dins de les clàusules `WHEN`.

```
CASE expressio_de_cerca
WHEN expressio [, expressio [ ... ]] THEN
sentències
[ WHEN expressio [, expressio [ ... ]] THEN
sentències

... ]
```

```
[ ELSE
sentències  ]
END CASE;
```

Si s'ha trobat una coincidència llavors s'executen les sentències corresponents i llavors el control d'execució passa a la sentència següent posterior a `END CASE` (les expressions `WHEN` subsegüents no són avaluades). Si no hi ha coincidències, les declaracions que hi ha dins `ELSE` s'executen, però si `ELSE` no hi és, llavors es llença una excepció `CASE_NOT_FOUND`.

Aquí en tenim un exemple:

```
CASE x
WHEN 1, 2 THEN
msg := 'un o dos';
ELSE
msg := 'un valor diferent d'un o dos';
END CASE;
```

2.8.5 CASE examinat

La forma de recerca de `CASE` proporciona l'execució condicional basada en la veritat de les expressions booleanes. Cada clàusula `WHEN` inclou una expressió booleana que s'avalua al seu torn, fins que se'n troba una en què el resultat és veritat. A continuació, s'executen les sentències corresponents, i després el control passa a la instrucció posterior següent `END CASE`.

```
CASE
WHEN expressió_booleana THEN
sentències
[ WHEN expressió_booleana THEN
sentències
... ]
[ ELSE
sentències  ]
END CASE;
```

Si no hi ha cap resultat en què la condició sigui veritat, s'executen les declaracions que hi ha dins `ELSE`, però si `ELSE` no hi és, llavors es llença una excepció `CASE_NOT_FOUND`.

Aquí tenim un exemple:

```
CASE
WHEN x BETWEEN 0 AND 10 THEN
msg := 'el valor és entre zero i deu';
WHEN x BETWEEN 11 AND 20 THEN
msg := 'value is between eleven and twenty';
END CASE;
```

2.9 Estructures iteratives

El PL/PgSQL proporciona tres modalitats de bucles:

- `LOOP`
- `WHILE...LOOP`
- `FOR`

2.9.1 LOOP

L'estructura `LOOP`, amb la sintaxi següent:

```
loop
<conjunt_de_sentències>
exit when <condició_sortida>
<conjunt_de_sentències>
exit when <condició_sortida>
<conjunt_de_sentències>
...
end loop;
```

L'estructura `LOOP` permet diferents condicions de sortida (`EXIT WHEN`). En arribar a la instrucció `END LOOP`, el programa repeteix el bucle.

2.9.2 WHILE...LOOP

L'estructura `WHILE ... LOOP`, amb la sintaxi següent:

```
while <condició> loop
<conjunt_de_sentències>
end loop;
```

L'estructura `WHILE ... LOOP` és idèntica a la de la majoria de llenguatges.

2.9.3 FOR

La sentència `FOR`, amb la sintaxi següent:

```
for <variable> in [reverse] <rang_mínim>..<rang_màxim> loop
<conjunt_de_sentències>
end loop;
```

L'estructura `FOR` no exigeix que `variable` estigui definida. Si no ho està, la defineix el bucle i desapareix en finalitzar l'execució del bucle. Els valors `rang_mínim` i `rang_màxim` han de ser expressions de resultat enter. El conjunt d'instruccions d'un bucle `FOR` no pot modificar el contingut de la variable. El bucle `FOR ... IN` inicialitza el valor de `variable` amb `rang_mínim` i, a cada repetició del bucle, incrementa el valor una unitat fins a sobrepassar el `rang_màxim`.

El bucle `FOR ... IN REVERSE` inicialitza el valor de `variable` amb `rang_màxim` i, a cada repetició del bucle, redueix el valor una unitat fins a ultrapassar el `rang_mínim`.

3. Cursors i control d'errors

Un cop vist quina és la estructura d'una funció i quin tipus de sentències es poden emprar, veurem dos aspectes importants quant a una bona implementació de funcions en PL/pgSQL:

1. El control d'errors.
2. La utilització de cursors.

3.1 Control d'errors

En aquest apartat estudiarem com fem el control dels errors produïts en temps d'execució dins d'un bloc PL/PgSQL.

3.1.1 Captura d'errors

Per defecte, qualsevol error que es produeix en una funció PL/PgSQL avorta l'execució de la funció i, de fet, de les transaccions següents. Si volem capturar els errors i recuperar-nos-en caldrà fer-ho mitjançant l'ús d'un bloc `BEGIN` amb una clàusula `EXCEPTION`.

La sintaxi és una extensió de la sintaxi normal d'un bloc `BEGIN`:

```
[ <<etiqueta>> ]
[ DECLARE
declaracions ]
BEGIN
sentències
EXCEPTION
WHEN condició [ OR condició ... ] THEN
sentència_gestora
[ WHEN condició [ OR condició ... ] THEN
sentència_gestora
... ]
END;
```

Si no hi ha error, aquesta forma de bloc simplement executa totes les sentències, i després el control passa a la instrucció següent després d'aquest `END`. Però si es produeix un error dins d'una sentència, el tractament de les sentències posteriors s'abandona, i el control passa a la llista d'excepcions. Dins la llista es busca la condició que coincideixi l'error que s'ha produït. Si es troba una coincidència, la sentència gestora corresponent (`handler_statement`) s'executa, i després el

control passa a la instrucció següent després d'aquest `END`. Si no hi ha coincidència, l'error es propaga com si la clàusula d'excepció no hi fos en absolut: l'error pot ser atrapat per un bloc que el conté amb l'excepció, o si no n'hi ha cap que anul·li el processament de la funció.

Els noms de condició poden ser qualssevol dels que es mostren a les taules següents.

El nom d'una categoria, o classe, coincideix amb qualsevol error que pertanyi a aquesta categoria.

La condició especial `OTHERS` coincideix amb cada tipus d'error excepte amb `QUERY_CANCELED`.

Els noms de condició es poden donar en minúscules o majúscules. També una condició d'error pot ser especificada per `SQLSTATE`.

```
1 WHEN division_by_zero THEN ...
2 WHEN SQLSTATE '22012' THEN ...
```

Quan un error és capturat per una clàusula `EXCEPTION`, les variables locals de la funció PL/PgSQL romanen com estaven quan va ocórrer l'error, però tots els canvis d'estat de la base de dades persistents dins del bloc es desfan. Com a exemple, aquest fragment:

```
1  INSERT INTO mytab(firstname, lastname) VALUES('Pere', 'Martí');
2  BEGIN
3  UPDATE mytab SET firstname = 'Josep' WHERE lastname = 'Martí';
4  x := x + 1;
5  y := x / 0;
6  EXCEPTION
7  WHEN division_by_zero THEN
8  RAISE NOTICE 'capturat error division_by_zero';
9  RETURN x;
10 END;
```

Quan el control arriba a l'assignació de **y**, fallarà amb un error de **division_by_zero**. Aquest serà capturat per la clàusula `EXCEPTION`.

El valor retornat en el compte de retorn és el valor incrementat de `x`, però els efectes de l'ordre `UPDATE` s'han revertit.

La instrucció `INSERT` anterior al bloc no es reverteix, però, de manera que el resultat final és que la base de dades conté "Pere Martí" i no "Josep Martí".

Clàusula d'excepció

Un bloc que conté una clàusula d'excepció és molt més car per entrar-hi i sortir-ne que un bloc sense una. Per tant, no utilitzeu una excepció sense necessitat.

Dins d'un manejador d'excepcions, la variable `SQLSTATE` conté el codi d'error que correspon a l'excepció que s'ha plantejat (vegeu les taules següents per a una llista de possibles codis d'error). La variable `SQLERRM` conté el missatge d'error associat a l'excepció. Aquestes variables no estan definides fora de controladors d'excepció.

Aquest exemple utilitza un manegador d'excepcions per fer cada `UPDATE` o `INSERT`, de manera apropiada:

```
CREATE TABLE db (a INT PRIMARY KEY, b TEXT);

CREATE FUNCTION merge_db(key INT, data TEXT) RETURNS VOID AS
$$
BEGIN
LOOP
-- Primerament intentem actualitzar amb el valor de key
UPDATE db SET b = data WHERE a = key;
IF found THEN
RETURN;
END IF;
-- Aqui haurem finalitzat, però si algú intenta inserir la clau
-- de manera concurrent,
-- podem tenir una errada de clau no única
BEGIN
INSERT INTO db(a,b) VALUES (key, data);
RETURN;
EXCEPTION WHEN unique_violation THEN
-- no fer res i tornem a intentar actualitzar un altre cop
END;
END LOOP;
END;
$$
LANGUAGE plpgsql;



SELECT merge_db(1, 'David');
SELECT merge_db(1, 'Marcel');
```

3.1.2 Errors i missatges

El PostgreSQL no disposa d'un model gaire acurat de tractament d'excepcions. Quan l'analitzador, l'optimitzador o l'executor decideixen que una sentència no es pot processar, la transacció completa s'avorta i el sistema torna per processar la consulta següent de l'aplicació.

L'única cosa que fa PL/PgSQL quan es produeix un avortament d'execució durant l'execució d'una funció o disparador és enviar missatges de depuració al nivell `DEBUG`, indicant en quina funció i on (número de línia i tipus de sentència) ha succeït l'error.

Es pot utilitzar la sentència `RAISE` per enviar missatges:

```
RAISE level 'format' [,identificador[...]];
```

en què `level` pot ser, per exemple, `NOTICE` o `EXCEPTION`.

- `NOTICE` escriu en la bitàcola de la base de dades i ho envia a l'aplicació del client.
- `EXCEPTION` escriu en la bitàcola de la bases de dades i avorta la transacció.

```
RAISE [ level ] 'format' [, expression [, ... ]] [ USING option = expression [, ... ] ];
RAISE [ level ] condition_name [ USING option = expression [, ... ] ];
RAISE [ level ] SQLSTATE 'sqlstate' [ USING option = expression [, ... ] ];
RAISE [ level ] USING option = expression [, ... ];
RAISE ;
```

L'opció de nivell especifica la gravetat de l'error.

Els nivells permesos són `DEBUG`, `LOG`, `INFO`, `NOTICE`, `WARNING` i `EXCEPTION`, i `EXCEPTION` és el nivell per defecte.

`EXCEPTION` genera un error (que normalment avorta la transacció actual) i els altres nivells només generen missatges de diferents nivells de prioritat.

Si els missatges són d'una prioritat especial, s'informa el client, o per escrit en el registre del servidor, o tots dos, i es controlen amb les variables de configuració `log_min_messages` i `client_min_messages`.

Després del nivell, si s'escau, es pot escriure un format (que ha de ser una simple cadena literal, no una expressió).

La cadena de format especifica el text del missatge d'error de què s'informa. La cadena de format pot ser seguida per les expressions amb argument opcional que s'insereixen en el missatge. Dins del format de cadena `%` se substitueix pel valor de l'argument següent.

Cal escriure `% %` per emetre un literal `%`.

En aquest exemple, el valor de `v_job_id` substituirà `%` dins la cadena:

```
RAISE NOTICE 'Cridant cs_create_job(%)', v_job_id;
```

Es pot adjuntar informació addicional a l'informe d'error emprant `USING` seguit per `opció_clau = expressió`. Les `opcions_clau` permeses són `MESSAGE`, `DETAIL`, `HINT` i `ERRCODE`, en què cada expressió pot ser una cadena.

- `MESSAGE` estableix el text del missatge d'error (aquesta opció no es pot utilitzar en la forma de `RAISE` que inclou una cadena de format abans de `USING`).
- `DETAIL` proporciona un missatge de detall d'error, mentre que `HINT` proporciona un missatge de suggeriment. `ERRCODE` especifica el codi d'error (`SQLSTATE`) de què s'informa, ja sigui pel nom de la condició, com es mostra en la taula, o directament amb un codi de 5 caràcters `SQLSTATE`.

Aquest exemple avortaria la transacció i donaria el missatge d'error i suggeriment següents:

```
RAISE EXCEPTION 'No existeix ID --> %', user_id
USING HINT = 'confirma identificador de l\'usuari';
```

Aquest dos exemples mostren d'informar mitjançant `SQLSTATE`:

```
RAISE 'Identificador de l\'usuari duplicat: %', user_id USING ERRCODE = '
    unique_violation';
RAISE 'Identificador de l\'usuari duplicat: %', user_id USING ERRCODE = '23505'
    ;
```

Aquí tenim una segona sintaxi de `RAISE` en la qual el principal argument és el nom de la condició o el codi `SQLSTATE`, com per exemple:

```
RAISE division_by_zero;
RAISE SQLSTATE '22012';
```

En aquesta sintaxi, `USING` es pot emprar per proporcionar un missatge d'error personalitzat, detall o pista. Una altra manera d'escriure l'exemple anterior seria:

```
RAISE unique_violation USING MESSAGE = 'Identificador de l\'usuari duplicat: '
    || user_id;
```

Encara tenim una altra variant: escriure `RAISE USING` o `RAISE level USING` i posar qualsevol cosa dins de la llista `USING`.

L'última variant de `RAISE` no té cap paràmetre. Aquesta forma només es pot utilitzar dins d'una clàusula `EXCEPTION` dins del bloc `BEGIN`, que provoca que l'error actual es gestioni per ser rellançat en el bloc inclòs següent.

Si en `RAISE EXCEPTION` no s'especifica cap `SQLSTATE` o nom de condició el valor escollit serà el genèric `RAISE_EXCEPTION (P0001)`.

Si no s'especifica cap missatge de text s'utilitza per defecte el nom de la condició o `SQLSTATE` com a text del missatge.

Es recomana evitar llençar els codis d'error que acaben en tres zeros, perquè es tracta de codis de categoria i només pot ser atrapada per la captura tota la categoria.

Codis d'error

Quan s'especifica un codi d'error `SQLSTATE`, els possibles codis que es poden emprar no es limiten als codis d'error predefinits. Això vol dir que podem seleccionar qualsevol codi d'error que consti de cinc dígits o lletres majúscules ASCII i que no sigui `00000`.

3.1.3 Codis d'error

A tots els missatges emesos pel servidor PostgreSQL s'assignen cinc caràcters de codi d'error que segueixen les convencions estàndard SQL per a codis `SQLSTATE`. Les aplicacions que necessiten saber quina condició d'error s'ha produït en general treballen amb el codi d'error en comptes de buscar el missatge d'error textual.

És menys probable que canviïn els codis d'error per mitjà de comunicats de PostgreSQL, i no estan subjectes a canvis a causa de la localització de missatges d'error. Tingueu en compte que alguns, però no tots, els codis d'error produïts pel PostgreSQL, estan definits per l'estàndard SQL, i alguns altres codis d'error addicionals, a causa de condicions no definides per la norma, han estat inventats o presos de sistemes gestors de bases de dades.

Podeu consultar les taules corresponents als codis d'error en la secció "Annexos" del web del mòdul.

Segons la norma, els dos primers caràcters d'un codi d'error denoten una classe d'error, mentre que els tres últims caràcters indiquen una condició específica dins

d'aquesta classe. Per tant, una aplicació que no reconeix el codi d'error específic encara pot ser capaç d'inferir a quina classe d'error pertany .

3.2 Cursors

Fins ara hem estat emprant cursors implícits. Aquest tipus de cursors presenten diversos problemes. El més important és que la subconsulta cal que retorni un sol tuple, ja que en cas contrari es produiria un error.

En lloc d'executar una consulta completa d'una vegada, és possible establir un cursor que encapsuli la consulta, i després llegeixi el resultat de la consulta una o unes quantes files a la vegada. Una de les raons per fer això és per evitar desbordament de memòria quan el resultat conté un gran nombre de files. (No obstant això, normalment els usuaris de PL/PgSQL no s'han de preocupar per això, ja que per als bucles `FOR` de manera automàtica es fa ús d'un cursor internament per evitar problemes de memòria.)

Un ús més interessant és el de retornar una referència a un cursor en què s'ha creat una funció, i permetre que el que fa la crida pugui llegir les files. Això proporciona una manera eficaç de retornar grans conjunts de files des de les funcions.

Hi ha quatre operacions bàsiques per treballar amb un cursor explícit:

- Declaració del cursor: `CURSOR`. Aquest tipus de variable es declara a la zona `DECLARE`.
- Obertura del cursor: `OPEN`. S'obre aquest cursor a la zona d'instruccions. Això comporta que s'executi automàticament la sentència `SELECT` associada i el seu resultat s'emmagatzema en estructures internes de memòria gestionades pel cursor.
- Recollida d'informació `FETCH`. En aquest instant es recull la informació d'una fila i s'emmagatzema en les variables corresponents.
- Tancament del cursor: `CLOSE`. Quan el cursor no torna a ser emprat cal tancar-lo.

3.2.1 Declaració de variables de cursor

Tots els accessos als cursors en PL/PgSQL passen a través de variables de cursor, que són sempre del tipus de dades especial *refcursor*. Una manera de crear una variable de cursor és declarar una variable com del tipus *refcursor*. Una altra manera és utilitzar la sintaxi de la declaració del cursor, que en general és:

```
1 name [ [ NO ] SCROLL ] CURSOR [ ( arguments ) ] FOR query ;
```

- `FOR` pot ser substituït per IS per assolir compatibilitat amb Oracle.
- `SCROLL`; si s'especifica, el cursor és capaç de desplaçar-se cap enrere.
- `NO SCROLL`; si s'especifica un desplaçament cap enrere serà rebutjat.
- la llista d'arguments, si s'especifica, és una llista separada per comes de variables de diferents tipus de dades que defineixen els noms per ser reemplaçats per valors dels paràmetres de la consulta donada. Els valors actuals per substituir aquests noms s'especificaran més endavant, quan s'obri el cursor.

Alguns exemples:

```
DECLARE
curs1 refcursor;
curs2 CURSOR FOR SELECT * FROM tenk1;
curs3 CURSOR (key integer) IS SELECT * FROM tenk1 WHERE unique1 = key;
```

Totes tres variables tenen tipus de dades *refcursor*, però la primera es pot utilitzar amb qualsevol consulta, mentre que la segona té una consulta totalment especificada ja fixada a la variable `curs2`, i l'última té una consulta amb paràmetres vinculats a aquesta. (`key` serà reemplaçat per un valor de paràmetre de nombre enter, quan s'obre el cursor.)

La variable `curs1` es diu que és independent, ja que el cursor no està vinculat a cap consulta en particular. Les variables `curs2` i `curs3` estan vinculades.

3.2.2 Obertura de cursors

Abans d'emprar un cursor per recuperar les files, cal que aquest s'obri. (Aquesta és l'acció equivalent a la instrucció SQL `DECLARE CURSOR`.) PL/PgSQL té tres formes de la sentència `OPEN`, dues de les quals utilitzen variables de cursor no vinculades, mentre que el tercer utilitza una variable de cursor vinculada.

Les variables de cursor vinculades...

...també es poden utilitzar sense obrir explícitament el cursor, per mitjà de la instrucció `FOR`, que es descriu posteriorment.

OPEN FOR query

La sintaxi de la declaració OPEN FOR query és:

```
OPEN unbound_cursorvar [ [ NO ] SCROLL ] FOR query ;
```

La variable de cursor s'obre i utilitza la consulta especificada per a l'execució. El cursor ja no es pot obrir, ja que ha estat declarada com una variable cursor no vinculada (és a dir, com una variable *refcursor* simple). La consulta ha de ser un `SELECT`, o alguna altra cosa que retorna files (com `EXPLAIN`). La consulta es farà de la mateixa manera que altres ordres SQL en PL/PgSQL: els noms de les variables PL/PgSQL són substituïts, i el pla de consulta s'emmagatzema per a la reutilització possible. Quan una variable PL/PgSQL se substitueix en la consulta

de cursor, el valor que se substitueix és el que té en el moment de l'`OPEN`, els canvis posteriors a la variable no afectaran el comportament del cursor. Les opcions de desplaçament `SCROLL` i `NO SCROLL` tenen el mateix significat que per a un cursor vinculat.

Un exemple:

```
OPEN curs1 FOR SELECT * FROM foo WHERE key = mykey;
```

OPEN FOR EXECUTE

La sintaxi de la declaració OPEN FOR EXECUTE és:

```
OPEN unbound_cursorvar [ [ NO ] SCROLL ] FOR EXECUTE query_string
[ USING expression [, ... ] ];
```

La variable de cursor s'obre i genera l'execució de la consulta especificada. El cursor no es pot tornar a obrir, i aquest ha d'haver estat declarat com una variable del cursor no vinculada (és a dir, com una variable **refcursor** simple). La consulta s'especifica com una expressió de cadena, de la mateixa manera que en la instrucció `EXECUTE`.

Com de costum, això li dóna flexibilitat perquè el pla de consulta pot variar d'una execució a una altra, i això també significa que la substitució de variables no es faci. Igual que amb `EXECUTE`, els valors dels paràmetres es poden inserir de manera dinàmica emprant `USING`. Les opcions de desplaçament `SCROLL` i `NO SCROLL` tenen el mateix significat que per a un cursor vinculat.

Un exemple:

```
OPEN curs1 FOR EXECUTE 'SELECT * FROM ' || quote_ident(tabname)
|| ' WHERE col1 = $1' USING keyvalue;
```

En aquest exemple, el nom de la taula s'insereix en la consulta de manera textual, emprant la funció `quote_ident()`. Es recomana evitar la injecció SQL. El valor de comparació per a `COL1` és inserit mitjançant `USING`, i per això no necessiten cometes.

Obertura d'un cursor vinculat

Aquesta forma d'obertura s'utilitza per obrir una variable de cursor que té la consulta vinculada quan es va declarar.

```
OPEN bound_cursorvar [ ( argument_values ) ];
```

El cursor ja no es pot tornar a obrir. Ha d'aparèixer una llista d'expressions d'argument si i només si el cursor es va declarar que tenia arguments. Aquests valors se substitueixen en la consulta. Tingueu en compte que les opcions `SCROLL` o `NO SCROLL` no poden ser determinades, ja que el comportament de desplaçament del cursor ja s'ha determinat.

