Administració de sistemes gestors de bases de dades

CFGS.ASX.M10/0.12

Administració de sistemes informàtics en xarxa

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
Maribel Perramon Cuadrat

Redacció de continguts
Joan Anton Pérez Braña

Agraïments

A la meva companya Mª Àngels per la seva comprensió i paciència. A les meves filles Estel i Ona a qui he restat temps de dedicació i que han fet possible la redacció d'aquests materials gràcies a l'alegria que hem donen. A la meva mare sempre present.

Joan Anton Pérez Braña

Als meus fills Adrià i Helena, al meu company José Luis i als meus pares. A totes les persones que m'han ajudat i han fet possible aquest material.

Maribel Perramon Cuadrat

Primera edició: Febrer 2012

Material realitzat per Eureca Media, SL
Dipòsit legal: DL B 12718-2016

Introducció

Els coneixements sobre bases de dades són uns coneixements fonamentals per a qualsevol professional del món de la informàtica. Tant en l'àmbit del desenvolupament d'aplicacions com en de l'administració de sistemes, els coneixements de programació sobre bases de dades són imprescindibles per a gestionar qualsevol sistema informàtic, i per a configurar-lo per tal que aquest sistema tingui el funcionament que vol.

En aquest mòdul s'imparteixen coneixements que habiliten els estudiants per a:

- Implantar i gestionar bases de dades instal·lant i administrant el software de gestió en condicions de qualitat, segons les característiques de la explotació.
- Assegurar el sistema i les dades segons les necessitats d'ús i les condicions de seguretat establertes per a prevenir fallides i atacs externs.
- Administrar usuaris d'acord a les especificacions d'explotació per a garantir els accessos i la disponibilitat dels recursos del sistema.

La primera 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; per la unitat "Programació de bases de dades", en què s'ensenya a programar usant la extensió procedimental del llenguatge SQL PL/pgSQL.

En la segona unitat formativa, "Instal.lació i ajustament d'un SGBD corporatiu", es mostren les característiques dels SGBD, la seva arquitectura, la connectivitat client/servidor, els avantatges de l'ús dels sistemes de bases de dades distribuïts i les diferents tècniques emprades en l'administració d'una base de dades.

Com a última consideració cal tenir en compte que, aquest mòdul té un alt component pràctic i per aprendre els conceptes que apareixen en cada unitat i aplicar amb agilitat les tècniques esmentades, serà imprescindible implementar els exemples ilustratius, efectuar totes les activitats proposades i els exercicis d'autoavaluació. Si no es fa així, serà molt difícil assolir els coneixements que es presenten i avançar en la comprensió de nous conceptes.

Resultats d'aprenentatge

En finalitzar aquest mòdul l'alumne/a ha de ser capaç de:

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.

Instal.lació i ajustament d'un SGBD corporatiu

1. Implanta sistemes gestors de bases de dades corporatius analitzant les seves característiques i ajustant-se als requeriments del sistema.
2. Configura el sistema gestor de bases de dades corporatiu interpretant les especificacions tècniques i els requisits d'explotació.
3. Aplica criteris de disponibilitat analitzant-los i ajustant la configuració del sistema gestor de bases de dades corporatiu.
4. Optimitza el rendiment del sistema aplicant tècniques de monitoratge i realitzant adaptacions.

Continguts

Llenguatges SQL: DCL i extensió procedimental

Unitat 1

Gestió d'usuaris

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

Unitat 2

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

Instal.lació i ajustament d'un SGBD corporatiu

Unitat 3

Configuració i administració d'un SGBD

1. Implantació de sistemes gestors de bases de dades corporatius
2. Sistemes de bases de dades i comunicacions
3. SGBD distribuïts
4. Administració d'un SGBD

Gestió d'usuaris

Joan Anton Pérez Braña

Administració de sistemes gestors de bases de dades

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
```

```
6
7  SELECT
8
9  2 * 10 + 2
10
11 demo-# \g
12
13 ?column?
14
15 ----------
16
17 22
18
19 (1 row)
20
21 demo=#
```

El client també disposa d'una ordre **\r** que permet esborrar completament la memòria intermèdia per començar de nou amb la sentència:

```
1  demo=# select 'Hola Mon'\r
2
3  Query buffer reset (cleared).
4
5  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`:

```
CREATE USER nom_usuari [ [ WITH ] opcions [ ... ] ];
```

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.

També es poden crear usuaris des de l'intèrpret de comandes, o shell del sistema, amb la instrucció `createuser`.

Les opcions poden ser:

```
SYSID ID_usuari

CREATEDB | NOCREATEDB

CREATEUSER | NOCREATEUSER

IN GROUP nom_grup [, ...]

[ ENCRYPTED | UNENCRYPTED ]

PASSWORD 'password'

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`:

```
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`:

```
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:

```
createuser nom_del_paper

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:

```
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 :

```
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:

```
GRANT DELETE on Llibres to PUBLIC;
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.

```
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.

```
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.

```
ALTER VIEW nom_vista OWNER TO nou_propietari
```

Reanomenar una vista

```
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.

```
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

Administració de sistemes gestors de bases de dades

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

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.

CREATE

Si es declaren diverses columnes amb *serial* en una taula, es crearà una seqüència i un índex per a cada una.

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"

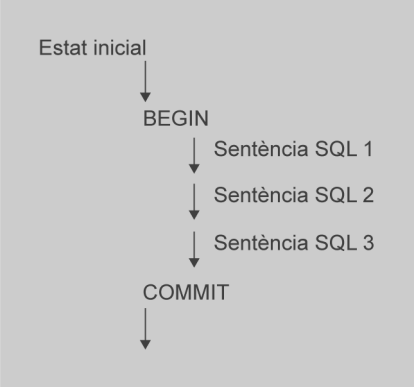


Figura 1.2. Funcionament d'una acció "ROLLBACK"

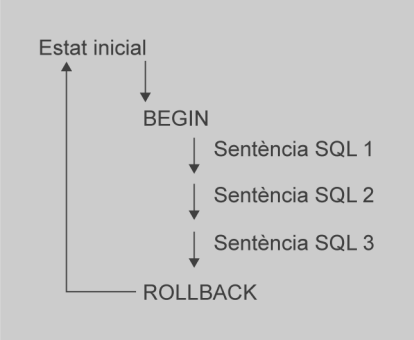
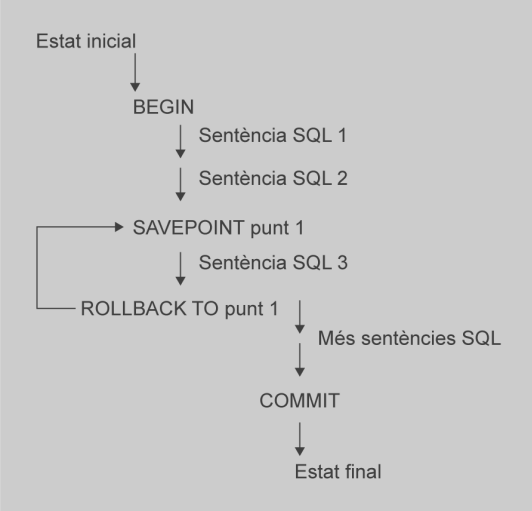


Figura 1.3. Funcionament d'una acció "SAVEPOINT"



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* :

```
sessio1=#\i sample.sql
CREATE TABLE
CREATE TABLE
...
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:

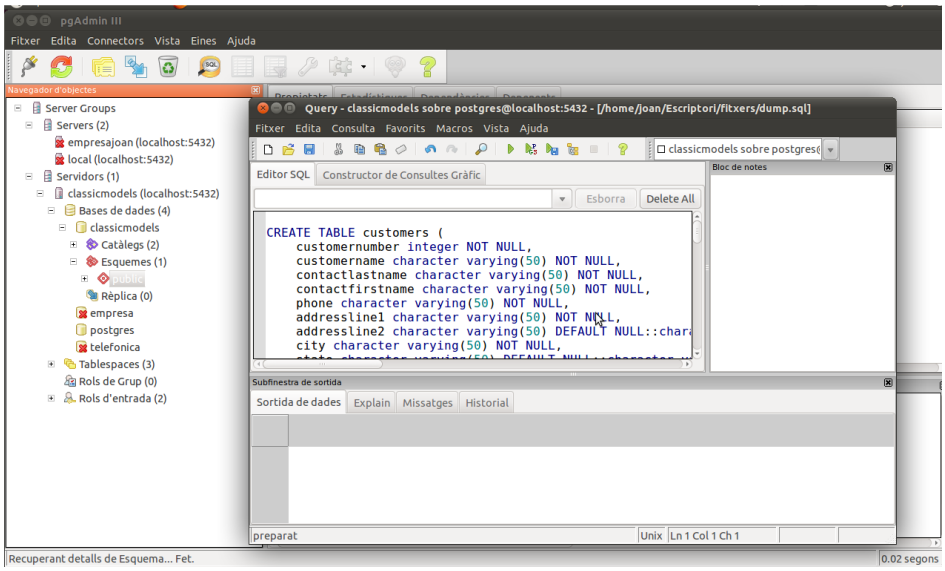
```
$ psql demo < demo.psql
```

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
```

```
21 END;
22 RAISE NOTICE 'la quantitat val aquí %', quantitat;
23 -- mostra 50
24 RETURN quantitat;
25 END;
26 $$ 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:

```
1 user_id integer;
2 quantity numeric(5);
3 url varchar;
4 myrow tablename%ROWTYPE;
5 myfield tablename.columnname%TYPE;
6 arow RECORD;
```

La sintaxi general d'una declaració de variable és:

```
1 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`.

Els camps del tipus de fila hereten la mida del camp de la taula o la precisió dels tipus de dades com `char(n)`.

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.

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:

```
CREATE FUNCTION sales_tax(subtotal real) RETURNS real AS $$
BEGIN
RETURN subtotal * 0.06;
END;
$$ 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:

```
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:

```
CREATE FUNCTION sales_tax(real) RETURNS real AS $$
DECLARE
subtotal ALIAS FOR $1;
BEGIN
RETURN subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;
```

Alguns exemples més:

```
CREATE FUNCTION instr(varchar, integer)
RETURNS integer AS $$
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:

```
CREATE FUNCTION suma_un (int4) RETURNS int4 as $$
BEGIN
return $1 + 1;
end;
$$ 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:

```
CREATE FUNCTION suma_un (float8) RETURNS float8 as $$
BEGIN
return $1 + 1;
end;
$$ language plpgsql;
```

I veurem que ara accepta qualsevol de les execucions següents:

```
SELECT suma_un(3);
SELECT suma_un(3.1);
```

Per veure el codi d'aquestes funcions podem executar:

```
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`.

```
if <condició> then
<conjunt_de_sentències_si_la_condició_s'avalua_com_a_certa>;
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`.

```
if <condició> then
<conjunt_de_sentències_si_la_condició_s'avalua_com_a_certa>;
else
<conjunt_de_sentències_si_la_condició_s'avalua_com_a_falsa>;
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.

```
WHEN division_by_zero THEN ...
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:

```
INSERT INTO mytab(firstname, lastname) VALUES('Pere', 'Martí');
BEGIN
UPDATE mytab SET firstname = 'Josep' WHERE lastname = 'Martí';
x := x + 1;
y := x / 0;
EXCEPTION
WHEN division_by_zero THEN
RAISE NOTICE 'capturat error division_by_zero';
RETURN x;
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:

```
CREATE TRIGGER nom { BEFORE | AFTER } { esdeveniment [ OR ... ] }
ON taula [ FOR [ EACH ] { ROW | STATEMENT } ]
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:

```
1  CREATE TABLE empleat (
2  nom_empleat text,
3  salari integer,
4  darrera_modif timestamp,
5  darrer_usuari text
6  );
```

Crearem la funció segons el que hem comentat:

```
1  CREATE FUNCTION reg_empleat() RETURNS trigger AS $reg_empleat$
2  BEGIN
3     -- Comprova que s'informi el nom_empleat i el salari.
4     IF NEW.nom_empleat IS NULL THEN
5        RAISE EXCEPTION 'nom_empleat no pot ser null';
6     END IF;
7     IF NEW.salari IS NULL THEN
8        RAISE EXCEPTION 'el salari de % no pot ser null,
9        NEW.nom_empleat;
10    END IF;
11    -- Comprova que el salari sigui positiu.
12    IF NEW.salari < 0 THEN
13       RAISE EXCEPTION 'el salari  de % no pot ser negatiu',
14       NEW.nom_empleat;
15    END IF;
16 -- Registre de qui ha fet la modificació
17    NEW.darrera_modif := 'now';
18    NEW.darrer_usuari := current_user;
19    RETURN NEW;
20 END;
21 $reg_empleat$ 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:

```
ALTER TRIGGER nom ON taula RENAME TO nou_nom
```

O eliminar-lo:

```
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:

```
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.

Configuració i administració d'un SGBD

Joan Anton Pérez Braña

Administració de sistemes gestors de bases de dades

Índex

Introducció

Actualment, a la majoria d'empreses i organitzacions els seria molt difícil de dur a terme la seva activitat sense el suport dels sistemes informàtics, i més concretament sense les dades que aquests emmagatzemen.

Bona part de la informació de les organitzacions s'estructura en bases de dades (BD), les quals són gestionades mitjançant un programari conegut com a sistema gestor de bases de dades (SGBD).

En aquesta unitat formativa veurem les característiques dels diferents components lògics i físics emprats en l'emmagatzematge de dades.

En l'apartat "Implantació de sistemes gestors de bases de dades corporatius" farem una valoració de les característiques dels sistemes gestors de bases de dades respecte al guany que suposen quant a prestacions i eficiència en relació amb els fitxers i d'altres sistemes de gestió de la informació; també veurem les característiques dels diferents components lògics i físics emprats en l'emmagatzematge de dades, i aprendrem a reconèixer l'entorn específic en el qual posteriorment instal·larem un SGBD. Així mateix, farem una valoració dels principals aspectes que s'han de tenir en compte per escollir el producte més adient per a les nostres necessitats.

La possibilitat d'oferir la informació globalment, a través de les xarxes, juntament amb l'evolució de la tecnologia de maquinari, ha fet replantejar la manera d'organitzar les dades. En l'apartat "Sistemes de bases de dades i comunicacions" veurem les raons per les quals s'ha estès l'ús d'arquitectures client-servidor. En l'apartat "SGBD distribuïts" s'analitza la utilitat de distribuir les BD en diverses fonts o nodes.

Les persones que configuren els SGBD i gestionen les BD que aquests emmagatzemen són els administradors de BD (DBA, acrònim de *database administrator*, en anglès). Els DBA són un perfil professional especialitzat a instal·lar, configurar, administrar i desinstal·lar SGBD, i gestionar l'explotació de les BD de manera segura i eficient. En l'apartat "Administració d'un SGBD" veurem el conjunt de tasques que corresponen a l'administrador del SGBD.

La gran quantitat de conceptes que s'introdueixen en aquesta unitat formativa fa necessari posar-los a la pràctica en el SGBDR amb el que treballarem, en el nostre cas PostgreSQL.

És molt recomanable, doncs, la pràctica en els vostres ordinadors tant provant els exemples que es descriuen en el material en paper, com aprofundint en els annexos i duent a terme i posant en pràctica les activitats del material web.

Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Implanta sistemes gestors de bases de dades corporatius analitzant les seves característiques i ajustant-se als requeriments del sistema.
2. Configura el sistema gestor de bases de dades corporatiu interpretant les especificacions tècniques i els requisits d'explotació.
3. Aplica criteris de disponibilitat analitzant-los i ajustant la configuració del sistema gestor de bases de dades corporatiu.
4. Optimitza el rendiment del sistema aplicant tècniques de monitoratge i realitzant adaptacions.

1. Implantació de sistemes gestors de bases de dades corporatius

Comencem aquesta unitat formativa amb l'estudi de les característiques dels diferents components emprats en l'emmagatzematge de dades tant des del punt de vista lògic com físic.

Veurem també les característiques i l'evolució dels sistemes gestors de bases de dades i n'analitzarem l'arquitectura amb l'objectiu de poder fer la instal·lació d'un SGBD d'acord amb les característiques de l'entorn de treball.

1.1 Utilització de fitxers segons la seva organització

En els dispositius d'emmagatzematge secundari, la base de dades s'organitza en un o diversos fitxers.

Cada fitxer està format per una sèrie de registres, i cada registre es divideix en diversos camps.

Normalment, els registres corresponen a entitats: persones, objectes, esdeveniments, etc., i els camps són aquells atributs o propietats que volem conèixer sobre les entitats: el nom de cada persona, el color de l'objecte, la data de l'esdeveniment, etc.

Exemple de fitxer de la plantilla d'una empresa

El fitxer amb informació sobre la plantilla d'una empresa estaria format per registres d'empleats amb els camps següents: número de l'empleat, nom, cognoms, càrrec, DNI i número de l'oficina a què pertany.

Quan un usuari demana les dades de l'empleat EG37, l'SGBD identifica el bloc de disc que conté les dades esmentades i les demana al sistema operatiu. Aquest utilitzarà les seves rutines d'accés a fitxers per obtenir el bloc i l'emmagatzemarà en les memòries intermèdies de l'SGBD en memòria principal.

Un bloc o pàgina és la unitat mínima de transferència entre disc i memòria principal.

Normalment, caben diversos registres de dades en un mateix bloc. Si un registre no cap en un sol bloc, es repartirà entre uns quants.

L'ordre en el qual es col·loquen els registres en un fitxer depèn de la seva estructura. Els principals tipus d'estructures de fitxers són els següents:

- **Fitxers desordenats**. En aquests fitxers els registres no segueixen un ordre específic.

- **Fitxers ordenats.** En aquests fitxers els registres estan ordenats pel valor d'un camp determinat.
- **Fitxers dispersos.** En aquests fitxers els registres s'emmagatzemen en la posició del fitxer que indica una funció matemàtica en ser aplicada sobre un camp determinat.
- **Agrupament.** Diversos fitxers intercalen els seus registres de manera que aquests quedin agrupats pel valor d'algun camp que tenen en comú.

Els passos que s'han de dur a terme per emmagatzemar un registre d'un fitxer i accedir-hi és el que es denomina un *mètode d'accés*. Hi ha diferents mètodes d'accés, alguns dels quals només es poden utilitzar amb determinades estructures de fitxers.

1.2 Índexs

Els índexs són estructures d'accés que s'utilitzen per accelerar l'accés als registres en resposta a certes condicions de cerca.

Alguns tipus d'índexs no afecten l'emplaçament físic dels registres al disc i el que fan és proporcionar camins d'accés alternatius per trobar els registres de manera eficient basant-se en els camps d'indexació.

Tanmateix, els tipus d'índexs que més s'utilitzen són els que es basen en fitxers ordenats (índexs d'un sol nivell) i les estructures en forma d'arbre (índexs multinivell, arbres B i arbres B+). A més, els índexs es poden construir mitjançant dispersió o altres estructures de dades.

1.3 Dispositius d'emmagatzematge

Per assegurar la perdurabilitat de les dades, cal emmagatzemar-les en dispositius d'emmagatzemament no volàtil. Aquest és el cas dels anomenats *discos durs*. Les característiques principals d'un disc dur són la seva capacitat d'emmagatzematge, actualment d'uns quants gigabytes (GB) a un terabyte (TB), la velocitat de transferència (*throughput*, en MB/s), i el temps d'accés (en ms), que alhora està condicionat per la velocitat de rotació (rpm o rotacions per minut). Així doncs, la capacitat i velocitat dels discos són els dos aspectes bàsics i més importants que cal tenir en compte a l'hora de triar els que es volen posar en els servidors que allotjaran el nostre SGBD.

Fa uns anys, les dues opcions bàsiques en els discos durs eren els ATA (també coneguts com a IDE i més barats) utilitzats en equips de sobretaula, enfront

dels discos SCSI, més cars i destinats a servidors. Això es coneix com a *emmagatzematge adjuntat directament* (DAS).

Fiabilitat

Cal tenir present que si falla un disc totes les particions que conté queden inaccessibles i no es pot accedir a la informació que emmagatzemen.

La necessitat de sistemes d'emmagatzemament a gran escala i la importància de la fiabilitat ha dut a la introducció de noves configuracions, com per exemple RAID (*redundant array of independent/inexpensive disks*) i la distribució de les dades en l'emmagatzemament vinculat a la xarxa, com per exemple NAS (*network attached storage*).

Tant en DAS com en SAN (*storage area network*) les aplicacions i programes d'usuaris fan les seves peticions de dades al sistema de fitxers directament. La diferència entre totes dues tecnologies és la manera com el sistema de fitxers obté les dades requerides. En una DAS, l'emmagatzematge és local respecte al sistema de fitxers, mentre que en una SAN l'emmagatzematge és remot. En la banda oposada es troba la tecnologia NAS (*network attached storage* o emmagatzematge adjuntat a la xarxa), en què les aplicacions fan les peticions de dades als sistemes de fitxers de manera remota.

1.3.1 Direct attached storage (DAS)

Direct attached storage (DAS), o emmagatzematge adjuntat directament, és el mètode tradicional d'emmagatzematge i el més senzill. Consisteix a connectar el dispositiu d'emmagatzematge directament al servidor o estació de treball, és a dir, físicament connectat al dispositiu que en fa ús.

Fonamentalment hi ha quatre tipus de dispositius:

1. *Intelligent drive electronics* (IDE).
2. *Serial ATA* (SATA).
3. *Small computer system interface* (SCSI).
4. *Serial attached SCSI* (SAS).

Característiques

Els principals protocols utilitzats en DAS són SCSI, SAS i *fibre channel*. Habitualment un sistema DAS habilita capacitat extra d'emmagatzematge en un servidor, mentre manté altes l'amplada de banda i les taxes d'accés.

El típic sistema DAS està format per un o més dispositius d'emmagatzematge, com discos durs, i un o més controladors. La interfície amb el servidor o amb l'estació de treball està feta per mitjà d'un *host bus adapter* (HBA).

Un típic sistema DAS proveeix controladors embeguts. La gestió del RAID és sense càrrega o simplement sense RAID. Els HBA poden ser usats amb reducció de costos.

Desavantatges

Els DAS han estat considerats com a "illes d'informació". Els seus desavantatges inclouen la incapacitat per compartir dades o fonts no usades amb altres servidors. Tant les arquitectures NAS com les SAN intenten adreçar això, però presenten alguns aspectes nous, també, com l'alt cost inicial i la gestió, seguretat i contenció de recursos.

1.3.2 RAID

L'enfocament RAID (matriu redundant de discos independents) és la manera estàndard per manejar tant el rendiment com la fiabilitat de les limitacions de les unitats de disc individuals. Un arranjament RAID es basa en un sistema d'emmagatzemament de la informació que combina diversos discos durs d'igual capacitat que davant del sistema funcionen com una única unitat lògica.

En alguns casos la fiabilitat addicional prové de la informació, que es coneix com a *informació de paritat*, que s'escriu a la matriu. La paritat és una forma de suma de comprovació de les dades, cosa que permet la reconstrucció, encara que alguna part de la informació es perdi.

Els nivells de RAID que usen la paritat són eficients des del punt de vista de l'espai en l'escriptura de dades d'una manera que permet sobreviure a errors en el disc, però la sobrecàrrega de càlcul de paritat pot ser important per a les aplicacions de base de dades.

Les formes bàsiques més comunes d'arranjaments RAID utilitzats són:

- **RAID 0:** també es coneix com a creació de bandes. S'utilitzen diversos discos al mateix temps, de manera que es llegeix i s'escriu en cadascun en paral·lel. Això pot ser gairebé una millora lineal, ja que dos discos de lectura seran el doble de ràpid que un de sol, però cal tenir en compte que una fallida en qualsevol volum del conjunt provocarà la pèrdua de totes les dades.

- **RAID 1:** també anomenat *mirall*. Cada unitat està duplicada amb una unitat de suport. Per tant, amb quatre unitats de disc, dos són de còpia. La informació es distribueix entre els discos. En general, la raó de RAID 1 és la redundància: si una unitat falla, el sistema funcionarà amb l'altra, que fa de mirall i en què les dades emmagatzemades són idèntiques a la que ha fet fallida.

- **RAID 10 o 1+0:** és una forma mixta dels dos anteriors. Replica un grup de RAID 1 en un grup de discos amb RAID 0. El resultat proporciona un alt rendiment i la capacitat de tolerar qualsevol error en el disc. És especialment adequat per a entorns d'escriptura pesada, en què la sobrecàrrega de càlcul

de paritat dels RAID 5 o RAID 6 pot baixar el rendiment de disc en operacions d'escriptura. En conseqüència, és el nivell de RAID preferit per als sistemes de base de dades d'alt rendiment.

- **RAID 5:** també es coneix com a *bandes de paritat*. Aquest enfocament es troba a mig camí entre 0 i 1. S'agreguen dades redundants a una unitat de paritat. Permet accés independent amb paritat distribuïda. Si un dels discos de la matriu es perd, les dades que falten es poden tornar a calcular a partir de la informació de paritat. En resulta un excel·lent rendiment i bona tolerància a fallides. És una de les configuracions més utilitzades, encara que cal tenir en compte la sobrecàrrega de feina que implica l'actualització de la informació en la unitat de paritat.