Tingueu en compte que a causa de la substitució de variables que es fa a la consulta del cursor obligat, hi ha dues maneres de passar els valors a l'indicador: ja sigui amb un argument explícit per obrir, o implícitament per referència a una variable de PL/PgSQL a la consulta. No obstant això, només les variables declarades abans que el cursor es va declarar obligat seran substituïdes. En qualsevol cas, el valor que es passa es determina en el moment de l'obertura.

Exemples:

```
OPEN curs2;
OPEN curs3(42);
```

3.2.3 Emprant cursors

Una vegada que el cursor s'ha obert, es pot manipular amb les sentències descrites a continuació:

- `FETCH`
- `MOVE`
- `UPDATE/DELETE WHERE CURRENT OF`
- `CLOSE`

Aquestes manipulacions no han de passar necessàriament en la mateixa funció en què inicialment es va obrir el cursor. Una funció pot retornar un valor *refcursor* i deixar que posteriorment s'operi amb aquest cursor.

Internament, un valor *refcursor* és simplement el nom de la cadena d'un denominat *portal* que conté la consulta activa per al cursor. Aquest nom pot ser distribuït, assignat a altres variables *refcursor*, i així successivament, sense pertorbar el portal mateix.

Tots els portals es tanquen de manera implícita al final de la transacció. Per tant, un valor *refcursor* es pot utilitzar per fer referència a un cursor obert només fins al final de la transacció.

FETCH

`FETCH` recupera la fila següent del cursor i es converteix en un objectiu, que podria ser una variable de fila, una variable de registre, o una llista separada per comes de variables senzilles, com `SELECT INTO`.

```
FETCH [ direction { FROM | IN } ] cursor INTO target;
```

Si no hi ha una fila a continuació, l'objectiu s'estableix en `NULL(s)`. Igual que amb `SELECT INTO`, la variable especial `FOUND` es pot comprovar per veure si una fila s'ha obtingut o no.

La clàusula de direcció pot ser qualsevol de les variants permeses en les instruccions SQL FETCH, NEXT, PRIOR, FIRST, LAST, ABSOLUTE compte, RELATIVE compte, ALL, FORWARD [compte | ALL], o BACKWARD [compte | ALL].

Cursor cal que sigui un nom d'una variable que faci referència a un portal de cursor *refcursor* obert.

```
FETCH curs1 INTO rowvar;
FETCH curs2 INTO foo, bar, baz;
FETCH LAST FROM curs3 INTO x, y;
FETCH RELATIVE -2 FROM curs4 INTO x;
```

MOVE

MOVE reposiciona un cursor sense retornar cap dada.

```
MOVE [ direction { FROM | IN } ] cursor ;
```

MOVE treballa exactament com la sentència FETCH, exceptuant que encara que es reposiciona el cursor no retorna cap fila. De la mateixa manera que la sentència SELECT INTO, la variable especial FOUND ens permet comprovar per veure si hi ha una fila al costat per fer el moviment.

La clàusula de direcció pot ser qualsevol de les variants permeses en les instruccions SQL FETCH, NEXT, PRIOR, FIRST, LAST, ABSOLUTE compte, RELATIVE compte, ALL, FORWARD [compte | ALL], o BACKWARD [compte | ALL].

L'omissió de la direcció és la mateixa que especifica NEXT. Els valors de la direcció que requereixen moviment cap enrere és probable que fallin, tret que el cursor s'hagi declarat o s'obri amb l'opció SCROLL.

Exemples:

```
MOVE curs1;
MOVE LAST FROM curs3;
MOVE RELATIVE -2 FROM curs4;
MOVE FORWARD 2 FROM curs4;
```

UPDATE/DELETE WHERE CURRENT OF

Quan un cursor està situat en una fila de la taula, la fila pot ser actualitzada o esborrada amb el cursor per identificar la fila.

```
UPDATE table SET ... WHERE CURRENT OF cursor ;
DELETE FROM table WHERE CURRENT OF cursor ;
```

Quan un cursor està situat en una fila de la taula, la fila pot ser actualitzada o esborrada amb el cursor per identificar la fila. Hi ha restriccions en el que pot ser la consulta del cursor (en particular, sense agrupació) i el millor és utilitzar FOR UPDATE a l'indicador.

Un exemple:

```
UPDATE foo SET dataval = myval WHERE CURRENT OF curs1;
```

CLOSE

`CLOSE` tanca en el portal subjacent un cursor obert.

```
CLOSE cursor ;
```

Això es pot utilitzar per alliberar els recursos abans del final de la transacció, o per alliberar la variable de cursor si cal obrir-la de nou.

És molt recomanable tancar un cursor al final d'usar-lo.

Un exemple:

```
CLOSE curs1;
```

Retornar cursors

Les funcions PL/PgSQL poden retornar cursors quan s'executen. Això és útil per retornar diverses files o columnes, especialment amb conjunts de resultats molt grans. Per això, la funció obre el cursor i retorna el nom del cursor (o simplement s'obre el cursor amb un nom de portal especificat o conegut per l'usuari que executa la funció). L'usuari que fa la crida pot captar files del cursor. El cursor el pot tancar l'usuari que fa la crida, o aquest es tancarà automàticament quan es tanqui la transacció.

El nom del portal utilitzat per un cursor pot ser especificat pel programador o generat de manera automàtica. Per especificar un nom de portal, només ha d'assignar una cadena a la variable *refcursor* abans d'obrir-lo. El valor de la cadena de la variable *refcursor* serà utilitzada per `OPEN` com a nom del portal subjacent. No obstant això, si la variable *refcursor* és nul·la, `OPEN` genera automàticament un nom que no sigui incompatible amb qualsevol portal ja existent, i s'assigna a la variable *refcursor*.

Nom d'una variable de cursor

Una variable de cursor vinculada s'inicialitza amb el valor de cadena que representa el seu nom, de manera que el nom del portal és el mateix que el nom de la variable de cursor, llevat que el programador la sobreescrigui abans d'obrir el cursor. No obstant això, una variable cursor no vinculada prendrà un valor nul al principi, per la qual cosa rebrà un nom generat automàticament únic, llevat que es reemplaci.

L'exemple següent mostra una manera com el cursor pot tenir un nom de cursor subministrat per qui fa l'execució de la funció:

```
CREATE TABLE test (col text);
INSERT INTO test VALUES ('123');
CREATE FUNCTION reffunc(refcursor) RETURNS refcursor AS
BEGIN
OPEN $1 FOR SELECT col FROM test;
RETURN $1;
END;
' LANGUAGE plpgsql;
BEGIN;
SELECT reffunc('funccursor');
FETCH ALL IN funccursor;
COMMIT;
```

L'exemple següent utilitza la generació automàtica de nom de cursor:

```
CREATE FUNCTION reffunc2() RETURNS refcursor AS '
DECLARE
ref refcursor;
BEGIN
OPEN ref FOR SELECT col FROM test;
RETURN ref;
END;
' LANGUAGE plpgsql;
-- need to be in a transaction to use cursors.
BEGIN;
SELECT reffunc2();
reffunc2
--------------------
<unnamed cursor 1>
(1 row)
FETCH ALL IN "<unnamed cursor 1>";
COMMIT;
```

L'exemple següent mostra una manera de retornar múltiples cursors en una sola funció:

```
CREATE FUNCTION myfunc(refcursor, refcursor) RETURNS SETOF refcursor AS $$
BEGIN
OPEN $1 FOR SELECT * FROM table_1;
RETURN NEXT $1;
OPEN $2 FOR SELECT * FROM table_2;
RETURN NEXT $2;
END;
$$ LANGUAGE plpgsql;
-- necessitem emprar una transacció per emprar cursors.



BEGIN;
SELECT * FROM myfunc('a', 'b');
FETCH ALL FROM a;
FETCH ALL FROM b;
COMMIT;
```

3.2.4 Iterant a través del resultat del cursor

Hi ha una variant de la instrucció FOR que permet recórrer en iteració les files retornades per un cursor.

La seva sintaxi és:

```
[ <<label>> ]
FOR recordvar IN bound_cursorvar [ ( argument_values ) ] LOOP

statements
END LOOP [ label ];
```

La variable de cursor ha d'haver estat vinculada a alguna consulta quan va ser declarada, i no ha d'haver estat oberta encara.

La instrucció FOR obre automàticament el cursor, i es tanca el cursor de nou quan se surt del bucle.

Cal que aparegui una llista de valors d'argument si, i només si, el cursor es va declarar que tenia arguments. Aquests valors se substitueixen a la consulta, de la mateixa manera que durant un `OPEN`.

La variable **recordvar** es defineix automàticament com a tipus de registre i només existeix dins del bucle (qualsevol definició existent de nom de la variable es té en compte dins del bucle). Cada fila retornada pel cursor s'assigna successivament a aquesta variable de registre i el cos del bucle s'executa.

4. Disparadors

Els disparadors (*triggers*) són procediments emmagatzemats a la base de dades que s'executen automàticament quan es du a terme una operació `INSERT`, `DELETE` o `UPDATE` sobre alguna taula en concret i permeten als usuaris executar funcions introduïdes per altres usuaris sense conèixer-ne la implementació.

No es poden dur a terme en una operació `SELECT`.

Aquesta acció podria servir per fer una comprovació de consistència en un conjunt de valors per ser inserits, en el format de les dades abans de ser introduïdes, en una modificació en una taula o en una modificació d'un conjunt de files.

- Formen part del PostgreSQL a partir de la versió 7.3.
- Els disparadors són una manera d'estendre la funcionalitat igual que els procediments emmagatzemats.
- No requereixen intervenció de l'usuari, i s'invoquen automàticament.
- Es poden utilitzar per comprovar la consistència de les dades.
- Es defineixen perquè s'executin abans o després (`BEFORE | AFTER`) de `INSERT`, `UPDATE` o `DELETE`.
- Poden actuar per cada fila (`ROW`) o una vegada per instrucció d'SQL (`STATEMENT`).

4.1 Creació d'un disparador

PostgreSQL permet que els disparadors estiguin escrits en SQL, C o un altre llenguatge procedimental definit.

Un disparador cal que tingui un nom diferent de la resta de disparadors d'una mateixa taula.

La sintaxi per a la creació d'un disparador és:

```
1 CREATE TRIGGER nom { BEFORE | AFTER } { esdeveniment [ OR ... ] }
2 ON taula [ FOR [ EACH ] { ROW | STATEMENT } ]
3 EXECUTE PROCEDURE nom_funció ( arguments )
```

On podem distingir les clàusules següents:

- `BEFORE | AFTER` per determinar quan s'executa el disparador abans o després de l'esdeveniment. `BEFORE` s'executa abans de la instrucció i abans

que s'operi sobre una fila. `AFTER` s'executa després que s'afecti una fila i abans d'acabar la instrucció.

- *esdeveniment*= `INSERT`, `UPDATE` o `DELETE`. Es poden especificar múltiples esdeveniments separats per un `OR`.

- `FOR EACH ROW | FOR EACH STATEMENT`: per cada clàusula `FOR EACH` determina si el disparador ha de ser llançat per cada fila afectada o per cada declaració. El valor per defecte és `FOR EACH STATEMENT`.

Hi ha dos tipus de disparadors: els disparadors per fila i els disparadors per sentència. En els disparadors per fila la funció s'invoca una vegada per cada fila afectada per la sentència que ha disparat el disparador. Al contrari, en un disparador per declaració s'invoca sols una vegada quan s'executa la sentència apropiada sense tenir en compte el nombre de files afectades per aquesta declaració. Aquests tipus de disparador sempre retornen `NUL`.

Per exemple, suposem que volem tenir constància de qui modifica la taula d'empleats, ja que hi consten els sous de tots els guies i altres treballadors. En l'exemple següent veiem que cada vegada que es fa una inserció o una modificació en una taula es registra el nom de l'usuari que l'ha fet i l'hora en la fila inserida o modificada. A més, comprova que el nom de l'empleat no estigui a nul i que el salari tingui un valor positiu.

La taula *empleat* la creem així:

```
CREATE TABLE empleat (
nom_empleat text,
salari integer,
darrera_modif timestamp,
darrer_usuari text
);
```

Crearem la funció segons el que hem comentat:

```
CREATE FUNCTION reg_empleat() RETURNS trigger AS $reg_empleat$
BEGIN
   -- Comprova que s'informi el nom_empleat i el salari.
   IF NEW.nom_empleat IS NULL THEN
      RAISE EXCEPTION 'nom_empleat no pot ser null';
   END IF;
   IF NEW.salari IS NULL THEN
      RAISE EXCEPTION 'el salari de % no pot ser null,
      NEW.nom_empleat;
   END IF;
   -- Comprova que el salari sigui positiu.
   IF NEW.salari < 0 THEN
      RAISE EXCEPTION 'el salari  de % no pot ser negatiu',
      NEW.nom_empleat;
   END IF;
-- Registre de qui ha fet la modificació
   NEW.darrera_modif := 'now';
   NEW.darrer_usuari := current_user;
   RETURN NEW;
END;
$reg_empleat$ LANGUAGE plpgsql;
```

En posar % treu el nom de l'empleat, en el cas que el valor sigui negatiu.

Ara caldrà crear el disparador `reg_empleat` i provar-lo:

```
CREATE TRIGGER reg_empleat BEFORE INSERT OR UPDATE ON empleat
FOR EACH ROW EXECUTE PROCEDURE reg_empleat();
```

Per provar-lo inserim una fila:

```
INSERT INTO empleat VALUES ('mila',1000,null,null);
```

I fem la consulta per veure quin usuari ha accedit al registre d'un determinat empleat:

```
SELECT * FROM empleat WHERE nom_empleat ='mila';
```

El resultat és la fila inserida amb les dades de qui l'ha inserida i en quin moment.

Si intentem fer aquesta inserció:

```
INSERT INTO empleat VALUES ('miquel',-500,null,null);
```

Ens avisarà amb el missatge que hem previst en la funció que hem creat:

```
ERROR: el salari de Miquel no pot ser negatiu.
```

El mateix passarà si no introduïm cap `salari`.

4.1.1 Variables especials associades a un disparador

Quan una funció PL/PgSQL és cridada per un disparador es creen diverses variables especials:

Taula 4.1.

Nom de la variable	Tipus de dada	Descripció
NEW	RECORD	Variable que conté la nova fila de la base de dades per a operacions INSERT / UPDATE en disparadors a escala de fila (*row-level*). Aquesta variable és NULL en els disparadors a escala d'instrucció (*statement-level*)i per a les operacions DELETE.
OLD	RECORD	Variable que conté el valor antic de la fila per a operacions UPDATE/DELETE en disparadors a escala de fila (*row-level*). Aquesta variable és NULL en els disparadors a escala d'instrucció (*statement-level*) i per a les operacions INSERT.
TG_NAME	name	Variable que conté el nom del disparador que en realitat s'ha disparat

TAULA 4.1 (continuació)

Nom de la variable	Tipus de dada	Descripció
TG_WHEN	text	Una cadena BEFORE o AFTER, depenent de la definició del disparador.
TG_LEVEL	text	Una cadena ROW o STATEMENT, depenent de la definició del disparador.
TG_OP	text	Una cadena INSERT, UPDATE, DELETE o TRUNCATE, que diu quina operació ha llençat el disparador.
TG_RELID	oid	L'OID (identificador d'objecte) de la taula que ha causat la invocació.
TG_RELNAME	name	El nom de la taula que va causar la invocació del disparador. Això ara està en desús, i podria desaparèixer en futures versions. Millor emprar TG_TABLE_NAME.
TG_TABLE_NAME	name	El nom de la taula que va causar la invocació del disparador
TG_TABLE_SCHEMA	name	El nom de l'esquema de la taula que va causar la invocació del disparador
TG_NARGS	integer	El nombre d'arguments donats en el procediment d'activació en la declaració CREATE TRIGGER
TG_ARGV[]	text array	Els arguments de la sentència CREATE TRIGGER. L'índex compta de 0. Índexs invàlids (més petit que 0 o més gran o igual que tg_nargs) donen com a resultat un valor nul.

Aquest exemple de disparador assegura que cada vegada que s'insereixi o actualitzi una fila a la taula el nom d'usuari i l'hora s'emmagatzemaran a la fila. I comprova que es dóna el nom d'un empleat i que el salari tingui un valor positiu.

```
1  CREATE TABLE emp (
2  empname text,
3  salary integer,
4  last_date timestamp,
5  last_user text
6  );
7
8
9  CREATE FUNCTION emp_stamp() RETURNS trigger AS $emp_stamp$
10 BEGIN
11 -- Check that empname and salary are given
12 IF NEW.empname IS NULL THEN
13 RAISE EXCEPTION 'empname cannot be null';
14 END IF;
15 IF NEW.salary IS NULL THEN
16 RAISE EXCEPTION '% cannot have null salary', NEW.empname;
17 END IF;
18 -- Who works for us when she must pay for it?
19 IF NEW.salary < 0 THEN
20 RAISE EXCEPTION '% cannot have a negative salary', NEW.empname;
21 END IF;
22 -- Remember who changed the payroll when
23 NEW.last_date := current_timestamp;
24 NEW.last_user := current_user;
```

```
RETURN NEW;
END;
$emp_stamp$ LANGUAGE plpgsql;

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON emp
FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```

Una altra manera de registrar els canvis en una taula consisteix a crear una nova taula que conté una fila per cada inserció, actualització o eliminació que es produeix. Aquest enfocament pot ser considerat com l'auditoria de canvis en una taula.

Aquest exemple de disparador exemple s'assegura que per a qualsevol inserció, actualització o eliminació d'una fila a la taula *emp* es registri (és a dir, s'auditi) en la taula *emp_audit*. L'hora i el nom d'usuari s'han enregistrat a la fila, juntament amb el tipus d'operació que s'hi fa:

```
CREATE TABLE emp (
empname text NOT NULL
salary integer
);
,

CREATE TABLE emp_audit(
operation char(1) NOT NULL,
stamp timestamp NOT NULL,
userid text NOT NULL,

empname text NOT NULL,
salary integer
);

CREATE OR REPLACE FUNCTION process_emp_audit() RETURNS TRIGGER AS $emp_audit$
BEGIN
--
-- Create a row in emp_audit to reflect the operation performed on emp,
-- make use of the special variable TG_OP to work out the operation.
--
IF (TG_OP = 'DELETE') THEN
INSERT INTO emp_audit SELECT 'D', now(), user, OLD.*;
RETURN OLD;
ELSIF (TG_OP = 'UPDATE') THEN
INSERT INTO emp_audit SELECT 'U', now(), user, NEW.*;
RETURN NEW;
ELSIF (TG_OP = 'INSERT') THEN
INSERT INTO emp_audit SELECT 'I', now(), user, NEW.*;
RETURN NEW;
END IF;
RETURN NULL; -- result is ignored since this is an AFTER trigger
END;
$emp_audit$ LANGUAGE plpgsql;

CREATE TRIGGER emp_audit
AFTER INSERT OR UPDATE OR DELETE ON emp
FOR EACH ROW EXECUTE PROCEDURE process_emp_audit();
```

4.2 Altres tipus de disparadors

Podem considerar altres aspectes quant als disparadors:

- Disparadors múltiples.
- Disparadors en cascada.

4.2.1 Disparadors múltiples

Si es defineix més d'un disparador s'executen en ordre alfabètic per nom. En el cas de `BEFORE` el resultat d'un es converteix en l'entrada del següent i si un retorna `NULL` s'abandona l'operació. Típicament, els disparadors `BEFORE` s'utilitzen per comprovar o modificar les dades que s'introduiran o s'actualitzaran. Per exemple, un disparador `BEFORE` podria ser utilitzat per introduir el temps actual en una columna del tipus *timestamp*, o comprovar que dos elements de la fila són coherents.

4.2.2 Disparadors en cascada

Si una funció executa instruccions SQL, aquestes podrien originar l'execució d'altres disparadors. D'aquest fet pot resultar una recursivitat infinita. Un disparador per a un `INSERT` pot utilitzar una funció que insereixi una fila en una taula, i provoqui un nou `INSERT` i així successivament. No hi ha cap limitació en el nombre de nivells de la cascada. És responsabilitat del programador del disparador evitar la recursivitat infinita.

4.3 Modificació i eliminació d'un disparador

Igual que es pot crear un disparador també hi ha la possibilitat de modificar-lo:

```
1 ALTER TRIGGER nom ON taula RENAME TO nou_nom
```

O eliminar-lo:

```
1 DROP TRIGGER nom ON taula [ CASCADE | RESTRICT ]
```

Si eliminem una funció que té un disparador definit, el disparador fallarà, i tornar a crear la funció amb el mateix nom no corregirà el problema. Cal tornar a crear el disparador després de tornar a crear la funció.

- Amb `CASCADE` se suprimeixen automàticament els objectes que depenen del disparador.
- `RESTRICT` és el valor per defecte i es nega a suprimir el disparador si hi ha objectes que en depenen.

Eliminem el disparador que hem definit en l'exemple:

```
1 DROP TRIGGER reg_empleat ON empleat;
```

Els disparadors ja existents en el PostgreSQL estan emmagatzemats en la vista del sistema `pg_trigger`. Si fem la consulta d'aquesta vista veurem que hi ha diversos disparadors ja creats i els que hem creat nosaltres amb les restriccions.

Bases de dades objecte-relacionals

Aina del Tio Esteve

Índex

Introducció

Les bases de dades objecte-relacionals agrupen les particularitats de les bases de dades relacionals i les bases de dades orientades a objectes. Entre altres característiques, aquestes base de dades:

1. estableixen interconnexions entre les dades per mitjà de les relacions,
2. encapsulen les dades emmagatzemant-les en estructures complexes i protegint-les d'accessos il·lícits per part d'usuaris externs,
3. estenen la funcionalitat d'unes estructures de dades a unes altres fomentant la reutilització, i
4. permeten crear nous tipus d'usuari proporcionant flexibilitat.

Per desenvolupar les funcionalitats anteriors farem un recorregut al llarg de dos llenguatges. Un d'ells, el **llenguatge de definició de dades** (*data definition language,* DDL) permet a l'usuari definir tant estructures de dades com funcions o procediments que permetran consultar-les; l'altre, el **llenguatge de manipulació de dades** (*data manipulation language,* DML) permet a l'usuari dur a terme tasques de consulta o manipulació de dades.

Hi ha gestors de bases de dades objecte-relacionals que faciliten l'administració i gestió de les bases de dades. En aquesta unitat treballarem amb el **PostgreSQL** pel conjunt de funcionalitats avançades que suporta i pel seu funcionament multiplataforma. Cal tenir en compte que la migració de bases de dades allotjades en productes comercials a PostgreSQL es facilita gràcies al fet que suporta àmpliament l'estàndard SQL.

La versió del PostgreSQL que s'ha utilitzat durant la redacció d'aquest material, i en els exemples, és la 9.1, última versió estable en aquest moment.

Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Gestiona la informació emmagatzemada en bases de dades d'objectes relacionals, avaluant i utilitzant les possibilitats que proporciona el sistema gestor.

- Descriu les característiques de les bases de dades objecte-relacionals.
- Crea tipus de dades objecte, els seus atributs i mètodes.
- Crea taules d'objectes i taules de columnes tipus objecte.
- Crea tipus de dades col·lecció.
- Realitza consultes.
- Modifica la informació emmagatzemada mantenint la integritat i consistència de les dades.

1. Bases de dades d'objectes relacionals

El terme *base de dades d'objectes relacionals* (BDOR) s'utilitza per descriure una base de dades que ha evolucionat des del model relacional cap a una altra més amplia que incorpora conceptes del paradigma orientat a objectes. Per tant, un**sistema de gestió d'objectes relacional**, *object database management system* (**ODBMS)**, conté totes dues tecnologies: la relacional i la d'objectes.

Un paradigma és una manera de representar i manipular el coneixement. Representa un enfocament particular o filosofia per a la construcció del programari. No és millor un que un altre, sinó que cada un té avantatges i desavantatges.

Les bases de dades **relacionals** han estat el model utilitzat fins a l'actualitat per modelitzar problemes reals i administrar dades dinàmicament. Compleixen el model relacional, i la seva idea fonamental és la utilització de relacions. Aquestes permeten establir interconnexions entre les dades i treballar-hi amb conjuntament.

Les bases de dades **orientades a objectes** tracten d'emmagatzemar dades d'objectes complets. Quan s'integren les característiques d'una base de dades amb les d'un llenguatge de programació orientat a objectes, el resultat és un ODBMS que estén els llenguatges amb dades persistents de manera transparent, control de concurrència, recuperació de dades i consultes associatives.

Les bases de dades orientades a objectes tracten d'emmagatzemar dades d'objectes complets.

Els sistemes de gestió d'objectes relacional, ODBMS, són una bona elecció per als sistemes que necessitin un bon rendiment en la manipulació de tipus de dades complexes. En relació amb altres sistemes de gestió, proporcionen els costos més baixos de desenvolupament i també el millor rendiment quan s'utilitzen estructures d'objectes, gràcies al fet que el model d'objecte utilitzat en l'aplicació s'emmagatzema exactament en el disc, de manera que té una integració transparent entre la base de dades i el programa escrit en el llenguatge de programació orientat a objectes. En emmagatzemar exactament el model d'objecte utilitzat en l'aplicació, els costos de desenvolupament i manteniment es redueixen.

1.1 Característiques de les bases de dades objecte-relacional

Una idea bàsica de les bases de dades objecte-relacional (BDOR) és que l'usuari pugui crear els seus propis tipus de dades, per ser utilitzats en la tecnologia que permeti la implementació de tipus de dades predefinits. A més, també permeten crear mètodes per a aquests tipus de dades, proporcionant flexibilitat i seguretat.

Un **objecte** és una instància d'un tipus objecte. Un **tipus objecte** és un paquet cohesionat que consisteix en un tipus concret de dada. S'usa per agrupar camps relacionats i mètodes, i descriu les regles que marquen el comportament de l'objecte.

Els sistemes gestors de BDOR són compatibles en sentit ascendent amb les bases de dades relacionals tradicionals, és a dir, es poden exportar les aplicacions sobre bases de dades relacionals al nou model sense que calgui reescriure-les.

Els **conceptes principals** de les BDOR són l'encapsulació, l'herència i el polimorfisme.

Seguidament veurem els conceptes principals:

1. **L'encapsulació** és l'ocultació de les dades d'un objecte de manera que només es poden modificar mitjançant les operacions definides per aquest objecte. L'aïllament protegeix les dades associades a un objecte de la modificació o eliminació sense permís per part d'un usuari extern.

2. **L'herència** és la relació entre un tipus objecte general i un altre de més especific. Un tipus objecte es deriva de l'altre estenent la seva funcionalitat, essent el mecanisme fonamental per implementar el polimorfisme i la reutilització.

3. **El polimorfisme** fa referència a la sobreescriptura de mètodes a l'hora d'implementar un tipus objecte utilitzant l'herència. No s'ha de confondre amb la sobrecàrrega de funcions que permeten altres llenguatges de programació.

La sobrecàrrega és la característica que permet tenir, en un mateix tipus objecte, diferents mètodes amb el mateix nom, mentre que la sobreescriptura (polimorfisme) és la característica que permet canviar la implementació d'un mètode, heretat del tipus objecte base, en el tipus objecte derivat.

1.1.1 Característiques del PostgreSQL

El PostgreSQL és un gestor de bases de dades relacionals amb suport per a objectes, que ens facilita l'administració i gestió de les bases de dades. Com que és un gestor que funciona àmpliament amb l'estàndard SQL, facilita la migració de dades allotjades en productes comercials.

Història

El PostgreSQL s'inicia el 1986 amb un projecte del professor Michael Stonebraker i un equip de desenvolupadors de la Universitat de Berkeley (Califòrnia). El nom original va ser Postgres. En el seu disseny es van incloure alguns conceptes avançats en bases de dades i suport parcial a l'orientació a objectes.

El Postgres va ser comercialitzat per Illustra, una empresa que posteriorment va formar part d'Informix. Va arribar un moment en què mantenir el projecte absorbia massa temps als investigadors i acadèmics, per la qual cosa el 1993 es va presentar la versió 4.5 i oficialment es va donar per acabat el projecte.

El 1994, Andrew Yu i Jolly Chen van incloure SQL a Postgres per posteriorment presentar-ne el codi al Web amb el nom de Postgres95. El projecte incloïa molts canvis en el codi original que en milloraven el rendiment i legibilitat.

El 1996 el nom va canviar a PostgreSQL reprenent la seqüència original de versions, per la qual cosa es va presentar la versió 6.0. L'última versió estable oficial, de 2004, és la 7.4.6, mentre que la versió 8.0 està en fase final d'estabilització.

Llicència BSD

És la llicència de programari principalment per als sistemes BSD (Berkeley Software Distribution), que pertany al grup de llicències de programari lliure. Aquesta té menys restriccions comparada amb d'altres com la GPL (General Public License), i és més propera al domini públic. El projecte PostgreSQL continua publicant versions principals anualment, com també versions menors de correcció d'errors (*bugs*), totes disponibles sota la llicència BSD, i basades majorment en contribucions de proveïdors comercials, empreses que hi donen suport i programadors de codi obert.

Prestacions

PostgreSQL destaca per l'àmplia llista de prestacions que el fan capaç de competir amb qualsevol sistema gestor de bases de dades comercial:

1. Esta desenvolupat en C, amb eines com Yacc i Lex.
2. Disposa d'un conjunt de tipus de dades, que en permet l'extensió mitjançant tipus i operadors definits i programats per l'usuari.
3. L'API d'accés a l'SGBD (Sistema Gestor de Bases de Dades) es troba disponible en C, C++, Java, Perl, PHP, Python i TCL, entre altres.
4. L'administració es basa en usuaris i privilegis.
5. Les seves opcions de connectivitat abasteixen TCP/IP, sòcols UNIX i sòcols NT.
6. És altament segur en relació amb l'estabilitat.
7. Es pot estendre amb biblioteques externes per suportar encriptació, cerques per similitud fonètica, etc.
8. Controla la concurrència multiversió, millorant sensiblement les operacions de bloqueig i transaccions en sistemes multiusuari.
9. Inclou suport per a vistes, claus foranes, integritat referencial, disparadors, procediments emmagatzemats, subconsultes i la majoria dels tipus i operadors suportats a SQL92 i SQL99.

   Limitacions

Trobareu més informació sobre l'ús de les transaccions, punts de recuperació i *tablespaces* en l'apartat "DDL i DML de les bases de dades objecte-relacionals" d'aquesta unitat.

Les limitacions d'aquest tipus de gestor de bases de dades objecte-relacionals se solen identificar en analitzar les prestacions que hi ha previstes per a les pròximes versions. Actualment, les limitacions principals són:

1. Punts de recuperació dins de transaccions. Actualment, les transaccions avorten completament si es troba un error durant l'execució. La definició de punts de recuperació permetria recuperar millor transaccions complexes.
2. No suporta *tablespaces* per definir on s'ha d'emmagatzemar la base de dades, l'esquema, els índexs, etc.
3. El suport a l'orientació a objectes és una extensió simple que ofereix prestacions com l'herència, però no dóna un suport complet.

1.1.2 El model orientat a objectes

En els llenguatges de programació orientats a objectes, els tipus objecte s'anomenen **classes**.

PostgreSQL inclou extensions d'orientació a objectes, tot i que no és un sistema gestor de bases de dades orientat a objectes complet. Els conceptes següents el relacionen amb aquest paradigma:

1. Objectes
2. Propietats
3. Encapsulació
4. Herència
5. Persistència
6. Adaptació a un sistema gestor de bases de dades

Objectes

En el món que ens envolta hi ha diferents objectes d'un mateix tipus -també podríem dir d'una mateixa classe. Per exemple, hi pot haver més d'un cotxe del mateix tipus. Si utilitzem la terminologia d'orientació a objectes, direm que el nostre cotxe és una instància del tipus objecte *cotxes*.

Un **objecte** és un tipus de dades que encapsula les dades necessàries i les funcions per accedir-hi.

Tots els cotxes tenen atributs (color, motor, potència) i mètodes (accelerar, frenar); tot i així cada cotxe tindrà un estat particular independent de la resta. Per això, podem dir que cada objecte té **identitat** pròpia: encara que dos objectes tinguin els mateixos valors seran entitats diferents. En els models orientats a objectes, la identitat es representa amb l'identificador d'objecte (*object identifier,* OID), que és únic per a cada objecte.

Els OID s'encarreguen de fer referència a un objecte des d'un altre; d'aquesta manera es creen les relacions entre objectes.

Un **tipus objecte** o classe defineix les característiques (atributs) i els comportaments (mètodes) que són comuns per a tots els objectes d'un cert tipus.

Tots els objectes que comparteixen les mateixes propietats pertanyen al mateix tipus objecte. Per mitjà d'aquest es defineixen les propietats dels objectes i s'utilitza com a plantilla per crear-los. Anomenem *instància de classe* la utilització de la definició d'un tipus objecte per obtenir un determinat objecte. Com a concepte, la instància és equivalent a un *tuple* (en català *n*-pla) concret en una taula d'una base de dades.

Els elements fonamentals en l'orientació a objectes són els objectes i els tipus objecte.

Propietats

Les propietats dels objectes poden ser dinàmiques o estàtiques. Els **atributs** representen una propietat estàtica d'un objecte. Els **mètodes** representen una propietat dinàmica, és a dir, una modificació sobre un atribut o una acció que es pot efectuar.

El conjunt de valors dels atributs en un moment donat es coneix com a **estat de l'objecte**. Els operadors actuen sobre l'objecte canviant-ne l'estat. La seqüència d'estats pels quals passa un objecte en executar els mètodes en defineixen el **comportament**.

Les propietats dels objectes són el seu estat i el seu comportament.

Encapsulació

Per a l'usuari l'estructura interna dels objectes ha d'estar oculta; no necessita conèixer-la per interactuar amb aquests.

El mètode és la implementació d'una operació.

En la **interfície** d'un objecte se'n defineixen els atributs i els mètodes, i és la part pública de l'objecte. L'estructura interna d'un objecte s'anomena **implementació**.

L'encapsulació comporta els avantatges següents:

1. La modificació interna de la implementació d'un objecte per modificar-lo o millorar-lo no afecta l'usuari.
2. Se simplifica l'ús de l'objecte, ja que se n'oculten els detalls del funcionament i aquest es presenta en termes de les seves propietats. En elevar el nivell d'abstracció es disminueix el nivell de complexitat d'un sistema.
3. Constitueix un mecanisme d'integritat. La dispersió d'un error per tot el sistema és menor, ja que en presentar una divisió entre interfície i implementació, els errors interns d'un objecte troben la barrera de l'encapsulació abans de propagar-se per tot el sistema.
4. Permet substituir objectes amb la mateixa interfície i diferent implementació.

   S'han de diferenciar els mètodes de les operacions. Les operacions no són exclusives dels tipus d'objecte; en canvi, els mètodes sí.

Una **operació** especifica què s'ha de fer, i un **mètode** com s'ha de fer.

Aquesta diferència permet tenir múltiples mètodes per a una mateixa operació.

Herència

En aquest apartat ens referirem als tipus objecte com a **classes**, nomenclatura que s'utilitza en el paradigma de l'orientació a objectes.

L'**herència** es defineix com el mecanisme pel qual s'utilitza la definició d'una classe anomenada *pare* per definir un nova classe anomenada *fill* que en pot heretar els atributs i les operacions.

L'herència de tipus

En l'herència de tipus el que hereta la subclasse són els atributs de la superclasse, però no necessàriament la seva implementació, ja que els pot tornar a implementar.

Les classes *fill* també es coneixen com a **subclasses**, i a les classes *pare* com a **superclasses**. La relació d'herència entre classes genera el que s'anomena j*erarquia de classes*.

Podem trobar diferents tipus d'herència. Parlem d'**herència de tipus** quan la subclasse hereta la interfície d'una superclasse, és a dir, els atributs i els mètodes. Parlem d'**herència estructural** quan la subclasse hereta la implementació de la superclasse, és a dir, les variables d'instància i els mètodes.

Exemple d'herència de tipus

Un cavall és-un animal. Totes les propietats de la classe "animal" les té la classe "cavall". Però un animal no-és necessàriament un cavall. Totes les propietats de cavall no les tenen tots els animals.

L'herència de tipus defineix relacions **és-un** entre classes, en què la subclasse té totes les propietats de la superclasse, però la superclasse no té totes les propietats de la subclasse.

Considerem una variable que és de tipus Animal, en el llenguatge de programació Java:

```
public class Animal {

Integer: velocitat;

public void correr(){

this.velocitat += 5;

}

}

Animal elmeuanimal = new Animal();

System.out.println(elmeuanimal.correr());
```

Sortida: 5

Pot fer referència a objectes de tipus Animal, o a tipus derivats d'aquest, com gos, gat o iguana.

```
public class Iguana extends Animal {

public void correr(){

```

```
5  this.velocitat +=1;
6
7  }
8
9  }
10
11 elmeuanimal = new Iguana();
12
13 System.out.println(elmeuanimal.correr());
```

Sortida: 1

Es construeix una nova variable Iguana i s'hi fa referència com a Animal.

La propietat de substituir objectes que descendeixen de la mateixa superclasse, com ja s'ha vist anteriorment, es coneix com a *polimorfisme* i és el mecanisme de reutilització.

La referència al tipus Animal és una referència polimòrfica, ja que es pot referir a tipus derivats d'Animal. A partir d'una referència polimòrfica es poden sol·licitar operacions sense conèixer-ne el tipus exacte:

```
1  elmeuanimal.correr();
```

El mètode correr() té el mateix significat per a tots els tipus Animal. Com ja hem vist, cadascú utilitzarà un mètode diferent per executar la instrucció.

Una classe pot heretar les propietats de dos superclasses mitjançant el que es coneix com a *herència múltiple*.

En una herència múltiple, podem trobar que en les dues superclasses hi hagi propietats amb els mateixos noms: això s'anomena **col·lisió de noms**. Podríem trobar els casos següents:

1. Els noms són iguals perquè es refereixen a la mateixa propietat. No hi ha conflicte perquè és la mateixa, només s'ha de definir una implementació adequada.
2. Els noms són iguals però tenen significats diferents. Hi ha conflicte i es pot resoldre canviant els noms de les propietats heretades que el tinguessin.

Persistència

La persistència es defineix com la capacitat d'un objecte per sobreviure al temps d'execució d'un programa. Per implementar-la, s'utilitza l'emmagatzematge secundari.

S'han proposat diferents mecanismes per implementar la persistència en els llenguatges de programació, entre els quals destaquen els següents:

1. **Arxius nets**. Es creen arxius per emmagatzemar els objectes en el format que vol el programador. Els objectes es carreguen en obrir el programa i es desen en finalitzar. Aquesta és l'opció més accessible per a tots els llenguatges de programació.
2. **Bases de dades relacionals**. Els objectes són mapats a taules; un mòdul del programa s'encarrega de fer les transformacions objecte-relacionals.

Aquest enfocament consumeix molt de temps a l'hora de dur a terme el mapatge. Hi ha eines que fan els mapatges de manera automàtica.

3. **Bases d'objectes**. Els objectes s'emmagatzemen de manera natural en una base d'objectes, i la consulta i recuperació és administrada pel gestor; d'aquesta manera les aplicacions no necessiten saber els detalls de la implementació.

4. **Persistència transparent**. El sistema emmagatzema i recupera els objectes quan ho creu convenient, sense que l'usuari hagi de fer cap sol·licitud explícita de consulta, actualització o recuperació d'informació a una base d'objectes. No es requereix un altre llenguatge per interactuar amb les bases de dades.

Adaptació a un sistema gestor de base de dades

Un dels conceptes fonamentals en l'orientació a objectes és el concepte de classe. Hi ha dos enfocaments per associar el concepte de classe amb el model relacional:

1. Les classes defineixen **tipus de taules**.

2. Les classes defineixen **tipus de columnes**.

Atès que en el model relacional les columnes estan definides per tipus de dades, el més correcte és associar les columnes amb les classes (vegeu la taula 1.1).

TAULA 1.1. Relació entre enfocaments i conceptes

	1r enfocament	2n enfocament
Objectes	Valors	n-plens
Classes	Dominis	Taules

Les bases de dades objecte-relacionals permeten fer una migració gradual de sistemes relacionals als sistemes orientats a objectes i que tots dos tipus d'aplicacions coexisteixin. No és fàcil aconseguir la coexistència dels dos models de dades, i és necessari equilibrar els conceptes de cadascun dels models sense que entrin en conflicte.

PostgreSQL implementa els objectes com a n-plans i les classes com a taules. Tot i que també és poden definir nous tipus de dades mitjançant els mecanismes d'extensió.

1.1.3 Mapatge d'objectes relacional

El **mapatge d'objectes relacional** (o*bject-relational mapping,* ORM) és una tècnica de programació per convertir dades de llenguatges de programació orientats a objectes en la seva representació en bases de dades relacionals, mitjançant la definició de les correspondències entre els diferents sistemes.

La gestió de dades en programació orientada a objectes es fa mitjançant la **manipulació d'objectes**, que quasi sempre presenten valors no escalables.

Considerant l'exemple d'una entrada d'una llibreta d'adreces, que representa una persona amb zero o més adreces i zero o més telèfons, en orientació a objectes això es representaria com un objecte "persona", del qual penjarien les seves dades personals, les adreces i els telèfons -aquestes dues darreres dades també serien objectes. Les bases de dades relacionals només poden emmagatzemar valors escalars, com cadenes de caràcters o enters i organitzar-los en taules. El programador ha de convertir els valors representats en forma d'objectes en valors simples o agrupats per emmagatzemar en la base de dades i implementar el procés invers. Alternativament podria treballar només amb valors escalars, i perdria els avantatges de la programació orientada a objectes. Per poder gaudir de la potència d'aquesta i estalviar-se els processos de conversió d'objecte a registre de la base de dades i el seu invers, s'ha creat l'automatització d'aquest procés, mitjançant el mapatge d'objectes relacional.

En utilitzar bases de dades objecte-relacional, és a dir, bases de dades dissenyades específicament per treballar amb valors d'objectes, eliminem la necessitat de convertir dades en objectes i des de la seva forma SQL, ja que les dades s'emmagatzemen de manera automàtica en la seva representació original.

El **mapatge d'objectes relacional**(ORM) elimina la necessitat de convertir dades en objectes, ja que aquestes s'emmagatzemen en la seva representació original.

La majoria de les eines ORM permeten crear el model mitjançant l'esquema de la base de dades; és a dir, en crear la base de dades l'eina llegeix automàticament l'esquema de les taules i les relacions, i crea un model ajustat. Una vegada el model està creat, el desenvolupament és més ràpid, ja que hi ha una abstracció que permet obviar les consultes a la base de dades, i el sistema ORM genera de manera automàtica les consultes a la base de dades per convertir els registres en objectes.

La característica més important dels sistemes ORM és el llenguatge propi que ofereix per a les consultes a la base de dades. Un conjunt de tipus objecte i mètodes permet extreure les dades de la base de dades deixant de banda la sintaxi de l'SQL (*sttructured query language*) i utilitzar el propi llenguatge de l'eina.

1.2 Gestió servidor-client PostgreSQL

El PostgreSQL està basat en una arquitectura client-servidor, la qual cosa vol dir que hi ha una relació entre les entitats client i servidor. Un procés servidor pot atendre exclusivament un sol client, és a dir, calen tants processos servidor com clients hi hagi. L'encarregat d'executar un nou servidor per a cada client que sol·liciti una connexió és el procés *postmaster*.

1.2.1 Instal·lació del programari

Per instal·lar el programari en primer lloc cal que descarregueu el paquet d'instal·lació de la pàgina web oficial: http://www.postgresql.org/download/.

La distribució PostgreSQL oficial està disponible en diferents formats binaris i en codi font. El paquet complet inclou:

1. El servidor PostgreSQL amb documentació completa: html, man.
2. Diverses eines de línia d'ordres: psql, pg_ctl, pg_dump, pg_restore, etc.
3. Biblioteca en C, libpq; i processador C incorporat, ecpg.
4. Nombrosos llenguatges per a funcions/procediments emmagatzemats: plpgsql, pltcl, plperl, plpython.
5. Nombrosos paquets addicionals: metaphone, pgcrypto, etc.

Una vegada tinguem el paquet descarregat, procedirem a la seva instal·lació i posterior execució.

1.2.2 Accés al servidor PostgreSQL

Abans d'intentar connectar-nos amb el servidor, ens hem d'assegurar que està funcionant i admet connexions, tant locals (l'SGBD s'està executant a la mateixa màquina que intenta la connexió) com remotes.

Una vegada comprovat el funcionament correcte del servidor, hem de disposar de les credencials necessàries per connectar-nos-hi. Per simplificar, suposarem que disposem de les credencials de l'administrador de la base de dades, que normalment són l'usuari PostgreSQL i la seva contrasenya.

El client psql

Per connectar-se amb un servidor es requereix, lògicament, un programa client. En la distribució de PostgreSQL s'inclou un client, psql, fàcil d'utilitzar, que permet la introducció interactiva d'ordres en mode text.

Per fer una connexió es requereixen les dades següents:

1. Servidor. Si no s'especifica, s'utilitza *localhost*.
2. Usuari. Si no s'especifica, s'utilitza el nom d'usuari UNIX que executa psql.
3. Base de dades.

Executem en el terminal l'SQL Shell psql que proporciona el programari i introduïm les dades que ens demana; si no n'especifiquem cap el client psql utilitza els valors per defecte (els que apareixen entre claudàtors):