- **RAID 6:** similar al RAID 5, amb l'excepció que conté més informació de paritat, cosa que permet la supervivència, fins i tot amb dos discos. Permet l'accés independent amb doble paritat. Té els mateixos avantatges i desavantatges fonamentals. RAID 6 és una forma cada vegada més comuna per fer front al fet que la reconstrucció d'un RAID 5 després d'una pèrdua de disc pot trigar molt de temps en unitats modernes d'alta capacitat.

La tècnica del RAID millora el rendiment, en distribuir la informació entre diverses unitats, i pot oferir redundància per augmentar la seguretat.

El RAID pot ser per programari o per maquinari. Si és per programari és més lent, i si és per maquinari és transparent al sistema operatiu.

1.3.3 SAN

Una xarxa d'àrea d'emmagatzematge, en anglès *storage area network* (SAN), és una xarxa concebuda per connectar servidors, matrius (*arrays*) de discos i biblioteques de suport. Principalment, està basada en tecnologia *fibre channel* i més recentment en iSCSI. La seva funció és la de connectar de manera ràpida, segura i fiable els diferents elements que la conformen.

Una xarxa SAN es distingeix d'altres formes d'emmagatzematge en xarxa per la manera d'accés a baix nivell. El tipus de trànsit en una SAN és molt similar al dels discos durs com ATA, SATA i SCSI. En altres mètodes d'emmagatzematge (com SMB o NFS) el servidor requereix un fitxer determinat, per exemple, */home/usuari/fitxer*. En una SAN el servidor requereix *el bloc 6000 del disc 4*. La majoria de les SAN actuals utilitzen el protocol SCSI per accedir a les dades de la SAN, encara que no facin servir interfícies físiques SCSI. Aquest tipus de xarxes de dades s'han utilitzat i s'utilitzen tradicionalment en grans ordinadors centrals, com en IBM, SUN o HP.

Dos protocols de xarxa utilitzats en una SAN són *fibre channel* i iSCSI. Una xarxa de canal de fibra és molt ràpida i no està atabalada pel trànsit de la xarxa LAN de l'empresa.

Estructura bàsica d'una SAN

Les SAN proveeixen connectivitat d'E/S a través dels ordinadors centrals i els dispositius d'emmagatzematge combinant els beneficis de tecnologies *fibre channel* i de les arquitectures de xarxes, i ofereix així una aproximació més robusta, flexible i sofisticada que supera les limitacions de DAS emprant la mateixa interfície lògica SCSI per accedir a l'emmagatzematge.

Les SAN es componen de tres capes:

- **Capa d'hoste**. Aquesta capa consisteix principalment en servidors, dispositius o components (HBA, GBIC, GLM) i programari (sistemes operatius).
- **Capa de fibra**. Aquesta capa la conformen els cables (fibra òptica) i els concentradors i commutadors SAN com a punt central de connexió per a la SAN.
- **Capa d'emmagatzematge**. Aquesta capa la componen les formacions de discos (matrius de discos, memòria cau, RAID) i cintes emprats per emmagatzemar dades.

La xarxa d'emmagatzematge pot ser de dos tipus:

- **Xarxa *fibre channel***. La xarxa *fibre channel* és la xarxa física de dispositius *fibre channel* que empra commutadors *fibre channel* i directors i el protocol *fibre channel* (FCP) per a transport (SCSI-3 en sèrie sobre *fibre channel*).
- **Xarxa IP**. Empra la infraestructura de l'estàndard LAN amb concentradors o commutadors Ethernet interconnectats. Una SAN IP fa servir iSCSI per a transport (SCSI-3 en sèrie sobre IP).

1.3.4 NAS

NAS (de l'anglès *network attached storage*) és el nom donat a una tecnologia d'emmagatzematge dedicada a compartir la capacitat d'emmagatzematge d'un computador (servidor) amb ordinadors personals o servidors clients a través d'una xarxa (normalment TCP/IP), fent ús d'un sistema operatiu optimitzat per donar accés amb els protocols CIFS, NFS, FTP o TFTP.

Generalment, els sistemes NAS són dispositius d'emmagatzematge específics a què s'accedeix des dels equips per mitjà de protocols de xarxa (normalment TCP/IP). També es podria considerar un sistema NAS un servidor (Linux, Windows...) que comparteix les seves unitats per xarxa, però la definició se sol aplicar a sistemes específics.

Els protocols de comunicacions NAS estan basats en fitxers; d'aquesta manera el client demana el fitxer complet al servidor i el maneja localment, i per aquesta raó estan orientats a informació emmagatzemada en fitxers de petita grandària i gran quantitat. Els protocols usats són protocols de compartició de fitxers com NFS o el *Microsoft common Internet file system* (CIFS).

Molts sistemes NAS tenen un o més dispositius d'emmagatzematge per incrementar la seva capacitat total. Normalment, aquests dispositius estan disposats en RAID (*redundant arrays of independent disks*) o contenidors d'emmagatzematge redundant.

Un dispositiu maquinari simple, anomenat *NAS box* o *NAS head*, actua com a interfície entre el NAS i els clients. Els clients sempre es connecten al *NAS head* (més que als dispositius individuals d'emmagatzematge) per mitjà d'una connexió Ethernet. El NAS apareix a la LAN com un simple node, que és l'adreça IP del dispositiu *NAS head*.

Aquests dispositius NAS no requereixen pantalla, ratolí o teclat, sinó que posseeixen interfície web.

1.4 Suports de còpies de seguretat

Els dispositius de còpia de seguretat són els aparells físics que s'utilitzen per fer còpies de seguretat de la informació dels servidors. Normalment les còpies són procediments que triguen hores a enllestir-se, i també es triga una hora o més a recuperar els fitxers del dispositiu en què s'ha emmagatzemat la informació.

Davant del problema de copiar la informació de l'organització per evitar-ne la pèrdua hi ha molts dispositius (aparells físics) i diferents tècniques. Hem de saber escollir la combinació que s'ajusti millor a les nostres necessitats.

Per fer còpies de seguretat hi ha disponibles diversos dispositius: cintes, biblioteques de còpia, gravadores DVD i discos durs. L'elecció del dispositiu de còpia de seguretat depèn sempre de la mida de l'organització, el volum de dades, el cost, etc.

1.4.1 Cintes

Fins fa poc les cintes eren el dispositiu més habitual als servidors. Quan es parla de capacitats de les unitats de cinta cal tenir en compte que n'hi ha amb compressió o sense.

Hi ha diferents tipus de dispositius de cinta:

- *Digital audio tape* (DAT)

- *Digital linear tape* (DLT)
- *Advanced intelligent tape* (AIT)
- *Linear tape open* (LTO)

1.4.2 Biblioteques de còpia

Es pot donar el cas que la nostra organització manipuli quantitats de dades que ocupin diverses cintes de còpia al dia. En aquest cas, una sola persona es passaria el dia fent còpies de seguretat, i no acabaria mai. Quina és la solució per a aquests volums d'informació tan grans? Hi ha uns dispositius anomenats *biblioteques de còpia*. Són externs, amb uns braços articulats, i contenen des de vint fins a dues mil cintes de còpia de seguretat (són com robots).

1.4.3 Enregistrament de discos òptics

Una unitat de cinta és un component car i moltes vegades resulta difícil accedir (demana un temps considerable) a les dades que contenen. Per això, de vegades es considera la gravadora de DVD com una opció de còpies de seguretat. Ens permetrà fer còpies de DVD gravables i regravables de 4,7 i 9,4 GB. Tot i que aquests són els volums més estàndard, podem trobar volums de DVD gravables una vegada fins de 17 GB.

Actualment, els DVD gravables una vegada són una opció que cal tenir en compte en plantejar-se les còpies de seguretat, ja que tenen els avantatges següents:

- El cost de compra del dispositiu és baix.
- El cost de les unitats de còpia és baix.
- Són de gran capacitat (fins a 17 GB).
- Ofereixen una gran facilitat per accedir a la informació guardada.

En poc temps veurem com el Blu Ray substitueix el DVD com a dispositiu de còpia.

1.4.4 Discos durs

En sistemes crítics, i més tenint en compte el cost i la capacitat actual d'aquests dispositius, no s'ha de descartar mai la possibilitat de fer una còpia de seguretat (o fins i tot de copiar tota la informació) en un altre disc dur només dedicat a aquesta

funció. L'estratègia és fer una primera còpia de seguretat en aquest disc dur (es pot fer amb un procediment automàtic i diverses vegades al dia, si cal), i d'aquest disc, posteriorment, se'n farà una còpia de seguretat en un altre dispositiu (que pot ser una cinta).

De vegades, aquesta estratègia és necessària si el procediment de còpia necessita bloquejar la informació a la qual accedeix i és, per exemple, una gran base de dades de la qual depèn tota l'organització. La còpia de disc a disc, en funcionar internament, pels busos del sistema i amb velocitats de transferència molt elevades, necessita bloquejar molt poc temps la informació per fer la còpia. Així, doncs, la interrupció per fer aquesta tasca és pràcticament imperceptible.

Gràcies a la proliferació de xarxes SAN o dispositius NAS que permeten una gran quantitat d'espai per emmagatzemar, s'utilitzen cada cop més els discos com a dispositiu de còpia. Els proveïdors mateixos ofereixen eines específiques que permeten fer aquestes còpies transparents al sistema mateix.

1.5 Altres tipus d'emmagatzematge d'informació

Independentment dels sistemes de fitxers esmentats i dels dispositius físics emprats per a l'emmagatzemament d'informació, avui dia s'utilitzen altres sistemes per emmagatzemar informació. Aquests són els *fitxers XML* i els *serveis de directori*.

1.5.1 XML

Als anys noranta, en resposta a la guerra comercial entre els navegadors web Netscape i Explorer, que provocava greus desviacions del llenguatge HTML respecte del seu propòsit original, la comunitat internacional va crear el W3C (World Wide Web Consortium), que havia de vetllar per una estandardització real dels formats de la Xarxa.

Una de les primeres tasques del W3C fou oficialitzar un versió unificada d'HTML.

World Wide Web Consortium

Podeu consultar el web del World Wide Web Consortium a: www.w3.org

Com que l'HTML no podia donar resposta a alguns dels usos que s'esperava de la Xarxa (com ara el comerç electrònic, la cerca d'informació, la personalització, etc.), l'any 1998 el W3C va publicar la versió 1.0 de l'especificació XML (*extensible markup language*). L'objectiu d'aquesta especificació és crear un SGML senzill, és a dir, un subconjunt d'SGML que n'elimini les parts més complexes i l'optimitzi per a l'ús a la Xarxa sense deixar de ser, això sí, extensible (el nombre d'etiquetes permeses no és tancat, sinó que és possible definir-ne de pròpies mitjançant la definició d'un DTD o un XML Schema). La finalitat de tot plegat és la de crear un llenguatge prou senzill perquè la indústria consideri rendible invertir en la creació d'eines per tractar-les.

Especificació XML

Podeu trobar l'especificació d'XML: www.w3.org/TR/2006/REC-xml11-20060816/

L'XML és un llenguatge de marques generalitzat que pretén dotar els continguts (documents) d'estructura lògica de manera que puguin ser manipulats com a dades per una màquina i es pugui intercanviar informació amb independència de la plataforma, l'arquitectura i la base de dades.

XML en 10 punts

Podeu visitar: www.w3.org/XML/1999/XML-in-10-points.ca.html

Un document XML és un text en una codificació de caràcters concreta. El text XML es pot considerar com un conjunt de dades de caràcter i marques. Les marques són text que comença pel caràcter < i acaba pel caràcter >, i també, text que comença pel caràcter & i acaba pel caràcter ;. Les dades de caràcter són tot allò que no són marques.

A més alt nivell, un document XML té dues parts, la capçalera i la instància de document:

```
<? xml version="1.0" encoding="ISO-8851-1" ?>
<? xml-stylesheet type="text/xsl" href="mail.xsl" ?>
<! DOCTYPE SYSTEM "mail.dtd">
<!-- això és un comentari. Permet anotar la fi de la capçalera -->

<!-- A partir d'aquest punt tenim la instància del document -->
<mail>
  <from>joan@ioc.cat</from>
  <to>estudiants@ioc.cat</to>
  <subject>Exemple XML </subject>
  <body> Això és un exemple de XML.</body>
</mail>
```

A la capçalera apareix informació per a la gestió del document, com ara codificació, definició del tipus de document, presentació associada, etc.

XML

Teniu una aproximació al llenguatge XML a la Wikipedia: ca.wikipedia.org/wiki/Extensible_Markup_Language

La instància de document representa el contingut real del document i està constituïda per elements, atributs i dades. S'anomena *instància* perquè és una cadena de caràcters que representa un exemplar concret d'un document o d'una tipologia determinada de document.

1.5.2 Serveis de directori (LDAP)

Considerarem una organització que vol posar les dades dels seus empleats a disposició dels diferents membres de l'organització; entre aquestes dades tindríem el nom i cognoms de l'empleat, l'adreça postal i electrònica, telèfon i fax etc. Anteriorment aquestes organitzacions creaven directoris físics dels empleats i els distribuïen per tota l'organització. Encara deveu recordar els directoris físics dels clients de les companyies telefòniques. En general un directori és una llista d'informació sobre algun tipus d'objecte, com per exemple persones. Els directoris es poden emprar per cercar informació sobre un objecte concret o en sentit invers per poder trobar aquells objectes que compleixen un determinat requisit. En l'exemple dels directoris físics telefònics aquells que permeten fer cerques en sentit directe els anomenem *pàgines blanques*, mentre que aquells que permeten cerques en sentit invers els anomenem *pàgines grogues*.

Avui dia, amb un mon interconnectat, continua vigent la necessitat d'aquests directoris i fins i tot és més important usar-ne. No obstant això, aquests directoris, en lloc d'emprar un suport físic, utilitzen un suport informàtic.

Protocols d'accés a directoris

La informació de directori es pot deixar disponible mitjançant interfícies web, com fan les companyies telefòniques. Aquestes interfícies són bones quan són les persones les que accedeixen a aquesta informació; no obstant això, també els programes accedeixen a aquest tipus d'informació.

S'han desenvolupat diversos protocols d'accés a directoris per oferir una manera normalitzada d'accés a les dades dels directoris. Entre aquests, el més utilitzat avui dia és el protocol d'accés lleuger a directoris (*lightweight directory access protocol*, LDAP).

Evidentment, tots els tipus de dades dels exemples que vénen a continuació s'emmagatzemen sense gaire problemes en sistemes de bases de dades i s'hi accedeix mitjançant protocols com JDBC o ODBC. La pregunta, llavors, és el motiu per haver creat un protocol especialitzat per a l'accés a la informació de directori. Inicialment ja tenim dues respostes a aquesta pregunta.

En primer lloc, els protocols d'accés a directoris són protocols simplificats que atenen un tipus limitat d'accés a les dades. Han evolucionat en paral·lel amb els protocols d'accés a les bases de dades.

En segon lloc, i el que és més important, els sistemes de directori ofereixen un mecanisme senzill per anomenar els objectes de manera jeràrquica, semblant als noms dels directoris d'un sistema d'arxius, i es poden utilitzar en un sistema distribuït de directori per especificar la informació que s'emmagatzema en cada servidor de directori. Per exemple, pot ser que un servidor de directori concret emmagatzemi la informació dels empleats de Laboratoris Well de Ripoll i que un altre emmagatzemi la informació dels empleats de Laboratoris Well a Badalona, la qual cosa permet a tots dos llocs autonomia per controlar les dades locals.

Es pot utilitzar el protocol d'accés al directori per obtenir dades dels dos directoris per la xarxa. El que és més important, el sistema de directoris es pot configurar perquè enviï de manera automàtica a un lloc les consultes formulades en l'altre, sense intervenció de l'usuari.

Per aquests motius algunes organitzacions tenen sistemes de directoris per fer que la informació de l'organització estigui disponible en connexió.

El protocol d'accés lleuger a directoris LDAP (lightweight directory access protocol)

En general, els sistemes de directoris s'implementen como un o diversos servidors que atenen diversos clients. Els clients utilitzen la interfície de programació d'aplicacions definida pel sistema de directoris per comunicar-se amb els servidors

de directoris. Els protocols d'accés a directoris també defineixen un model de dades i el control dels accessos.

LDAP Podeu trobar més informació sobre LDAP a: goo.gl/oq34C

El protocol d'accés a directoris X.500, definit per l'Organització Internacional per a la Normalització (International Organization for Standardization, ISO), és una norma per a l'accés a informació dels directoris. No obstant això, el protocol és bastant complex i no s'utilitza gaire. El protocol d'accés lleuger a directoris (*lightweight directory access protocol*, LDAP) ofereix moltes de les característiques d'X.500, però amb menys complexitat, i s'utilitza bastant.

1.6 Evolució i funcions dels SGBD

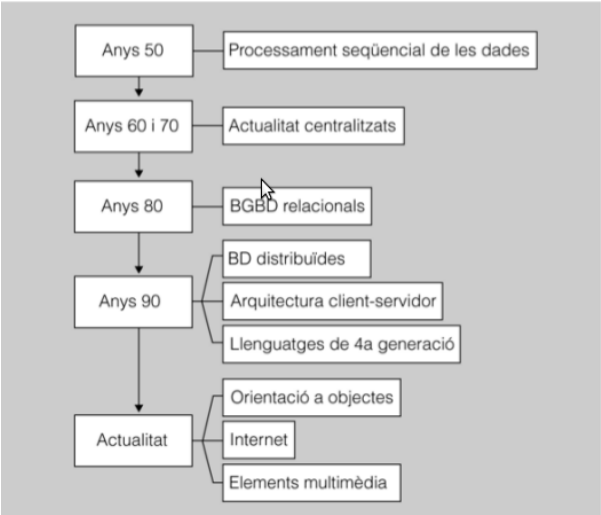
Els SGBD són un tipus de programari que té com a finalitats la gestió i el control de les BD.

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.

1.6.1 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.

Figura 1.1. Evolució de les bases de dades | |cc



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

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 fer 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 fer 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 faran 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 fer 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).

Magatzems de dades

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 fer anàlisis estratègiques d'índole financera, de mercats, etc.

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).

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.

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.

1.6.2 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, l'SGBD ha de ser capaç de respondre ell mateix a la consulta formulada, sense que sigui necessari recórrer a cap aplicació externa.

Quan no hi havia 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 una llista 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 per l'SGBD com pels 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 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 per 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.

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

Són dades íntegres les que es mantenen senceres i correctes.

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 de l'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, l'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.

Les regles d'integritat poden ser de dos tipus:

- **Restriccions del model.** Són regles inherents al model de dades que utilitza l'SGBD (com ara el model relacional). El sistema les incorpora predefinides, i sempre s'acompleixen.

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.

- **Restriccions de l'usuari.** Són regles definides pels usuaris (dissenyadors i administradors, fonamentalment) de la BD, que no incorpora *a priori* l'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 hagi alumnes matriculats en cicles formatius de grau superior que siguin 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 fer 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 fer 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

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).

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

- 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, l'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.

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 les remuneracions dels empleats, com ara el salari mitjà.

"COMMIT" i "ROLLBACK"

La instrucció `COMMIT` indica, a l'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.

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

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.

Exemple de dades confidencials

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 té el càrrec de director general, que les podrà consultar per tal d'optimitzar la ubicació dels treballadors i l'organigrama en funció del perfil respectiu, 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, tret 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.

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.

Un altre aspecte que cal tenir en compte en parlar de la seguretat de les dades és el xifratge. Molts SGBD ofereixen aquesta possibilitat, en alguna mesura.

Les tècniques de xifratge 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.

El xifratge 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 xifrar les contrasenyes.

1.7 Models de bases de dades

Un model de dades és bàsicament una "descripció" d'una cosa coneguda com a *contenidor de dades* (on es desa la informació), i també dels mètodes per desar i recuperar informació d'aquests contenidors. Els models de dades no són coses físiques: són abstraccions que permeten la implementació d'un sistema eficient de base de dades; per norma general es refereixen algoritmes i conceptes matemàtics.

Conèixer les característiques dels diferents models de bases de dades ens permetrà posteriorment fer una tria correcta quan hàgim de fer la implantació d'un SGBD corporatiu.

1.7.1 Bases de dades en xarxa

L'organització Conference on Data Systems Languages (CODASYL), formada per venedors de maquinari, venedors de programari i els usuaris més importants, va normalitzar el llenguatge COBOL al començament dels anys seixanta. Al final d'aquesta dècada, el consorci va anomenar un subgrup, el Database Task Group (DBTG) per tal que desenvolupés les normes que havien de regir els SGBD. Aquest grup estava fortament influenciat pel primer SGBD, l'Integrated Data Store (IDS), desenvolupat per General Electric, que estava basat en estructures de xarxa. Tot i que el DBTG de CODASYL va remetre diversos informes a l'ANSI, aquest

no va impulsar mai un estàndard pel que fa a això. Tot i així, aquests informes van servir de base per a alguns SGBD en xarxa, com són IDS de Honeywell i IDMS de Computer Associates.

La estructura de tres nivells corresponent a una base de dades: conceptual o esquema, extern (vistes d'usuari) o sotsesquema i intern, ja està implícita en la seva implementació.

Tipus de registre i conjunts

Els sistemes en xarxa només tenen dues estructures bàsiques: el tipus de registre i conjunts.

El tipus de registre d'un client podria incloure, per exemple les dades següents: `ID_Client`, `Nom`, `Adreça`, `Quantitat_endeutada`, `Data_darrer_pagament`. Aquest seria el tipus de registre `CLIENT`. També tindríem tipus de registre `FACTURA`, `VENEDOR`...

Un conjunt representa una interrelació entre tipus de registre. Per exemple, un conjunt en aquest sentit podria ser la interrelació d'un a molts entre clients i factures. Aleshores, el client seria el propietari i la factura el membre.

Les entitats del model conceptual seran tipus de registre en el model en xarxa. Els atributs seran camps del tipus de registre.

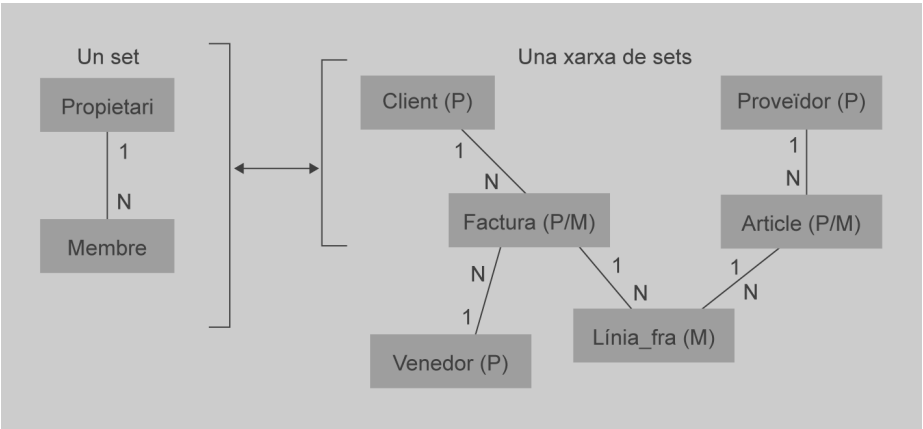
Les interrelacions 1:*N* es transformen en conjunts en què el tipus de registre que correspon a l'1 serà el propietari i el que correspon a *N* el membre.

Per a interrelacions *N*:*M* es crea un tipus de registre intermedi i es transformen en dues 1:*N*.

En les interrelacions *n*-àries també es crea un tipus de registre d'enllaç.

A la figura 1.2 podem veure una exemplificació del model CODASYL.

Figura 1.2. Model CODASYL



DDL de DBTG

Les definicions es fan a partir de l'esquema, que consta de tres seccions: esquema, tipus de registre i conjunts.

Definim l'esquema

```
SCHEMA NAME IS REGISTRE_DE_COMPTABILITAT
```

Definim els registres

Registre de client

```
RECORD NAME IS CLIENT
ID_CLIENT TYPE IS NUMERIC INTEGER
NOM TYPE IS CHARACTER 15
ADRESSA TYPE IS CHARACTER 40
BALANS_COMPTE TYPE IS NUMERIC (5,2)
```

Registre de factura

```
RECORD NAME IS FACTURA
NUM_FACTURA TYPE IS NUMERIC INTEGER
DATA TYPE IS CHARACTER 9
QUANTITAT TYPE IS NUMERIC (5,2)
ESTAT TYPE IS CHARACTER 12
```

Registre d'articles

```
RECORD NAME IS ARTICLE
CODI_ARTICLE TYPE IS NUMERIC INTEGER
DESCRIPCIO TYPE IS CHARACTER 20
PREU TYPE IS NUMERIC (4,2)
```

Definim els conjunts

Relació propietari-membre entre client i factura

```
CLIENT_FACTURA
OWNER IS CLIENT
MEMBER IS FACTURA
```

DML de DBTG

A diferència de les BD relacionals, aquí el DML processa els registres d'un en un i ha d'estar allotjat en un llenguatge amfitrió com ara COBOL. Hi ha ordres de diversos tipus:

- Navegació: `FIND`
- Consulta: `GET`
- Actualització de registres: `ERASE`, `STORE`, `MODIFY`
- Actualització de conjunts: `CONNECT`, `DISCONNECT`, `RECONNECT`

Com que es processen registres, cal l'ordre FIND per localitzar el registre per processar. Aquesta ordre té dues sintaxis, que utilitzarem en els exemples següents:

```
FIND ANY <nom_registre> [USING <llista_camps>]
FIND DUPLICATE <nom_registre> [USING <llista_camps>]
```

Restriccions d'integritat en l'IDMS

Les associacions de conjunts poden ser MANDATORY (un cop un registre membre forma part d'una ocurrència de conjunt, no es pot eliminar l'ocurrència de conjunt fins que no s'elimini el registre) o OPTIONAL (no hi ha restriccions d'integritat).