```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres: *****
psql (9.1.1)
Type "help" for help.
postgres=#
```

El símbol # significa que el psql està llest per llegir l'entrada de l'usuari.

Les sentències SQL s'envien directament al servidor per tal que les interpreti. Les ordres internes tenen la forma `\ordre` i ofereixen opcions que no estan incloses en l'SQL i que són interpretades internament pel psql.

Per acabar la sessió amb psql, utilitzem l'ordre `\q` o podem polsar les tecles Ctrl + D.

Introducció de sentències

Les sentències SQL que escriguem al client hauran d'acabar amb ; o bé amb \g:

```
postgres=# select user;
current_user
--------------
postgres
(1 row)
```

Quan una ordre ocupa més d'una línia, l'indicador canvia de forma i va assenyalant l'element que encara no s'ha completat:

```
postgres=# select
postgres-# user\g
current_user
--------------
postgres
(1 row)
```

TAULA 1.2. Indicadors de PostgreSQL

Indicador	Significat
=#	Espera una nova sentència.
-#	La sentència encara no s'ha acabat amb `;` o `\g`.
‘“#	Una cadena en cometes dobles no s'ha tancat.
’#	Una cadena en cometes simples no s'ha tancat.
(#	Un parèntesi no s'ha tancat.

El client psql emmagatzema la sentència fins que se li dóna l'ordre d'enviar-la a l'SGBD. Per visualitzar el contingut de la memòria intermèdia (*buffer*) en què ha emmagatzemat la sentència, disposem de l'ordre `\p`:

```
postgres=# select
postgres-# 2 * 10 + 1
postgres-# \p
select
2 * 10 + 1
postgres-# \g
?column?
----------
21
(1 row)
```

El client també disposa d'una ordre que permet esborrar completament la memòria intermèdia per començar de nou amb la sentència:

```
postgres=# select 'Hola'\r
Query buffer reset (cleared).
```

Expressions i variables

El client psql disposa de multitud de prestacions avançades, entre les quals (com ja hem comentat) hi ha el suport per a la substitució de variables, similar al dels *shells* de l'UNIX:

```
postgres=# \set var1 demostracio
```

Aquesta sentència crea la variable `var1` i li assigna el valor `demostració`. Per recuperar el valor de la variable, simplement l'haurem d'incloure precedida de `:` en qualsevol sentència o bé veure'n el valor mitjançant l'ordre `echo`:

```
postgres=# \echo :var1
demostracio
```

De la mateixa manera, psql defineix algunes variables especials que poden ser útils per conèixer detalls del servidor al qual estem connectats:

```
postgres=# \echo :DBNAME :ENCODING :HOST :PORT :USER;
postgres UTF8 localhost 5432 postgres ;
```

Procés per lots i formats de sortida

El psql pot processar ordres per lots emmagatzemats en un arxiu del sistema operatiu mitjançant la sintaxi següent:

```
$ psql postgres –f arxiu.psql
```

El mateix intèrpret psql ens proporciona mecanismes per emmagatzemar en un fitxer el resultat de les sentències:

1. Especificant el fitxer de destinació directament en finalitzar una sentència:

```
postgres=# select user \g /tmp/a.txt
```

2. Mitjançant una *pipe* a través de la qual enviem la sortida a una ordre UNIX:

```
postgres=# select user \g | cat > /tmp/b.txt
```

Pipe

És un dispositiu lògic destinat a comunicar processos.El seu funcionament és el d'una cua de caràcters, amb una longitud fixa en què els processos poden llegir i escriure.

3. Mitjançant l'ordre `\o`, que permet indicar quina ha de ser la sortida de les sentències SQL que s'executin en endavant:

```
postgres=# \o /tmp/sentencies.txt
postgres=# select user;
postgres=# select 1+1+4;
postgres=# \o
postgres=# select 1+1+4;
?column?
----------
6
(1 row)
```

Notació

A l'ordre \o se li ha d'especificar un fitxer o bé una ordre que anirà rebent els resultats mitjançant una pipe.Quan es vulgui tornar a la sortida estàndard (stdout) simplement es donarà l'ordre \o sense cap paràmetre.

4. Es pot especificar el format de sortida dels resultats d'una sentència –per defecte, psql els mostra en forma tabular mitjançant text. Per a canviar-lo s'ha de modificar el valor de la variable interna `format` mitjançant l'ordre `\pset`. En especificar que es vol la sortida en HTML, es redirigeix a un fitxer i es genera un arxiu HTML que permet veure el resultat de la consulta mitjançant un navegador web convencional.Vegem, en primer lloc, l'especificació del format de sortida:

```
postgres=# \pset format html
Output format is html.
postgres=# select user;
<table border="1">
<tr>
<th align="center">current_user</th>
</tr>
<tr valign="top">
<td align="left">postgres</td>
</tr>
</table>
<p>(1 row)<br />
</p>
```

Hi ha d'altres formats de sortida, com `aligned`, `unaligned`, `html` i `latex`. Per defecte, psql mostra el resultat en format `aligned`.

També tenim multitud de variables per ajustar els separadors entre columnes, el nombre de registres per pàgina, el separador entre registres, el títol de la pàgina HTML, etc. Vegem-ne un exemple:

```
postgres=# \pset format unaligned
Output format is unaligned.
postgres=# \pset fieldsep ','
Field separator is ",".
postgres=# select user, 1+2+3 as resultat;
current_user,resultat
postgres,6
(1 row)
```

Per a poder posar en pràctica els exemples de la resta d'aquest apartat, s'ha de processar el contingut del fitxer *arxiu.psql* tal com es transcriu a continuació.

> Des del terminal, situats en el directori en què hàgiu creat el fitxer, amb l'usuari per defecte postgres, executeu l'ordre `$ psql postgres -f arxiu.psql`.

El contingut del fitxer *arxiu.psql* és el següent:

```
--drop table productes;
--drop table proveidors;
--drop table preus;
--drop table guany;
create table productes (
partvarchar(20),
tipusvarchar(20),
especificacio varchar(20),
psuggeritfloat(6),
clauserial,
primary key(clau)
);
insert into productes (part,tipus,especificacio,psuggerit) values ('Processador
    ','2 GHz','32 bits',null);
insert into productes (part,tipus,especificacio,psuggerit) values ('Processador
    ','2.4 GHz','32 bits',35);
insert into productes (part,tipus,especificacio,psuggerit) values ('Processador
    ','1.7 GHz','64 bits',205);
insert into productes (part,tipus,especificacio,psuggerit) values ('Processador
    ','3 GHz','64 bits',560);
insert into productes (part,tipus,especificacio,psuggerit) values ('RAM','128MB
    ','333 MHz',10);
insert into productes (part,tipus,especificacio,psuggerit) values ('RAM','256MB
    ','400 MHz',35);
insert into productes (part,tipus,especificacio,psuggerit) values ('Disc Dur','
    80 GB','7200 rpm',60);
insert into productes (part,tipus,especificacio,psuggerit) values ('Disc Dur','
    120 GB','7200 rpm',78);
insert into productes (part,tipus,especificacio,psuggerit) values ('Disc Dur','
    200 GB','7200 rpm',110);
insert into productes (part,tipus,especificacio,psuggerit) values ('Disc Dur','
    40 GB','4200 rpm',null);
insert into productes (part,tipus,especificacio,psuggerit) values ('Monitor','
    1024x876','75 Hz',80);
insert into productes (part,tipus,especificacio,psuggerit) values ('Monitor','
    1024x876','60 Hz',67);

create table proveidors (
empresa varchar(20) not null,
credit bool,
efectiubool,
primary key(empresa)
);

insert into proveidors (empresa,efectiu) values ('Tecno-k', true );
insert into proveidors (empresa,credit) values ('Patito', true );
```

```
insert into proveidors (empresa,credit,efectiu) values ('Nacional', true, true
    );

create table guany(
vendavarchar(16),
factordecimal (4,2),
primary key (venda)
);

insert into guany values('al engros',1.05);
insert into guany values('al detall',1.12);

create table preus (
empresavarchar(20) not null,
clauint not null,
preufloat(6),
primary key (empresa, clau),
foreign key (empresa) references proveidors,
foreign key (clau)references productes
);

insert into preus values ('Nacional',001,30.82);
insert into preus values ('Nacional',002,32.73);
insert into preus values ('Nacional',003,202.25);
insert into preus values ('Nacional',005,9.76);
insert into preus values ('Nacional',006,31.52);
insert into preus values ('Nacional',007,58.41);
insert into preus values ('Nacional',010,64.38);
insert into preus values ('Patito',001,30.40);
insert into preus values ('Patito',002,33.63);
insert into preus values ('Patito',003,195.59);
insert into preus values ('Patito',005,9.78);
insert into preus values ('Patito',006,32.44);
insert into preus values ('Patito',007,59.99);
insert into preus values ('Patito',010,62.02);
insert into preus values ('Tecno-k',003,198.34);
insert into preus values ('Tecno-k',005,9.27);
insert into preus values ('Tecno-k',006,34.85);
insert into preus values ('Tecno-k',007,59.95);
insert into preus values ('Tecno-k',010,61.22);
insert into preus values ('Tecno-k',012,62.29);
```

Gestió de la base de dades

L'ordre següent informa sobre les bases de dades existents en l'SGBD:

```
postgres=# \l
List of databases
Name | Owner | Encoding
-----------+-----------+-----------
postgres | postgres | UTF8
template0 | postgres | UTF8
template1 | postgres | UTF8
(3 rows)
```

L'ordre `\c` permet connectar-se a una base de dades:

```
postgres=# \c postgres
You are now connected to database "postgres" as user "postgres".
```

La taula que conté la base de dades *postgres* es consulta mitjançant l'ordre `\d`:

```
postgres=# \d

List of relations

Schema | Name| Type| Owner

————————+————————————+————————+————————

public | guanys| table | postgres

public | preus| table | postgres

public | productes| table | postgres

public | productes_clau_seq | sequence | postgres

public | proveidors| table | postgres

(5 rows)
```

> L'ordre `\d` és útil per mostrar informació sobre l'SGBD: taules, índexs, objectes, variables, permisos, etc. Podeu obtenir totes les variants d'aquesta sentència introduint `\?` en l'intèrpret d'ordres.

Vegeu com es mostra una consulta de les columnes de cada una de les taules:

```
postgres=# \d proveidors

Table "public.proveidors"

Column | Type | Modifiers

————————+————————————+————————

empresa | character varying(20) | not null

credit | boolean |

efectiu | boolean |

Indexes:

"proveidors_pkey" PRIMARY KEY, btree (empresa)

Referenced by:

TABLE ""preus CONSTRAINT ""preus_empresa_fkey FOREIGN KEY (empresa) REFERENCES
    proveidors (empresa)
```

Per crear una nova base de dades, utilitzarem la sentència `create database`:

```
postgres=# create database prova;
```

Per eliminar una base de dades, utilitzarem la sentència `drop database`:

```
postgres=# drop database prova;
```

2. Objectes i col·leccions

El tipus objecte és un paquet cohesionat que descriu les regles que marquen el comportament de l'objecte. Disposa tant d'una interfície com d'una estructura; la primera descriu com es pot interactuar amb el tipus objecte, mentre que la segona defineix com s'emmagatzemen les dades a la base de dades. En el moment que es vol agrupar un nombre indefinit d'elements, si aquests elements són del mateix tipus, l'anomenem *col·lecció*.

2.1 Tipus de dades objecte

Els sistemes gestors de bases de dades inclouen un conjunt important de tipus de dades que s'adapten a moltes aplicacions i permeten als usuaris definir els seus propis tipus de dades. Els tipus d'usuari s'emmagatzemen en les files d'una taula i tenen un identificador únic (OID). Aquest identificador es pot utilitzar per referenciar altres tipus d'usuari i així representar relacions d'associació i d'agregació.

Un **tipus** defineix una estructura i un comportament comuns per a un conjunt de dades.

2.1.1 Estructura d'un tipus objecte

Un tipus objecte representa una entitat del món real i es compon dels elements següents:

1. Un **nom** que serveix per identificar el tipus objecte.
2. Uns **atributs** que modelitzen l'estructura i els valors de les dades del tipus. Cada atribut pot ser d'un tipus de dades **bàsic** o d'un tipus **d'usuari**.
3. Uns **mètodes** que són procediments o funcions escrits en el llenguatge PL/PgSQL (emmagatzemats en la BDOR), o escrits en un llenguatge orientat a objectes (emmagatzemats externament).

   L'estructura d'un tipus objecte consta de dues parts: especificació i cos.

L'**especificació** és la interfície del tipus objecte a les aplicacions. És on es declaren les estructures de dades o conjunt d'atributs i els mètodes necessaris per manipular les dades. El **cos** defineix els mètodes, és a dir, implementa l'especificació.

Els tipus objecte es poden interpretar com a plantilles per als objectes de cada tipus.

En la figura 2.1 podeu veure un exemple de definició de tipus de dades en pseudocodi, i com s'utilitza aquest tipus de dades per definir altres tipus objecte:

FIGURA 2.1. Definició de tipus de dades en pseudocodi

```
tipus Model
        atributs
                enter: potència;
                real: consum;
                real: cilindrada;
                real: acceleració;
fi Model.
tipus Vehicle
        atributs
                cadena_caràcters: marca;
                Model: model_vehicle;
                cadena_caràcters: color;
                enter: rodes;
        mètodes
                accelerar();
                frenar();
fi Vehicle.
```

En les especificacions es troba tota la informació que un client necessita per utilitzar els mètodes, d'aquí que es consideri una interfície operacional. És possible modificar el cos sense necessitat de modificar l'especificació i d'aquesta manera no afectar les aplicacions client.

Una de les característiques de l'orientació a objectes és que en una especificació de tipus objecte els atributs s'han de declarar abans que qualsevol dels mètodes. Per tant, si una especificació de tipus només declara atributs, el cos és innecessari perquè no hi ha mètodes a implementar. Tampoc no es poden declarar atributs en el cos del tipus objecte.

En l'especificació del tipus totes les declaracions són públiques, és a dir, visibles fora del tipus objecte. En canvi, el cos pot contenir declaracions privades, que defineixen mètodes necessaris per al funcionament intern del tipus objecte. L'àmbit de les declaracions privades és local en el cos de l'objecte.

En les **especificacions** del tipus objecte les declaracions són **públiques**, en canvi en el **cos** les declaracions són **privades**.

Per entendre aquesta estructura utilitzarem l'exemple de la figura 2.2, en què es defineix un tipus objecte i una sèrie d'operacions associades.

FIGURA 2.2. Exemple d'especificació d'un tipus objecte Rectangle

```
// Especificació dels atributs i mètodes
tipus Rectangle
        atributs
                enter: x;
                enter: y;
                enter: ample;
                enter: alt;
        mètodes
                calcularArea() -> enter;
                desplaçar(enter dx, enter dy);
fi Rectangle.
// Implementació dels mètodes
mètode calcularArea()
inici
        retorna(ample*alt);
fi
mètode desplaçar (enter dx, enter dy)
inici
        x <- x + dx;
        y <- y + dy;
fi
```

La implementació dels mètodes utilitza els atributs declarats dins el tipus objecte. En l'exemple anterior es calcula l'àrea del rectangle amb els valors dels atributs `ample` i `alt`, i el resultat es retorna com a paràmetre de sortida. En canvi, per calcular el desplaçament es modifiquen els atributs del tipus objecte amb els valors dels paràmetres d'entrada del mètode.

2.1.2 Components d'un tipus objecte

Un tipus objecte encapsula dades i operacions, per la qual cosa en l'especificació només es poden declarar atributs i mètodes, però no constants, excepcions, cursors o tipus. Com a mínim en un tipus objecte hi ha d'haver un atribut; pel que fa als mètodes, són opcionals.

Atributs

Un atribut es declara mitjançant un nom i un tipus. El nom ha de ser únic dins del tipus objecte, i el tipus pot ser qualsevol que suporti l'SGBD. Els tipus de dades bàsics són els següents: lògics, numèrics, de caràcters, de dates i de taules (vegeu la figura 2.3).

FIGURA 2.3. Tipus objecte amb diferents atributs

```
tipus Persona
        atributs
                cadena_caràcters: nom_cognom;
                enter: edat;
                data: data_naixement;
                cadena_caràcters: telèfon;
fi Persona.
```

El tipus objecte `Persona` es representa com un *tuple*, en què apareixen atributs de diferents tipus de dades.

Les estructures de dades poden arribar a ser molt complexes, fins al punt que el tipus d'un atribut pot ser un altre tipus objecte, anomenat *tipus objecte imbricat*. Això permet construir tipus d'objectes complexos a partir d'objectes simples. Alguns objectes com cues, llistes i arbres són dinàmics, creixen a mesura que s'utilitzen (vegeu la figura 2.4).

FIGURA 2.4. Tipus objecte complex Client

```
tipus Adreça
        atributs
                cadena_caràcters: carrer;
                cadena_caràcters: ciutat;
                cadena_caràcters: codi_postal;
fi Adreça.
tipus Client
        atributs
                enter: numeroCli;
                cadena_caràcters: nomCli;
                Adreça: adreçaCli;
        mètodes
                númeroClient() -> enter;
                dadesClient() -> cadena_caràcters;
fi Client.
```

El tipus objecte `Client` és un tipus objecte complex, ja que conté el tipus objecte `Adreça`com a atribut.En aquest cas podem veure un exemple de reutilització, ja que el tipus `Adreça` s'ha encapsulat com a tipus independent i aquest es pot utilitzar en la declaració de tipus.

Mètodes

Un **mètode** és un **subprograma declarat en una especificació de tipus**. El mètode no pot tenir el mateix nom que el tipus objecte ni el de cap dels seus atributs.

Els mètodes consten d'especificació i cos, i l'especificació consisteix en el nom del mètode, una llista opcional de paràmetres i, en el cas de les funcions, un tipus de retorn. Pel que fa al cos, el codi que conté s'executa per portar a terme una operació específica.

Per cada especificació de mètode hi ha d'haver el cos corresponent del mètode. En un tipus objecte, els mètodes poden fer referència als atributs i als altres mètodes sense qualificador.

Els mètodes es poden executar sobre els objectes del seu mateix tipus. Si `client_jove` és una variable PL/PgSQL que emmagatzema objectes del tipus `Client`, llavors `client_jove.numeroClient()` és un mètode que retorna el número del client emmagatzemat a `client_jove`.

Sobrecàrrega

Els mètodes del mateix tipus es poden sobrecarregar. Perquè això passi haurem d'utilitzar el mateix nom en diversos mètodes si els seus paràmetres formals són

diferents en número, ordre o tipus de dades. Quan s'invoca un dels mètodes, el PL/PgSQL troba el cos adequat comparant la llista de paràmetres actuals amb cadascuna de les llistes de paràmetres formals.

L'operació de sobrecàrrega no és possible en els casos següents:

1. Si els paràmetres formals es diferencien només en el mode.
2. Si les funcions només es diferencien en el tipus de retorn.

2.2 Definició de tipus d'objecte

Un cop sabem com volem emmagatzemar les dades, hem de crear l'estructura que les acollirà. Per realitzar aquest procés el PostgreSQL ens ofereix diferents tipus d'estructures: el tipus simple i el tipus compost. Aquestes estructures es poden utilitzar per definir el tipus d'un atribut dins d'una taula i per definir una taula d'objectes on els camps del tipus també són els camps de la taula.

Atès que el gestor de bases de dades seleccionat dóna suport parcial a orientació a objectes, hi ha alguns conceptes que no es podran desenvolupar, com la definició de mètodes en l'especificació d'un tipus.

2.2.1 Creació de tipus

PostgreSQL permet a l'usuari crear nous tipus de dades, el nom del tipus de les quals ha de ser diferent del nom de qualsevol tipus o domini que hi hagi en la base de dades. Preveu dos tipus de dades definides per l'usuari:

1. Un tipus de dades **simple**, per utilitzar en les definicions de columnes de les taules.
2. Un tipus de dades **compost**, per utilitzar com a tipus de retorn en les funcions definides per l'usuari.

   Ofereix les declaracions següents a l'hora de crear un tipus:

```
1  CREATE TYPE nom AS
2  ( nom_columnatipus [, ... ] )
3
4  CREATE TYPE nom (
5  INPUT = funcio_entrada,
6  OUTPUT = funcio_sortida
7  [ , RECEIVE = funcio_rebre ]
8  [ , SEND = funcio_enviament ]
9  [ , TYPMOD_IN = modificador_tipus_funcio_entrada ]
10 [ , TYPMOD_OUT = modificador_tipus_funcio_sortida ]
11 [ , ANALYZE = funcio_analitzar ]
12 [ , INTERNALLENGTH = { longitud_interna | VARIABLE } ]
13 [ , PASSEDBYVALUE ]
14 [ , ALIGNMENT = alineament ]
15 [ , STORAGE = emmagatzematge ]
16 [ , LIKE = tipus_com ]
17 [ , CATEGORY = categoria ]
18 [ , PREFERRED = preferit ]
```

```
[ , DEFAULT = per_defecte ]
[ , ELEMENT = element ]
[ , DELIMITER = delimitador ]
)
```

Els **tipus objecte** es poden definir com a **tipus simples** o com a**tipus compostos**.

Tipus simple

Per al tipus de dades simple, la definició és més complexa, ja que a les funcions se'ls ha d'especificar que tractaran amb aquest tipus. D'aquesta manera les funcions el podran utilitzar en operacions, assignacions, etc.

Definirem els tipus objecte com a tipus simples, indicant-los els atributs i definint els mètodes que implementaran les seves operacions.

Tractar amb el tipus de dades simple
Habitualment, les funcions que tractaran amb aquest tipus de dades s'escriuran en llenguatge C.

A tall d'exemple, crearem el tipus `nombre complex`, tal com fa la documentació de PostgreSQL. En primer lloc, hem de definir l'estructura en què emmagatzemarem el tipus:

```
TYPEDEF STRUCTComplex {
DOUBLEx;
DOUBLEy;
} Complex;
```

Després, les funcions que el rebran o tornaran:

```
PG_FUNCTION_INFO_V1(complex_in);
Datum
complex_in(PG_FUNCTION_ARGS) {
char*str = PG_GETARG_CSTRING(0);
double x, y;
Complex *result;

if (sscanf(str, " ( %lf , %lf )", &x, &y) != 2)
ereport(ERROR,(errcode(ERRCODE_INVALID_TEXT_REPRESENTATION), errmsg"(invalid
    input syntax for complex: "\%s""\, str)));

result = (Complex *) palloc(sizeof(Complex));
result->x = x;
result->y = y;
PG_RETURN_POINTER(result);
}

PG_FUNCTION_INFO_V1(complex_out);
Datum
complex_out(PG_FUNCTION_ARGS) {
Complex *complex = (Complex *) PG_GETARG_POINTER(0);
char*result;
result = (char *) palloc(100);
snprintf(result, 100, "(%g,%g)", complex->x, complex->y); PG_RETURN_CSTRING(
    result);
}
```

Ara estem en condicions de definir les funcions, i el tipus:

```
CREATE FUNCTION complex_in(cstring)
RETURNS complex
AS 'filename'
LANGUAGE c IMMUTABLE STRICT;
CREATE FUNCTION complex_out(complex)
RETURNS cstring
AS 'filename'
LANGUAGE c IMMUTABLE STRICT;
CREATE TYPE complex (
internallength = 16,
input = complex_in,
output = complex_out,
alignment = double
);
```

Tipus compost

La primera forma de `CREATE TYPE` crea un tipus compost. El tipus compost s'especifica mitjançant una llista de noms d'atributs i tipus de dades, i representa l'estructura d'una fila o registre. És el mateix que el tipus fila d'una taula, però evita crear taules dins la base de dades quan només es vol definir un tipus.

El PostgreSQL permet que els tipus compostos s'utilitzin com a tipus simples. Per exemple:

1. Una columna d'una taula es pot declarar com a tipus de dades.
2. En els mètodes o funcions es poden utilitzar com a argument o com a tipus de retorn de la funció.

Tipus d'objectes

Els tipus objecte es poden definir com a tipus simples o com a tipus compostos.

Definirem els tipus objecte com a tipus compostos, encapsulant els atributs del mateix o diferent tipus sota un nom únic i sense mètodes. A continuació podem veure un exemple de definició de tipus objecte:

```
CREATE TYPE complex AS (
r double precision,
i double precision
);
```

La sintaxi de definició d'un tipus objecte (`CREATE TYPE ... AS`) és comparable amb la sintaxi de definició d'una taula (`CREATE TABLE`), excepte que només els noms i tipus de camp es poden especificar, **no admetrestriccions.**Tingueu en compte que la paraula clau AS és essencial; sense aquesta, el sistema produirà un error de sintaxi.

Una vegada definits els tipus, aquests es poden utilitzar per definir nous tipus i taules que emmagatzemen objectes d'aquest tipus, o per definir el tipus dels atributs d'una taula.

Agafem com a exemple el tipus objecte, `inventari_element`. Aquest conté tres atributs de diferents tipus, que es definiran en les columnes `nom`, `proveidor_id` i `preu`:

```
CREATE TYPE inventari_element AS (
nom text,
proveidor_id integer,
preu numeric
);
```

Aquest tipus objecte el podem utilitzar per definir el tipus d'un atribut dins d'una taula, és a dir, per definir la columna de la taula, amb la sentència SQL `CREATE TABLE`:

```
CREATE TABLE a_mà (
element inventari_element,
quantitat integer
);
```

També podem utilitzar el tipus objecte com a paràmetre d'una funció:

```
CREATE FUNCTION preu_extensió (inventari_element, integer) RETURNS numeric
```

La sentència SQL `DROP TABLE` permet eliminar taules:

```
DROP TABLE a_mà;
```

Les taules creades en PostgreSQL poden incloure diverses columnes ocultes que emmagatzemen informació sobre l'identificador de transacció en què poden estar implicades, la localització física del registre dins de la taula (per localitzar-la molt ràpidament) i, els més importants, l'OID i el TABLEOID.

Aquestes últimes columnes estan definides amb un tipus de dades especial anomenat **identificador d'objecte** (*object identifier,* OID) que s'implementa com un enter positiu de 32 bits. Quan s'insereix un nou registre en una taula s'hi pot assignar un número consecutiu com a `OID`, i el `TABLEOID` de la taula que li correspon.

> `L'OID` no s'assigna per defecte a les taules creades per l'usuari: s'ha d'especificar quan es crea la taula amb l'ordre `WITH OIDS` o activant la variable de configuració `default_with_oids`.

En la programació orientada a objectes, el concepte d'OID és de vital importància, ja que es refereix a la **identitat pròpia**de l'objecte, la qual cosa el diferencia dels altres objectes.

Si ho apliquem en l'exemple anterior, definim la taula de la manera següent:

```
CREATE TABLE a_mà (
element inventari_element,
quantitat integer
) WITH OIDS;
```

Per observar les columnes ocultes hi hem de fer referència específicament en l'ordre de consulta `select`, indicant els camps `oid` i `tableoid` com també la resta de camps de la taula `a_mà`. Veurem amb més deteniment les ordres de consulta i

inserció en apartats posteriors, però de moment suposem que hem inserit dades en la taula anterior amb la sentència SQL `insert into`. L'exemple següent mostra un possible retorn d'una consulta d'aquest tipus:

```
oid | tableoid | nom | proveidor_id | preu | quantitat

--------+----------+-----------+--------------+------+-----------

17242 | 17240 | galetes | 42 | 1.99 | 1000

(1 rows)
```

Aquestes columnes s'implementen per servir d'identificadors en la realització d'enllaços des d'altres taules.

Dominis

La creació d'un domini en PostgreSQL consisteix en un tipus, definit per l'usuari o inclòs en l'SGBD, més un conjunt de restriccions. A diferència del tipus objecte, en la definició de dominis es poden especificar restriccions. La sintaxi la podem veure en els exemples següents:

```
CREATE DOMAIN country_code char(2) NOT NULL;
CREATE DOMAIN complex_positiu complex NOT NULL CHECK
(complex_major(value,(0,0)))
```

1. El primer domini s'ha creat basant-se en un tipus definit pel sistema, `country_code`, en què l'única restricció és la seva longitud.
2. En el segon domini hauríem d'haver definit l'operador `complex_major` que rebés dos nombres complexos i indiqués si el primer és més gran que el segon.

2.2.2 Taules d'objectes

Una taula d'objectes és una classe especial de taula que emmagatzema un objecte en cada fila i que facilita l'accés als atributs d'aquests objectes com si fossin columnes de la taula. Prenem com a exemple el tipus `inventari_element`. Aquest conté tres atributs de diferent tipus, que es definiran en les columnes `nom`, `proveidor_id` i `preu`:

```
CREATE TYPE inventari_element AS (
nom text,
proveidor_id integer,
preu numeric
);
```

Aquest tipus el podem utilitzar per definir el tipus d'una taula, és a dir, indicar de quin tipus seran els objectes que s'emmagatzemaran a la taula. Utilitzarem la sentència SQL `CREATE TABLE/OF`**:**

```
CREATE TABLE a_mà OF inventari_element (
PRIMARY KEY (nom),
preu WITH OPTIONS DEFAULT 10
);
```

En crear la taula associada a un tipus, aquesta pren l'estructura del tipus, i fa que les columnes quedin determinades pels atributs del tipus. No obstant això, l'ordre `CREATE TABLE` pot afegir valors i restriccions a la taula.

La taula està lligada al seu tipus, de manera que si s'elimina el tipus també s'elimina la taula associada, per exemple amb la sentència `DROP TYPE ... CASCADE`.

Quan creem el tipus `inventari_element` no s'accepten atributs de tipus `serial`. La clau primària ja s'indica en crear la taula.

Trobareu més informació sobre l'ús del tipus `serial` en l'apartat "Herència i identificadors".

Suposem que sol·licitem a la base de dades que ens mostri totes les columnes dels registres de la taula `a_mà`. A continuació podem veure el retorn d'una consulta d'aquesta taula. Es pot observar com les columnes de la taula són els atributs del tipus `inventari_element`:

```
nom| proveidor_id | preu
———————+———————————+———————
galetes|42| 1.99
(1 rows)
```

2.3 Herència

PostgreSQL ofereix com a característica particular l'herència entre taules, que permet definir una taula que hereti (atributs i mètodes) d'una altra prèviament definida, segons la definició d'herència que hem vist anteriorment.

Utilitzem la taula següent, anomenada `persona`, especificant que volem treballar amb OID:

```
CREATE TABLA persona (
nom varchar (30),
adreça varchar (30)
) WITH OIDS;
```

A partir d'aquesta, declarem la taula `estudiant` com a derivada de `persona`:

```
CREATE TABLE estudiant (
carrera varchar (50),
grup char,
grau int
) INHERITS (persona);
```

En la taula `estudiant` es defineixen les columnes `carrera`, `grup` i `grau`, però si sol·licitem informació de l'estructura de la taula observem que també inclou les columnes definides en la taula `persona`, `nom` i `adreça`.

En aquest cas, la taula `persona` l'anomenem *pare* i la taula `estudiant`, *fill*.

Cada registre de la taula `estudiant` contindrà cinc valors perquè té cinc columnes, tres de la taula `estudiant` i dues que hereta de la taula `persona`.

L'herència no solament permet que la taula *fill* contingui les columnes de la taula *pare*, sinó que estableix una relació conceptual entre aquestes.

Suposem que prèviament hem inserit les dades següents en la taula `estudiant`:

1. un estudiant de nom: 'Joan Miquel',
2. amb adreça: 'Treboles 21',
3. que estudia la carrera: 'Eng. computacio',
4. pertany al grup 'A' i grau 3.

   Si a continuació fem una consulta del contingut de la taula `estudiant` se'ns mostrarà un sol registre. No s'hereten les dades, únicament els camps de l'objecte –els atributs–:

```
nom | adreca | carrera | grup | grau

——————————+————————————— +————————————————————+———————+———————

Joan Miquel | Treboles 21 | Eng. Computació | A | 3

(1 rows)
```

I si fem una consulta de la taula `persona`, aquesta mostrarà un nou registre. Podem veure el retorn de la consulta:

```
nom | adreca
————————————— + ————————————
Joan Miquel | Treboles 21
(1 rows)
```

El registre que es mostra és el que es va inserir en la taula `estudiant`; tanmateix, l'herència defineix una relació conceptual en la qual un estudiant és una persona. Per tant, en consultar quantes persones estan registrades en la base de dades, tots els estudiants s'inclouen en el resultat.

No és possible esborrar una taula *pare* si no s'esborren primer les taules *fill*.

Com és lògic, en esborrar la fila del nou estudiant que hem inserit aquesta s'esborra de les dues taules, tant si l'esborrem des de la taula `persona` com si l'esborrem des de la taula `estudiant`.

2.3.1 Herència i identificadors

Els OID (identificadors d'objectes) permeten que es diferenciïn els registres de totes les taules, encara que siguin heretades: el nostre estudiant tindrà el

mateix OID en les dues taules, ja que es tracta d'una única instància de la taula `estudiant`que hereta de la taula`persona`:

Retorn de la consulta de la taula estudiant :

```
oid | tableoid | nom | adreca | carrera | grup | grau

———————+——————————+———————————— +—————————————+——————————————————+———————+———————

**16434 **| 16431 | Joan Miquel | Treboles 21 | Eng. Computació | A | 3

(1 rows)
```

Retorn de la consulta de la taula persona:

```
oid | tableoid | nom | adreca

————————+————————————+—————————————————+—————————————————

**16434 | **16427 | Joan Miquel | Treboles 21

(1 rows)
```

Com que no es recomana l'ús d'OID en bases de dades gaire grans, i s'ha d'incloure explícitament en les consultes per examinar-ne el valor, és convenient utilitzar una seqüència compartida per a *pares* i tots els seus *descendents* si es requereix un identificador.

En el PostgreSQL, una alternativa per no utilitzar els OID és crear una columna de tipus `serial` en la taula *pare*, així serà heretada en la taula *fill*. El tipus `serial` defineix una seqüència de valors que s'anirà incrementant de manera automàtica i, per tant, constitueix una bona manera de crear claus primàries, igual que el tipus `AUTO_INCREMENT` en MySQL. Tornarem a crear la taula `persona` amb l'atribut `id` de tipus `serial`:

```
CREATE TABLA persona (
id serial,
nom varchar (30),
adreça varchar (30)
);
```

En definir un tipus serial, hem creat implícitament una seqüència independent de la taula. Una vegada la taula estigui creada, el sistema ens notificarà aquest missatge, indicant que la columna `id` es considera clau primària:

```
NOTICE: CREATE TABLE will create implicit sequence 'persona_id_seq' for SERIAL
     column 'persona.
NOTICE: CREATE TABLE / UNIQUE will create implicit index 'persona_id_key' for
     table 'persona'
```

Creem novament la taula `estudiant` heretant els atributs i mètodes de `persona`:

```
CREATE TABLE estudiant (
carrera varchar (50),
grup char,
grau int
) INHERITS (persona);
```

`Estudiant` heretarà la columna `id` i al ser de tipus serial el valor s'incrementarà utilitzant la mateixa seqüència. Suposem que inserim en la taula alguns registres d'exemple, ometent el valor per a la columna `id`:

- Inserim a la taula persona: ('Josep Claramunt' , 'Montoliu 12');
- Inserim a la taula persona: ('Lluís Arnau', 'Barcelona 3');
- Inserim a la taula estudiant: ('Anna Guillen' , 'Tarragona 19', 'Psicologia', 'C', 2);

Si fem una consulta sobre la taula `estudiant`, veurem que aquesta contindrà un sol registre, però el seu identificador serà el número 3:

```
id | nom | adreca | carrera | grup | grau

————+————————————+————————————+————————————+————————+————————

3 | Anna Guillen | Tarragona 19 | Psicologia | C | 2

(1 row)
```

Tots els registres de la taula `persona` segueixen una mateixa seqüència sense importar si són *pares* o *fills*:

```
id | nom | adreca
————+————————————————+————————————
1 | Josep Claramunt | Montoliu 12
2 | Lluis Arnau | Barcelona 3
3 | Anna Guillen | Tarragona 19
(3 row)
```

L'herència és útil per definir taules que de manera conceptual mantenen elements en comú, però també requereixen dades que les fan diferents. Un dels elements que convé definir com a comuns són els identificadors de registre.

2.3.2 Herència en taula d'objectes

Si tenim el tipus següent d'objecte `estudiant`:

```
CREATE TYPE estudiant AS (
id int,
nom varchar (20),
adrecavarchar (30),
carrera varchar (50),
grup varchar(5),
grau int
);
```

Es pot definir una taula per emmagatzemar els estudiants del curs 2012 i una altra per emmagatzemar els estudiants que cursaran pràctiques durant l'any 2012 de la manera següent:

```
CREATE TABLE estudiants_2012 OF estudiant (
PRIMARY KEY (id)
);
CREATE TABLE estudiants_en_pràctiques (
empresa_pràctiques varchar(30),
) INHERITS (estudiants_2012);
```

La diferència entre la primera i la segona taula és que la primera emmagatzema objectes de tipus `estudiant`, i la segona és una especialització de la taula anterior. És a dir, la segona taula hereta les característiques de la primera i afegeix les seves pròpies.

Exemple d'herència en taula d'objectes

Podríem executar una de les instruccions següents. La taula estudiants_2012 es considera una taula amb diverses columnes en què els valors són els especificats.

En la taula estudiants_en_pràctiques inserim:

```
(1000,'Roger Llorac Arnau', 'Castalia 33', 'Psicologia', 'C', 3,
    'Software Catalunya');
```

En incloure el registre a la taula fillestudiants_en_pràctiques, aquest s'afegirà a les dues taules, ja que en heretar els atributs compleix totes dues especificacions: és alumne en el curs 2012 i fa pràctiques a l'empresa.

En la taula estudiants_2012 inserim:

```
(1005,'Lluís Boella Domenec', 'Montoliu 55', 'Antropologia', 'B',
     2);
```

En aquest cas el registre s'inclou només en la taula pareestudiants_2012, ja que no hi ha cap especialització, no s'especifica l'empresa de pràctiques.

Les regles d'integritat i de clau primària, com també la resta de propietats que es defineixin sobre una taula, només afecten els objectes d'aquesta taula, és a dir, no es refereixen a tots els objectes del tipus assignat a aquesta.

2.4 Identificadors, referències i restriccions

Com ja hem vist anteriorment, en els models orientats a objectes, hi ha el concepte d'identificador d'objecte (en anglès *object identifier,* OID), que representa la identitat de l'objecte, única per cada un.

Els OID s'encarreguen de fer referència a un objecte des d'un altre, creant d'aquesta manera les relacions entre objectes.

2.4.1 Identificadors

Hem vist que el PostgreSQL inclou en les taules creades els identificadors d'objecte, aquests es poden utilitzar per referenciar altres tipus d'usuari i així representar relacions d'associació i d'agregació entre objectes.

També som conscients que en la programació orientada a objectes, el concepte d'OID és de vital importància, ja que es refereix a la identitat pròpia de l'objecte, la qual cosa el diferencia dels altres objectes. A continuació veurem un exemple de la utilització d'OID per enllaçar dues taules.

Ens basem en la taula `persona` dels exemples anteriors:

```
CREATE TABLE persona (
nom varchar (15),
adreça varchar (30)
) WITH OIDS;
```

Definim un nou tipus i una nova taula per emmagatzemar els telèfons:

```
CREATE TYPE telefon AS (
tipusvarchar (10),
numero varchar (20),
propietarioid
);
CREATE TABLE llista_telefons OF telefon (
PRIMARY KEY(propietari)
) ;
```

La taula `llista_telefons` inclou la columna `propietari` de tipus OID, que emmagatzemarà la referència als registres de la taula `persona`. Suposem que agreguem dos telèfons a '`Joan Miquel`'; per fer la inserció utilitzem el seu OID, que és 16434. En la taula `llista_telefons`inserim les dades següents:

```
( 'mòbil' , '12345678', 16434 );
( 'casa' , '987654', 16434 );
```

Les dues taules estan vinculades per l'OID de persona, en aquest cas el `propietari`, que com podem comprovar és el mateix:

```
-- Consulta de la taula llista_telefons
tipus | numero | propietari
----------- +---------------- +----------------
mòbil | 12345678 | 16434
casa | 987654 | 16434
(2 rows)
```

Consulteu l'apartat "Criteris de selecció" per obtenir més informació sobre l'operació `join`.

Hi ha una operació que ens permet unir les dues taules, `join`.En aquest cas uneix `llista_telefons` i `persona`, utilitzant la igualtat de les columnes`llista_telefons.propietari` i `persona.oid`.El resultat de la consulta anterior seria:

```
tipus | numero | propietari | nom | adreca

----------- + ---------------- + ---------------- +
    ----------------+----------------

mòbil | 12345678 | 16434| Joan Miquel | Treboles 21

casa | 987654 | 16434 | Joan Miquel | Treboles 21

(2 rows)
```

Els OID del PostgreSQL presenten algunes deficiències:

1. Tots els OID d'una base de dades es generen a partir d'una única seqüència centralitzada, la qual cosa provoca que, en bases de dades amb molta activitat d'inserció i eliminació de registres, el comptador de 4 bytes es desbordi i pugui lliurar OID ja lliurats. Això passa, per descomptat, amb bases de dades molt grans.
2. En les taules enllaçades mitjançant OID no s'obté cap avantatge, en termes d'eficiència, d'utilitzar operadors de composició respecte a una clau primària convencional.
3. Els OID no milloren el rendiment. En realitat, són una columna amb un nombre enter com a valor.

Els desenvolupadors de PostgreSQL proposen l'alternativa següent per utilitzar OID d'una manera absolutament segura:

1. Crear una restricció de taula perquè l'OID sigui únic, almenys en cada taula. L'SGBD anirà incrementant de manera seqüencial l'OID fins a trobar-ne un sense usar.
2. Usar la combinació `OID-TABLEOID` si es necessita un identificador únic per a un registre vàlid en tota la base de dades.

2.4.2 Referències

Els identificadors únics assignats als objectes que s'emmagatzemen en una taula permeten que aquests es puguin referenciar des dels atributs d'altres objectes.

Si volem que un atribut emmagatzemi una referència a un objecte del tipus definit, l'ordre que utilitzem és `REFERENCES`, que implementa una relació d'associació entre els dos tipus objecte.

En l'exemple següent apliquem l'ordre `REFERENCES` a un atribut:

```
1  CREATE TYPE tipus_persona AS (
2  dni varchar (9),
3  nom  varchar (30),
4  adreça varchar (30)
5  );
6  CREATE TABLA persona OF tipus_persona (
7  PRIMARY KEY (dni)
8  );
9  CREATE TYPE telefon (
10 tipusvarchar (10),
11 numero varchar (20)
12 );
13 CREATE TABLE llista_telefons (
14 tlftelefon,
15 propietari varchar(9) REFERENCES persona
16 ) ;
```

Quan s'afegeixin les dades d'un nou telèfon, el PostgreSQL verificarà que el valor de propietari faci referència a una persona, i en cas contrari emetrà un missatge d'error.

2.4.3 Restriccions

Les restriccions permeten especificar condicions que hauran de complir les taules o columnes per mantenir la integritat de les dades. Algunes restriccions vénen imposades pel model concret que s'implementa, mentre que altres tenen l'origen en les regles del client o els valors que poden prendre alguns camps, entre d'altres.

'Null' i 'Not Null'

Sovint el valor d'una columna és desconegut, no és aplicable o no existeix. En aquests casos, els valors zero, cadena buida o fals són inadequats, per la qual cosa utilitzarem `null` per especificar l'absència de valor. En definir la taula podem indicar quines columnes podran contenir valors nuls i quines no.