Obliga que els conjunts que s'hagin d'esborrar quan s'elimina el registre amo estiguin inclosos en tots els subesquemes que continguin aquell registre.

Utilitat del model en xarxa

El model en xarxa és útil per a bases de dades amb les característiques següents:

- Mida gran
- Consultes repetitives ben definides
- Transaccions ben definides
- Aplicacions ben definides

1.7.2 Bases de dades jeràrquiques

No parteixen ni d'un model matemàtic (BD relacionals) ni d'un esforç d'estandardització (DB en xarxa) sinó del dia a dia. Com que no hi ha un estàndard, només ens queda analitzar els SGDB que l'implementen, entre els quals no hi ha dubte que destaca l'IMS d'IBM, tot i que n'hi ha hagut d'altres: TDMS (*system development corporation's time-shared data management system*), MARK IV (*control data corporation's multi-access retrieval system*) i el System-2000 (SAS Institute).

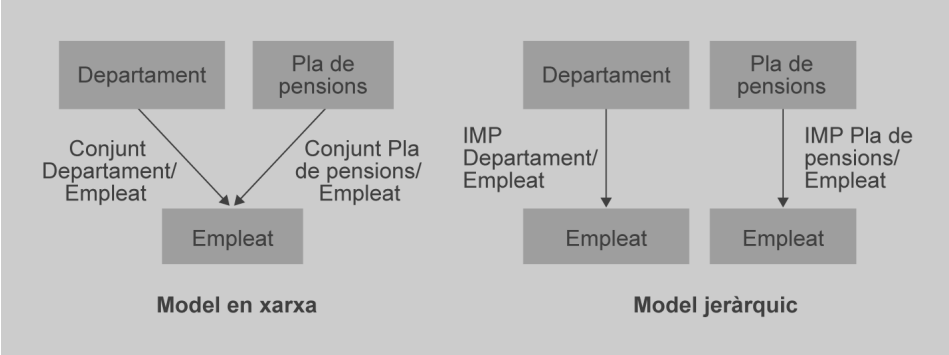
Naixen al començament dels anys seixanta, com el model en xarxa. De fet, l'IMS va sorgir entre IBM i North American Aviation (la que es convertí en Rockwell) per fer un sistema de suport al projecte lunar Apollo: les peces s'organitzaven de manera jeràrquica, ja que les més petites formaven les més grans.

Tot i que decreix el nombre d'instal·lacions d'IMS, en sistemes que requereixen respostes molt ràpides encara és un sistema competitiu.

Tal com en les BD en xarxa hi ha amo i membre, aquí tindrem pare i fill. Ara bé, el model jeràrquic no accepta qualsevol graf sinó només arbres. Consta de tipus

de segments (semblants als registres de les xarxes) i tipus d'interrelacions pare-fill (tipus IPH, semblants als conjunts del model en xarxa).

Figura 1.3. Comparativa de models en xarxa i jeràrquic



1.7.3 Bases de dades relacionals

El model relacional per a la gestió d'una base de dades és un model de dades basat en la lògica de predicats i en la teoria de conjunts. És el model més utilitzat actualment per modelitzar problemes reals i administrar dades dinàmicament. Després de ser postulades les seves bases el 1970 per Edgar Frank Codd, dels laboratoris IBM a San José (Califòrnia), no es va trigar a consolidar com un nou paradigma en els models de base de dades.

El model relacional és un model de dades i, com a tal, té en compte els tres aspectes següents de les dades:

1. **L'estructura**, que ha de permetre representar la informació que ens interessa del món real. Aquesta estructura es representa mitjançant un conjunt de relacions.

2. **La manipulació**, a la qual dóna suport mitjançant les operacions d'actualització i consulta de les dades. La consulta de les dades consisteix en l'obtenció de dades deduïbles a partir de les relacions que conté la base de dades. El llenguatge SQL, en les sentències de consulta, combina construccions de l'àlgebra relacional i del càlcul relacional amb un predomini de les construccions del càlcul. Així mateix, el llenguatge SQL ens permet fer operacions d'inserció, actualització i eliminació de dades.

3. **La integritat**, que és facilitada mitjançant l'establiment de regles d'integritat, és a dir, condicions que les dades han de complir:

    - Regla d'integritat d'unicitat de la clau primària.
    - Regla d'integritat d'entitat de la clau primària.
    - Regla d'integritat referencial.
    - Regla d'integritat de domini.

1.7.4 Bases de dades XML natives

XML és part del present i segurament el futur de l'administració de dades, ja que ha permès trencar certes barreres i crear un format estàndard per processar informació.

Doncs bé, XML està provocant l'aparició de noves tecnologies, entre aquestes l'aparició d'una nova generació de bases de dades que, encara que ara per ara es troben en una fase d'investigació i desenvolupament, en un futur poden ser una alternativa a les bases de dades relacionals. Són les anomenades *native XML database*, les bases de dades XML natives (BD:XML).

Tractament XML en les bases de dades relacionals

Aquests tipus de bases de dades són completament diferents de les relacionals, les quals en aquests moments ja donen suport per a XML, però encara continuen emmagatzemant la informació de manera relacional, és a dir, en forma tabular, o emmagatzemen tot el document en format *binary large object*. A més, no estan ben preparades per emmagatzemar estructures de tipus jeràrquic com són els documents XML per les raons següents:

- Les BD relacionals tenen una estructura regular, mentre que els documents XML tenen caràcter heterogeni.
- Els documents XML solen contenir molts nivells d'imbricació, mentre que les dades relacionals són "planes", és a dir, tenen un nombre de columnes per a cada fila.
- Els documents XML tenen un ordre intrínsec, mentre que les dades relacionals són no ordenades.
- Les dades relacionals són generalment denses, és a dir, cada columna té un valor i acostumen a tenir pocs valors nuls; en canvi, els documents XML són dispersos, ja que poden presentar mancança d'informació mitjançant l'absència de l'element.

S'han creat teories per encaixar objectes o estructures jerarquitzades en bases de dades relacionals.

Comparativa entre les XML:DB i les bases de dades relacionals

La principal característica de les XML:DB és la capacitat d'obtenir resultats de consultes amb format XML, i per això aquestes bases de dades pertanyen a la categoria *XML-enabled database*.

L'organització XML:DB Initiative for XML Databases descriu una base de dades d'aquest tipus com un "model lògic" per a documents XML que emmagatzema i recupera documents d'acord amb aquest model.

Totes les bases de dades relacionals són centrades en les dades, *data-centric databases*, ja que el que aquestes emmagatzemen són dades atòmiques. En canvi, una BD:XML, ni té camps, ni emmagatzema dades atòmiques, ja que el que emmagatzema són documents XML, per la qual cosa aquest tipus de bases de dades s'anomenen *bases de dades centrades en documents*, *document-centric databases*.

Característiques generals de les XML:DB

Diversos productes ofereixen diferents característiques per a les bases de dades natives XML, i són generalment aquestes:

- **Aspectes generals:** Són bases de dades i suporten transaccions, accés multiusuari, llenguatges de consulta, etc., dissenyades especialment per emmagatzemar documents XML.

- **Emmagatzemament:** Per deducció lògica una base de dades nativa en XML emmagatzema informació en format XML, però això no és solament una deducció lògica, ja que aquest tipus de bases de dades té repositoris amb un format de tipus XML com pot ser DOM o Infoset. En aquest mateix repositori s'emmagatzemen els índexs que es generen per a cada document XML emmagatzemat. És a dir, emmagatzema el document XML sencer en format text i proporciona alguna funcionalitat a la base de dades per accedir-hi.

- **Emmagatzemament de documents en col·leccions:** Les col·leccions tenen en les bases de dades natives el paper de les taules en les bases de dades relacionals. Els documents s'agrupen en funció de la informació que contenen en col·leccions, que a la vegada poden contenir altres sub-col·leccions dins seu.

- **Validació de documents:** Cal que els documents que s'emmagatzemin siguin "XML ben formats".

- **Processament de dades:** El processament de dades en aquest tipus de bases de dades sembla molt àgil, però això no és del tot així a causa del format jeràrquic en el qual està emmagatzemada la informació. Moltes bases de dades necessiten que es recuperi tot el document XML , s'actualitzi en l'API XML que s'estigui fent servir i posteriorment es torni a emmagatzemar en el repositori. Això és perquè no hi ha un llenguatge estàndard que permeti l'actualització, inserció o eliminació d'elements d'un document XML. Hi ha un llenguatge que permet fer actualitzacions en un document XML, encara que alguns gestors d'aquest tipus de bases de dades no el suporten; aquest llenguatge és l'XUpdate.

eXist

Una de les XML:DB més populars és eXist. Podeu consultar les seves característiques a: exist.sourceforge.net

XUpdate

Podeu veure les especificacions de l'XUpdate a: xmldb-org.sourceforge.net/xupdate/xupdate-wd.html

XQuery Update extensions

Podeu veure exemples de les *XQuery update extensions* que utilitza l'XML:DB eXist a: exist.sourceforge.net/update_ext.html

XPath

Podeu veure les especificacions d'XPath a: www.w3.org/TR/xpath20/

XQuery

Podeu veure les especificacions d'XQuery a: www.w3.org/TR/xquery/

- **Consultes:** Aquest tipus de bases de dades no utilitzen SQL com a llenguatge de consulta. En lloc d'això utilitzen Xpath i XQuery. Algunes bases de dades permeten seleccionar els elements que han de tenir índexs, mentre que d'altres indexen tot el contingut del document. El problema que tenen les bases de dades que només permeten cerques amb XPath és que no permeten fer cerques gaire complicades, ja que no permeten l'ordenació ni el *cross join*, ja que inicialment XPath va ser creat per fer cerques dins d'un document i no en una base de dades. No obstant això, cal dir que tot això està en evolució contínua. Ara la majoria de les BD:XML suporten un o més llenguatges de consulta. Un dels més populars és XQuery.

- **Indexació XML:** S'ha de permetre la creació d'índexs que accelerin les consultes fetes freqüentment.

- **Creació d'identificadors únics:** A cada document XML s'associa un identificador únic pel qual serà reconegut dins del repositori.

1.8 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.

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 condicions millors d'utilitzar-los d'una manera òptima.

1.8.1 Esquemes i nivells

Per gestionar les BD, els SGBD n'han de conèixer l'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.

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

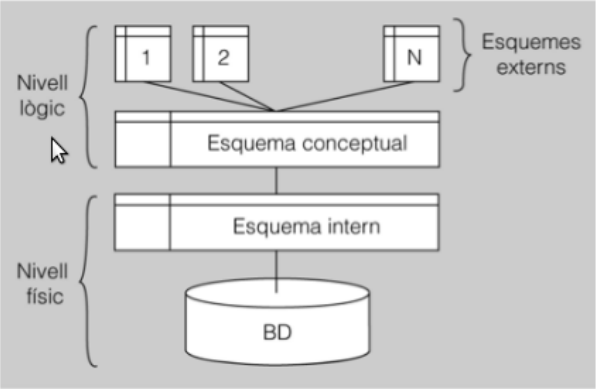
- En el **nivell extern** se situen les diferents visions lògiques que els processos usuaris (programes d'aplicació i usuaris directes) tenen de les parts de la BD que utilitzen. Aquestes visions s'anomenen *esquemes externs*.

- 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*.

A la figura 1.4 podem veure una esquematització d'aquests nivells.

Figura 1.4. Nivells en una base de dades



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

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.

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

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.

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.

1.8.2 Components funcionals dels SGBD

El programari que conforma els SGBD es divideix en diferents mòduls, encarregats de les funcionalitats respectives 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 requisits 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 de la velocitat de la unitat central de processament, és molt important que l'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 pels 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 mida 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 l'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, en un nivell lògic, en un llenguatge no procedimental (típicament, SQL), a una seqüència d'operacions de 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 de baix nivell que pot interpretar el motor d'avaluació de consultes. En fer la traducció esmentada, un bon compilador DML també s'encarregarà de fer una optimització de consultes triant, entre totes les alternatives, la de menys cost.
- **Motor d'avaluació de consultes.** Executa les instruccions de baix nivell generades pel compilador DML.

1.9 Tipus d'usuari de l'SGBD

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.

1.9.1 Usuaris del 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 fer 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.

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.

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.

1.9.2 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.

1.10 Instal·lació i desinstal·lació d'un SGBD

Com a administradors d'un Sistema gestor de bases de dades haureu de fer, primer de tot, un estudi exhaustiu de l'entorn que us envolta i en el qual heu d'instal·lar l'SGBD.

També heu d'estar assabentats de la finalitat i l'ús que es donarà al vostre sistema al final del procés d'instal·lació. Per exemple, és important fer una aproximació del nombre d'usuaris finals que tindrà connectats en un determinat moment el nostre sistema, així com conèixer aproximadament quin tipus d'operacions es faran de forma reiterada (consultes de dades, insercions de noves dades, modificacions de dades existents), ja que depenent d'aquests factors ens decantarem per un o altre sistema gestor dels existents en el mercat actualment.

1.10.1 Creació de l'entorn per a l'SGBD

Una de les tasques més importants que s'ha de fer com a DBA és escollir i instal·lar un SGBD adient per tal de satisfer les necessitats del client. Això comporta tenir ben present, entre altres coses:

DBA és l'abreviatura de data base administrator, que en català és administrador de bases de dades.

- El maquinari del qual disposa el client per muntar el servidor.
- La plataforma o sistema operatiu que utilitzarà per moure'l.
- L'ús final que es donarà al sistema.

Les principals tasques per crear un entorn de treball amb un o diversos SGBD són:

- Justificació de l'elecció.
- Escollir un SGBD adient.
- Escollir l'arquitectura de l'SGBD.
- Escollir el tipus de clusterització.

Justificació de l'elecció de l'SGBD

Escollir un sistema gestor de bases de dades ha deixat de ser una feina excessivament complicada, ja que els grans fabricants d'aquest tipus de programari són pocs.

Hem de tenir en compte que a la vida real són poques les empreses que disposen i fan ús d'un únic programari de gestió de bases de dades. No és gens estrany trobar clients amb bases de dades fetes amb l'Acces o el Filemaker, que a més utilitzen per a les necessitats de l'empresa algun tipus de servidor amb l'SQL Server de Microsoft o l'Oracle.

El problema resideix a saber quins de tots aquests programaris són els millors i si poden o han de conviure en el mateix entorn. També ens podem trobar amb una actualització de programari, amb la qual cosa tindrem dos o més sistemes gestors convivint en el mateix entorn. També hem de tenir cura d'esborrar antics SGBD o actualitzar-los, ja que hi pot haver problemes de compatibilitat a l'hora d'extreure les dades antigues.

Escollir un SGBD adient

Hem d'intentar reduir el nombre d'SGBD diferents en el nostre entorn de treball. Així doncs, davant la possibilitat de triar diversos SGBD segons la plataforma o el programari, és aconsellable escollir sempre basant-se en la plataforma (llevat que el programari aconselli el contrari).

Al mercat hi ha moltes empreses que ofereixen el seu SGBD, però només algunes tenen renom i el suport més qualificat. Sempre que es pugui, hem de triar productes amb una qualitat contrastada. Actualment hi ha grans productes en el mercat, que són:

1. DB2 Universal Database (IBM Corporation).
2. Oracle (Oracle Corporation).
3. SQL Server (Microsoft Corporation).
4. PostgreSQL (programari lliure amb llicència BSD).

Tot el que no sigui escollir un d'aquests programaris qualificats és assumir riscos de més o menys grau. També és important fer notar que un programari de menys renom pot trigar de 18 a 24 mesos a incorporar les eines i funcionalitats de què disposen avui dia els programaris més coneguts. Aquests programaris també tenen referents en el mercat, com ara:

1. MySQL.
2. SQLite.

Per escollir un SGBD hem de tenir en compte, entre altres coses:

- **El suport per als nostres sistemes operatius**, tant el present com el futur si tenim previst fer una actualització.
- **Tipus de client que ens demana l'SGBD**. Tipus de seguretat que vol a les dades, quantitat de diners a invertir en la compra del programari de gestió, etc.

- **El joc de proves** (**benchmarking**). Una sèrie de proves que du a terme un organisme no lucratiu (**TCP**) donen una idea bàsica de com es comporten els programaris del mercat en diverses condicions de treball.

- **L'ampliació d'usuaris i/o maquinari d'emmagatzematge està suportada**. L'SGBD, podrà suportar les bases de dades de la grandària que necessiteu?

- **Els tècnics qualificats**. Us podrien ajudar a implementar i mantenir el sistema? L'SGBD és un programari tan nou que podem tenir problemes per saber com implementar segons quines bases de dades.

- **El cost del producte**, incloent-hi el preu de les llicències per client i el programari necessari per gestionar l'SGBD i fer-ne ús.

- **L'actualització de versions del producte**. Cada quant s'actualitza el programari que comprarem? Quin cost associat tindrà? Estem en una organització conservadora (que vol pocs canvis) o en una més arriscada (no importen els canvis si són a millor)?

TCP

El Transaction Processing Performance Council (TCP) és un organisme que fa proves de càrrega sobre els principals programaris del mercat. Ens podem guiar pels seus resultats per escollir el nostre SGBD, segons l'ús final que li donarem.

Per fer una bona elecció haurem de tenir aquests paràmetres ben presents. El millor SGBD serà el que ens ofereixi millor funcionalitat per als nostres fins. Aquests indicadors ens poden donar una bona idea de quin producte estem buscant.

Arquitectures dels SGBD

Els diferents fabricants de programari de gestió de bases de dades ofereixen diversos tipus d'arquitectura del mateix programari.

> Entenem com a **arquitectura de l'SGBD** el disseny intern del programari preparat per donar servei a un determinat grup de tipus d'usuari.

Els principals venedors de programari han establert tres nivells d'arquitectura per als seus programaris:

1)La versió enterprise. Està dissenyada per suportar un gran grau d'ampliació i ha de ser capaç de suportar un gran nombre de bases de dades de mida gran, un gran nombre d'usuaris connectats de manera concurrent i també diverses aplicacions. Aquests tipus de distribucions acostumen a funcionar sobre un maquinari molt potent, generalment supercomputadors amb múltiples microprocessadors i amb sistemes operatius robustos, com l'UNIX o el Linux.

2)La versió departamental. Està pensada per a grups d'usuaris reduïts, com mitjanes o petites empreses. La diferència entre aquesta versió i l'*enterprise* és ben petita, ja que els sistemes operatius amb què funciona aquest programari també són Linux, Unix o alguna versió del Windows Server. Aquests grups reduïts poden servir per provar alguna funcionalitat que després s'incorporarà a un sistema *enterprise*.

Versió lite

Una versió *lite* d'un programa és una versió retallada. Normalment, però, ofereix la majoria de les funcionalitats de la versió *enterprise*.

3)La versió personal. Està pensada per a usuaris individuals, i funcionen en màquines estàndard amb sistemes operatius com el Windows. Un dels programaris més estès en aquest entorn és el Microsoft Access. Hem de destacar que sovint els grans proveïdors d'SGBD ofereixen el seu programari en una versió *lite,* com per exemple el Personal Oracle d'Oracle.

És important que, en la mesura que es pugui, s'escullin paquets del mateix venedor de programari per a cadascun dels nivells de desenvolupament i d'usuaris.

Clusterització dels SGBD

Entenem com a **clusterització** l'ús de sistemes de computació múltiples i independents com un de sol. La majoria dels programaris actuals ofereixen aquesta funcionalitat.

Es poden distingir dos tipus de clusterització:

- **Compartició de disc** (**shared-disk**). Es tracta de tenir diverses CPU i memòries treballant en els mateixos discos durs, els quals estan connectats en xarxa. Aquest sistema de clusterització és el més utilitzat per a grans supercomputadors. Interconnectem poques CPU potents amb molts discos; és, doncs, un sistema escalable però car, pel fet que cal treballar amb supercomputadors.
- **Arquitectura no compartida** (**shared-nothing**). En aquest cas es tracta d'aprofitar ordinadors no tan potents (un simple PC domèstic), els quals contenen una CPU mitjanament bona, una memòria adient i uns discos de gran capacitat. La interconnexió s'ha de fer mitjançant la xarxa i connectant les CPU, que es passaran informació de l'una a l'altra. Les dades físiques no es comparteixen, sinó que cada disc és independent de l'altre. El sistema s'encarrega d'enviar cada consulta a la CPU que disposa de la informació demanada.

1.10.2 Instal·lació de l'SGBD

Una vegada hem escollit el nostre SGBD, l'hem d'instal·lar. Aquest procés no és tan simple com introduir un CD i fer clics fins a finalitzar.

És del tot necessari planificar amb cura tots els passos que cal seguir, fent especial atenció a mantenir els mínims requisits específics de cada SGBD.

Primer de tot, és primordial tenir ben present que s'han de complir els prerequisits que estableixi el sistema gestor per tal de dur a terme la instal·lació. Aquests prerequisits poden anar des de la utilització de la versió correcta del sistema operatiu fins a tenir prou memòria RAM o prou espai al disc dur per instal·lar-

hi el programari (deixem de banda les dades que posteriorment introduirem en l'SGBD).

El segon pas que hem de fer si no volem tenir problemes greus en un futur pròxim és llegir el manual d'instal·lació de cap a peus.

Requeriments generals de maquinari

Com és natural, l'SGBD necessitarà una CPU mínima, és a dir, una CPU amb una velocitat de processament per sota la qual el nostre SGBD no funcionarà. Lògicament, la majoria dels SGBD treballaran amb una CPU estàndard, però cal fer notar que els grans supercomputadors poden obtenir millors resultats si el programari de l'SGBD té microcodi específic per al microprocessador en qüestió. És important escollir l'SGBD que millor ens vagi, i si cal, fins i tot ajustar el maquinari per complir els requisits del nostre SGBD.

Requeriments d'emmagatzematge

Qualsevol SGBD requereix un disc dur per poder funcionar. És lògic, ja que necessita un lloc on crear les bases de dades i emmagatzemar informació. L'SGBD utilitza el disc per desar, entre altres coses:

- El catàleg (diccionari de dades) usat per l'SGBD per cercar les bases de dades emmagatzemades.
- Els fitxers LOG, que ens ajuden a saber què ha canviat a les nostres bases de dades.
- Els fitxers de configuració de l'SGBD.
- Fitxers d'errors i bolcats de memòria.
- Estructures temporals que creen les aplicacions sota demanda.

> És important **descentralitzar les dades**, és a dir, hem de procurar repartir-les entre diversos discos. Si fem això, cada accés a dades diferents pot anar en discos diferents. Això es tradueix en més velocitat en servir les dades i evitem el coll d'ampolla que crearia un sol disc si són molts els clients connectats de manera concurrent.

També està bé tenir els fitxers LOG en algun dispositiu diferent, que pugui ser contínuament llegit sense destorbar les operacions de consulta de dades de l'SGBD. A més a més, s'han de preveure les còpies de seguretat i en quin suport físic aniran.

Requeriments de memòria

Per a tot programa d'avui en dia la memòria és un bé molt preuat. Gairebé es podria dir que és el més preuat de tots, ja que és en aquest espai de memòria on l'SGBD farà totes les funcions bàsiques per al seu funcionament.

El principal handicap que ens trobarem als SGBD per tal de poder oferir les dades com més ràpid millor és que sempre hem d'anar a buscar-les al disc dur, i sabem que aquest dispositiu és el més lent davant la CPU i/o la memòria RAM. Així doncs, és necessària la memòria RAM per tal que el nostre SGBD intenti minimitzar els accessos al disc dur, i això ho farà prenent totes les dades possibles (més de les que demanem) cada vegada que accedeixi al disc. Com més dades tingui la memòria més possibilitats hi ha que la nostra consulta obtingui les dades demanades directament i que no accedeixi al disc dur. Per tant, obtindrem una resposta més ràpida.

És imprescindible que l'SGBD tingui una memòria prou gran per poder fer totes les operacions que requereix el sistema. Penseu que com més gran és la memòria més memòries intermèdies o *buffers*de dades pot contenir, i per tant més possibilitats d'evitar l'accés a disc.

1.10.3 Configuració de l'SGBD

Configurar l'SGBD no és res més que portar un control dels recursos de què disposarà el nostre sistema i com els utilitzarà. Cada sistema disposa d'uns paràmetres diferents per portar a terme aquesta configuració.

És habitual que durant la instal·lació es vagi perfilant una configuració bàsica mitjançant decisions que us aniran sortint en pantalles, com: l'ús final del gestor de bases de dades, el tipus d'accés, el nombre d'accessos concurrents, etc.

Tot i aquesta petita configuració prèvia, sempre podrem editar manualment els fitxers de configuració per ajustar tant com es pugui el funcionament del sistema.

Exemple de nexe interconnector

Un exemple de nexe interconector són els programes de control o *drivers* ODBC i JDBC, que permeten fer visible diferents bases de dades del sistema per a aplicacions Windows o permetre l'accés a la base de dades mitjançant programari implementat en java respectivament.