```
1 CREATE TABLE persona (
2 nom varchar(40) not null,
3 treball varchar(40) null,
4 correu varchar(20)
5 );
```

El nom d'una persona no pot ser nul, però és possible que la persona no tingui correu, ja que en no especificar una restricció `not null` s'assumeix que la columna pot contenir valors nuls.

'Unique'

Aquesta restricció s'utilitza quan no volem que els valors que conté una columna es puguin duplicar.

```
1 CREATE TABLE persona (
2 nom varchar(40) not null,
3 conjugevarchar(40) unique
4 );
```

Així, `conjuge` no pot contenir valors duplicats: no hem de permetre que dues persones tinguin simultàniament el mateix cònjuge.

'Primary key'

Aquesta restricció especifica la columna o columnes que escollim com a clau primària. Hi pot haver múltiples columnes `unique`, però només hi ha d'haver una clau primària. Els valors que són únics poden servir per identificar una fila de la taula de manera unívoca, per la qual cosa se les anomena *claus candidates*.

```
CREATE TABLE persona (
dni varchar(10) PRIMARY KEY,
conjugevarchar(40) UNIQUE
);
```

En definir una columna com a `primary key`, es defineix implícitament com a `unique`. El `dni` no solament és únic sinó que també l'utilitzarem per identificar les persones.

Referències i 'foreign key'

En el model relacional, establim les relacions entre entitats mitjançant la inclusió de claus foranes en altres relacions. El PostgreSQL i l'SWL ofereixen mecanismes per expressar i mantenir aquesta integritat referencial. En l'exemple següent, els `MeusAnimals` tenen com a propietari una `persona`:

```
CREATE TABLE MeusAnimals (
nom varchar(20),
propietari varchar(10) REFERENCES persona
);
```

Una referència per defecte és una clau primària, per la qual cosa `propietari` es refereix implícitament al `dni`de `persona`. Quan es capturen les dades d'un nou animal, el PostgreSQL verifica que el valor de `propietari` faci referència a un `dni` que existeixi de `persona`, en cas contrari emetrà un missatge d'error. No es permet assignar un nou animal a una persona que no existeix.

També és possible especificar a quina columna de la taula es fa referència:

```
CREATE TABLE MeusAnimals (
nom varchar(20),
propietari varchar(10) REFERENCES persona(dni)
);
```

O el seu equivalent:

```
CREATE TABLE MeusAnimals (
nom varchar(20),
propietari varchar(10),
FOREIGN KEY propietari REFERENCES persona(dni)
);
```

Es podria donar el cas que la clau primària de la taula referenciada tingués més d'una columna. En aquest cas, la clau forana també hauria d'estar formada pel mateix nombre de columnes:

```
CREATE TABLE t1 (
a integer PRIMARY KEY,
binteger,
c integer,
FOREIGN KEY (b,c) REFERENCES other_table (c1,c2)
);
```

Si no s'especifica una altra acció, la persona que tingui un animal no es podrà eliminar per omissió, perquè l'animal es quedaria sense propietari. Per poder

eliminar una persona, abans s'han d'eliminar els animals que pugui tenir. Aquest comportament no sembla el més adequat, així que per modificar aquest comportament disposem de les **regles d'integritat referencial** del llenguatge SQL, que PostgreSQL també suporta.

En l'exemple següent es permet que en eliminar una persona els animals quedin sense propietari:

```
CREATE TABLE MeusAnimals (
propietari varchar(10) REFERENCES persona ON DELETE SET NULL
);
```

Amb la clàusula `on delete`es poden especificar les accions següents:

1. `set null`. La referència pren el valor `null`. Si s'elimina `Persona`, el seu `Animal` es quedarà sense propietari.
2. `set default`. La referència pren el valor per omissió.
3. `Cascade`. L'acció s'efectua en cascada. Si s'elimina `Persona` automàticament s'eliminen els seus animals.
4. `Restrict`. No permet esborrar el registre. No es pot eliminar una `Persona` que tingui animals. Aquesta és l'acció que es pren per omissió.

Si es modifica la clau primària de la taula referenciada, es disposa de les mateixes accions que en el cas anterior, que especificarem amb la clàusula `ON UPDATE`.

La restricció`Check` fa l'avaluació prèvia d'una expressió lògica quan s'intenta efectuar una assignació. Si el resultat és cert, accepta el valor per a la columna; en cas contrari, emet un missatge d'error i rebutja el valor.

```
CREATE TABLE Persona (
edatint CHECK (edat > 10 and edat < 80),
correuvarchar(20) CHECK (correu Autoria desconeguda2012–08–28T11:18:35Al
    maquetador: aquesta és una expressió aleatòria d'error. Si dóna problemes
    en l'exportació parlem–ne, doncs probablement es puguin substituir els car
    àcters reservats per d'altres que no facin petar l'exportador.~ '.+@.+\..+
    ' ),
ciutatvarchar(30) CHECK (ciutat <> '')
);
```

S'han restringit els valors que s'acceptaran en la columna de la manera següent:

1. `edat` ha de tenir un valor entre onze i setanta-nou anys.
2. `ciutat` no ha de ser una cadena buida.
3. `correu` ha de contenir una arrova.

Qualsevol d'aquestes restriccions pot tenir nom, de manera que es facilita la referència a les restriccions especifiques per esborrar-les, modificar-les, etc. Per assignar un nom a una restricció ho farem de la manera següent:

```
CONSTRAINT nom_de_restriccio <restriccio>
```

Restriccions de taula

Quan les restriccions s'indiquen després de les definicions de les columnes, algunes d'aquestes poden quedar afectades simultàniament. Llavors parlem de *restriccions de taula*:

```
CREATE TABLE persona (
dni int,
nom varchar (30),
conjuge varchar(30),
cap int,
correu varchar(20),
PRIMARY KEY (dni),
UNIQUE (conjuge),
FOREIGN KEY (cap) REFERENCES persona,
CHECK (correu ~ '@' )
);
```

Aquesta notació permet que la restricció pugui abarcar diverses columnes.

```
CREATE TABLE curs (
materia varchar(30),
grup varchar (4),
dia int,
hora time,
aula int,
PRIMARY KEY (materia,grup),
UNIQUE (dia, hora, aula)
);
```

Un curs s'identifica pel grup i la matèria, i dos cursos no poden estar a la mateixa aula el mateix dia i a la mateixa hora.

Igual que amb la restricció de columna, a les restriccions de taula se'ls pot assignar un nom:

```
CREATE TABLE persona (
dniint,
nom varchar (30),
conjuge varchar(30),
cap int,
correu varchar(20),
CONSTRAINT identificador PRIMARY KEY (dni),
CONSTRAINT monogàmia UNIQUE (conjuge),
CONSTRAINT un_cap FOREIGN KEY (cap) REFERENCES persona,
CHECK (correu ~ '@' )
);
```

La sentència `ALTER TABLE` permet afegir (`ADD`) o treure (`DROP`) restriccions que ja s'han definit:

```
ALTER TABLE persona DROP CONSTRAINT monogàmia
ALTER TABLE ADD CONSTRAINT monogàmia UNIQUE (conjuge);
```

2.4.4 Indexació

El PostgreSQL crea un índex per a les claus primàries de totes les taules. Quan necessitem crear índexs addicionals, utilitzarem l'expressió següent:

```
1 CREATE INDEX taula_camp_index ON taula (camp);
```

Prenent com a exemple la taula `persona`, quan es fa una consulta a la taula segons el valor d'un camp concret, el sistema ha d'escanejar-la sencera, fila per fila, per trobar totes les entrades coincidents. Aquest és un mètode ineficient, ja que no discrimina el fet que hi hagi poques o moltes entrades: trigarà molt a fer la consulta a l'haver de fer un registre exhaustiu. Però si al sistema se li indica que mantingui un índex en una de lescolumnes a mode d'identificador, es pot utilitzar un mètode més eficient per localitzar els registres coincidents. Per exemple, només hauria d'avançar uns nivells de profunditat en un arbre de cerca.

Es pot utilitzar l'ordre següent per crear un índex en la columna `dni`:

```
1 CREATE INDEX persona_dni_index ON persona (dni);
```

Per eliminar un índex, s'utilitza l'ordre següent:

```
1 DROP INDEX persona_dni_index;
```

2.5 Tipus de dades col·lecció

Per poder implementar relacions 1:N, l'SGBD ha de permetre definir *tipus col·lecció*. Un tipus col·lecció està format per un nombre indefinit d'elements, tots del mateix tipus. D'aquesta manera, és possible emmagatzemar en un atribut un conjunt de *tuples* en forma d'*array* o en forma de taula imbricada.

Relació 1:N

Representa la interrelació entre dos entitats dins del model entitat-relació, la cardinalitat un a varis (1:N) especifica el tipus de relació.

Les **col·leccions** emmagatzemen objectes del mateix tipus.

El tipus de dades *array* és una de les característiques especials del PostgreSQL. Permet l'emmagatzematge de més d'un valor del mateix tipus en la mateixa columna.

El PostgreSQL permet que les columnes d'una taula es defineixin com a taules multidimensionals de longitud variable. Es poden crear taules de tipus simples definits per l'usuari, tipus enumerats o tipus compostos, però les taules de dominis encara no són compatibles.

Per il·lustrar l'ús de tipus *array*, creem la taula següent:

```
CREATE TYPE telefon AS (
tipusvarchar (10),
numero varchar (20)
);
CREATE TABLE llista_telefons (
nom varchar (10),
tlftelefon[]
);
```

La taula `llista_telefons` conté un atribut de tipus *array* que emmagatzemarà un conjunt d'objectes de tipus `telefon`.

La sintaxi permet especificar la mida exacta del tipus *array*, per exemple:

```
CREATE TABLE llista_telefons (
nom varchar (30),
tlftelefon[10]
);
```

No obstant això, definir la mida dins la sentència `CREATE TABLE` no afecta el comportament en temps d'execució.

Una sintaxi alternativa que s'ajusta a l'estàndard SQL utilitza la paraula clau `ARRAY` per a taules d'una dimensió. Si ho apliquem a l'exemple anterior es podria definir de la manera següent:

```
tlftelefon ARRAY
tlf telefon ARRAY[10]
```

Els valors de l'*array* s'escriuen sempre entre claudàtors, i com passa amb qualsevol columna quan no s'especifica el contrari, s'accepten valors nuls. Vegeu un exemple de com s'inserieix un registre en la taula `llista_telefons`amb un objecte de tipus `telefon`:

```
INSERT INTO llista_telefons VALUES ('Agenda feina', ARRAY[ROW('mobil','
    623533123')::telefon]);
```

Com veurem més endavant, en la sentència `INSERT INTO` s'utilitza l'ordre `ROW`, perquè el tipus de l'atribut `tlf` és un tipus compost.

Un cop tenim dades a `llista_telefons` podem executar algunes consultes sobre la taula. Per exemple, podem mostrar les dades que s'han inserit anteriorment. Ho faríem de la manera següent:

```
SELECT nom, tlf[1] FROM llista_telefons;
nom |tlf
————————————+————————————————————
Agenda feina| (mobil, 623533123)
```

PostgreSQL ofereix un conjunt d'operadors i funcions compatibles amb el tipus *array*. Per exemple, si utilitzem la funció `array_dims()` podem conèixer les dimensions d'una taula:

```
SELECT array_dims(tlf) FROM llista_telefons;
[1:1]
```

Aquests operadors de tipus *array* permeten comparar i concatenar els elements de les taules. La taula 2.1 mostra els tipus d'operadors que podem trobar.

TAULA 2.1. Tipus d'operadors

Operador	Descripció	Exemple	Resultat
=	igual	`ARRAY[1.1,2.1,3.1]::int[] = ARRAY[1,2,3]`	t
<>	diferent	`ARRAY[1,2,3] <> ARRAY[1,2,4]`	t
<	més petit que	`ARRAY[1,2,3] < ARRAY[1,2,4]`	t
>	més gran que	`ARRAY[1,4,3] > ARRAY[1,2,4]`	t
<=	més petit o igual que	`ARRAY[1,2,3] <= ARRAY[1,2,3]`	t
>=	més gran o igual que	`ARRAY[1,4,3] >= ARRAY[1,4,3]`	t
@>	conté	`ARRAY[1,4,3] @> ARRAY[3,1]`	t
<@	és continguda per	`ARRAY[2,7] <@ ARRAY[1,7,4,2,6]`	t
&&	superposició (tenen elements en comú)	`ARRAY[1,4,3] && ARRAY[2,1]`	t
||	concatenació taula-a-taula	`ARRAY[1,2,3] || ARRAY[4,5,6]`	{1,2,3,4,5,6}
||	concatenació element-a-taula	`3 || ARRAY[4,5,6]`	{3,4,5,6}
||	concatenació taula-a-element	`ARRAY[4,5,6] || 7`	{4,5,6,7}

També podem trobar funcions que faciliten a l'usuari la interacció amb les taules. En la taula 2.2 podeu veure les diferents funcions que hi ha.

TAULA 2.2. Funcions

Funció	Tipus Retorn	Descripció	Exemple	Resultat
`array_append(anyarray, anyelement)`	anyarray	Afegeix un element al final de la taula.	`array_append (ARRAY[1,2],3)`	{1,2,3}
`array_cat(anyarray, anyarray)`	anyarray	Concatena dues taules.	`array_cat(ARRAY[1,2,3], ARRAY[4,5])`	{1,2,3,4,5}
`array_ndims(anyarray)`	int	Retorna el nombre de dimensions de la taula.	`array_ndims(ARRAY [[1,2,3],[4,5,6]])`	2
`array_dims(anyarray)`	text	Retorna una representació de text de les dimensions de la taula.	`array_dims (ARRAY[[1,2,3],[4,5,6]])`	[1:2][1:3]
`array_fill(anyelement, int[],[,int[]])`	anyarray	Retorna una taula inicialitzada amb el valor i dimensions indicats.	`array_fill(7,ARRAY[3], ARRAY[2])`	[2:4]={7,7,7}
`array_length(anyarray,int)`	int	Retorna la longitud de la dimensió de la taula sol·licitada.	`array_length(array [1,2,3],1)`	3
`array_lower(anyarray,int)`	int	Retorna el límit inferior de la dimensió de la taula sol·licitada	`array_lower('[0:2]= {1,2,3}'::int[],1)`	0
`array_prepend(anyelement, anyarray)`	anyarray	Afegeix un element al principi d'una taula.	`array_prepend(1, ARRAY[2,3])`	{1,2,3}
`array_to_string(anyarray, text)`	text	Concatena elements d'una taula utilitzant el delimitador indicat.	`array_to_string(ARRAY[1, 2, 3],'~^~')`	1~^~2~^~3
`array_upper(anyarray,int)`	int	Retorna el límit superior de la dimensió de la taula sol·licitada.	`array_upper(ARRAY [1,2,3,4],1)`	4
`string_to_array(text, text)`	text[]	Divideix la cadena de caràcters en elements d'una taula utilitzant el delimitador indicat.	`string_to_array ('xx~^~yy~^~zz','~^~')`	{xx,yy,zz}
`unnest(anyarray)`	setof anyelement	Amplia una taula a un conjunt de files.	`unnest(ARRAY[1,2])`	1 2 (2 rows)

3. DDL i DML de les bases de dades objecte-relacionals

Moltes vegades no n'hi ha prou amb la gestió de dades que proporciona el llenguatge SQL, ja que sovint ens interessarà automatitzar processos repetitius i/o prendre decisions de gestió de dades en funció del seu contingut. Per fer-ho caldrà disposar d'una extensió procedimental al llenguatge SQL que el PostgreSQL proporciona amb el llenguatge PL/pgSQL.

PL/PgSQL és un llenguatge procedimental per a sistemes de bases de dades PostgreSQL.

Llenguatge de definició de dades

Un llenguatge de definició de dades, data definition language (DDL), és un llenguatge proporcionat pel sistema de gestió de bases de dades que permet als usuaris dur a terme tasques de definició de les estructures que emmagatzemaran les dades com també els procediments o funcions que permetran consultar-les.

Els llenguatges procedimentals executen i processen, mitjançant ordres DDL (*data definition language*, llenguatge de definició de dades) i DML (*data manipulation language*, llenguatge de manipulació de dades), diferents operacions directament en el servidor. Un dels avantatges principals d'executar programació en el servidor de base de dades és que les consultes i el resultat no han de ser transportats entre el client i el servidor, ja que les dades resideixen en el mateix servidor.

Llenguatge de manipulació de dades

Un llenguatge de manipulació de dades, data manipulation language (DML), és un llenguatge proporcionat pel Sistema de Gestió de Bases de Dades que permet als usuaris dur a terme les tasques de consulta o manipulació de les dades.

El llenguatge PL/pgSQL, desenvolupat en C, s'executa des del mateix client de PostgreSQL (pgsql). Els objectius del seu disseny com a llenguatge procedimental són:

1. Que s'utilitzi per crear funcions i disparadors (*triggers*).

2. Que afegeixi estructures de control al llenguatge SQL.

3. Que hereti totes les definicions d'usuari com tipus, funcions i operadors.

   El PL/pgSQL disposa d'estructures de control repetitives i condicionals, a més de donar la possibilitat de crear funcions que es poden cridar en sentències SQL o executar en esdeveniments de tipus disparador (*trigger*).

Les funcions escrites en PL/PgSQL poden acceptar com a arguments qualsevol dada escalar o *array* de dades que estiguin suportats per l'SGBD i poden retornar un resultat de qualsevol d'aquests tipus. També podem acceptar o retornar un tipus de dada compost (*row type*) especificat pel nom. Es pot declarar una funció PL/PgSQL que retorni un registre, la qual cosa significa que el resultat serà un tipus de fila les columnes de la qual han estat determinades per l'especificació en la crida de la consulta.

3.1 Declaració i inicialització d'objectes

Una vegada s'ha definit un tipus objecte es pot utilitzar en qualsevol bloc PL/pgSQL. Les instàncies dels tipus objecte es creen en temps d'execució, de manera queaquests objectes segueixen les regles normals d'àmbit i instància. En un bloc o subprograma, els objectes locals són instanciats quan s'entra en el bloc o subprograma i deixen d'existir quan se'n surt.

3.1.1 Estructura del PL/pgSQL

En el PostgreSQL, en ser un llenguatge estructurat en blocs, el codi de la definició d'una funció es considera un bloc. Un bloc es defineix de la manera següent:

```
[ <<etiqueta>> ]
[ DECLARE
<declaracions de constants i variables>]
BEGIN
<sentències executables>;
[EXCEPTION
<declaració d'excepcions>;]
END [ etiqueta ];
```

L'etiqueta només és necessària si volem identificar el bloc per utilitzar-lo en una sentència `EXIT`, o per qualificar els noms de les variables declarades en el bloc. Si una etiqueta s'escriu després d'`END`, ha de coincidir amb l'etiqueta al principi del bloc.

A continuació podem veure un exemple de bloc i el codi de la definició de la funció `taxa_vendes(subtotal real)`:

```
CREATE FUNCTION taxa_vendes(subtotal real) RETURNS real AS $$
BEGIN
RETURN subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;
```

> Hi pot haver qualsevol nombre de subblocs en la secció de la sentència d'un bloc.

Hi ha dos tipus de comentaris a PL/pgSQL. Amb dos guions `(-)`comença un comentari que s'estén fins al final de la línia mentre que els caràcters `/*` inicien un bloc de comentaris que s'estén fins que es troba una altra vegada el caràcter `*/`.

A continuació podeu veure un exemple d'una funció escrita amb PL/pgSQL que conté subblocs, `aquestafunc()`:

```
CREATE FUNCTION aquestafunc() RETURNS integer AS $$
DECLARE
quantitat integer := 30;
```

```
BEGIN
RAISE NOTICE 'La quantitat aquí conté %', quantitat; -- Mostra 30
quantitat := 50;
-- Creem un sub-bloc
DECLARE
quantitat integer := 80;
BEGIN
RAISE NOTICE 'La quantitat aquí conté %', quantitat; -- Mostra 80
END;
RAISE NOTICE 'La quantitat aquí conté %', quantitat; -- Mostra 50
RETURN quantitat;
END;
$$ LANGUAGE plpgsql;
```

Al final de la funció es defineix el llenguatge procedimental que utilitzarem mitjançant la clàusula `LANGUAGE`.

És important no confondre l'ús de `BEGIN/END` per agrupar declaracions en PL/pgSQL amb els noms de les ordres SQL per al control de la transacció. `BEGIN/END` només són per a l'agrupació, no per iniciar o finalitzar una transacció.

Trobareu més informació sobre l'ús de la transacció en l'apartat "Modificació d'objectes" d'aquesta unitat.

També podem canviar la definició de la funció amb l'ordre `ALTER FUNCTION`. Podem modificar el nom d'una funció tal com es mostra tot seguit:

```
ALTER FUNCTION taxa_vendes(subtotal real) RENAME TO impost_vendes;
```

3.1.2 Declaració d'objectes

La declaració d'objectes es pot dividir en diferents parts: la que defineix els paràmetres d'entrada i de retorn d'una funció; la que defineix el tipus objecte com a *variable de fila*; la que defineix l'estratègia que s'utilitza per declarar les col·leccions; i la que ens permet copiar el tipus de dades d'una estructura a una altra.

Declaració de paràmetres d'una funció

Totes les variables i constants, `CONSTANT`, utilitzades en un bloc o en un subbloc s'han de declarar en la secció `DECLARE` del bloc. Les variables en PL/pgSQL poden emmagatzemar qualsevol tipus de dades de SQL i el seu valor per omissió és `null`.

Prenent com a exemple la funció anterior `taxa_vendes(subtotal real)`amb una declaració de variables:

```
CREATE FUNCTION taxa_vendes(subtotal real) RETURNS real AS $$
DECLARE
percentatge real := 0.06;
BEGIN
RETURN subtotal * percentatge;
```

```
END;
$$ LANGUAGE plpgsql;
```

Els paràmetres passats a les funcions s'anomenen amb els identificadors `$1`, `$2`, etc. De manera opcional, es poden declarar àlies utilitzant el número del paràmetre, `$n`, per a més llegibilitat. Qualsevol àlies o identificador numèric es pot utilitzar per fer referència al valor del paràmetre.

Es poden crear àlies de dues maneres diferents. La millor manera és donar un nom al paràmetre de la funció: aquest es pot definir en crear la funció com en l'exemple anterior en què se li assigna l'àlies `subtotal`.