Una vegada tinguem configurat el nostre sistema gestor, serà convenient fer les connexions pertinents perquè les diferents aplicacions puguin interactuar amb ell.

2. Sistemes de bases de dades i comunicacions

L'arquitectura d'un sistema de bases de dades està influenciada en gran mesura pel sistema informàtic subjacent en què s'executa, en particular per aspectes de l'arquitectura de l'ordinador com la connexió en xarxa, el paral·lelisme i la distribució:

1. **La connexió en xarxa** de diverses computadores permet que algunes d'aquestes s'executin en un sistema servidor i que altres s'executin en els sistemes clients. Aquesta divisió de treball ha conduït al desenvolupament de sistemes de bases de dades client-servidor.

2. **El processament paral·lel** dins d'un ordinador permet accelerar les activitats del sistema de base de dades, i proporciona a les transaccions unes respostes més ràpides i la capacitat d'executar més transaccions per segon. Les consultes es poden processar de manera que s'exploti el paral·lelisme ofert pel sistema informàtic subjacent. La necessitat del processament paral·lel de consultes ha conduït al desenvolupament dels sistemes de bases de dades paral·lels.

3. **La distribució de dades** a través de les diferents seus o departaments d'una organització permet que aquestes dades es trobin on han estat generades o on són més necessàries, però continuen essent accessibles des d'altres llocs o departaments diferents. El fet de desar diverses còpies de la base de dades en diferents llocs permet que puguin continuar les operacions sobre la base de dades encara que algun lloc es vegi afectat per algun desastre natural com una inundació, un incendi o un terratrèmol. S'han desenvolupat els sistemes distribuïts de bases de dades per manejar dades distribuïdes geogràficament o administrativament al llarg de múltiples sistemes de bases de dades.

Estudiarem, doncs, l'arquitectura dels sistemes de bases de dades començant amb els tradicionals sistemes centralitzats i tractant, més endavant, els sistemes de bases de dades client-servidor, paral·lels i distribuïts.

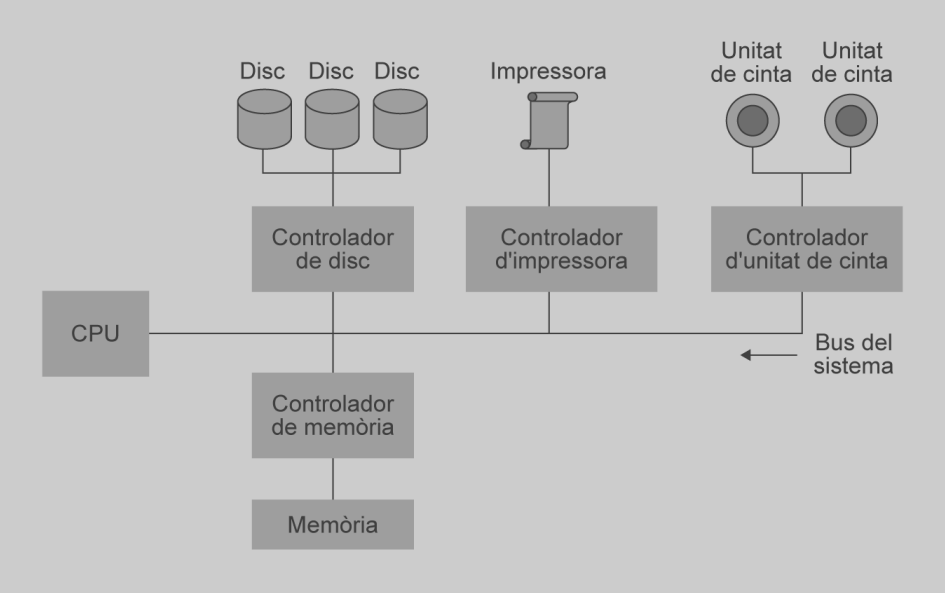
Els **sistemes de bases de dades centralitzats** són aquells que s'executen en un únic sistema informàtic sense interaccionar amb cap altre ordinador. Aquests sistemes comprenen el rang des dels sistemes de bases de dades monousuari que s'executen en ordinadors personals fins als sistemes de bases de dades d'alt rendiment que s'executen en grans sistemes.

D'altra banda, els sistemes client-servidor tenen la seva funcionalitat dividida entre el sistema servidor i múltiples sistemes clients.

2.1 Sistemes centralitzats

Un ordinador modern de propòsit general consisteix en una o poques unitats centrals de processament i un nombre determinat de controladors per als dispositius que es troben connectats per mitjà d'un bus comú, el qual proporciona accés a la memòria compartida. Les CPU (unitats centrals de processament) tenen memòries cau locals on s'emmagatzemen còpies de certes parts de la memòria per accelerar l'accés a les dades. Cada controlador s'encarrega d'un tipus específic de dispositius (per exemple, una unitat de disc, una targeta de so o un monitor). Les CPU i els controladors de dispositius es poden executar concurrentment i competeixen així per l'accés a la memòria. La memòria cau redueix la disputa per l'accés a la memòria, ja que la CPU necessita accedir a la memòria compartida un nombre de vegades menor (figura 2.1).

Figura 2.1. Funcionament de la CPU i els controladors de dispositius



Es distingeixen dues maneres d'utilitzar les computadores: com a sistemes monousuari o multiusuari.

En la primera classe tindrem els ordinadors personals i les estacions de treball. Un sistema monousuari típic és una unitat de sobretaula utilitzada per una única persona que disposa d'una sola CPU, d'un o dos discos fixos i que treballa amb un sistema operatiu que només permet un únic usuari.

En canvi, un sistema multiusuari típic té més discos i més memòria, pot disposar de diverses CPU i treballa amb un sistema operatiu multiusuari. S'encarrega de donar servei a un gran nombre d'usuaris que estan connectats al sistema per mitjà de terminals.

Normalment, els sistemes de bases de dades dissenyats per funcionar sobre sistemes monousuari no solen proporcionar moltes de les facilitats que ofereixen els sistemes multiusuari. En particular, no tenen control de concurrència, que no és necessari quan només un usuari pot generar modificacions.

Les facilitats de recuperació en aquests sistemes o no existeixen o són primitives; per exemple, fer una còpia de seguretat de la base de dades abans de qualsevol modificació.

La majoria d'aquests sistemes no admeten SQL i proporcionen un llenguatge de consulta molt simple que, en alguns casos, és una variant de QBE. En canvi, els sistemes de bases de dades dissenyats per a sistemes multiusuari suporten totes les característiques de les transaccions que s'han estudiat abans.

Encara que avui dia els ordinadors de propòsit general tenen diversos processadors, utilitzen paral·lelisme de gra gruixut, i disposen d'uns pocs processadors (normalment dos o quatre) que comparteixen la mateixa memòria principal. Les bases de dades que s'executen en aquestes màquines habitualment no intenten dividir una consulta simple entre els diferents processadors, sinó que executen cada consulta en un únic processador i així es possibilita la concurrència de diverses consultes. Així, aquests sistemes suporten més productivitat, és a dir, permeten executar més transaccions per segon, tot i que cada transacció individualment no s'executi més ràpid.

Les bases de dades dissenyades per a les màquines monoprocessador ja disposen de multitasca i això permet que diversos processos s'executin al mateix temps en el mateix processador, usant temps compartit, mentre que amb vista a l'usuari sembla que els processos s'estan executant en paral·lel. D'aquesta manera, des d'un punt de vista lògic, les màquines paral·leles de gra gruixut semblen idèntiques a les màquines monoprocessador, i poden adaptar fàcilment els sistemes de bases de dades dissenyats per a màquines de temps compartit per tal que es puguin executar sobre màquines paral·leles de gra gruixut.

Per contra, les màquines paral·leles de gra fi tenen un gran nombre de processadors i els sistemes de bases de dades que s'executen sobre aquestes intenten fer paral·leles les tasques simples (consultes, per exemple) que demanen els usuaris.

2.2 Bases de dades client/servidor

Com les computadores personals són ara més ràpides, més potents i més barates, els sistemes s'han anat distanciant de l'arquitectura centralitzada.

Els terminals connectats a un sistema central han estat suplantats per ordinadors personals. De la mateixa manera, la interfície d'usuari, que solia estar gestionada directament pel sistema central, està passant a ser gestionada, cada vegada més, per les computadores personals.

Com a conseqüència, els sistemes centralitzats actuen avui com a sistemes de servidor que satisfan les peticions generades pels sistemes clients.

Aquest model es basa en la distribució de funcions entre dos tipus de processos independents i autònoms, servidors i clients.

Un **client** és un procés que sol·licita serveis específics als processos d'un servidor. Un **servidor** és un procés que proporciona serveis sol·licitats pels clients.
Els processos client i servidor poden estar al mateix ordinador o en diferents, sempre que estiguin connectats per una xarxa.

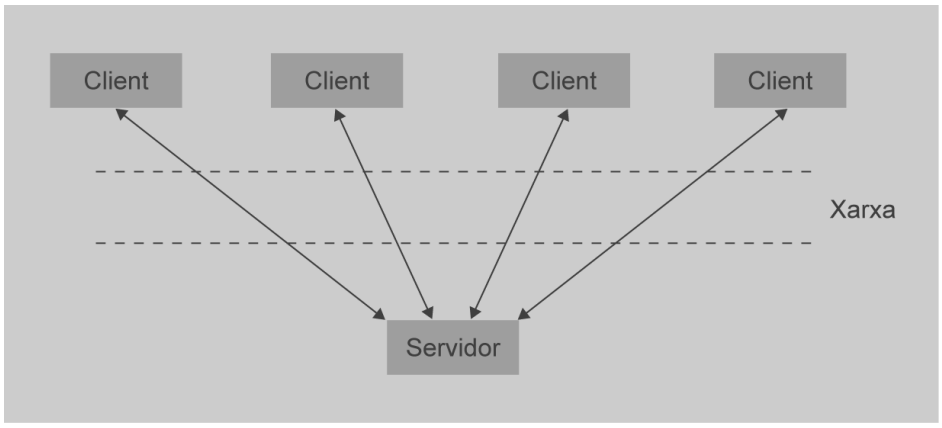
Segons el grau en què es comparteix el processament entre el client i el servidor, un servidor o un client es pot catalogar com a *pesat* o *lleuger*. Un **client lleuger** és el que fa un mínim de processament pel costat client, mentre que un **client pesat** és el que suporta una proporció relativament gran de càrrega de processament. De la mateixa manera, un **servidor pesat** fa la part preponderant de càrrega de processament, mentre que un **servidor lleuger** s'encarrega de menys càrrega de processament.

Finalment, val a dir que els sistemes client/servidor també es classifiquen com de 2 o 3 capes.

2.2.1 Sistema client/servidor de 2 capes

En un sistema client/servidor de 2 capes un client sol·licita serveis directament al servidor. A la figura 2.2 es representa l'estructura general d'un sistema client-servidor.

Figura 2.2. Estructura d'un sistema client-servidor



En el cas del PostgreSQL, es pot accedir a les nostres bases de dades via **sòcols**.

Un sòcol és un concepte abstracte pel qual dos programes (possiblement situats en ordinadors diferents) poden intercanviar qualsevol flux de dades, generalment de manera fiable i ordenada. Tot sòcol està definit per una adreça de sòcol. L'adreça de sòcol és una combinació de tres elements: una adreça IP, un protocol de transport i un número de port (per exemple: 84.88.125.15, TCP, 5432).

Perquè dos programes es puguin comunicar entre si és necessari que es compleixin certs requisits:

- Que un programa sigui capaç de localitzar l'altre.
- Que tots dos programes siguin capaços d'intercanviar qualsevol seqüència de bytes, és a dir, dades rellevants a la seva finalitat.

Per a això són necessaris els tres recursos que originen el concepte de sòcol:

- Un protocol de comunicacions, que permet l'intercanvi de bytes.
- Una adreça del protocol de xarxa (adreça IP, si s'utilitza el protocol TCP/IP), que identifica un ordinador.
- Un número de port, que identifica un programa dins d'un ordinador.

Els sòcols permeten implementar una arquitectura client-servidor. La comunicació ha de ser iniciada per un dels programes, que s'anomena *programa client*. El segon programa espera que un altre inicii la comunicació, i per aquest motiu s'anomena *programa servidor*.

Un sòcol és un fitxer informàtic existent a la màquina client i a la màquina servidora, que serveix en última instància perquè el programa servidor i el client llegeixin i escriguin la informació. Aquesta informació serà la transmesa per les diferents capes de xarxa.

En el cas del PostgreSQL inicialment el servidor queda configurat amb sòcols Unix. Podem comprovar que està funcionant el servidor (*postmaster*) des de l'intèrpret d'ordres amb:

En la secció "Annexos" del web del mòdul podeu veure com es fa la configuració del PostgreSQL mitjançant els arxius **pg_hba.conf** i **postgresql.conf**.

```
# pgrep post
```

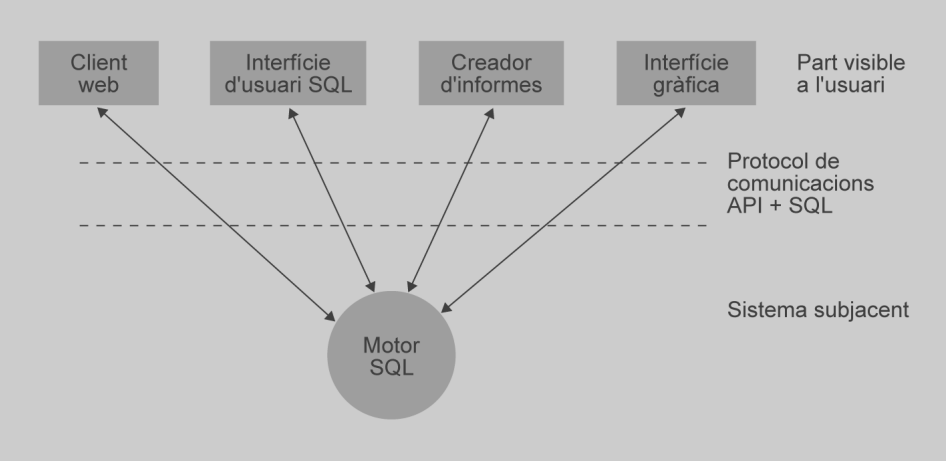
Per utilitzar el protocol TCP/IP per a connexions des de diferents màquines cal editar els fitxers **pg_hba.conf** i **postgresql.conf**.

2.2.2 Sistema client/servidor de 3 capes

En un sistema client/servidor de 3 capes, les sol·licituds del client són manejades per servidors intermedis, i són aquests els que s'encarreguen de coordinar l'execució de les sol·licituds del client amb altres servidors subordinats.

Com es mostra en la figura 2.3, la funcionalitat d'una base de dades es pot dividir a grans trets en tres parts: la part visible a l'usuari i el sistema subjacent s'interconnecten per una capa intermèdia.

FIGURA 2.3. Funcionament d'un sistema client-servidor de 3 capes



El sistema subjacent gestiona l'accés a les estructures, l'avaluació i optimització de consultes, el control de concurrència i la recuperació. La part visible a l'usuari d'un sistema de base de dades està formada per eines com formularis, dissenyadors d'informes i facilitats gràfiques d'interfície d'usuari. La interfície entre la part visible a l'usuari i el sistema subjacent pot ser SQL o una aplicació.

Les normes com ODBC i JDBC es van desenvolupar per fer d'interfície entre clients i servidors. Qualsevol client que utilitzi interfícies ODBC o JDBC es pot connectar a qualsevol servidor que proporcioni aquesta interfície.

Eines de desenvolupament d'aplicacions

Algunes de les eines de desenvolupament d'aplicacions més famoses són PowerBuilder, Magic i Borland Delphi. El Visual Basic també s'utilitza bastant en el desenvolupament d'aplicacions.

Interfícies client-servidor

Certes aplicacions com els fulls de càlcul i els paquets d'anàlisi estadística disposen d'una interfície client-servidor per accedir a les dades del servidor subjacent. De fet, proporcionen interfícies visibles especials per a diferents tasques.

En les primeres generacions de sistemes de bases de dades, la manca d'aquestes normes feia que fos necessari que la interfície visible i el sistema subjacent fossin proporcionats pel mateix distribuïdor de programari. Amb l'augment de les interfícies estàndard, sovint diferents distribuïdors proporcionen la interfície visible a l'usuari i el servidor del sistema subjacent. Les eines de desenvolupament d'aplicacions s'utilitzen per construir interfícies d'usuari; proporcionen eines gràfiques que es poden utilitzar per construir interfícies sense programar.

Alguns sistemes de processament de transaccions proporcionen una interfície de crida a procediments remots per a transaccions per connectar els clients amb el servidor. Aquestes trucades apareixen per al programador com a crides normals a procediments, però totes les crides a procediments remots fetes des d'un client s'engloben en una única transacció al servidor final. D'aquesta manera, si la transacció es cancel·la, el servidor pot desfer els efectes de les crides a procediments remots de manera individual.

2.3 Arquitectura client/servidor

Una arquitectura client/servidor es basa en components de programari i maquinari que interactuen per formar un sistema.

2.3.1 Components d'una arquitectura client/servidor

Aquest sistema inclou 3 components principals: clients, servidors i programari intermediari de comunicacions.

- El **client** és qualsevol ordinador que sol·licita serveis al servidor. També es coneix com a **aplicació frontal**.
- El **servidor** és qualsevol procés de computadora que proporciona serveis als clients. També es coneix com a **aplicació dorsal**.
- El **programari intermediari de comunicacions** és qualsevol procés o conjunt d'aquests mitjançant els quals el procés client i el procés servidor es comuniquen. També és conegut com a *middleware* o *estrat de comunicacions* i es compon de diversos estrats de programari que ajuden a transmetre dades i a controlar la informació entre clients i servidors. El programari intermediari de comunicacions en general s'associa a una xarxa. Totes les sol·licituds del client i les respostes del servidor viatgen a través de la xarxa en forma de missatges que contenen dades i informació de control.

2.3.2 Principis client/servidor

Els components de l'arquitectura client/servidor cal que s'ajustin a alguns principis bàsics per assolir una interacció adequada. Aquest principis cal que siguin aplicables tant a clients, servidors i components de programari intermediari.

- ***Independència de maquinari***: aquest principi requereix que tant els processos client, servidor i intermediari funcionin en diferents plataformes de maquinari sense cap diferència de funcionament.

- ***Independència de programari***: requereix que els processos client, servidor i intermediari suportin diferents sistemes operatius, diversos protocols (IPX i TCP/IP) i diferents aplicacions.

- ***Accés obert a serveis***: cal que tots els clients en el sistema tinguin accés obert a tots els serveis proveïts a la xarxa i que aquests serveis no depenguin de la ubicació del client o del servidor.

- ***Distribució de processos***: els processos client i servidor cal que siguin entitats autònomes amb límits i funcions clarament definides. Això permet delimitar la funcionalitat per cada costat i millorar la modularitat i flexibilitat. Cal assolir el màxim d'utilització dels recursos, i per això el procés

servidor cal que sigui compartit per tots els clients, és a dir, podrà atendre diferents sol·licituds de diferents clients. Cal que els processos client i servidor siguin fàcils d'actualitzar. Cal que el procés servidor i els clients s'integrin fàcilment per formar un sistema.

- ***Estàndards***: tots aquests principis exposats s'han d'implementar seguint diferents estàndards, ja que així assolim que els diferents components interactuïn de manera ordenada per assolir els resultats que volem.

2.3.3 Protocol RDA (remote database access)

RDA és un protocol de comunicacions per a l'accés a remot a bases de dades que ha estat adoptat com un estàndard internacional per l'Organització Internacional de Normalització (ISO) i la Comissió Electrotècnica Internacional (IEC). També s'ha adoptat com a norma ANSI.

Proporciona protocols estàndard per establir una connexió remota entre un client i un servidor de base de dades. El client està actuant com un programa d'aplicació mentre el servidor fa d'interfície d'un procés que controla les transferències de dades cap a una base de dades i des d'aquesta.

L'objectiu és promoure la interconnexió de les aplicacions de bases de dades entre entorns heterogenis.

RDA va ser publicat com una combinació d'ANSI/ISO/IEC el 1993. Es compon de dues parts:

- Part 1 – RDA genèric ANSI/ISO/IEC 9579-1:1993
- Part 2 – Especialització SQL ANSI/ISO/IEC 9579-2:1993

La part 1 especifica el model general, el servei i el protocol per a una connexió a la base de dades i la part 2 especifica els protocols addicionals per a la connexió de bases de dades d'acord amb el llenguatge SQL.

Com a característiques generals podem dir que:

- L'aplicació interactua amb la base de dades mitjançant sentències SQL.
- Les seves sentències poden recuperar, actualitzar, eliminar o inserir registres.
- Les declaracions es poden agrupar en les transaccions, de manera que una transacció té èxit o fracassa en el seu conjunt.
- La ubicació de la base de dades és transparent.

2.4 Components del client

Com hem esmentat, un client correspon a qualsevol procés que sol·licita serveis a un procés servidor. El client és proactiu, és a dir, sempre iniciarà la conversa amb el servidor. Les característiques desitjables tant de programari com de maquinari que cal que tingui un client són:

- Maquinari poderós
- Sistema operatiu multitasca
- Interfície gràfica d'usuari (GUI)
- Capacitats de comunicació

Una aplicació client funciona sobre un sistema operatiu i es connecta al programari intermediari de comunicacions per accedir als serveis disponibles de la xarxa.

Es poden emprar tant llenguatges de 3a. generació com de 4a. generació (4GL) per crear l'aplicació frontal amb la qual interactuarà l'usuari.

La majoria de les aplicacions frontals estan basades en GUI per ocultar a l'usuari la complexitat dels components client/servidor.

Així aquest poder de processament permetrà el desenvolupament d'aplicacions amb capacitats multimèdia.

2.5 Sistemes de servidor

Els sistemes de servidor es poden dividir en servidors de transaccions i servidors de dades.

- **Els sistemes de servidor de transaccions**, també anomenats *sistemes de servidor de consultes*, proporcionen una interfície per mitjà de la qual els clients poden enviar peticions per fer una acció que el servidor executarà, i els resultats es tornaran al client. Normalment, les màquines client envien les transaccions als sistemes de servidor, lloc en què aquestes transaccions s'executen, i els resultats es retornen als clients, que són els encarregats de visualitzar les dades. Les peticions es poden especificar utilitzant SQL o mitjançant la interfície d'una aplicació especialitzada.
- **Els sistemes de servidor de dades** permeten als clients interaccionar amb els servidors fent peticions de lectura o modificació de dades en unitats com ara arxius o pàgines. Per exemple, els servidors de fitxers proporcionen una interfície de sistema d'arxius per mitjà de la qual els clients poden crear, modificar, llegir i esborrar fitxers. Els servidors de dades dels

sistemes de bases de dades ofereixen moltes més funcionalitats, suporten unitats de dades de mida menor que els arxius, com pàgines, tuples o objectes. Proporcionen facilitats d'indexació de les dades, i també facilitats de transacció, de manera que les dades mai no es queden en un estat inconsistent si falla una màquina client o un procés.

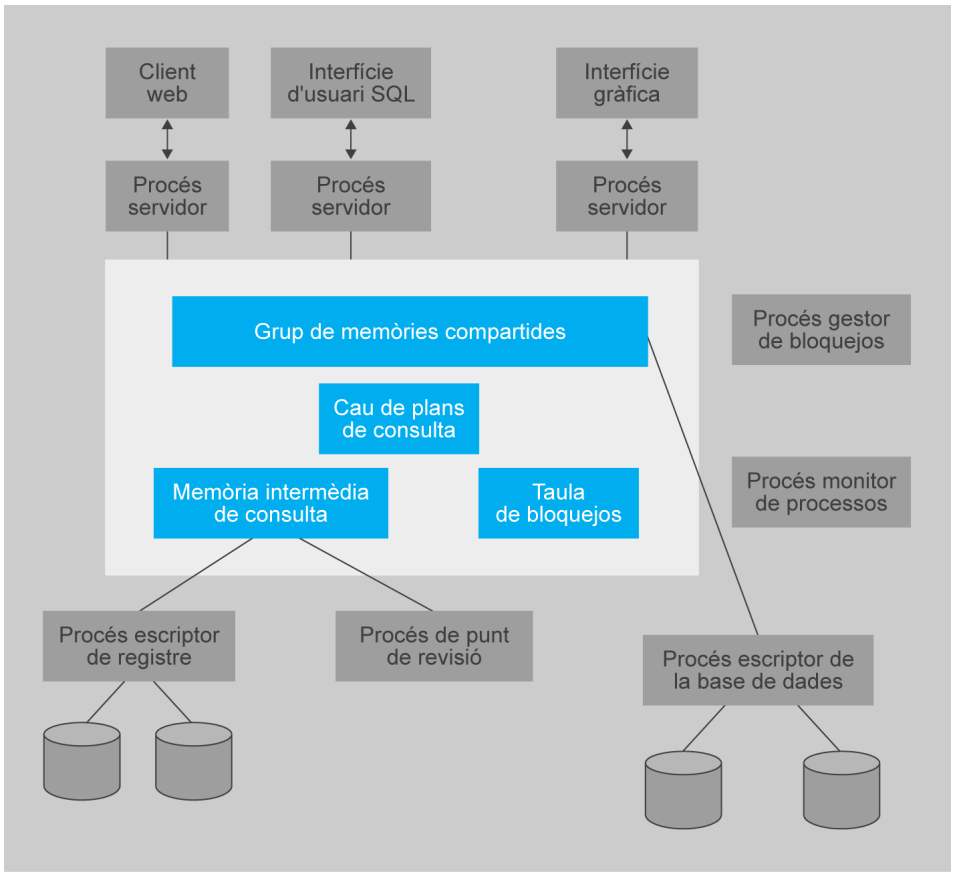
Podeu veure l'arquitectura del PostgreSQL en la secció "Annexos" del web del mòdul.