```
CREATE FUNCTION taxa_vendes(subtotal real) RETURNS real AS $$
```

L'altra manera, que era l'única disponible abans de la versió del PostgreSQL 8.0, és declarar de manera explícita un àlies utilitzant la sintaxi de declaració següent:

```
nom ALIAS FOR $n;
```

L'exemple anterior es podria modificar com es mostra a continuació:

```
CREATE FUNCTION taxa_vendes (real) RETURNS real AS $$
DECLARE
percentatge real := 0.06;
subtotal ALIAS FOR $1;
BEGIN
RETURN subtotal * percentatge;
END;
$$ LANGUAGE plpgsql;
```

Aquests dos exemples no són perfectament equivalents. En el primer cas, subtotal es podria referenciar com a `taxa_vendes.subtotal`, en canvi, en el segon cas no es podria.

Quan una funció PL/pgSQL es declara amb paràmetres de sortida, als paràmetres de sortida se'ls dóna els noms `$n`i àlies opcionals de la mateixa manera que als paràmetres d'entrada. Un paràmetre de sortida és una variable que comença amb valor `null`, i s'hi ha d'assignar un valor durant l'execució de la funció. El valor final del paràmetre és el que es retorna. Per exemple, l'impost a les vendes també es podria fer com es mostra a continuació:

```
CREATE FUNCTION taxa_vendes(subtotal real, OUT taxa real) AS $$
DECLARE

percentatge real := 0.06;

BEGIN

taxa := subtotal * percentatge;

END;
$$ LANGUAGE plpgsql;
```

Declaració d'un tipus objecte

Una variable de tipus objecte s'anomena *variable de fila* (variable *row-type*). Aquesta pot contenir una fila sencera del resultat d'una consulta `SELECT` o `FOR`, mentre que el conjunt de consultes d'una columna coincideix amb el tipus declarat de la variable.

```
nomnom_taula%ROWTYPE;
nomnom_tipus_objecte;
```

Als valors dels camps individuals de la fila s'hi accedeix usant la notació de punts habitual, `variable_fila.camp`.

En cas que els paràmetres d'una funció siguin de tipus objecte, l'identificador corresponent `$n` serà una variable de fila, i els camps es podran seleccionar com a `$1.id_usuari`.

A continuació podem veure un exemple de l'ús dels tipus objecte. Els tipus `taula1` i `taula2` ja existeixen, i podem veure com apareix la sentència `select` per fer una consulta:

```
CREATE FUNCTION combinació(t_fila taula1) RETURNS text AS $$
DECLARE
t2_fila taula2%ROWTYPE;
BEGIN
SELECT * INTO t2_fila FROM taula2 WHERE ... ;
RETURN t_fila.f1 || t2_fila.f3 || t_fila.f5 || t2_fila.f7;
END;
$$ LANGUAGE plpgsql;
```

De retorn d'una funció

Les funcions poden retornar tipus bàsics o compostos, fins i tot es poden retornar estructures de taules. Les estructures de control són probablement la part més útil del PL/pgSQL. Amb aquestes estructures es poden manipular les dades PostgreSQL d'una manera molt flexible.

Dins les estructures de control trobem dues ordres disponibles que permeten retornar les dades d'una funció: `RETURN` i `RETURN NEXT`.

L'ordre `RETURN` amb una expressió acaba la funció i retorna el valor de l'expressió a la crida. Aquest mètode és utilitzat per les funcions PL/pgSQL que no retornen un conjunt.

```
RETURN expressió;
```

Quan es retorna un tipus escalar es pot utilitzar qualsevol expressió. El resultat de l'expressió es converteix automàticament en el tipus de retorn de la funció. Per retornar el valor d'un tipus objecte (fila), s'ha de declarar l'expressió amb una variable de registre o fila.

La funció següent incrementa un nombre enter fent ús d'un paràmetre d'entrada, i retorna el resultat següent:

```
CREATE FUNCTION increment(i integer) RETURNS integer AS $$
BEGIN
RETURN i + 1;
END;

$$ LANGUAGE plpgsql;
```

Quan una funció PL/pgSQL es declara com a retorn `SETOF`, el procediment a seguir és lleugerament diferent. En aquest cas, els elements individuals a retornar s'especifiquen mitjançant una seqüència d'ordres `RETURN NEXT` o `RETURN QUERY`, i a continuació s'utilitza una ordre final `RETURN` sense arguments per indicar que la funció s'ha acabat d'executar.

L'ordre `RETURN NEXT` es pot utilitzar amb tipus escalars i tipus objecte. Si s'utilitza com a resultat, un tipus objecte retornarà una "taula" de resultats. L'ordre `RETURN QUERY` afegeix els resultats de l'execució d'una consulta al conjunt de resultats de la funció. Aquestes dues ordres poden ser barrejades en un sol conjunt d'una funció, en aquest cas els seus resultats es concatenen.

Si declarem la funció amb paràmetres de sortida, hem d'escriure `RETURN NEXT` sense expressió. En cada execució, els valors actuals de la variable o variables de sortida es desaran per a un eventual retorn com una fila del resultat. Recordeu que hem de declarar la funció com a retorn de registre, `SETOF record`, quan hi hagi diferents paràmetres de sortida, o `SETOF sometype`, quan hi hagi un sol paràmetre de sortida de tipus *sometype*, amb la finalitat de crear una funció amb un retorn que sigui un conjunt dels paràmetres de sortida.

A continuació veurem l'exemple d'una funció que utilitza `RETURN NEXT`:

```
CREATE TABLE foo (fooid INT, foosubid INT, fooname TEXT);

CREATE OR REPLACE FUNCTION getAllFoo() RETURNS SETOF foo AS
$BODY$
DECLARE
r foo%rowtype;
BEGIN
FOR r IN SELECT * FROM foo
WHERE fooid > 0
LOOP
-- camp per realitzar les tasques de processament
RETURN NEXT r; -- retorna la fila actual del SELECT
END LOOP;
RETURN;
END
$BODY$
LANGUAGE 'plpgsql' ;
```

Declaració de col·leccions

El bucle `FOREACH` és molt semblant a un bucle `FOR`, però en comptes d'iterar mitjançant les files retornades per una consulta SQL, aquest itera mitjançant els elements d'una col·lecció. La instrucció `FOREACH` té la sintaxi que es mostra en la figura 3.1.

FIGURA 3.1. Sintaxi de 'FOREACH'

```
[ <<etiqueta>> ]
FOREACH variable destinació [ SLICE número ] IN ARRAY expressió LOOP
    declaracions
END LOOP [ etiqueta ];
```

Sense SLICE, o si s'especifica SLICE 0, el bucle es repeteix mitjançant elements individuals de l'*array* produïda per l'avaluació de l'expressió.

La variable de destinació, *target*, s'assigna a cada valor d'element en seqüència, i el cos del bucle s'executa per a cada element. A continuació veurem un exemple d'un bucle mitjançant els elements d'una col·lecció d'enters:

```
1  CREATE FUNCTION suma(int[]) RETURNS int8 AS $$
2  DECLARE
3  s int8 := 0;
4  x int;
5  BEGIN
6  FOREACH x IN ARRAY $1-- $1: paràmetre d'entrada int[]
7  LOOP
8  s := s + x;
9  END LOOP;
10 RETURN s;
11 END;
12 $$ LANGUAGE plpgsql;
```

Els elements són iterats en ordre d'emmagatzematge, independentment del nombre de dimensions de la col·lecció. Tot i que la variable de destinació és en general única, pot ser una llista de variables que iteri per mitjà d'una col·lecció de valors compostos (registres). En aquest cas, per cada element de la matriu, les variables s'assignen a partir de columnes successives del valor compost.

Amb un valor positiu del paràmetre SLICE, el bucle FOREACH itera per mitjà de talls de la col·lecció en lloc d'elements individuals. El valor del segment, SLICE, ha de ser un enter constant no més gran que el nombre de dimensions de la col·lecció. La variable destinació ha de ser una col·lecció, i ha de rebre talls successius del valor de la col·lecció, en què cada sector és el nombre de dimensions especificades pel segment. A continuació es mostra un exemple d'iteració mitjançant un segment d'una dimensió:

```
1  CREATE FUNCTION cercar_files(int[]) RETURNS void AS $$
2
3  DECLARE
4  x int[];
5
6  BEGIN
7  FOREACH x SLICE 1 IN ARRAY $1
8
9  LOOP
10 RAISE NOTICE 'row = %', x; -- El que mostrarà la funció
11 END LOOP;
12
13 END;
14
15 $$ LANGUAGE plpgsql;
```

La crida següent utilitza la funció anterior passant-li per paràmetre una matriu:

```
SELECT cercar_files(ARRAY[[1,2,3],[4,5,6],[7,8,9],[10,11,12]]);
-- La sortida que mostra en executar-se la funció:
NOTICE: row = {1,2,3}
NOTICE: row = {4,5,6}
NOTICE: row = {7,8,9}
NOTICE: row = {10,11,12}
```

Tipus de còpia

L'ordre %TYPE ofereix el tipus de dades d'una variable o columna d'una taula. La podem utilitzar per declarar les variables que contindran els valors de la base de dades. Per exemple, si tenim una columna anomenada `id_usuari` en la taula d'usuaris, per declarar una variable amb el mateix tipus de dades faríem el següent:

```
id_usuari usuaris.id_usuari%TYPE;
```

Mitjançant l'ús de %TYPE no cal conèixer el tipus de dades de l'estructura a la qual fem referència, i el més important, si el tipus de dades de l'estructura referenciada canvia en el futur, no haurem de modificar la definició de la funció.

Polimorfisme

És la capacitat de diversos tipus objecte derivats d'un antecessor per utilitzar el mateix mètode de manera diferent.

L'ordre %TYPE és imprescindible en funcions polimòrfiques, ja que els tipus de dades necessàries per a les variables internes poden canviar d'una crida a la següent.

3.1.3 Inicialització d'objectes

Les variables declarades en la secció de declaracions (DECLARE) s'inicialitzen amb el seu valor per defecte cada vegada que s'inicia el bloc, no cada vegada que es fa la crida a la funció. Per exemple, l'assignació de la funció `ara()` a una variable de tipus `timestamp` fa que la aquesta emmagatzemi el temps del moment en què s'ha fet la crida, no el moment en què la funció es va compilar.

Quan s'inicialitzen les variables també es poden definir valors per defecte, DEFAULT, o com a constants, CONSTANT, tal com es pot veure tot seguit:

```
quantitat integer DEFAULT 32;
url varchar := 'http://elmeuespai.com';
id_usuari CONSTANT integer := 10;
```

També es pot dur a terme la inicialització mitjançant una funció,en què les dades del tipus objecte es passen per mitjà dels paràmetres d'entrada. Tot seguit es mostra el tipus de dades `tipus_persona`, la taula `persona` d'aquest tipus i la funció que l'inicialitza:

```
CREATE TYPE tipus_persona AS (
dnivarchar (9),
nom varchar (15),
adreçavarchar (30)
);

```

```
CREATE TABLA persona OF tipus_persona (
PRIMARY KEY (dni)
);

CREATE FUNCTION alta_persona(varchar (9), varchar (15), varchar (30)) AS $$
BEGIN
--Inserim els valors que es passen a través dels paràmetres
INSERT INTO persona VALUES ($1, $2, $3);
END;
$$ LANGUAGE plpgsql;
```

3.2 Ús de la sentència 'SELECT'

Les consultes de selecció s'utilitzen per indicar al servidor que retorni informació de les bases de dades. Aquesta informació retornada pot ser un valor, un *tuple* o una taula. La sintaxi bàsica d'una consulta de selecció és la que es mostra en la figura 3.2.

Figura 3.2. Sintaxi bàsica d'una consulta de selecció

```
SELECT camps FROM taula;
```

En aquesta sintaxi, *camps* representa la llista de camps que volem recuperar i *taula* és on hi ha la informació emmagatzemada.

Creem un tipus d'objecte `persona` i una taula d'objectes de tipus `persones` que inclourà diferents valors. Consultem els atributs `id`, `nom` i `telefon` dels objectes de la taula:

```
CREATE TYPE persona AS (
id integer,
nom varchar(15),
cognom varchar(20),
aniversaridate,
adreçavarchar(30),
telefonvarchar(15)
);
CREATE TABLE persones OF persona;
-- Consulta dels objectes de la taula persones
SELECT id, nom, telefon FROM persones;
```

També ens podem referir a tots els camps d'una taula amb el símbol `*`. De manera que la consulta anterior també es podria fer de la manera següent:

```
SELECT * FROM persones;
```

De manera addicional es pot especificar l'ordre en què es volen recuperar els objectes de la taula mitjançant la clàusula `ORDER BY llista_camps`, en què `llista_camps` representa els camps a ordenar:

```
SELECT id, nom, telefon FROM persones ORDER BY id;
```

Aquesta consulta retorna les dades de les persones emmagatzemades a la taula `persones` però ordenades en funció del seu identificador.

Per indicar que volem recuperar els objectes segons l'interval de valors d'un camp hem d'utilitzar l'operador `BETWEEN`. La seva sintaxi és la que es mostra en la figura 3.3.

Figura 3.3. Sintaxi de l'operador 'BETWEEN'

```
camp [NOT] BETWEEN valor1 AND valor2
```

En aquest cas la consulta retornaria els objectes que continguin a *camp* un valor inclòs en l'interval *valor1*, *valor2* (tots dos inclosos). Si avantposem la condició `NOT`, es retornaran els valors no inclosos en l'interval:

```
1 BEGIN
2 SELECT * FROM persones
3 WHERE id BETWEEN 1 AND 100;
4 END;
```

La consulta retornarà les persones que tinguin el valor de l'atribut identificador entre 1 i 100.

3.2.1 Accés a les dades

Cada tipus de funció pot prendre els tipus bàsics, els tipus compostos o combinacions d'aquests com a arguments (paràmetres). A més, cada tipus de funció pot retornar un tipus bàsic o un tipus compost.

Accés als tipus compostos

Prenem com a exemple el tipus objecte *inventari_element.* Aquest conté tres atributs de diferent tipus:

```
1 CREATE TYPE inventari_element AS (
2 nom text,
3 proveidor_id integer,
4 preu numeric
5 );
```

Aquest tipus objecte el podem utilitzar per definir el tipus d'un atribut dins d'una taula, és a dir, definir la columna de la taula.

```
1 CREATE TABLE a_mà (
2 element inventari_element,
3 quantitat integer
4 );
```

Tanmateix, una funció pot prendre com a argument un tipus compost.

Tot seguit podeu veure com una funció utilitza un tipus compost en el seu cos:

```
CREATE FUNCTION preu_extensió(inventari_element, integer) RETURNS numeric AS $$
BEGIN
RETURN $1.preu * $2;
END;
$$ LANGUAGE plpgsql;
```

I ara podeu veure una consulta que utilitza la funció anterior, on el retorn de la funció és el paràmetre de la consulta:

```
SELECT preu_extensió(t.element, 10) FROM a_mà t;
```

Per accedir al camp d'una columna composta, s'escriu un punt i el nom del camp. Per exemple, si volem seleccionar alguns subcamps de la taula `a_ma`, com el nom dels elements amb preus superiors a les 9,99 unitats, ho faríem amb la sentència següent:

```
SELECT element.nom FROM a_mà WHERE element.preu > 9.99;
```

Però això no funcionarà, ja que el nom de l'element es pren com a un nom de taula, no com a nom d'una columna de la taula `a_mà`, segons les regles de la sintaxi SQL. Hauríem d'escriure, doncs, el següent:

```
SELECT (element).nom FROM a_mà WHERE (element).preu > 9.99;
```

O, en cas que necessitem utilitzar el nom de la taula, l'escriuríem de la manera següent:

```
SELECT (a_mà.element).nom FROM a_mà WHERE (a_mà.element).preu > 9.99;
```

Ara l'objecte entre parèntesis s'interpreta correctament com una referència a la columna d'elements i, a continuació el subcamp es pot seleccionar des d'aquesta.

Problemes sintàctics similars tenen lloc cada vegada que seleccionem un camp d'un valor compost. Per exemple, per seleccionar un sol camp a partir del resultat d'una funció que retorni un valor compost, la sintaxi seria la següent:

```
SELECT (la_meva_funció(...)).camp FROM ...
```

Sobrenoms

En una base de dades amb tipus i objectes, el més recomanable és utilitzar sobrenoms per als noms de les taules. El sobrenom d'una taula ha de ser únic en el context de la consulta. Els sobrenoms serveixen per accedir al contingut de la taula, però s'han d'utilitzar adequadament en les taules que emmagatzemen objectes.

Vegeu tot seguit com s'han d'utilitzar els sobrenoms:

```
BEGIN
CREATE TYPE persona AS (nom varchar(15));
CREATE TABLE ptaula1 OF persona;
CREATE TABLE ptaula2 (p1 persona);
...
END;
```

La diferència entre les dues primeres taules és que la primera emmagatzema objectes del tipus `persona`, mentre que la segona té una columna en què s'emmagatzemen valors del tipus `persona`. Considerant les consultes següents, vegeu en la figura 3.4 com s'accedeix a aquestes taules.

FIGURA 3.4. Consultes utilitzant sobrenoms

1. SELECT nom FROM ptaula1;	Correcte
2. SELECT p1.nom FROM ptaula2;	Incorrecte
3. SELECT *sobrenom*.nom FROM ptaula2 *sobrenom*;	Correcte

En la primera consulta, `nom` es considera com una de les columnes de la taula `ptaula1`, ja que els atributs dels objectes es consideren columnes de la taula d'objectes. En canvi, en la segona consulta es requereix l'ús d'un sobrenom per indicar que `nom` és el nom d'un atribut de l'objecte de tipus `persona` que s'emmagatzema en la columna `p1`. Per resoldre aquest problema no es poden utilitzar els noms de les taules directament: `ptaula2.p1.nom` seria incorrecte.

Per facilitar la formulació de consultes i evitar errors es recomana utilitzar sobrenoms per accedir a totes les taules que continguin objectes amb identitat o sense i per accedir a les columnes de les taules en general.

3.2.2 Criteris de selecció

El PL/pgSQL tambépermet filtrar els objectes amb la finalitat de recuperar només els que compleixin unes condicions preestablertes.

La clàusula 'WHERE'

La clàusula `WHERE` es pot utilitzar per determinar quins objectes de les taules enumerades en la clàusula `FROM` apareixeran en els resultats de la sentència `SELECT`.

Per exemple, per obtenir només les persones que tinguin el cognom `arnau`, la consulta adequada seria:

```
BEGIN
SELECT * FROM persones
WHERE cognom LIKE '%arnau';
END;
```

En l'exemple següent podeu veure la unió de la taula `pel·licules` amb la taula `distribuidors`:

```
BEGIN
SELECT p.titol, p.did, d.nom, p.data_prod, p.tipus
FROM distribuidors d, pel·licules p
WHERE p.did = d.did
END;
```

Resultat de la consulta:

```
titol | did | nom | data_prod | tipus
------------------------+-----+-------------+------------+----------
The Third Man | 101 | British Lion | 1949-12-23 | Drama
The African Queen | 101 | British Lion | 1951-08-11 | Romantica

...
```

Les clàusules 'GROUP BY' i 'HAVING'

Després de passar el filtre `WHERE`, la taula d'entrada resultant es pot agrupar, amb la clàusula `GROUP BY`, i es poden eliminar les files del grup amb la clàusula `HAVING`.