D'aquestes, l'arquitectura del servidor de transaccions és, amb molt, l'arquitectura més àmpliament utilitzada.

2.5.1 Estructura de processos del servidor de transaccions

Avui dia, un sistema servidor de transaccions típic consisteix en múltiples processos que accedeixen a les dades en una memòria compartida, com mostra la figura 2.4.

Figura 2.4. Estructura d'un sistema servidor de transaccions



Els processos que formen part del sistema de bases de dades inclouen:

- **Processos servidor:** són processos que reben consultes de l'usuari (transaccions), les executen, i retornen els resultats. Les consultes s'han d'enviar als processos servidor des de la interfície d'usuari, o des d'un procés d'usuari que executa SQL incorporat, o mitjançant JDBC, ODBC o protocols similars. Alguns sistemes de bases de dades utilitzen un procés diferent per a

cada sessió d'usuari, i algunes utilitzen un únic procés de la base de dades per a totes les sessions de l'usuari, però amb múltiples fils, de manera que es poden executar concurrentment múltiples consultes. (Un fil és com un procés, però diversos fils s'executen com a part del mateix procés, i tots els fils dins d'un procés s'executen en el mateix espai de memòria virtual. Dins d'un procés es poden executar concurrentment múltiples fils.) Alguns sistemes de bases de dades utilitzen una arquitectura híbrida, amb processos múltiples, cadascun amb diverses cadenes.

- **Procés gestor de bloquejos:** aquest procés implementa una funció de gestió de bloquejos que inclou concessió de bloquejos, alliberament de bloquejos i detecció d'interbloquejos.
- **Procés escriptor de bases de dades:** hi ha un o més processos que bolquen al disc els blocs de memòria intermèdia modificats de manera contínua.
- **Procés escriptor del registre:** aquest procés genera entrades del registre en l'emmagatzematge estable a partir de la memòria intermèdia del registre. Els processos servidor simplifiquen l'addició d'entrades a la memòria intermèdia del registre en memòria compartida i, si cal forçar l'escriptura del registre, demanen al procés escriptor del registre que bolqui les entrades del registre.
- **Procés punt de revisió:** aquest procés fa periòdicament punts de revisió.
- **Procés monitor de procés:** aquest procés observa altres processos i, si qualsevol falla, fa accions de recuperació per al procés, com ara cancel·lar qualsevol transacció que estigués executant el procés fallit, i reinicia el procés.

La memòria compartida conté totes les dades compartides, com:

- Grup de memòries intermèdies.
- Taula de bloquejos.
- Memòria intermèdia del registre, que conté les entrades del registre que esperen a ser abocades en l'emmagatzematge estable.
- Plans de consulta en memòria cau, que es poden reutilitzar si s'envia de nou la mateixa consulta.

Tots els processos de la base de dades poden accedir a les dades de la memòria compartida. Ja que múltiples processos poden llegir o fer actualitzacions en les estructures de dades en memòria compartida, hi ha d'haver un mecanisme que asseguri que només un està modificant una estructura de dades en un moment donat, i que cap procés no està llegint una estructura de dades mentre altres l'escriuen. Tal exclusió mútua es pot implementar per mitjà de funcions del sistema operatiu anomenades *semàfors*.

Implementacions alternatives, amb menys sobrecàrregues, utilitzen instruccions atòmiques especials suportades pel maquinari de l'ordinador, un tipus d'instrucció

atòmica comprova una posició de la memòria i l'estableix a un automàticament. Es poden trobar més detalls sobre l'exclusió mútua en qualsevol llibre de text d'un sistema operatiu estàndard. Els mecanismes d'exclusió mútua també s'utilitzen per implementar sòcols.

Per evitar la sobrecàrrega del pas de missatges, en molts sistemes de bases de dades els processos servidor implementen el bloqueig actualitzant directament la taula de bloquejos (que està en memòria compartida), en lloc d'enviar missatges de sol·licitud de bloqueig a un procés administrador de bloquejos. El procediment de sol·licitud de bloquejos executa les accions que faria el procés administrador de bloquejos per processar la petició de bloqueig. Les accions de la sol·licitud i l'alliberament de bloquejos són les habituals, però amb dues diferències significatives:

- Atès que diversos processos servidor poden accedir a la memòria compartida, s'ha d'assegurar l'exclusió mútua en la taula de bloquejos.
- Si no es pot obtenir un bloqueig immediatament a causa d'un conflicte de bloquejos, el codi de la sol·licitud de bloqueig continua observant la taula de bloquejos fins a adonar-se que s'ha concedit el bloqueig. El codi d'alliberament de bloqueig actualitza la taula de bloquejos per indicar a quin procés s'ha concedit el bloqueig.

Per evitar comprovacions repetides de la taula de bloquejos, el codi de sol·licitud de bloqueig pot utilitzar els semàfors del sistema operatiu per esperar una notificació d'una concessió de bloqueig. El codi d'alliberament de bloqueig ha d'utilitzar llavors el mecanisme de semàfors per notificar a les transaccions que estan esperant que els seus bloquejos hagin estat concedits.

Fins i tot si el sistema gestiona les sol·licituds de bloqueig per mitjà de memòria compartida, encara utilitza el procés administrador de bloquejos per a la detecció d'interbloquejos.

2.5.2 Servidors de dades

Els sistemes de servidor de dades s'utilitzen en xarxes d'àrea local en què s'assoleix una alta velocitat de connexió entre els clients i el servidor, les màquines clients són comparables al servidor quant a poder de processament i s'executen tasques de còmput intensiu.

En aquest entorn té sentit enviar les dades a les màquines client, fer allà tot el processament (que pot durar un temps) i després enviar les dades de tornada al servidor. Noteu que aquesta arquitectura necessita que els clients tinguin totes les funcionalitats del sistema subjacent.

Les arquitectures dels servidors de dades s'han fet particularment populars en els sistemes de bases de dades orientades a objectes.

En aquesta arquitectura sorgeixen alguns aspectes interessants, ja que el cost en temps de comunicació entre el client i el servidor és alt comparat amb el d'accés a una memòria local (mil·lisegons enfront de menys de 100 nanosegons):

- **Enviament de pàgines o enviament d'elements.** La unitat de comunicació de dades pot ser de gra gruixut, com una pàgina, o de gra fi, com un tuple (o, en el context dels sistemes de bases de dades orientades a objectes, un objecte). S'usarà el terme *element* per referir-se tant a tuples com a objectes. Si la unitat de comunicació de dades és un únic element, la sobrecàrrega per la transferència de missatges és alta comparada amb el nombre de dades transmeses. En comptes de fer això, quan es necessita un element, pren sentit la idea d'enviar al costat d'aquell altres elements que probablement hagin de ser utilitzats en un futur pròxim. Es denomina *preextracció* l'acció de cercar i enviar elements abans que sigui estrictament necessari. Si diversos elements es troben en una pàgina, l'enviament de pàgines es pot considerar com una forma de preextracció, ja que, quan un procés vulgui accedir a un únic element de la pàgina, s'enviaran tots els elements d'aquesta pàgina.

- **Bloqueig.** La concessió del bloqueig dels elements de dades que el servidor envia als clients la fa habitualment el servidor mateix. Un inconvenient de l'enviament de pàgines és que els clients poden rebre bloquejos de gra gruixut: el bloqueig d'una pàgina bloqueja implícitament tots els elements que s'hi troben. El client adquireix implícitament bloquejos sobre tots els elements preextrets, fins i tot encara que no estigui accedint a alguns d'aquests. D'aquesta manera, pot aturar innecessàriament el processament d'altres clients que necessiten bloquejar aquests elements. S'han proposat algunes tècniques per a l'alliberament de bloquejos en què el servidor pot demanar als clients que li tornin el control sobre els bloquejos dels elements preextrets. Si el client no necessita l'element preextret pot retornar els bloquejos sobre aquest element al servidor perquè aquests puguin ser assignats a altres clients.

- **Memòria cau de dades.** Les dades que s'envien al client en favor d'una transacció es poden allotjar en una memòria cau del client fins i tot una vegada completada la transacció, si disposa de prou espai d'emmagatzematge lliure. Les transaccions successives en el mateix client poden fer ús de les dades en memòria cau. No obstant això, es presenta el problema de la coherència de cau: si una transacció troba les dades en la memòria cau, s'ha d'assegurar que aquestes dades estan al dia, ja que, després d'haver estat emmagatzemades en la memòria cau, poden haver estat modificades per un altre client. Així, s'ha d'establir una comunicació amb el servidor per comprovar la validesa de les dades i poder adquirir un bloqueig sobre aquestes.

- **Memòria cau de bloquejos.** Els bloquejos també poden ser emmagatzemats en la memòria cau del client si la utilització de les dades està pràcticament dividida entre els clients, de manera que un client poques vegades necessita dades que estan essent utilitzats per altres clients.

Exemple de cost en la comunicació

Suposem que es troben a la memòria cau tant l'element de dades que es busca com el bloqueig requerit per accedir-hi. Llavors, el client pot accedir a l'element de dades sense necessitat de comunicar res al servidor. No obstant això, el servidor ha de seguir el rastre dels bloquejos en memòria cau; si un client demana un bloqueig al servidor, aquest ha de comunicar a tots els bloquejos sobre l'element de dades que es troba en les memòries cau d'altres clients. La tasca es torna més complicada quan es tenen en compte els possibles errors de la màquina. Aquesta tècnica es diferencia de l'alliberament de bloquejos, en què la memòria cau de bloqueig es fa per mitjà de transaccions; d'una altra manera, les dues tècniques serien similars.

2.6 Programari intermediari de comunicacions

El programari intermediari proporciona les funcions següents:

- Homogeneïtzar els diferents components del maquinari, sistemes operatius i protocols de comunicació.
- En sistemes distribuïts, ocultar el fet que normalment una aplicació està formada per diferents parts interconnectades que s'executen en diferents localitzacions.
- Proporcionar interfícies uniformes d'alt nivell per als desenvolupadors i integradors d'aplicacions, i facilitar la composició, reutilització, portabilitat i interoperabilitat d'aquestes aplicacions.
- Proveir un sistema de serveis comuns per fer diverses funcions de finalitats generals, per tal d'evitar la duplicació d'esforços i facilitar la col·laboració entre les aplicacions.

2.6.1 Components del programari intermediari

Així doncs, el programari intermediari de base de dades consta de tres components principals:

- Interfície de programació d'aplicacions (API): la utilització dependrà de l'aplicació client. Aquesta API permet al programador escriure codi SQL genèric.
- Traductor de base de dades: tradueix aquestes sol·licituds fetes en format genèric a una sintaxi específica per a aquell servidor de bases de dades que serà utilitzat, ja que podria tenir algunes característiques no estàndard.
- Traductor de xarxa: que gestionarà els diferents protocols de comunicació de xarxa emprats per la part client i la part servidor.

2.6.2 Classificació del programari intermediari

El programari intermediari es pot classificar d'acord amb la manera com els clients i els servidors es comuniquen a través de la xarxa:

- **Programari intermediari orientat a missatges (MOM)**. És un programari que suporta la infraestructura d'enviament i recepció de missatges entre sistemes distribuïts. MOM permet els mòduls d'aplicació per a la distribució a través de plataformes heterogènies i redueix la complexitat del desenvolupament d'aplicacions que abasten múltiples sistemes operatius i protocols de xarxa. El programari intermediari crea una capa de comunicacions distribuïdes que aïlla el desenvolupador de l'aplicació dels detalls del sistema operatiu i les interfícies de xarxa.
- **Programari intermediari basat en crides a procediments remots (basat en RPC)**. RPC és una tecnologia que permet a un programa fer que una subrutina o procediment s'executi en un altre espai d'adreces (habitualment en un altre ordinador en una xarxa compartida) sense que s'hagin de programar explícitament els detalls d'aquesta interacció remota. Un RPC és instanciat pel client enviant un missatge de petició a un servidor remot conegut per executar el procediment especificat fent servir paràmetres subministrats. La resposta serà retornada al client de manera que continuarà amb el seu procés.
- **Programari intermediari basat en objectes**. Un exemple notable de programari intermediari orientat a objectes és la *common object request broker architecture* (**CORBA**). CORBA és un estàndard, no un producte, i va ser desenvolupat per l'Object Management Group (OMG), que és un consorci de gairebé tots els fabricants de programari importants i alguns grans usuaris. Malgrat la seva procedència, poques vegades es veu en les implementacions. Una de les raons de la manca d'implementació CORBA era la seva complexitat. Però, possiblement, la principal raó que CORBA mai no es vagi implementar a fons va ser el sorgiment de la tecnologia de components.

2.6.3 Iniciatives de programari intermediari

Algunes iniciatives de creació de programari intermediari són les següents:

- *Call level interface*
- ODBC
- iODBC
- unixODBC

- JDBC
- IDAPI
- OLE DB

Call Level Interface

Call level interface (CLI) és un estàndard de programari definit en la norma ISO/IEC 9075-3:2003. La interfície de nivell de crida defineix la manera com un programa ha d'enviar consultes SQL per al sistema de gestió de bases de dades i com els conjunts de registres retornats han de ser manejats per l'aplicació d'una manera coherent. Desenvolupat en la dècada de 1990, l'API es defineix només per als llenguatges C i COBOL.

Evolució de SAG

L'SQL Access Group (SAG) va ser un grup d'empreses de programari que es va formar el 1989 per definir i promoure normes per a la portabilitat i la interoperabilitat de bases de dades. Els membres inicials van ser Oracle Corporation, Informix, Ingres, DEC, Tandem, Sun i HP.

El SAG va iniciar el desenvolupament de l'*SQL call level interface* (SQL CLI), que més tard es va publicar com una especificació X/Open.

L'SQL Access Group va transferir les seves activitats i actius a X/Open al final de 1994.

ODBC

L'ús més estès de l'estàndard CLI és la base de l'ODBC (open database connectivity), la qual és àmpliament utilitzada per permetre que les aplicacions puguin accedir de manera transparent als sistemes de base de dades de diferents proveïdors.

Microsoft, en col·laboració amb Simba Technologies, va crear ODBC per adaptar-lo a les especificacions SQL CLI. ODBC aconsegueix la independència de la plataforma i el llenguatge mitjançant l'ús d'un controlador ODBC, com una capa de traducció entre l'aplicació i l'SGBD. L'aplicació, per tant, només necessita conèixer la sintaxi d'ODBC, i el controlador pot passar la consulta a l'SGBD en el seu format natiu i tornar les dades en un format que l'aplicació pugui comprendre.

A causa de les diferencies entre les tecnologies d'emmagatzematge de dades, tots els controladors ODBC no poden implementar totes les interfícies possibles disponibles en l'estàndard ODBC. Microsoft descriu la disponibilitat d'una interfície com a "específica del controlador", ja que poden no ser aplicables en funció de la tecnologia d'emmagatzematge de dades.

Implementació d'ODBC en diferents sistemes operatius

Actualment diferents implementacions d'ODBC funcionen sobre diferents sistemes operatius, incloent-hi Microsoft Windows, Unix, Linux, OS/2, OS/400, IBM i5/OS, i Mac OS X. Hi ha gran quantitat de controladors ODBC per a diferents SGBD com Oracle, DB2, Microsoft SQL Server, Sybase, Pervasive SQL, IBM Lotus Domino, MySQL, PostgreSQL, OpenLink Virtuoso, i bases de dades d'escriptori com FileMaker i Microsoft Access.

iODBC

iODBC és una iniciativa de codi obert gestionada per OpenLink Software. Es tracta d'una plataforma independent SDK d'ODBC i permet el desenvolupament d'aplicacions compatibles amb ODBC i controladors que no siguin de plataforma Windows. Els principals objectius d'aquest projecte són els següents:

- Simplificar la portabilitat de les aplicacions basades en l'ODBC de Windows a altres plataformes.
- Simplificar la portabilitat dels controladors ODBC de Windows a altres plataformes.
- Crear dissenys coherents mitjançant l'experiència en la utilització d'ODBC en totes les plataformes.

unixODBC

unixODBC és un projecte de codi obert que implementa l'API d'ODBC. El codi es distribueix sota llicència GNU GPL/LGPL i pot ser construït i utilitzat en molts sistemes operatius diferents, incloent-hi la majoria de les versions de Unix, Linux, Mac OS X, IBM OS/2 i Interix de Microsoft.

Els objectius del projecte inclouen:

- Proporcionar als desenvolupadors les eines necessàries per portar aplicacions de Microsoft Windows ODBC a altres plataformes amb el mínim de canvis en el codi.
- Mantenir el projecte com un proveïdor SDK neutre d'interfície de base de dades.
- Proporcionar als desenvolupadors de controladors ODBC les eines per portar els seus controladors a les plataformes no-Windows.
- Proporcionar a l'usuari un conjunt d'eines de línia d'ordres i la interfície gràfica d'usuari per administrar l'accés de base de dades.
- Mantenir vincles amb la comunitat de programari lliure i els venedors de bases de dades comercials, per garantir la interoperabilitat.

JDBC

L'API JDBC (*Java database connectivity*) permet a les aplicacions en llenguatge Java accedir mitjançant una interfície comuna a les bases de dades per a les quals hi hagi JDBC.

Els controladors JDBC es poden classificar en 4 tipus:

- **JDBC tipus 1:** controladors que actuen com una passarel·la i que permeten l'accés a la base de dades per mitjà d'una altra tecnologia com és l'ODBC.
- **JDBC tipus 2**: controladors natius. Es tracta d'una barreja de controladors natius al sistema gestor de bases de dades i de programes de control de Java. Les crides JDBC es converteixen en crides natives al sistema gestor de bases de dades.
- **JDBC tipus 3**: en aquest cas els controladors converteixen les crides JDBC de les aplicacions Java a un protocol independent del sistema gestor de bases de dades. Posteriorment, una aplicació intermediària les converteix al protocol que requereix el sistema gestor, i permet que segueixi el model de 3 capes. Això el diferencia del controlador de tipus 4, ja que la lògica de la conversió de protocols no es troba en el client, sinó en el nivell intermedi. El mateix controlador serveix, doncs, per a diferents SGBD.
- **JDBC tipus 4**: els controladors converteixen les crides JDBC directament a un protocol que entén el sistema gestor de bases de dades. Com el protocol de base de dades és específic del proveïdor, el client JDBC requereix controladors diferents, que ofereix cada proveïdor de l'SGBD, per connectar-se a diferents tipus de bases de dades. Són els tipus de controladors que ofereixen una comunicació més ràpida i eficient amb el gestor de bases de dades.

Ús actual de la tecnologia BDE
Actualment no s'utilitza, ja que la tecnologia BDE va ser discontinuada per Borland, si bé Code Gear, companyia filial de Borland dedicada a eines de desenvolupament, encara proveeix documentació per a aquesta.

IDAPI

IDAPI (*integrated database application program interface* o *independent database application program interface*), va ser originalment un component del sistema de bases de dades Paradox de Borland. Més endavant es va convertir en la interfície per a aplicacions del BDE o *Borland database engine*.

OLE DB

OLE DB és la sigla d'*object linking and embedding for databases* ('enllaç i incrustació d'objectes per a bases de dades'), i és una tecnologia desenvolupada per Microsoft usada per tenir accés a diferents fonts d'informació, o bases de dades, de manera uniforme.

OLE DB permet separar les dades de l'aplicació que les requereix. Això es va fer així, ja que diferents aplicacions requereixen accés a diferents tipus i magatzems de dades, i no necessàriament volen conèixer com poden tenir accés a certa funcionalitat amb mètodes de tecnologies específiques. OLE DB està conceptualment dividit en consumidors i proveïdors: el consumidor és l'aplicació que requereix accés a les dades i el proveïdor és el component de programari que exposa una interfície OLE DB per mitjà de l'ús del *component object model* (COM).

Microsoft data access components:

OLE DB és part dels components de Microsoft per a accés a dades o *Microsoft data access components* (MDAC); MDAC és un grup de tecnologies de Microsoft que interactua en conjunt com una infraestructura que brinda als programadors de la nova era una manera de desenvolupar aplicacions amb accés a gairebé qualsevol magatzem de dades. Els proveïdors OLE DB poden ser creats per tenir accés a magatzems de dades, que van des de simples arxius de text i fulls de càlcul fins a bases de dades complexes com Oracle, Microsoft SQL Server o Sybase ASE.

3. SGBD distribuïts

En un sistema distribuït de bases de dades s'emmagatzema la base de dades en diverses computadores. Els diversos mitjans de comunicació, com les xarxes d'alta velocitat o les línies telefòniques, són els que poden posar en contacte els diferents ordinadors d'un sistema distribuït.

No comparteixen ni memòria ni discos. Els ordinadors d'un sistema distribuït poden variar en mida i funció i poden abastar des de les estacions de treball fins als grans sistemes.

Depenent del context en què s'esmentin hi ha diferents noms per referir-se a les computadores que formen part d'un sistema distribuït, com ara llocs o nodes. Per emfatitzar la distribució física d'aquests sistemes es fa servir principalment el terme *lloc*.

Les principals diferències entre les bases de dades paral·leles sense compartiments i les bases de dades distribuïdes són que les bases de dades distribuïdes normalment es troben en diversos llocs geogràfics diferents, s'administren de manera separada i tenen una interconnexió més lenta.

Una altra gran diferència és que en un sistema distribuït es donen dos tipus de transaccions, les locals i les globals.

Una transacció local és aquella que accedeix a les dades de l'únic lloc en el qual es va iniciar la transacció. D'altra banda, una transacció global és aquella que, o bé accedeix a les dades situades en un lloc diferent d'aquell en què es va iniciar la transacció, o bé accedeix a dades de diversos llocs diferents.

3.1 Evolució dels sistemes gestors de bases de dades distribuïdes

En els anys vuitanta van ocórrer tot un conjunt de canvis tecnològics i socials que van afectar el disseny de les bases de dades:

- Les operacions de negoci es van transformar en geogràficament descentralitzades.
- La competència es va incrementar a escala global.
- Les demandes dels clients i les necessitats del mercat van afavorir un estil d'administració descentralitzat.
- Els ràpids canvis tecnològics van permetre crear microcomputadores d'altes prestacions i de baix cost. A més, les corporacions van adoptar cada cop més l'ús de xarxes d'àrea local (LAN).

- El major nombre d'aplicacions basades en SGBD i les necessitats de protegir les inversions en programari van provocar que el concepte de compartir dades cada cop fos més atractiu.

Posteriorment el creixent ús d'Internet i l'increment de l'amplada de banda va permetre el plantejament del disseny de DDBMS.

3.1.1 Avantatges dels DDBMS

Els sistemes d'administració de bases de dades distribuïdes ofereixen tot un conjunt d'avantatges, ja que:

- Les dades es localitzen prop del lloc on hi ha més demanda.
- Permeten un accés més ràpid a les dades.
- Permeten un processament més ràpid de les dades.
- Faciliten el creixement, ja que permeten afegir nous llocs a la xarxa.
- Hi ha una millora en les comunicacions, ja que els llocs són més petits i estan més a prop dels clients.
- Reducció de costos, ja que és més barat afegir una nova estació de treball que actualitzar un ordinador central.
- Utilitzen interfícies gràfiques d'usuari fàcils d'usar.
- Menys perill de fallida en un sol punt, ja que en cas que una estació de treball es col·lapsi la seva càrrega de treball pot ser absorbida per altres.
- Independència del processador, ja que una sol·licitud d'un usuari pot ser processada per qualsevol processador disponible.

3.1.2 Desavantatges dels DDBMS

- Increment en el cost.
- Integritat de control més difícil.
- La manca de normes.
- Base de dades de disseny més complex.
- La complexitat de la gestió i control. Les sol·licituds han de reconèixer les dades d'ubicació i han de ser capaces d'unir les dades de diversos llocs.

- Tecnològicament difícil: la integritat de dades, gestió de transaccions, control de concurrència, seguretat, còpia de seguretat, recuperació, optimització de consultes, selecció de ruta d'accés, són totes les qüestions que s'han d'abordar.
- Els errors de seguretat augmenten quan les dades es troben en diverses ubicacions.
- La manca de normes a causa de l'absència de protocols de comunicació pot fer la transformació i la distribució de les dades difícils.
- Augment d'emmagatzematge i requisits d'infraestructura, perquè diverses còpies de les dades que es requereixen en diversos llocs separats requereixen més espai en disc.
- Augment dels costos a causa de la complexitat més gran de la formació.
- Es requereix duplicar la infraestructura (personal, programari i llicències, la ubicació física o del medi ambient) i això a vegades no pot compensar els estalvis operatius.

3.1.3 Funcions d'un sistema d'administració de bases de dades distribuïdes

Un sistema d'administració de bases de dades distribuïdes gestiona l'emmagatzemament i processament de dades lògicament relacionades a través d'ordinadors interconnectats en què les funcions de dades i les de processament es distribueixen en diferents llocs. Així doncs, un SGBD haurà de tenir les funcions següents per ser considerat com un sistema distribuït:

- Interfície d'aplicació per interactuar amb l'usuari final o amb programes d'aplicació i d'altres SGBD dins de la base de dades distribuïda.
- Validació per analitzar les sol·licituds de dades.
- Transformació per determinar quins components de sol·licitud de dades es distribueixen i quins no.
- Optimització de consultes per trobar la millor estratègia d'accés.
- Mapatge per determinar la ubicació de les dades de fragments locals o remots.
- Interfície d'E/S per llegir o escriure dades que estiguin en mitjans que permetin la persistència.
- Formatament per preparar les dades per a la presentació a l'usuari final o a un programa d'aplicació.
- Seguretat per proporcionar privacitat tant en bases de dades locals com en remotes.

- Còpia de seguretat i recuperació per garantir la disponibilitat i recuperabilitat de la base de dades en cas d'una fallida

- Funcions d'administració a escala global i local.

- Control de concurrència per manejar l'accés simultani a les dades i per garantir la consistència per mitjà dels fragments en l'SGDBD.

- Maneig de les transaccions per garantir que les dades passin d'un estat consistent a un altre. Aquesta activitat inclou la sincronització de transaccions locals i remotes i aquelles transaccions per mitjà de segments múltiples distribuïts.

3.2 Components d'un DDBMS

Un SGBD distribuït consta del següent:

- Estacions de treball (nodes o llocs) que conformen el sistema de xarxa. L'SGBD distribuït ha de ser independent del maquinari dels ordinadors.

- Components de programari i maquinari que es troben en cada estació de treball. Els components de xarxa permetran que els diferents llocs interactuïn i intercanviïn dades.

- Mitjans de comunicació que permetin el transport de les dades d'una estació de treball a una altra. L'SGBD distribuït ha de ser independent als mitjans de comunicació.

- Processador de transaccions (TP): és un component de programari que es troba en cada ordinador que sol·licita dades. Aquest TP rep i processa les sol·licituds de l'aplicació, ja siguin locals o remotes.

- Processador de dades (DP): és un component de programari resident en cada ordinador que emmagatzema i recupera dades localitzades en el lloc.

Cal tenir en compte tot un conjunt de protocols que:

- Permetran la comunicació per la xarxa amb l'objectiu de transportar dades i instruccions entre els DP i els TP.

- Sincronitzar totes les dades recuperades pels DP i encaminar-les cap als TP corresponents.

- Garantirà funcions de base de dades comuns en un SGBD distribuït.

3.3 Nivells de distribució de dades i processos

Els SGBD es poden classificar segons la distribució de dades i processos. A continuació mostrem diferents escenaris possibles:

- **Processament en un sol lloc i dades en un sol lloc**. Es fa en una única computadora amfitriona. Aquest escenari és típic dels SGBD centralitzats. El processador de transaccions i el processador de dades es troben ubicats en el mateix ordinador.
- **Processament en múltiples llocs i dades en un sol lloc**. Es coneix com a *arquitectura client servidor*.
- **Processament en llocs múltiples i dades en llocs múltiples**. Aquí ens trobem en el cas d'una base de dades totalment distribuïda.

SGBD distribuïts purs o fortament acoblats: són capaços de cooperar entre si per oferir a les aplicacions un accés integrat a les dades. Acostumen a ser sistemes homogenis. Són resultat d'un desenvolupament descendent.

SGBD federats o lleument acoblats: són aquells que per dificultats d'interoperabilitat no poden oferir un accés integrat a les dades. Acostumen a ser sistemes heterogenis. Poden sorgir d'un desenvolupament ascendent.

Podem detectar diferents nivells de distribució:

- Nivell de petició remota: cada petició efectuada al servidor de dades es tracta com una única transacció.
- Nivell d'unitat de treball remota: és possible enviar al servidor diverses sentències que es tracten com una única transacció.
- Nivell d'unitat distribuïda de treball: permet accedir a dos o més servidors distints en la mateixa transacció.
- Nivell de petició distribuïda: una mateixa sentència pot ser executada alhora per diversos servidors.

3.4 Transparència

No s'ha d'exigir als usuaris dels sistemes distribuïts de bases de dades que coneguin la ubicació física de les dades ni la manera com s'hi pot tenir accés en un lloc local concret. Aquesta característica, anomenada *transparència de les dades*, pot adoptar diverses formes:

- **Transparència de la fragmentació**. No s'exigeix als usuaris que coneguin la manera com s'ha fragmentat la relació.

- **Transparència de la rèplica.** Els usuaris veuen cada objecte de dades com a lògicament únic. Pot ser que el sistema distribuït repliqui els objectes per incrementar el rendiment del sistema o la disponibilitat de les dades. Els usuaris no s'han de preocupar pels objectes que s'hagin replicat ni per la ubicació d'aquestes rèpliques.

- **Transparència de la ubicació.** No s'exigeix als usuaris que coneguin la ubicació física de les dades. El sistema distribuït de bases de dades ha de poder trobar les dades sempre que la transacció de l'usuari faciliti l'identificador de les dades.

Els elements de dades (com les relacions, els fragments i les rèpliques) han de tenir noms únics. Aquesta propietat és fàcil d'assegurar en una base de dades centralitzada. En les bases de dades distribuïdes, però, cal anar amb compte per assegurar-se que dos llocs no utilitzin el mateix nom per a elements de dades diferents.

Una solució a aquest problema és exigir que tots els noms es registrin en un servidor de noms central. El servidor de noms ajuda a assegurar que aquest nom no s'utilitzi per a elements de dades diferents. També es pot utilitzar el servidor de noms per ubicar un element de dades, donat el nom de l'element. Aquest enfocament, però, presenta dos inconvenients principals. En primer lloc, pot ser que el servidor de noms es transformi en un coll d'ampolla per al rendiment quan els elements de dades se situen pels seus noms, fet que dóna lloc a un baix rendiment. En segon lloc, si el servidor de noms queda fora de servei, pot ser que no sigui possible que continuï funcionant cap altre lloc del sistema distribuït.

Un enfocament alternatiu més utilitzat exigeix que cada lloc anteposi el seu identificador de lloc propi a qualsevol nom que generi. Aquest enfocament assegura que dos llocs no generin mai el mateix nom (ja que cada lloc té un identificador únic). A més, no es necessita cap control centralitzat. Aquesta solució, però, no aconsegueix la transparència de la ubicació, ja que s'adjunten als noms els identificadors dels llocs. Així, es pot fer referència a la relació compte com a *lloc17.compte*, o *compte@lloc17*, en lloc de merament *compte*. Molts sistemes de bases de dades utilitzen l'adreça d'Internet dels llocs per identificar-los.

Per superar aquest problema el sistema de bases de dades pot crear un conjunt de noms alternatius o àlies per als elements de dades. Així, els usuaris es poden referir als articles de dades mitjançant noms senzills, que el sistema tradueix als noms complets. L'assignació dels àlies als noms reals es pot emmagatzemar en cada lloc. Amb els àlies l'usuari pot ignorar la ubicació física dels elements de dades. A més, l'usuari no es veurà afectat si l'administrador de la base de dades decideix traslladar un element de dades d'un lloc a un altre.

Els usuaris no han d'haver de fer referència a una rèplica concreta d'un element de dades. En comptes d'això, el sistema ha de determinar la rèplica a la qual

cal fer referència en les sol·licituds per llegir, i actualitzarà totes les rèpliques en les sol·licituds per escriure. Es pot assegurar que ho faci mantenint una taula de catàleg, que el sistema utilitza per determinar totes les rèpliques de l'element de dades.

3.5 Control de concurrència distribuït: transaccions, bloquejos i recuperació

L'atomicitat de les transaccions és un aspecte important de la construcció d'un sistema distribuït de bases de dades. Si una transacció s'executa al llarg de dos llocs, tret que els dissenyadors del sistema siguin curosos, es pot comprometre en un lloc i cancel·lar en un altre, cosa que conduiria a un estat d'inconsistència. Els protocols de compromís de transaccions asseguren que aquestes situacions no es produeixin. El protocol de compromís de dues fases (*C2F*) és el més utilitzat d'aquests protocols.

La idea bàsica del C2F és que cada lloc executa la transacció just fins abans del compromís, i llavors deixa la decisió del compromís a un únic lloc coordinador; es diu que en aquest punt la transacció està en estat preparat en el lloc. El coordinador decideix comprometre la transacció només si la transacció arriba a l'estat preparat a cada lloc on es va executar; en un altre cas (per exemple, si la transacció es va cancel·lar en algun lloc), el coordinador decideix cancel·lar la transacció. Tots els llocs on la transacció es va executar han d'acatar la decisió del coordinador. Si un lloc falla quan una transacció es troba en estat preparat, quan el lloc es recuperi de la decisió hauria d'estar en posició de comprometre o cancel·lar la transacció, depenent de la decisió del coordinador.

3.5.1 Compromís de dues fases

En primer lloc, es descriu la manera com opera el protocol de compromís de dues fases (*C2F*) durant el funcionament normal, després descriu la manera com maneja els errors i, finalment, la manera com executa la recuperació i el control de la concurrència.

Considerem una transacció *T* iniciada en el lloc *Li*, en què el coordinador de transaccions és *Ci*.

El protocol de compromís

Quan *T* completa l'execució (és a dir, quan tots els llocs on s'ha executat *T* informen *Ci* que *T* s'ha completat) *Ci* inicia el protocol *C2F*.

- **Fase 1.** *Ci* afegeix el registre *T preparar* al registre històric i obliga a desar el registre històric en un lloc d'emmagatzematge estable. Llavors envia un

missatge *preparar T* a tots els llocs on s'ha executat *T*. En rebre aquest missatge el gestor de transaccions del lloc determina si vol comprometre la seva part de *T*. Si la resposta és negativa, afegeix un registre *no T* al registre històric i respon enviant a *Ci* el missatge *avortar T*. Si la resposta és sí, afegeix un registre *T preparada* al registre històric i obliga que el registre històric (amb tots els registres del registre històric corresponents a *T*) es desi en un emmagatzematge estable. El gestor de transaccions contesta llavors a *Ci* amb el missatge *T preparada*.

- **Fase 2.** Quan *Ci* rep les respostes al missatge *preparar T* de tots els llocs, o quan ha transcorregut un interval de temps especificat amb anterioritat des que es va enviar el missatge *preparar T*, *Ci* pot determinar si la transacció *T* pot comprometre o avortar. La transacció *T* pot comprometre si *Ci* ha rebut el missatge *T preparada* de tots els llocs participants. En cas contrari, la transacció *T* ha d'avortar. En funció del resultat, s'afegeix al registre històric un registre *T compromesa* o un registre *T avortada* i s'obliga que el registre històric es desi en un emmagatzematge estable. En aquest moment la destinació de la transacció ja s'ha segellat. A partir d'aquest moment el coordinador envia un missatge *comprometre T* o *avortar T* a tots els participants. Quan un lloc rep aquest missatge, el desa en el registre històric.

Els llocs on es va executar *T* poden avortar de manera incondicional en qualsevol moment abans d'enviar el missatge *T preparada* al coordinador. Un cop enviat el missatge, es diu que la transacció està en estat preparat en el lloc. El missatge *T preparada* constitueix, en realitat, un compromís del lloc de seguir l'ordre del coordinador de *comprometre T* o d'*avortar T*. Per fer aquest compromís primer cal desar en un emmagatzematge estable la informació necessària. En cas contrari, si el lloc fallés després d'enviar el missatge *T preparada*, potser no era capaç de complir la seva promesa. A més, els bloquejos adquirits per la transacció s'han de mantenir fins que es completi la transacció.

Atès que s'exigeix la unanimitat per comprometre una transacció, la destinació de *T* queda segellada quan un lloc respongui *avortar T*. Atès que el lloc coordinador *Si* és un dels llocs on s'ha executat *T*, el coordinador pot decidir unilateralment *avortar T*. El veredicte final sobre *T* es determina en el moment en què el coordinador l'escriu (*comprometre* o *avortar*) en el registre històric i obliga a desar el veredicte en un emmagatzematge estable. En algunes implementacions del protocol C2F un lloc envia el missatge *justificant-de-rebut T* al coordinador al final de la segona fase del protocol. Quan el coordinador rep el missatge *justificant-de-rebut T* de tot arreu, afegeix el registre *T completada* al registre històric.

Tractament de les fallides

El protocol C2f respon de manera diferent a diversos tipus d'errors:

- **Decisió d'un lloc participant.** Si el coordinador *Ci* detecta que un lloc ha fallat emprèn les accions següents. Si el lloc falla abans de respondre a *Ci* amb el missatge *T preparada*, el coordinador dóna per fet que ha respost

amb el missatge *avortar T*. Si el lloc falla després que el coordinador hagi rebut del lloc el missatge *T preparada*, el coordinador executa la resta del protocol de compromís de manera normal, i ignora la decisió del lloc. Quan un lloc participant *Lk* es recupera d'una fallada ha d'examinar el seu registre històric per determinar la destinació de les transaccions que es trobaven en trànsit d'execució quan es va produir la sentència.

- **Decisió del coordinador.** Si el coordinador falla durant l'execució del protocol de compromís per a la transacció *T*, els llocs participants han de decidir la destinació de *T*. Es veurà que, en certs casos, els llocs participants no poden decidir si *comprometre T* o *avortar T*, i, per tant, han d'esperar a la recuperació del coordinador que ha fallat.
- **Divisió de la xarxa.** Quan una xarxa queda dividida, hi ha dues possibilitats:

1. **El coordinador i tots els participants segueixen en una de les particions**. En aquest cas, la decisió no té cap efecte sobre el protocol de compromís.
2. **El coordinador i els participants queden en múltiples particions**. Des del punt de vista dels llocs d'una de les particions, sembla com si els llocs de les altres particions hagin fallat. Els llocs que no es troben en la partició que conté el coordinador senzillament executen el protocol per tractar la decisió del coordinador.

El coordinador i els llocs que es troben en la seva mateixa partició segueixen el protocol de compromís habitual, amb el benentès que els llocs de les altres particions han fallat.

Per tant, el principal inconvenient del protocol C2F és que la decisió del coordinador pot donar lloc a un bloqueig, en el qual pot ser que calgui retardar la decisió sobre *comprometre T* o *avortar T* fins que es recuperi *Ci*.

3.5.2 Recuperació i control de la concurrència

El control de concurrència és una altra característica d'una base de dades distribuïda. Com una transacció pot accedir a elements de dades de diversos llocs, els administradors de transaccions de diversos llocs poden necessitar coordinar-se per implementar el control de la concurrència. Si s'utilitza bloqueig (com gairebé sempre succeeix en la pràctica), el bloqueig es pot fer de manera local en els llocs que contenen els elements de dades accedits, però també hi ha possibilitat d'un interbloqueig que involucri transaccions originades en múltiples llocs. Per tant, cal portar la detecció d'interbloquejos al llarg de múltiples llocs. Els errors són més comuns en els sistemes distribuïts, ja que no solament les computadores poden fallar, sinó que també poden fallar els enllaços de comunicacions.

La rèplica dels elements de dades, que és la clau per al funcionament continuat de les bases de dades distribuïdes quan ocorren errors, complica encara més el control de la concurrència.

Els models estàndard de transaccions, basats en múltiples accions dutes a terme per una única unitat de programa, són sovint inadequats per fer tasques que travessen els límits de les bases de dades que no poden o no cooperen per implementar protocols com el C2F. Per a aquestes tasques s'utilitzen generalment tècniques alternatives, basades en missatgeria persistent per a les comunicacions.

Quan les tasques per fer són complexes, i involucren múltiples bases de dades i múltiples interaccions amb humans, la coordinació de les tasques i assegurar les propietats de les transaccions per a les tasques es torna més complicat. Els sistemes de gestió de fluxos de treball són sistemes dissenyats per ajudar en la realització d'aquestes tasques.

En cas que una empresa hagi de triar entre una arquitectura distribuïda i una arquitectura centralitzada per implementar una aplicació, l'arquitecte del sistema ha de sospesar els avantatges enfront dels desavantatges de la distribució de dades. Ja s'han examinat els avantatges d'utilitzar bases de dades distribuïdes. El principal inconvenient dels sistemes distribuïts de bases de dades és la complexitat afegida que és necessària per garantir la coordinació adequada entre els llocs. Aquesta creixent complexitat té diverses facetes:

- **Cost de desenvolupament del programari**. La implementació d'un sistema distribuït de bases de dades és més difícil i, per tant, més costós.
- **Més probabilitat d'errors**. Com els llocs que constitueixen el sistema distribuït operen en paral·lel és més difícil assegurar la correcció dels algorismes, del funcionament especial durant els errors de part del sistema i de la recuperació. Són probables errors extremadament subtils.
- **Més sobrecàrrega de processament**. L'intercanvi de missatges i el còmput addicional necessari per aconseguir la coordinació entre els diferents llocs constitueixen una forma de sobrecàrrega que no sorgeix en els sistemes centralitzats.

Hi ha diversos enfocaments sobre el disseny de les bases de dades distribuïdes que abasten des dels dissenys completament distribuïts fins als que inclouen un alt grau de centralització.

3.6 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.

3.6.1 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 3.1 mostra una relació, anomenada PROVEIDOR, per tal d'exemplificar els mètodes principals de fragmentació.

Taula 3.1. Relació 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

La taula 3.2 i taula 3.3 mostren dos possibles fragments horitzontals de la relació PROVEIDOR. Els tuples s'han dividit en funció de la localitat de cada proveïdor.

Taula 3.2. Primer fragment horitzontal de la relació 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

Taula 3.3. Segon fragment horitzontal de la relació 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

3.6.2 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 3.4 i taula 3.5 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 3.4. Primer fragment vertical de la relació 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

Taula 3.5. Segon fragment vertical de la relació 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

3.6.3 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.

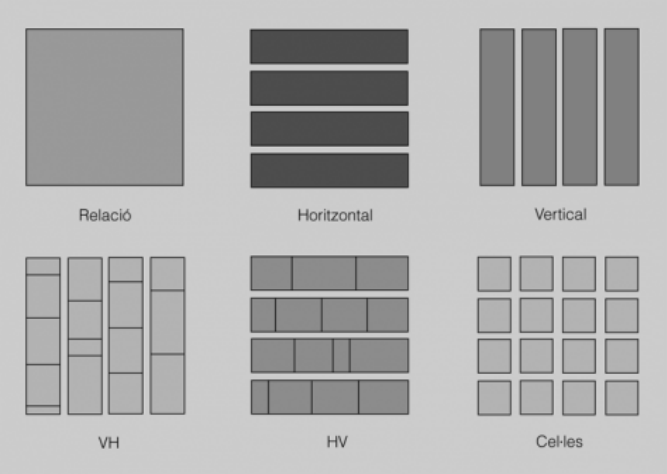
2. **Fragmentació HV**. S'aplica primer una divisió horitzontal i tot seguit es desenvolupa una altra de vertical sobre els fragments prèviament generats.

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.

La figura 3.1 mostra esquemàticament algunes possibilitats de fragmentació d'una relació.

FIGURA 3.1. Tipologies de fragmentació d'una relació



cc

3.6.4 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 tupla (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.

3.7 Replicació de dades

La replicació consisteix en l'emmagatzemament de còpies (rèpliques) de taules o fragments d'aquestes en diversos servidors de manera que les actualitzacions que s'hi duguin a terme a la còpia d'un servidor es propaguin a totes les altres.

Hi ha diverses avantatges i desavantatges en les rèpliques:

- **Disponibilitat**. Si algun dels llocs que conté la relació R falla, la relació pot trobar-se en un altre lloc diferent. Per tant, el sistema pot seguir processant les consultes que impliquin R, i tot a la decisió del lloc.
- **Paral·lelisme incrementat**. En cas que la majoria dels accessos a la relació R només resultin en la lectura de la relació, diversos llocs poden processar

en paral·lel les lectures que impliquin R tot a la decisió del lloc. Com més rèpliques de R hagi, major serà la possibilitat que les dades necessàries es troben en el lloc en que s'executa la transacció. Per tant, la rèplica de les dades minimitza el moviment de les dades entre els llocs.

- **Sobrecàrrega incrementada durant l'actualització**. El sistema ha d'assegurar que totes les rèpliques de la relació R siguin consistents, en cas contrari es poden produir còmputs erronis. Per això, sempre que s'actualitza R, cal propagar l'actualització a tots els llocs que contenen rèpliques. El resultat és una sobrecàrrega incrementada. Per exemple, en un sistema bancari, en què es replica en diversos llocs la informació dels comptes, cal assegurar-se que el saldo de cada compte concordi a tot arreu.

Així doncs podem dir que els avantatges d'un sistema distribuït es basen en disminuir els costos de comunicacions entre nodes i permetre que si cau un node de la xarxa pot seguir funcionant la base de dades.

Podem distingir dos tipus de replicació:

- **Replicació síncrona:** les actualitzacions que s'efectuen en un servidor es propaguen immediatament a totes les reproduccions que hi ha en altres servidors.
- **Replicació assíncrona:** els canvis es propaguen periòdicament a petició d'un administrador o per altres circumstàncies.

3.7.1 Caus persistents

El mecanisme de cau persistent consisteix a emmagatzemar en un servidor una còpia d'una part de les dades emmagatzemades en un altre servidor. Aquesta part és la que sembla que tingui més possibilitats de ser accedida.

Optimitzen l'accés per la lectura però les actualitzacions caldrà fer-les contra el servidor mestre.

3.8 Disseny de bases de dades distribuïdes

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.

3.8.1 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.

3.8.2 Estratègies: dissenys ascendent i descendent

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

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

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.

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 prevegi la migració a un SGBD distribuït, comú a tots els nodes.

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

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.

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.

3.8.3 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

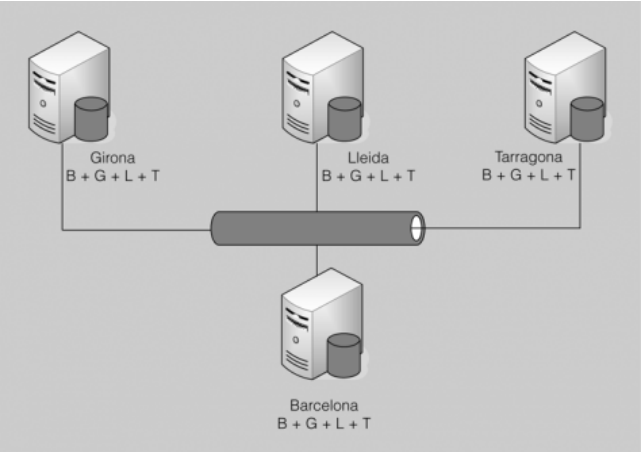
La multiplicació implica que la BD està replicada íntegrament a cada node del sistema. A la pràctica, aquest sistema és molt poc utilitzat.

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

Figura 3.2. BD multiplicada, on es mostra la ubicació de les diferents parts (B, G, L i T) de les dades de la BD



Divisió

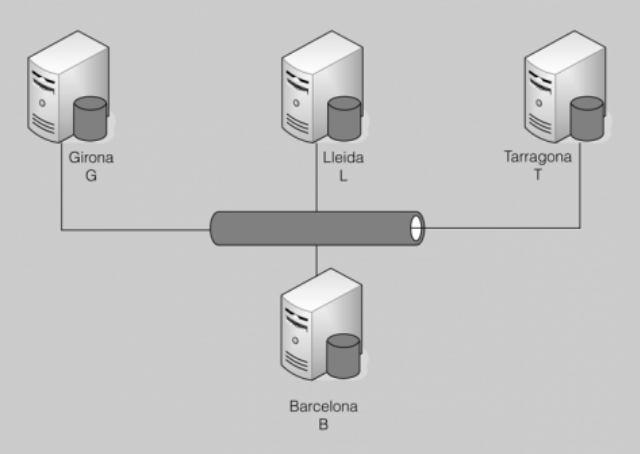
Amb el mètode de la divisió, la BD està distribuïda de tal manera que no hi ha cap part que estigui replicada en més d'un node.

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.

Com a contrapartides, podem dir que les operacions de consulta són sem- pre 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 3.3 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.

Figura 3.3. BD dividida, on es mostra la ubicació de les diferents parts (B, G, L i T) de les dades de la BD



Distribució amb node principal

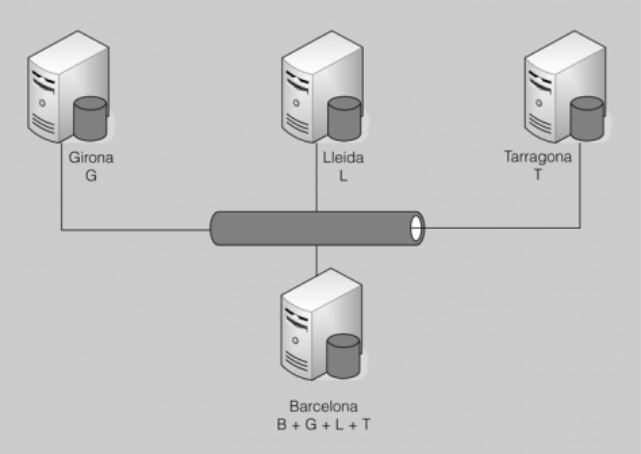
Quan s'utilitza el model de distribució amb node principal, un dels nodes, al qual es pot considerar principal, conté la BD sencera, i cadascun dels altres nodes conté replicada alguna part de la BD.