```
SELECT llista_selecció
FROM ...
[WHERE ...]
GROUP BY agrupació_referencies_columna [, ]...
```

Aquesta clàusula s'utilitza per agrupar les files d'una taula que tenen els mateixos valors en totes les columnes que es mostren. L'ordre en què les columnes s'enumeren no importa. L'efecte és combinar cada conjunt de files que tenen valors comuns a la fila grup que representa totes les files del grup. Això es fa per eliminar la redundància en la sortida. Per exemple:

```
SELECT * FROM test1;
x | y
---+---
a | 3
c | 2
b | 5
a | 1
(4 rows)
```

```
SELECT x, suma(y) FROM test1 GROUP BY x;
```

```
x | suma
---+-----
a | 4
b | 5
c | 2
(3 rows)
```

Un altre exemple: la consulta calcula les vendes totals de cada producte, en lloc de les vendes totals de tots els productes:

```
BEGIN
SELECT id_producte, p.nom, (suma(s.units) * p.preu) AS vendes
FROM productes p LEFT JOIN vendes s USING (id_producte)
GROUP BY id_producte, p.nom, p.preu;
END;
```

En aquest exemple, les columnes `id_producte`, `p.nom` i `p.preu` han d'estar en la clàusula `GROUP BY`, ja que se'n fa referència en la llista de selecció de la consulta. La columna `s.units` no ha d'estar en la llista `GROUP BY`, ja que només s'utilitza en una expressió d'agregació (`suma(...)`), que representa les vendes d'un producte. Per cada producte, la consulta retorna totes les vendes del producte.

Si una taula s'ha agrupat utilitzant `GROUP BY`, però només alguns grups són d'interès, la clàusula `HAVING` es pot utilitzar, de la mateixa manera que una clàusula `WHERE`, per eliminar grups del resultat. La sintaxi és:

```
SELECT llista_selecció FROM ... [WHERE ...] GROUP BY ... HAVING expressió
    _booleana
```

Les expressions amb la clàusula `HAVING` es poden referir tant a les expressions agrupades com a les no agrupades (que impliquen necessàriament una funció d'agregació). A continuació es pot veure un exemple seguint l'anterior:

```
SELECT x, suma(y) FROM test1 GROUP BY x HAVING suma(y) > 3;
x | suma
---+-----
a | 4
b | 5
(2 rows)
```

```
SELECT x, suma(y) FROM test1 GROUP BY x HAVING x < 'c';
```

```
x | suma
---+-----
a | 4
b | 5
(2 rows)
```

Prenent com a exemple, la taula `pel·licules` de l'apartat anterior, podem sumar la columna longitud (`long`) de totes les pel·lícules i agrupar els resultats per `tipus`:

```
SELECT tipus, suma(long) AS total FROM pel·licules GROUP BY tipus;
```

```
tipus | total
----------------+-------
Accio | 07:34
Comedia | 02:58
Drama | 14:28
Musical | 06:42
Romantica | 04:38
```

Si volem sumar la columna longitud, `long`, de totes les pel·lícules, cal que agrupem els resultats per tipus i mostrem només els grups amb una durada inferior a cinc hores. Ho faríem de la manera següent:

```
SELECT tipus, suma(long) AS total FROM pel·licules
GROUP BY tipus
HAVING suma(long) < interval '5 hores';
```

```
tipus | total
------------------+-------
Comedia | 02:58
Romantica | 04:38
```

L'operador 'JOIN'

Si distribuïm la informació en diferents taules evitem la redundància de dades i ocupem menys espai físic en el disc. L'operador `JOIN` relaciona dues o més taules per obtenir un resultat que inclogui dades (camps i registres) de totes dues, i les taules es combinen segons els camps comuns a totes dues taules.

Per exemple, la informació de `llibres` es podria separar en dues taules, una anomenada `llibre` i una altra `editorial`. A continuació definim les dues taules:

```
CREATE TABLE llibre (
codiserial,
titolvarchar(40) not null,
autor varchar(30) not null default 'Desconegut',
codi_editorialsmallint not null,
preudecimal,
PRIMARY KEY(codi)
);
CREATE TABLE editorial (
codiserial,
nomvarchar(20) not null,
PRIMARY KEY(codi)
);
```

D'aquesta manera evitem emmagatzemar cada vegada la informació de l'editorial en la taula `llibre`. Indiquem l'editorial de cada llibre amb el camp que fa referència al codi de l'editorial.

Quan fem una consulta de les dades de la taula llibre ens apareixerà el codi de l'editorial, però no obtindrem la informació de l'editorial. Per obtenir les dades del llibre i de l'editorial necessitem consultar les dues taules alhora. Tot seguit es mostra una consulta relacionant les dues taules:

```
SELECT * FROM llibre
JOIN editorial ON llibre.codi_editorial = editorial.codi;
```

L'operador 'UNION'

Aquest operador calcula la unió del conjunt de files retornades per les sentències `SELECT` involucrades. Una fila apareix en el conjunt de la unió si apareix en almenys un dels conjunts de resultats. Les dues sentències `SELECT` que representen els operands directes de la unió han de tenir el mateix nombre de columnes, i les columnes corresponents han de ser de tipus de dades compatibles.

La clàusula UNION té la sintaxi que es mostra en la figura 3.5.

FIGURA 3.5. Sintaxi de la clàusula 'UNION'

```
declaracio_select UNION [ ALL | DISTINCT ] declaracio_select
```

Així, declaracio_select fa referència a qualsevol declaració SELECT sense les clàusules ORDER BY, LIMIT, FOR UPDATE, o FOR SHARE. El resultat de l'operador UNION no conté registres duplicats tret que s'especifiqui l'opció ALL. Aquesta opció impedeix que s'eliminin els duplicats. En canvi, DISTINCT es pot escriure per especificar explícitament el comportament per defecte d'eliminar les files duplicades.

Prenent com a exemple la taula distribuidors de l'apartat anterior, tot seguit es mostra com es pot obtenir la unió de les taules distribuidors i actors, restringint els resultats als que comencen amb la lletra *W* en cada taula. Només interessen les files diferents, de manera que la paraula clau ALL s'omet.

```
distribuidors: actors:
did | nom id | nom
————+————————— ————+————————————
108 | Westward 1 | Woody Allen
111 | Walt Disney 2 | Warren Beatty
112 | Warner Bros. 3 | Walter Matthau
... ...
```

```
SELECT distribuidors. nom
FROM distribuidors
WHERE distribuidors. nom LIKE 'W%'
UNION
SELECT actors.nom
FROM actors
WHERE actors.nom LIKE 'W%';
```

```
nom
———————————
Walt Disney
Walter Matthau
Warner Bros.
Warren Beatty
Westward
Woody Allen
```

3.2.3 Subconsultes

Una subconsulta és una instrucció SELECT imbricada dins d'una instrucció SELECT, INSERT...INTO, DELETE o UPDATE o dins d'una altra subconsulta. Es pot utilitzar una subconsulta en lloc d'una expressió en la llista de camps d'una instrucció SELECT o en una clàusula WHERE.

En les subconsultes s'utilitza la instrucció SELECT per proporcionar un conjunt d'un o més valors per avaluar l'expressió de la clàusula WHERE de la consulta

principal. La subconsulta següent retorna com a resultat un conjunt de files que contenen només atributs dels objectes `persona`:

```
BEGIN
INSERT INTO treballadors
(SELECT * FROM persones WHERE cognom LIKE '%arnau');
END;
```

La subconsulta següent ens retorna el títol, l'autor i el preu del llibre més car:

```
BEGIN
SELECT titol, autor, preu FROM llibre
WHERE preu = (SELECT max(preu) FROM llibre);
END;
```

Fins ara hem vist que una subconsulta pot substituir una expressió. Aquesta subconsulta ha de retornar un valor escalar o una llista de valors d'un camp. Les subconsultes que retornen una llista de valors inclouen dins la clàusula `WHERE` la paraula clau `IN`.

El resultat de fer una subconsulta amb `IN` o `NOT IN` és una llista de valors. La sintaxi és la que es mostra en la figura 3.6.

FIGURA 3.6. Sintaxi d'una subconsulta amb "IN o 'NOT IN'

```
... WHERE expressió IN (subconsulta);
```

L'exemple següent mostra els noms de les editorials que han publicat llibres d'un determinat autor:

```
BEGIN
SELECT nom FROM editorial
WHERE codi IN (SELECT codi_editorial FROM llibre
WHERE autor = 'Joanne Kathleen');
END;
```

La subconsulta retorna una llista de valors d'un sol camp, en aquest cas el `codi`, que utilitzarà la consulta exterior.

També es poden buscar valors no coincidents, per exemple, les editorials que no han publicat llibres d'un autor específic:

```
BEGIN
SELECT nom FROM editorial
WHERE codi NOT IN (SELECT codi_editorial FROM llibre
WHERE autor = 'Joanne Kathleen');
END;
```

3.3 Inserció d'objectes

Per afegir objectes a una taula d'objectes s'utilitza la sentència `INSERT INTO`. La sintaxi és la que es mostra en la figura 3.7.

FIGURA 3.7. Sintaxi de la sentència 'INSERT INTO'

```
INSERT INTO taula (camp1, camp2, ..., campN) VALUES (valor1, valor2, ..., valorN);
```

Aquesta consulta emmagatzema en el camp1 el valor1, en el camp2 el valor 2 i així successivament. S'ha de tenir un tracte especial amb les cadenes de caràcters, ja que s'han de delimitar els valors literals utilitzant cometes simples (').

Les sentències següents són equivalents. En la primera no s'especifiquen els camps de la taula perquè s'insereixen tots els valors en l'ordre adequat de les columnes de la taula. En la segona podem veure que es remarca l'ordre dels camps que volem inserir. En aquest cas, els camps coincideixen:

```
1 BEGIN
2 CREATE TABLE persones OF persona;
3 ...
4 INSERT INTO persones VALUES (001, 'Lluis', 'Llatzer', 1985-04-17, 'Alabastros 5
      ', '612345421');
5 INSERT INTO persones (id, nom, cognom, aniversari, adreça, telefon) VALUES
      (001, 'Lluis', 'Llatzer', 1985-04-17, 'Alabastros 5', '612345421');
6 ...
7 END;
```

No cal introduir tots els valors dels atributs de l'objecte, però en aquest cas és necessari remarcar quins camps emmagatzemaran els valors que s'han introduït. Vegeu-ne un exemple tot seguit:

```
1 INSERT INTO persones (id, nom, cognom, telefon) VALUES (002, 'Isabel', 'Cervera
       Arnau', '634769835');
```

La sentència `INSERT INTO` utilitza l'ordre `ROW` quan un dels atributs de la taula és un tipus compost, i per accedir-hi s'han d'introduir les dades dins d'una estructura de tipus fila. Tal com mostra l'exemple següent:

```
1  BEGIN
2  CREATE TYPE persona AS (
3  id integer,
4  nom varchar(15),
5  cognom varchar(20),
6  ...
7  );
8  CREATE TABLE compte (
9  clientpersona,
10 sucursalinteger,
11 balançreal
12 );
13
14 INSERT INTO compte VALUES (ROW(002, 'Carme', 'Llatzer Roig'), 043, 2500.50);
15 END;
```

Un altre exemple podria ser una inserció en la taula `a_mà`, amb l'atribut `element` com a tipus compost, és a dir, com a columna de la taula:

```
1 BEGIN
2 CREATE TABLE a_mà (
3 element inventari_element,
4 quantitat integer
5 );
```

```
6 INSERT INTO a_mà VALUES (ROW('galetes', 42, 1.99), 1000);
7 END;
```

3.4 Modificació i esborrament d'objectes

Les **consultes d'acció** són les que no retornen cap element i modifiquen el valor d'un, alguns o tots els camps d'un, alguns o tots els registres d'una taula. Podem dir que la modificació i eliminació d'objectes són consultes d'acció.

Quan hi ha diverses consultes i es considera que s'han d'executar en bloc per mantenir la integritat de les dades, parlem de **transacció**.

3.4.1 Modificació d'objectes

Per modificar els valors dels atributs d'un objecte en una taula, basant-se en un criteri especific, s'utilitza la sentència `UPDATE`. La seva sintaxi és la que s'indica en la figura 3.8.

Figura 3.8. Sintaxi de la sentència 'UPDATE'

```
UPDATE taula SET camp1=valor1, camp2=valor2, ... campN=valorN
WHERE criteri;
```

Tot seguit en podeu veure un exemple:

```
1 BEGIN
2 CREATE TABLE persones OF persona;
3 ...
4 UPDATE persones SET adreça = '341 Oakdene Ave'
5 WHERE nom = 'Isabel' AND cognom = 'Cervera Arnau';
6 ...
7 END;
```

Aquesta sentència modifica l'adreça si es compleix el criteri assignat, que és que hi hagi la persona Isabel Cervera Arnau.

La sentència anterior no genera cap resultat. Per saber quins objectes es canviaran, s'ha d'examinar primer el resultat de la consulta de selecció que utilitzi el mateix criteri i després executar la consulta d'actualització.

```
1 BEGIN
2 SELECT * FROM persones WHERE nom = 'Isabel' AND cognom = 'Cervera Arnau';
3 UPDATE persones SET adreça = '341 Oakdene Ave'
4 WHERE nom = 'Isabel' AND cognom = 'Cervera Arnau';
5 END;
```

Si en una actualització esborrem la clàusula `WHERE` tots els objectes de la taula

indicada seran actualitzats; per tant, s'ha d'anar amb compte amb aquest tipus de sentència.

En determinades clàusules, i per a taules heretades, és possible limitar l'esborrament o l'actualització en la taula *pare* (sense que es propagui als registres de les taules *fill*) amb la clàusula `ONLY`. Recuperarem l'exemple utilitzat en l'herència de tipus:

Trobareu més informació sobre l'herència en l'apartat "Herència" d'aquesta unitat.

```
BEGIN
CREATE TABLA persona (
id **serial**,
nom varchar (30),
adreça varchar (30)
);

CREATE TABLE estudiant (
carrera varchar (50),
grup char,
grau int
) INHERITS (persona);

INSERT INTO persona (nom, adreca) VALUES ('Josep Claramunt' , 'Montoliu 12');
INSERT INTO persona (nom, adreca) VALUES ('Lluís Arnau', 'Barcelona 3');
INSERT INTO estudiant (nom, adreca, carrera, grup, grau) VALUES ('Anna Guillen'
     , 'Tarragona 19', 'Psicologia', 'C', 2);

SELECT * FROM persona;

END;
```

```
———————————————————————————— Sortida
id | nom | adreca
———+———————————— +————————————
1 | Josep Claramunt | Montoliu 12
2 | Lluis Arnau | Barcelona 3
3 | Anna Guillen | Tarragona 19
(3 row)
```

Només actualitzem els camps de la taula *pare* `persona`:

```
BEGIN
...
UPDATE ONLY persona SET nom='Sr. ' || nom;
SELECT * FROM persona;
...
END;
———————————————————————————— Sortida
id | nom | adreca
———+———————————— +————————————
1 | Sr. Josep Claramunt | Montoliu 12
2 | Sr. Lluis Arnau | Barcelona 3
3 | Anna Guillen | Tarragona 19
(3 rows)
```

Els valors de la tercera fila no es modifiquen perquè pertanyen a una instància de la taula *fill* `estudiant`; es modifiquen els objectes de la taula *pare* concatenant una paraula amb els noms.

La sintaxi bàsica per modificar objectes utilitzant subconsultes és la que es mostra en la figura 3.9.

FIGURA 3.9. Sintaxi bàsica per modificar objectes utilitzant subconsultes

```
UPDATE taula SET camp = nou_valor
        WHERE camp = (SUBCONSULTA);
```

La subconsulta següent actualitza el preu de tots els llibres de l'editorial Brooklyn:

```
BEGIN
UPDATE llibre SET preu = preu + (preu*0.2)
WHERE codi_editorial =
(SELECT codi FROM editorial
WHERE nom = 'Brooklyn');
END;
```

Transaccions

Les transaccions són un concepte fonamental en tots els sistemes de bases de dades. El punt essencial d'una transacció és que engloba múltiples operacions en un sòl pas; per exemple, si considerem la base de dades d'un banc que conté balanços per a diferents comptes de clients, com també el total dels dipòsits de les sucursals. Suposem que volem registrar el pagament de 100 € des del compte del `Client1` al compte del `Client2`, les sentències SQL a executar per a aquesta operació serien les següents:

```
UPDATE compte SET balanç = balanç - 100 WHERE nom = 'Client1';\\ UPDATE
    sucursal SET balanç = balanç - 100 WHERE nom = (SELECT nom_sucursal FROM
    compte WHERE nom = 'Client1');
UPDATE compte SET balanç = balanç + 100 WHERE nom = 'Client2';\\ UPDATE
    sucursal SET balanç = balanç + 100 WHERE nom = (SELECT nom_sucursal FROM
    compte WHERE nom = 'Client2');
```

Com es pot veure hi ha diferents actualitzacions involucrades per finalitzar l'operació. Els treballadors del banc han d'estar segurs que totes aquestes actualitzacions s'executen, o que en cas d'error no se n'executa cap, ja que es podria donar el cas que el `Client2` rebés 100 € sense que fossin extrets del compte del `Client1`. Agrupant les actualitzacions en una sola transacció es pot garantir que en cas d'error no s'executaria cap actualització.

La **transacció** és un conjunt d'ordres que s'executen formant una unitat de treball, és a dir, de manera indivisible o atòmica. No pot finalitzar en un estat intermedi i, si per alguna causa el sistema la cancel·la, es comencen a desfer les ordres executades fins a deixar la base de dades en el seu estat inicial, mantenint la integritat de les dades.

Atomicitat

Propietat que garanteix que una operació es dugui a terme, no quedant-se mai en estats intermedis.

En PostgreSQL les transaccions es configuren de manera que es tanquen en un bloc les ordres SQL que es volen incloure; el bloc ha de començar i acabar amb les ordres `BEGIN` i `COMMIT`, per exemple:

```
BEGIN
UPDATE compte SET balanç = balanç - 100 WHERE nom = 'Client1'...;
COMMIT;
```

En el moment en què la clàusula `COMMIT` es passa a PostgreSQL és quan s'escriuen les actualitzacions en la base de dades. Si en algun moment no volem fer `COMMIT` d'alguna operació, es pot utilitzar la clàusula `ROLLBACK` i totes les actualitzacions dins de la transacció es cancel·laran.

Si no es vol fer un `ROLLBACK` complet de la transacció es poden definir marcadors (*save-points*) que indiquen fins on es vol que s'arribi en la transacció, per exemple:

```
1  BEGIN;
2
3  UPDATE compte SET balanç = balanç - 100 WHERE nom = 'Client1';
4  SAVEPOINT marcador1;
5  UPDATE compte SET balanç = balanç + 100 WHERE nom = 'Client2';
6  -- es vol descartar l'actualització per al Client2 i fer-la per al Client3
7  ROLLBACK TO marcador1;
8  UPDATE compte SET balanç = balanç + 100 WHERE nom = 'Client3';
9
10 COMMIT;
```

En l'exemple anterior es veu l'ús de marcadors i *rollbacks;* en aquest cas s'ha fet una actualització sobre el compte del `Client2`, però a continuació es decideix que no s'abonarà la quantitat al `Client2`, sinó que es farà sobre el `Client3`. Per tant, es fa un `ROLLBACK` fins al marcador anterior, `marcador1`, i es continua amb l'actualització següent.

3.4.2 Esborrament d'objectes

Per eliminar objectes d'una taula d'objectes s'utilitza la sentència `DELETE`. Aquesta sentència elimina els objectes complets, no és possible eliminar el contingut d'algun camp en concret. En podeu veure la sintaxi en la figura 3.10.

Figura 3.10. Sintaxi de la sentència 'DELETE'

```
DELETE FROM taula WHERE criteri
```

Una vegada s'han eliminat els objectes utilitzats durant l'esborrament d'objectes no es pot desfer l'operació. Si volem saber quins objectes s'eliminaran, primer hem d'examinar els resultats d'una consulta de selecció que utilitzi el mateix criteri i després executar la sentència d'esborrament.

Per eliminar objectes de manera selectiva s'utilitza la clàusula `WHERE` com es mostra en l'exemple:

```
1  BEGIN
2  DELETE FROM persona
3  WHERE nom = 'Josep Claramunt';
4  END;
```

La sintaxi bàsica per eliminar objectes utilitzant subconsultes és la que es mostra en la figura 3.11.

FIGURA 3.11. Sintaxi bàsica per eliminar objectes utilitzant subconsultes

```
DELETE FROM taula
        WHERE camp IN (SUBCONSULTA);
```

La subconsulta següent elimina tots els llibres de les editorials que tenen publicats llibres d'un autor determinat:

```
1 BEGIN
2 DELETE FROM llibre WHERE codi_editorial IN
3 (SELECT e.codi FROM editorial AS e
4 JOIN llibre ON codi_editorial = e.codi
5 WHERE autor = 'Antoni Llorac');
6 END;
```