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 3.4 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 3.4. 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



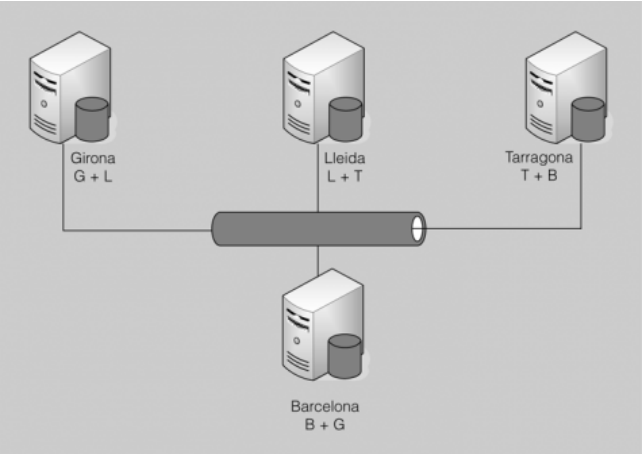
Distribució amb duplicacions en nodes seleccionats

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 3.5 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 3.5. 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



3.8.4 Regles fonamentals pel que fa al disseny de sistemes distribuïts

J. C. Date formula 12 regles que cal tenir en compte a l'hora de dissenyar i implementar una base de dades distribuïda. L'objectiu d'aquestes regles és que en cap cas l'usuari final tingui cap percepció sobre com estan distribuïdes les dades realment. Aquestes regles són les següents:

- **Màxima autonomia local:** cada un dels servidors ha de tenir la màxima autonomia pel que fa a seguretat, integritat de dades, optimització d'accés, gestió de transaccions.
- **Igualtat entre servidors:** no hi ha d'haver cap servidor central que s'encarregui de funcions essencials, de les quals depengui de manera general el funcionament dels altres servidors.
- **Operació continuada:** els servidors han d'estar disponibles, és a dir, les aplicacions hi han de poder accedir, encara que altres servidors hagin caigut o no sigui possible comunicar-s'hi.
- **Transparència de localització:** les aplicacions han de poder accedir tant a les estructures de dades com a altres elements de la BD sense necessitat d'especificar-ne la localització
- **Transparència de fragmentació:** les aplicacions han de poder ignorar la fragmentació física que, per motius de rendiment, presentin les dades lògicament similars.
- **Transparència de replicació:** les aplicacions han de funcionar de la mateixa manera amb independència que hi hagi dades replicades per millorar el rendiment.
- **Processament distribuït de consultes:** l'SGBD ha de ser capaç de processar i optimitzar les consultes sobre dades de més d'un servidor,

tenint en compte la localització de les dades, l'existència de reproducció o fragmentació i el cost del trànsit de xarxa.

- **Gestió distribuïda de transaccions:** les transaccions han de complir les propietats ACID a pesar de la distribució.
- **Independència de maquinari:** el funcionament del conjunt de servidors no s'ha de veure afectat per les diferències de maquinari.
- **Independència de sistema operatiu:** la utilització de sistemes operatius diferents en els servidors no ha de ser causa de limitacions en el funcionament.
- **Independència de protocols de xarxa:** la col·laboració entre diversos servidors no s'ha de restringir per l'existència de diferents protocols de comunicació per xarxa.
- **Independència d'SGBD:** la cooperació entre servidors ha de ser possible encara que hi hagi nodes que treballin amb SGBD diferents, tant si són del mateix fabricant com si són de fabricants diferents.

4. Administració d'un SGBD

Les tasques que corresponen a l'administració d'una base de dades solen ser bastant complexes i per això cal que el seu administrador tingui un perfil d'usuari molt experimentat, i sigui capaç d'enfrontar-se als problemes referents a la gestió dels diferents usuaris i a tots aquells que tinguin com a objectiu aconseguir un rendiment òptim del sistema.

4.1 Funcions de l'administrador de l'SGBD

En els diferents nivells i aplicacions de bases de dades hi ha la funció de l'administrador de la base de dades (DBA), encara que varia en complexitat. Aquesta és més senzilla quan es tracta d'una base de dades personal que quan es refereix a una base de dades de grups de treball, i aquesta al seu torn és més senzilla que en una base de dades organitzacional.

En una base de dades personal normalment l'usuari mateix és l'administrador de la base de dades, però les bases de dades de grups de treball requereixen una o dues persones que normalment no es dediquen a aquesta funció de temps complet, ja que tenen altres responsabilitats dins o fora de l'organització. En les bases de dades organitzacionals, que normalment permeten l'accés a desenes i fins i tot centenars d'usuaris, es requereix un administrador de base de dades de temps complet, a causa de l'alt volum de processos que han de desenvolupar, controlar i supervisar.

Un administrador de base de dades de temps complet normalment té aptituds tècniques per al maneig del sistema en qüestió i a més és desitjable que tingui nocions d'administració, maneig de personal i fins i tot un cert grau de diplomàcia. La característica més important que ha de tenir és un coneixement profund de les polítiques i normes de l'empresa i el criteri de l'empresa per aplicar-les en un moment donat.

La responsabilitat general del DBA és facilitar el desenvolupament i l'ús de la base de dades dins de les guies d'acció definides per l'administració de les dades.

El DBA és responsable primordialment del següent:

- Administrar l'estructura de la base de dades
- Administrar l'activitat de les dades
- Administrar el sistema gestor de base de dades
- Establir el diccionari de dades
- Assegurar la fiabilitat de la base de dades

- Confirmar la seguretat de la base de dades.

4.1.1 Administració de l'estructura de la base de dades

L'administració de l'estructura de la base de dades inclou participar en el disseny inicial i la posada en pràctica i també controlar i administrar-ne els requisits, i ajudar a avaluar alternatives, incloent-hi els SGBD per utilitzar i ajudar en el disseny general de BD. En els casos de grans aplicacions de tipus organitzacional, el DBA és un gerent que supervisa el treball del personal de disseny de la BD.

Definir les verificacions de seguretat i integritat

Les verificacions de seguretat i d'integritat es poden considerar part de l'esquema conceptual. El DDL lògic inclou els mitjans per especificar aquestes verificacions.

Les principals tasques de l'administrador de la base de dades són les següents:

- **Definició dels esquemes conceptual i lògic:** és tasca de l'administrador de dades decidir amb exactitud quina és la informació que s'ha de mantenir en la base de dades, és a dir, identificar les entitats que interessen a l'empresa i la informació que s'ha de registrar sobre aquestes entitats. Aquest procés en general es denomina *disseny conceptual de la base de dades*. Quan l'administrador de dades decideix el contingut de la base de dades en un nivell abstracte, el DBA crea a continuació el disseny lògic corresponent, utilitzant el DDL lògic. L'SGBD utilitzarà la versió objecte (compilada) d'aquest esquema per respondre les sol·licituds d'accés. La versió font sense compilar servirà com a document de referència per als usuaris del sistema.

- **Definició de l'esquema intern:** el DBA ha de decidir també com es representarà la informació a la base de dades emmagatzemada. Aquest procés se sol anomenar *disseny físic de la base de dades*. Un cop fet això el DBA haurà de crear la definició d'estructura d'emmagatzematge corresponent (és a dir, l'esquema intern) valent-se del DDL intern. A més, haurà de definir la correspondència pertinent entre els esquemes intern i conceptual. En la pràctica, ja sigui el DDL lògic o bé el DDL intern inclouen segurament els mitjans per definir aquesta correspondència, però les dues funcions (crear l'esquema i definir la correspondència) s'han de poder separar amb nitidesa. Igual que l'esquema lògic, l'esquema intern i la correspondència associada existiran tant en la versió font com en la versió objecte.

Una vegada dissenyada la BD, és posada en pràctica utilitzant productes de l'SGBD, i es procedeix llavors a la creació de les dades (captura inicial). El DBA participa en el desenvolupament de procediments i controls per assegurar la qualitat i l'alta integritat de la BD.

El DBA s'ha d'encarregar de la comunicació amb els usuaris, garantir la disponibilitat de les dades que requereixen i d'escriure –o ajudar els usuaris a escriure– els esquemes externs necessaris, emprant el DDL extern aplicable. A més, caldrà definir la correspondència entre qualsevol esquema extern i l'esquema

conceptual. En la pràctica, el DDL extern inclourà amb tota probabilitat els mitjans per especificar la correspondència, però en aquest cas també l'esquema i la correspondència s'hauran de poder separar amb claredat. Cada esquema extern i la correspondència associada existiran en totes dues versions font i objecte. Altres aspectes de la funció d'enllaç amb els usuaris inclouen les consultes sobre disseny d'aplicacions, la interpretació de la documentació tècnica, l'ajuda en la localització i resolució de problemes, i altres serveis professionals similars relacionats amb el sistema.

Els requisits dels usuaris es van modificant, ja que aquests troben noves formes o mètodes per assolir els seus objectius. Per altra banda, la tecnologia de la BD es va modificant i els fabricants de l'SGBD actualitzen els seus productes. Això implica que totes les modificacions en les estructures o procediments de BD requereixen una administració acurada.

Implicacions degudes a la modificació d'esquemes

Supervisar l'acompliment i respondre a canvis en els requisits

És responsabilitat del DBA organitzar el sistema de manera que s'obtingui l'exercici que sigui "millor per a l'empresa", i fer els ajustaments apropiats quan canviïn els requisits.

Les sol·licituds de modificació són inevitables una vegada que el sistema ha entrat en operació, poden aparèixer sol·licituds de nous requisits o poden resultar d'una comprensió inadequada d'aquests. En qualsevol cas, s'han d'efectuar modificacions en relació amb tota la comunitat de la BD, ja que l'impacte d'aquestes alteracions ressentirà més d'una aplicació. En alguns casos, es poden donar modificacions que presenten efectes negatius per a alguns usuaris, i aquests casos hauran de ser tractats esgrimint com a argument els beneficis globals que seran obtinguts d'aquestes alteracions.

Una administració eficaç de la BD ha d'incloure procediments i polítiques mitjançant les quals els usuaris puguin registrar les seves necessitats de modificacions, i així la comunitat podrà analitzar i discutir els impactes d'aquestes modificacions, i determinar llavors la posada o no en pràctica d'aquestes alteracions.

En raó de la mida i complexitat d'una BD i de les seves aplicacions, les modificacions poden tenir resultats inesperats. El DBA ha d'estar preparat per reparar la BD i reunir suficient informació per diagnosticar i corregir el problema provocat per la falla. Després d'un canvi la BD és més vulnerable a fallides.

Documentació

La responsabilitat final d'un DBA en l'administració de l'estructura d'una BD és la *documentació*. És molt important saber quines modificacions han estat efectuades, com van ser fetes i quan van ser establertes. Una modificació sobre l'estructura de la BD pot ocasionar un error que no aparegui a curt termini, i una vegada que aquest surti, sense la documentació adequada sobre les modificacions fetes, el diagnòstic resultaria extremadament complicat. En aquests casos, es faria necessària una seqüència de reexecucions per intentar detectar el punt en conflicte, i el risc d'aquest procediment és que és possible afectar la informació continguda a la BD. Per identificar un canvi és molt important mantenir un registre dels formats de prova i de les execucions de les proves efectuades. Si s'utilitzen procediments

de prova, formats de proves i mètodes de registre estandarditzats, el registre dels resultats de la prova no consumirà un temps excessiu.

Normalment el temps de la documentació és tediós i això fa que alguns DBA tendeixen a reduir o abreujar la informació que s'hi registra i fins i tot l'arriben a desatendre. Quan ocorre un sinistre, la documentació completa i organitzada pot ser la diferència entre resoldre o no un problema d'extrema importància en la majoria dels casos, i pot implicar costos molt grans a l'empresa.

La tasca de la documentació és cada vegada més lleugera i precisa quan s'utilitzen SGBD que integren eines CASE per a les tasques de disseny, manteniment i documentació. Aquestes eines CASE mateixes proporcionen en la majoria dels casos la facilitat de generar i mantenir de manera automàtica el diccionari de dades.

Una raó més per documentar consisteix en la necessitat de mantenir organitzades les dades històriques. Passa sovint que es vol fer una consulta sobre els suports per conèixer l'estat que tenia la informació en un període determinat que va transcórrer prèviament. Els registres de modificació existents en la documentació permetran resoldre problemes d'incompatibilitat entre les estructures que eren vigents en el període de suport i les que ho són ara, i permetran també el desenvolupament de mòduls d'ajust que facilitin la traducció de formats o escales per d'emmagatzematge.

En els casos de caigudes del sistema es presenta una situació semblant: les còpies de seguretat són requerides i s'ha de verificar l'estructura, format i escala per integrar-los en l'operació del sistema.

4.1.2 Administració de l'activitat de les dades

Encara que el DBA protegeix les dades, no les processa. El DBA no és usuari del sistema; en conseqüència, no administra valors de dades, sinó que administra l'activitat de les dades. Atès que la BD és un recurs compartit, el DBA ha de proporcionar estàndards, guies d'acció, procediments de control i la documentació necessària per garantir que els usuaris treballen de manera cooperativa i complementària en processar dades a la BD.

Com es pot suposar, hi ha una gran activitat a l'interior d'un SGBD. La concurrència de múltiples usuaris requereix estandarditzar els processos d'operació, el DBA és responsable d'aquestes especificacions i d'assegurar que aquestes arribin als qui en són afectats. Tot l'àmbit de la BD es regeix per estàndards, des de la manera com es captura la informació (tipus, longitud, format), fins a com és processada i presentada. El nivell d'estandardització arriba fins als aspectes més interns de la BD, com s'accedeix a un arxiu, com es determinen els índexs primaris i auxiliars, la foliació dels registres i d'altres.

S'ha de procurar sempre que els estàndards que s'han d'aplicar beneficiïn també els usuaris, privilegiant sempre l'optimització en l'operació de l'SGBD i l'afecció de les polítiques de l'empresa.

Una administració de BD efectiva ha de disposar sempre d'aquest tipus d'estàndards; entre les funcions del DBA es troba la de revisar-ho periòdicament per determinar-ne l'operativitat, i si escau ajustar-los, ampliar-los o cancel·lar-los. És també la seva responsabilitat que aquests es compleixin.

Quan es defineixen estàndards sobre l'estructura de la BD, aquests s'han de registrar en una secció del diccionari de dades a la qual tots aquells usuaris relacionats amb aquest tipus de procés poden accedir.

Un altre dels aspectes que l'administrador ha d'atendre és el de coordinar totes aquelles noves propostes de modificació dels drets d'accés a dades compartides i les propostes corresponents a noves aplicacions. Primerament, caldria que aquestes propostes fossin analitzades en conjunt amb els supervisors o directius de les àrees involucrades per determinar si cal fer aquestes modificacions, no fos cas que poguessin aparèixer problemes quan dos o més grups d'usuaris queden autoritzats per notificar les mateixes dades.

Un d'aquests conflictes és el de l'actualització perduda: es dóna quan el treball d'un usuari queda sobreescrit pel d'un segon usuari. El DBA queda responsabilitzat d'identificar la possible ocurrència d'aquests problemes, i també de crear normes i procediments per impedir-los. S'obtindran aquest tipus de garanties quan l'SGBD sigui capaç d'implementar les restriccions aplicables a l'accés concurrent, i aquest sigui utilitzat adequadament per programadors i usuaris, per evitar el que hem esmentat anteriorment. Per això es fa indispensable ajustar-se als estàndards corresponents al seguiment d'instructius i documentar mitjançant manuals totes aquells regles establertes per als diversos processaments i procediments que es duen a terme.

Concessió de l'autorització per a l'accés a dades

La concessió de diferents tipus d'autorització permet a l'administrador de la base de dades regular a quines parts de la base de dades podran accedir els diversos usuaris.

Entre les alternatives més utilitzades pel DBA per intentar resoldre o minimitzar aquest problema es troben les següents:

- Restringir l'accés als procediments per a certs usuaris.
- Restringir l'accés a les dades per a certs usuaris, procediments o dades.
- Evitar la coincidència d'horaris per a usuaris que comparteixen les mateixes dades.

Les tècniques de recuperació són una altra funció essencial del DBA en administrar l'activitat de dades.

Tot i que el SGBD porta a terme una part del procés de recuperació, els usuaris determinen de manera crítica l'operativitat d'aquests sistemes de protecció. El DBA ha d'anticipar fallides i definir procediments estàndard d'operació, els usuaris han de saber què han de fer quan el sistema està caigut i què és el primer que s'ha de fer quan el sistema està en marxa novament. El personal d'operació haurà de saber com iniciar el procés de recuperació de la BD, quines còpies de seguretat cal utilitzar, com cal programar la reexecució del temps perdut i de les tasques pendents. És important també establir un calendari per dur a terme aquestes activitats sense afectar altres sistemes dins de l'organització que facin ús dels mateixos recursos de còmput.

Destaquen per la seva importància en el procés de recuperació i al seu torn en l'atenció que presten a altres sectors de l'organització: els dispositius de comunicació remota, els sistemes d'interconnexió i altres accessoris d'ús compartit.

El DBA és el responsable de la publicació i manteniment de la documentació en relació amb l'activitat de les dades, incloent-hi els estàndards de la BD, els drets de recuperació i d'accés a la BD, els estàndards per a la recuperació de caigudes i el compliment de les polítiques establertes.

Els productes SGBD més populars que es troben en el mercat proporcionen eines per ajudar el DBA en l'administració de les dades i la seva activitat. Alguns sistemes registren automàticament els noms dels usuaris i de les aplicacions a les quals tenen accés i també altres objectes de la BD. Incorporen també utilitats que permetin definir en el diccionari de dades les restriccions perquè determinades aplicacions o mòduls d'aquestes només tinguin accés a segments específics de la BD.

4.1.3 Administració de l'SGBD

A més d'administrar l'activitat de dades i l'estructura de la BD, el DBA ha d'administrar l'SGBD mateix. Haurà de compilar i analitzar estadístiques relatives al rendiment del sistema i identificar àrees potencials del problema. Atès que la BD està servint molts grups d'usuaris, el DBA ha d'investigar totes les queixes sobre el temps de resposta del sistema, la precisió de les dades i la facilitat d'ús. Si es requereixen canvis el DBA se'ls ha de planejar i posar-los en pràctica.

El DBA haurà de vigilar periòdicament i contínuament les activitats dels usuaris a la BD. Els productes SGBD inclouen tecnologies que reuneixen i publiquen estadístiques. Aquests informes poden indicar quins van ser els usuaris actius, quins arxius i quins elements de dades han estat utilitzats, i fins i tot el mètode d'accés que s'ha aplicat. Poden capturar i reportar les taxes d'error i els tipus d'errors. El DBA analitzarà aquestes dades per determinar si cal una modificació en el disseny de la BD per gestionar-ne el rendiment o per facilitar les tasques dels usuaris; en aquest cas, el DBA la durà a terme.

Casos com l'addició de noves claus o l'eliminació, noves relacions entre les dades i altres situacions típiques han de ser analitzades per determinar el tipus de modificació escaient.

El DBA haurà d'analitzar les estadístiques de temps d'execució sobre l'activitat de la BD i el seu rendiment. Quan s'identifiqui un problema de rendiment, ja sigui mitjançant una queixa o un informe, el DBA ha de determinar si resulta apropiada una modificació en l'estructura de la BD o en el sistema.

Quan el fabricant de l'SGBD anuncïi una nova versió del producte, s'ha de fer una anàlisi de les característiques que incorpora i analitzar-les segons les necessitats de la comunitat d'usuaris. Si es decideix l'adquisició del producte, els usuaris n'han de ser notificats, i també capacitats per utilitzar-lo. El DBA haurà d'administrar i controlar la migració, tant de les estructures, com de les dades i les aplicacions.

El programari de suport i altres característiques de maquinari poden implicar també modificacions de les quals el DBA és responsable ocasionalment; aquestes modificacions tenen com a conseqüència canvis en la configuració o en alguns paràmetres d'operació de l'SGBD.

Les opcions de l'SGBD són ajustades al principi, és a dir, en la posada en marxa del sistema es coneix molt poca informació sobre les característiques de funcionament i resposta que proporcionarà als grups d'usuaris. L'anàlisi de l'experiència operacional i el seu rendiment en un període determinat de temps poden revelar que es requereix un camp. Si el rendiment sembla acceptable, el DBA pot considerar modificar algunes opcions i observar-ne l'efecte sobre el sistema, tot buscant optimitzar-lo o afinar-lo.

4.1.4 Assegurar la fiabilitat de la base de dades

Quan una empresa es decideix a utilitzar un sistema de base de dades, es torna dependent en grau màxim del funcionament correcte d'aquest sistema. En el cas que pateixi un dany qualsevol porció de la base de dades per causa d'un error humà o una fallida en l'equip o en el sistema que li dóna suport, és essencial poder reparar les dades implicades amb un mínim de retard i afectant el menys possible la resta del sistema. En teoria, per exemple la disponibilitat de les dades no danyades no s'hauria de veure afectada. El DBA ha de definir i posar en pràctica un pla de recuperació adequat que inclogui, per exemple una descàrrega o buidatge periòdic de la base de dades en un medi d'emmagatzematge de suport, i procediments per carregar una altra vegada la base de dades a partir del buidatge més recent quan sigui necessari.

4.2 Còpies de seguretat i recuperació de dades

Les coses es trenquen, és un fet. Quan tot és nou i ho acabem d'instal·lar tot rutlla segons el que està establert, però amb el temps es va afegint nou programari al sistema, noves aplicacions, nou maquinari, etc. Una de les feines més importants dels DBA és reaccionar davant els possibles errors que es produeixin en el nostre sistema. Aquests es poden classificar en tres grans grups:

1. **Problemes al sistema**. Inclouen problemes del sistema operatiu sobre el qual està instal·lat l'SGBD, problemes interns de l'SGBD o problemes relacionats amb algun programari.

2. **Problemes d'aplicació o transaccions**. Es donen quan hi ha una execució de programes o procediments fora d'un temps establert o les entrades del programa no són les correctes. Aquestes fallades poden crear dades incorrectes que haurem de detectar i corregir.

3. **Problemes de maquinari**. Aquest tipus de problemes vénen donats quan fallen els medis on desem les dades (disc dur, cintes, etc.). Aquests problemes cada vegada són menys freqüents i, si es donen, s'autocorregeixen si utilitzem mesures de seguretat com el RAID.

4.2.1 Còpies de seguretat

Documentació de la política de seguretat

S'ha de mantenir a cada moment una documentació de la política de còpies de seguretat i de quina manera està tot planificat (tipus de còpies, periodicitat, continguts).

És feina del DBA establir una planificació per tal de fer les còpies de seguretat de les dades adequades en el moment adequat.

Per saber amb quina freqüència hem de fer una còpia de desament de dades ens fixarem amb quant de temps és necessari per tal de restaurar-les, i per fer això ens fixarem en els factors següents:

- El nombre de registres que s'han de restaurar.
- El temps que es triga a muntar i desmuntar les cintes pertinents.
- El temps que es triga a processar els canvis dels registres una vegada han estat restaurats.

A més de fer això hem de tenir clar quan hem de fer les còpies de seguretat i en quina quantitat. Tot seguit teniu unes possibles i més que recomanables pautes a seguir en aquest procés:

- Fer, com a mínim, dues còpies per evitar les possibles errades del medi on s'emmagatzemen les còpies.
- Fer còpies locals i en la mesura que es pugui fer-les en medis fora del servidor (altres servidors, altres medis).
- Conservar dues rondes de còpies sempre. És a dir, si fem una còpia integral del sistema cada dilluns, mantenir les còpies dels dos dilluns anteriors.
- Ens hem d'assegurar d'incloure el catàleg del sistema, ja que si ha canviat el DDL (*data definition language*, llenguatge de definició de dades) tindríem error fins i tot havent restaurat el sistema.
- Comprovar, una vegada acabat el procés de desament, que la còpia és correcta.

Còpies de seguretat integrals i incrementals

Les **còpies integrals** són aquelles on les dades que es desen són totes les que hi ha al sistema en aquell moment.
Les **còpies incrementals** són aquelles que només desen les dades que han canviat des de l'última data en què es va fer una còpia integral o incremental.

Lògicament, les còpies incrementals ocupen menys espai físic i tarden menys a fer-se.

Exemple de realització d'una còpia de seguretat

Suposem que els dilluns fem una còpia integral, i els dimarts i dimecres en fem d'incrementals. Si per qualsevol causa hem de restaurar el sistema el dijous, hauríem de seguir els passos següents:

- Restaurar la còpia integral del dilluns.
- Restaurar la còpia incremental del dimarts.
- Restaurar la còpia incremental del dimecres.

Hem de fer una valoració de quin és el millor tipus de còpia de seguretat que podem utilitzar a cada moment o sistema gestor. Per tant, és imprescindible que, com a DBA, sapiguem quina quantitat de registres hem de desar i aproximadament quant de temps es triga a fer una còpia integral i incremental.

Hem de tenir present que el temps que ens estalviem fent còpies de seguretat incremental només és útil si el nombre de registres que canvien entre còpies de seguretat és reduït.

Altres aproximacions a les còpies de seguretat

És aconsellable fer còpies de seguretat utilitzant les eines que proporciona el mateix SGBD. Tot i així, hi ha diverses formes de fer còpies de seguretat alternatives, que només s'han d'utilitzar en moments molt concrets.

Fins ara hem estat parlant de còpies de tot el sistema, però de vegades aquest tipus de còpies de seguretat més que ajudar en la restauració ens poden donar problemes segons les dades que s'hagin perdut.

Veurem ara una forma de fer una còpia de seguretat lògica (ens fixarem en les dades emmagatzemades, no tant en el tipus de dades). La instrucció `EXPORT` o `UNLOAD`, segons l'SGBD, ens permetrà fer un bolcat de totes les dades indicades i així tenir una còpia lògica de les dades. Veiem en quins casos ens pot arribar a ser útil:

- **Restauració d'una fila o objecte en especial**. Si per error algú esborra només unes quantes files o dades d'una taula, pot arribar a ser molt complicat només restaurar aquesta part des d'una còpia de seguretat física.
- **Actualització de l'SGBD**. Si ens trobem davant una actualització del sistema gestor i no volem convertir totes les estructures, també pot ser útil fer una introducció massiva de les dades des d'una còpia lògica.
- **Migració de dades entre SGBD heterogenis**. Si volem transportar dades d'un sistema gestor a un altre que disposa d'estructures físiques ben diferenciades, també optarem per restaurar una còpia lògica al sistema destinatari de les dades.

- **Moviment de dades**. Per traslladar les dades entre aplicacions o fer-ho per exemple en fitxers de text, seria l'única manera d'aconseguir-ho.

Com que la còpia de seguretat lògica es fa amb el sistema en funcionament, l'únic problema de rendiment pot venir donat per l'accés concurrent a les dades.

4.2.2 Restauració de dades

La restauració de dades no és una tasca senzilla. És més que dur a terme una restauració d'una imatge prèviament feta. Hem de tractar de deixar el sistema tal com requereixi l'aplicació en qüestió, incorporant-hi tots els possibles canvis que hagin succeït a les bases de dades des del moment de la còpia de seguretat fins al moment exacte que nosaltres volem.

Estratègies de recuperació de dades

El primer que hem de fer davant una fallada del sistema és tenir ben present de quins recursos disposem per fer la restauració, és a dir, de quines còpies disposem i on es troben. El pas següent és descobrir exactament què ha produït la fallada del sistema. Hi ha una sèrie de preguntes que ens poden ajudar a enfocar els nostres esforços:

- Com s'ha apagat el sistema gestor? De manera fortuïta o per la nostra ordre?
- Té a veure amb el sistema operatiu?
- S'ha reiniciat el servidor?
- Hi ha errors en els fitxers de LOG del sistema operatiu o del servidor de bases de dades?
- Quina és la importància de les dades que s'han perdut?
- S'ha intentat algun tipus de restauració? Si és així, en quin punt ens trobem de la restauració?
- Què es necessita restaurar? La base de dades, una taula o un altre element?
- Tenim còpies lògiques recents?
- Es pot accedir als objectes de la base de dades?

Una vegada disposem de les respostes, estarem preparats per poder iniciar la restauració de les dades perdudes. Els passos que cal seguir per fer-la són els següents:

1. **Identificar** l'error que provoca la fallada del sistema: les aplicacions o els usuaris han de proporcionar el tipus d'error que els indica l'aplicació que accedeix a les dades.

2. **Analitzar** la situació: esbrinar fins on arriba exactament l'error i quines parts s'han vist realment compromeses.

3. **Determinar** què s'ha de restaurar exactament: mirar quines dades necessiten una restauració (pot portar molt de temps fer-ho).

4. **Identificar dependències** entre objectes de la base de dades: possibles restauracions de certs objectes poden comportar restaurar-ne altres com a efecte col·lateral.

5. **Localitzar** les còpies que s'han d'utilitzar i veure si són operatives: localitzar els dispositius i comprovar que s'hi pot accedir (no falla el medi d'emmagatzematge).

6. **Restaurar** les còpies de seguretat.

7. **Retornar** el sistema al punt exacte de funcionament desitjat: utilitzar els fitxers de moviment per deixar el sistema exactament al moment que necessitem.

Tipus de restauracions de dades

El primer tipus de restauració que veurem és, potser, un dels més comuns. Es tracta de la **recuperació fins al moment on va fallar**. Se sol donar a causa de problemes amb els medis d'emmagatzematge (discos durs per exemple).

Es necessita una còpia integral del sistema per començar la restauració, ja que serà el punt de partida. Una vegada restaurada aquesta imatge, hem d'utilitzar els fitxers LOG de moviments per deixar el sistema en el mateix estat que estava abans de la fallada. Si per qualsevol motiu no trobem una còpia integral actual, hem de recórrer a una còpia antiga i anar restaurant també les còpies incrementals successives fins a arribar al punt on podríem utilitzar els fitxers LOG de moviments.

Un altre tipus de restauració és aquell en què portem les bases de dades **a un estat determinat en el temps**. Per exemple, si sabem que el nostre sistema va fallar a les 13.43 del migdia, intentarem portar el nostre sistema fins a com estava a les 13.42, sigui l'hora que sigui. Aquest tipus de restauració se sol fer només per a una part del sistema gestor. Generalment no afecta tot el conjunt de dades del sistema. Hem de tenir clar com actuarem amb les transaccions que s'han efectuat al sistema des de la fallada fins al moment de fer la restauració. Es tracta, doncs, de fer una restauració del sistema aplicant els canvis correctes i rebutjant els canvis que no interessen i que han produït aquest error del sistema.

Hi ha dues maneres de fer aquest tipus de restauració, i l'aproximació per la qual optem vindrà donada pels mecanismes de què disposi el nostre SGBD i la quantitat de dades que s'han de restaurar. Podrem optar, doncs, per:

- Restaurar una còpia i aplicar els canvis que ens interessin dels fitxers LOG fins a arribar al punt que vulguem.

- No restaurar cap còpia. Desfarem els canvis erronis fixant-nos en els fitxers LOG fins a deixar el sistema en un punt consistent.

Si podem optar per totes dues solucions, escollirem sempre la que faci que el sistema estigui aturat menys temps.

Per finalitzar la tipificació de les possibles restauracions que ens podem trobar, ens queda per veure la **restauració de transaccions**. Les transaccions vénen definides per les aplicacions dels usuaris. Tenint en compte aquest fet, nosaltres, com a DBA, no podrem fer gaire cosa, ja que serà necessària l'ajuda d'un programari extern per reparar els possibles errors que s'hi hagin produït. Aquests errors sempre estan relacionats amb la pèrdua de certa informació important.

Diem que cal una aplicació externa a l'SGBD per fer la reparació perquè només l'usuari sap què esperava fer en la transacció i com ha quedat la informació de la transacció. Haurà de tornar enrere desfent totes les modificacions que ha fet i tornar a intentar l'operació una altra vegada. Per tirar enrere "sobre els seus propis passos", ho podrà fer amb les ordres `REDO` i `UNDO`.

Altres alternatives a les còpies de seguretat i les recuperacions

Hi ha altres maneres de protegir l'SGBD a part de les còpies de seguretat. Vegem-ne unes possibles alternatives que, per descomptat, no ens eximeixen en cap cas de tenir còpies de seguretat:

RAID

RAID és l'acrònim de *redundant array of inexpensive disks* ; en català, conjunt redundant de discos barats. Aquest sistema ofereix seguretat tot fent còpies simultànies de la mateixa informació en diferents discos.

CRC

CRC és l'acrònim de *control de redundància cíclica*. El CRC és un mecanisme que permet detectar errors en sistemes digitals mitjançant certes codificacions.

- **Replicació**. Es tracta de mantenir unes còpies redundants exactes de les bases de dades en discos i suports diferents. Com més còpies redundants tinguem més segurs estarem, davant possibles fallades del sistema, que podrem restaurar-lo en poc temps.
- **Discos mirall**. Es tracta d'aconseguir en el mercat el maquinari que proporciona aquest tipus de tecnologia, en què es mantenen diversos discos amb la mateixa informació replicada de maneres diferents (RAID) i amb una certa seguretat de dades incorporada (CRC). Aquest tipus de seguretat ens estalviaria la recuperació necessària en tenir una fallada d'algun dels discos.

4.2.3 Còpia de seguretat en el PostgreSQL

Els fitxers en què el PostgreSQL emmagatzema la BD són fitxers físics en disc des del punt de vista del sistema operatiu, i per això a mesura que s'incorpora informació, aquests van augmentant de mida.

Juntament amb la informació útil que es desa en els fitxers de la BD, s'emmagatzemen dades relatives a l'ordre d'aquesta, al tipus de dades, i altres necessàries per a l'accés i el funcionament correcte de l'SGBD. Això fa que de manera

estimada, els fitxers esmentats ocupin almenys el doble d'espai que les dades que emmagatzemen.

Per assegurar la durabilitat de totes aquestes dades, cal que l'administrador de la BD faci còpies de seguretat periòdiques, ja que malgrat que el PostgreSQL és molt estable, i fins i tot els discos durs tinguin redundància, res no permetria recuperar les dades en cas de robatori físic de l'equip, d'incendi, o d'altres accidents.

Les còpies de seguretat "totals" es poden fer d'almenys tres maneres:

- **Còpia dels fitxers de la BD en fred**. Per fer una còpia en fred, s'ha d'aturar l'SGBD i desar els fitxers on PostgreSQL emmagatzema les taules i la resta d'informació. Aquest mètode és poc recomanable, ja que implica no poder utilitzar el sistema durant el temps en què s'executa aquest procés. La restauració posteriorment també pot ser complicada, ja que només es podrà recuperar la còpia de manera total, i no serà possible fer recuperacions parcials.
- **Abocament des de rutines pròpies**. L'opció d'utilitzar rutines pròpies per fer l'abocament de les dades tampoc no és gaire recomanable, ja que pot passar que no es tinguin permisos d'accés a totes les taules, o que en fer l'abocament, les dades resultants no tinguin integritat referencial, per haver accedit a les taules de manera seqüencial.
- **Còpia en calent**. L'opció de fer una còpia de seguretat en calent, amb la utilitat *pg_dump* que el PostgreSQL mateix incorpora, sembla la més bona. Aquest programa serveix per fer un abocament total del contingut de la BD pel canal estàndard de sortida del sistema operatiu.

Si fem una canalització a un fitxer, aquest després es podrà tornar a utilitzar per fer la càrrega de dades. Si tenim en funcionament dues bases de dades, també es podran utilitzar canalitzacions *pipe-out* i *pipe-in*, per fer un traspàs directe de la informació entre aquestes. Aquesta última opció és especialment útil en fer migracions de versió, o en haver de substituir el maquinari, ja que els SGBD es poden estar executant en diferents ordinadors.

Exemple de backup

En primer lloc localitzarem l'executable **pg_dump** o **pg_dumpall** (en sistemes Windows) en el directori *../bin* d'allà on és instal·lat l'SGBD.

Tot seguit es pot utilitzar la instrucció amb els paràmetres:

```
pg_dump --host=\\nomServidor --port=5432 -username=postgres  --password >backup
    .txt
```

Les clàusules `host` i `port` són opcionals si s'executa des del mateix ordinador on hi ha la base de dades i si el port de comunicacions és l'estàndard.

En executar l'abocament, ens demanarà la contrasenya de l'usuari *postgres*.

El fitxer *backup.txt* contindrà l'abocament de les dades, i totes les instruccions necessàries per refer les taules, procediments emmagatzemats, usuaris i tots aquells paràmetres necessaris per aconseguir que es pugui reconstruir la BD original, en restaurar aquest fitxer sobre una BD buida.

A grans trets, l'ordre i contingut del fitxer *backup.txt* és (per blocs):

```
-- PostgreSQL database cluster dump'' Connexió a la tablespace de la qual es fa l'abocament.

-- Users Instruccions de creació dels usuaris existents.

-- Database creation Instruccions per a la creació de la BD.

-- Users Paràmetres addicionals dels usuaris de la BD.

-- PostgreSQL database dump Paràmetres de la BD (valors dels 'SET').

-- Name: DATABASE [nomTablespace]; Type: COMMENT; Schema: -; Owner: Comentaris DATABASE

-- Name: SCHEMA public; Type: COMMENT; [nomUsuari] Comentaris dels SCHEMA existents.

-- Name: [nomFuncio]() ; Type: FUNCTION; Schema: public; Owner: [nomUsuari] Codi plpgsql de les funcions existents.

-- Name: [nomLlenguatge]; Type: PROCEDURAL LANGUAGE; public; Owner:'' Instruccions de creació dels llenguatges instal·lats.

-- Name: [nomTipus]; [nomUsuari] Creació dels tipus.

-- Name: nom_taula; Type: [nomUsuari]; Tablespace: Creació de taules.

-- Data for Name: test; Type: TABLE DATA; Schema: public; Owner: [nomUsuari] Instruccions de l'abocament de les dades de la taula.

-- Name: [nomVista]; [nomUsuari] Creació de vistes.

-- Data for Name: [nomTaula]; Type: TABLE DATA; Schema: public; Owner: [nomUsuari] Instrucció d'abocament de dades de les taules, i les dades d'aquestes.

-- Name: [nomResticcio]; Type: (PK/FK) CONSTRAINT; Schema: public; Owner: [nomUsuari]; Tablespace: Modificació de taules per afegir les CONSTRAINT (claus primàries, índexs, etc).

-- Name: [nomDisparador]; Type: TRIGGER; Schema: public; Owner: [nomUsuari] Definició de disparadors

-- Name: public; Type: ACL; Schema: -; Owner: [nomUsuari] Assignació de privilegis d'accés dels usuaris sobre les dades.

-- PostgreSQL database dump complete Indicació de fi d'abocament de la BD.
```

L'avantatge d'aquest mètode de còpia de seguretat és que es pot editar el fitxer de còpia per cercar informació concreta. En aquest trobarem tant les dades contingudes en les taules, com les sentències de creació d'aquestes (amb les claus primàries i foranes), i fins i tot el codi *plpgsql* de les funcions i disparadors.

Exemple de restauració

El contingut d'una còpia de seguretat es pot restaurar totalment o parcialment.

En el cas de voler una restauració parcial, haurem d'editar el fitxer on s'ha emmagatzemat la còpia, i traspassar el que volem editar a un nou fitxer. Utilitzarem aquest últim per fer una restauració parcial.

La importació es farà utilitzant una canonada del sistema operatiu mateix:

```
pgsql NomBaseDades < backup.txt
```

Per restaurar tot el contingut d'una BD, primer l'esborraríem per assegurar no quedessin dades, després la crearíem, per finalitzar amb la restauració pròpiament dita.

Igual que en l'exemple anterior, utilitzarem una canonada del sistema operatiu per fer la importació de les dades:

```
dropdb NomBaseDades

createdb -E UTF-8 NomBaseDades

psql NomBaseDades < backup.txt
```

4.3 Manteniment rutinari de la base de dades

Hi ha una sèrie d'activitats que l'administrador d'un sistema gestor de bases de dades ha de tenir presents constantment i que haurà de fer periòdicament. En el cas del PostgreSQL, aquestes es limiten a un manteniment i neteja dels identificadors interns i de les estadístiques de planificació de les consultes, a una reindexació periòdica de les taules i al tractament dels fitxers de registre.

Tal com s'ha exposat al llarg del present mòdul, l'SGBD PostgreSQL gairebé no requereix manteniment. No obstant això, per assegurar-ne el bon rendiment, caldrà executar periòdicament els processos següents:

1. Eliminació dels registres marcats com a obsolets (generats a causa de l'ús de l'MVCC).
2. Regeneració dels clústers creats.
3. Regeneració de la informació estadística de les taules.

Així, caldrà automatitzar respectivament l'execució de la seqüència d'ordres següents:

1. VACUUM FULL;
2. CLUSTER;
3. ANALYZE;

Si incloem aquestes instruccions en un fitxer (amb nom *sql.txt*, per exemple), es podria automatitzar l'execució d'un *script* des del sistema operatiu de l'ordinador, perquè executés la línia:

```
..\rutaBinDelPostgreSQL\psql -U postgres -d nom_db -f sql.txt <password.txt
```

En què `plsql` és la instrucció que permet executar les ordres, `nom_db` és la base de dades, `postgres` és el nom d'usuari que ha d'executar les instruccions que hi ha al fitxer *sql.txt*, considerant que en l'interior del fitxer *password.txt* es troba la contrasenya d'aquest usuari.

4.3.1 Vacuum

Quan s'esborra una fila d'una taula, per qüestions de rendiment, la fila no s'esborra realment del fitxer que conté la taula, sinó que senzillament es desindexa dels índexs que l'afectin i es marca com a esborrada. Les noves insercions tampoc no faran servir aquest espai, sinó que s'afegiran sempre al final del fitxer.

Això fa que les operacions d'inserció i esborrament siguin molt més ràpides del que ho serien si haguessin d'anar movent dades d'aquí cap allà per ajustar la mida de cada fila amb la següent. Però també implica que si tenim un volum important d'operacions d'esborrament, l'espai utilitzat al disc creix molt més del realment necessari per a les dades. A més, la dispersió més gran de les dades dins la taula tampoc no afavoreix el temps de resposta en les consultes.

El procés que fa la neteja de la base de dades en PostgreSQL es diu `VACUUM`. L'operació `VACUUM` compacta els fitxers de les taules i eliminen definitivament les fileres esborrades i manté, lògicament, la correlació dels índexs. La necessitat de dur a terme processos de `VACUUM` periòdicament es justifica pels motius següents:

- Recuperar l'espai de disc perdut en esborraments i actualitzacions de dades.
- Actualitzar les estadístiques de dades utilitzades pel planificador de consultes SQL.
- Protegir-se davant de la pèrdua de dades per reutilització d'identificadors de transacció.

Per dur a terme un `VACUUM`, haurem d'executar periòdicament les sentències `VACUUM` i `ANALYZE`. A partir de la versió 8.1 del PostgreSQL, la base de dades mateixa es preocupa, per defecte, de fer els *vacuums* quan ho estima necessari.

Tot i que, si ens interessa, podem deshabilitar aquesta funcionalitat si preferim encarregar-nos nosaltres de programar els *vacuums* com més ens interessi.

Per exemple, passant-li com a paràmetre el nom d'una o més taules, podem fer *vacuum* exclusivament d'aquelles taules i evitar així perdre temps amb altres taules que sabem que tenen un índex de fragmentació molt més baix.

En cas que hi hagi algun problema o acció addicional per fer, el sistema ens ho indicarà.

4.3.2 Cluster

Quan es fa l'operació `CLUSTER` sobre una taula aquesta es reordena físicament segons la informació d'un índex especificat definit sobre aquesta taula. Si la taula s'actualitza posteriorment, de mica en mica aquesta ordenació es va perdent, és a dir, no s'intenten desar files noves o actualitzades d'acord amb el seu ordre de l'índex. Per tant, periòdicament caldrà emprar l'ordre `CLUSTER` de nou perquè es faci aquesta reestructuració.

La instrucció `CLUSTER` sense cap paràmetre reorganitza totes les taules prèviament agrupades. Quan una taula s'agrupa, aquesta adquireix bloqueig d'accés exclusiu. Això evita que es faci qualsevol operació de modificació sobre la taula fins que l'acció de la instrucció `CLUSTER` hagi finalitzat.

4.3.3 Analyze

A l'hora de planificar una consulta, el PostgreSQL té en compte molts factors, com ara els índexs de què disposa, la mida de la taula, i altres dades estadístiques sobre les dades emmagatzemades a la taula.

Obtenir tota aquesta informació a cada consulta costaria molt més que l'avantatge que ens reporta, i per això no es fa automàticament, sinó que es calcula un primer cop quan creem la taula i prou.

A mesura que es van fent insercions a la taula o es van modificant les dades, aquesta informació esdevé desactualitzada i pot fer que les decisions que prengui el planificador a l'hora d'avaluar les consultes siguin més dolentes.

Executant la sentència `ANALYZE`, fem que el PostgreSQL recalculi tota aquesta informació estadística, i aconseguim així que l'analitzador faci molt millor la seva feina. Amb una base de dades petita, pot ser que la diferencia no sigui gaire grossa, però quan es comencen a tenir taules d'uns pocs milions de registres, la diferència pot estar entre diversos segons i unes poques centèsimes d'execució per a la mateixa consulta.

4.3.4 Reindexació

La reindexació completa de la base de dades no és una tasca gaire habitual, però pot millorar de manera substancial la velocitat de les consultes complexes en taules amb molta activitat.

```
demo=# reindex database demo;
```

4.4 El planificador de consultes

La sentència `EXPLAIN` ens mostra, desglossada, la manera com el PostgreSQL planificaria una consulta i el cost aproximat que el planificador preveu que tindrà.

Això, per una banda, ens serveix per comparar i avaluar les diferents possibilitats que tenim a l'hora d'implementar una consulta per obtenir una informació determinada i així poder triar l'opció que resulti més eficient.

Per altra banda, si tenim una consulta molt pesada que volem optimitzar, com veurem més endavant, ens donarà informació molt útil per determinar quines són les parts de la consulta que li resulten més pesades i que més ens convé esforçar-nos a millorar.

Així doncs, la sentència `EXPLAIN` executada sobre una consulta ens mostra el següent:

- Aquesta instrucció mostra el pla d'execució que el planificador del PostgreSQL genera per a la consulta donada.
- El pla d'execució mostra la manera com seran escanejades les taules referenciades, ja sigui escaneig seqüencial pla, escaneig per índex, etc.
- En el cas que es referenciïn diverses taules, els algoritmes d'unió que seran utilitzats per agrupar els tuples requerits de cada taula d'entrada.
- L'opció `VERBOSE` emet la representació interna completa de l'arbre del pla, en comptes d'un resum (i també l'envia a l'arxiu *log* del *postmaster*). Normalment aquesta opció és únicament útil per a la correcció d'errors (depuració) del PostgreSQL.

Podem veure els valors de sortida següents:

- *Cost inici estimat*. Temps inicial que triga a retornar el primer tuple.
- *Cost total estimat*. Temps total que triguen a retornar tots els tuples: per exemple, si es limita el nombre de tuples per retornar amb una clàusula `LIMIT`, el planificador fa una interpolació apropiada entre els dos costos finals per estimar quin dels plans és realment el menys costós.

- *Nombre estimat de files escanejades*. Es compleix solament si l'execució de la consulta és completa.
- *Mida estimada de les files de sortida*. Es compleix solament si l'execució de la consulta és completa i indica la mida mitjana de les files expressada en bytes.

Exemple de pla de consulta simple

Per mostrar un pla de consulta per a una consulta simple sobre una taula amb una única columna de tipus *int4*:

```
postgres=#CREATE TABLE TEST1  ( id int4 primary key, text varchar
      );

NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "
     test1_pkey" for table "test1"
CREATE TABLE

postgres=#EXPLAIN SELECT * FROM TEST1;
                        QUERY PLAN
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 Seq Scan on test1  (cost=0.00..22.30 rows=1230 width=36)
(1 row)
```

Exemple de pla d'execució d'una consulta amb condició

Per a la mateixa taula amb un índex per aconseguir una condició *equijoin* a la consulta, `EXPLAIN` mostrarà un pla diferent:

```
postgres=#EXPLAIN SELECT * FROM TEST1 WHERE id=4;
                               QUERY PLAN
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 Index Scan using test1_pkey on test1  (cost=0.00..8.27 rows=1
     width=36)
   Index Cond: (id = 4)
(2 rows)
```

Exemple de pla de consulta amb una funció d'agregació

Per acabar, la mateixa taula amb un índex per aconseguir una condició *equijoin* a la consulta; Explain mostrarà el següent per a una consulta que utilitzi una funció d'agregació:

```
postgres=#EXPLAIN SELECT sum(id) FROM TEST1 WHERE id=4;
                                 QUERY PLAN
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Aggregate  (cost=8.27..8.28 rows=1 width=4)
->  Index Scan using test1_pkey on test1  (cost=0.00..8.27 rows=1
     width=4)
        Index Cond: (id = 4)
(3 rows)
```

4.5 Fitxers de registre

És una bona pràctica mantenir arxius de registre de l'activitat del servidor. Almenys, dels errors que origina. Durant el desenvolupament d'aplicacions pot ser molt útil disposar també d'un registre de les consultes efectuades, encara que en bases de dades de molta activitat, disminueix el rendiment del gestor i no és de gaire utilitat.

En qualsevol cas, és convenient disposar de mecanismes de rotació dels fitxers de registre; és a dir, que cada cert temps (12 hores, un dia, una setmana...), es faci una còpia d'aquests fitxers i se'n comencin de nous, cosa que ens permetrà mantenir un historial (tants com fitxers puguem emmagatzemar segons la mida que tinguin i les nostres limitacions d'espai en disc).

El PostgreSQL no proporciona directament utilitats per fer aquesta rotació, però en la majoria de sistemes Unix s'inclouen utilitats.

4.6 Gestió de l'espai d'emmagatzemament

A diferència d'altres SGBD, el PostgreSQL no reserva espai per als fitxers on emmagatzema les dades. L'agrupació d'aquests fitxers formen el que s'anomena *clúster* (no s'ha de confondre amb el clúster d'agrupació de dades de les taules), que és l'equivalent a les *tablespaces* de l'Oracle, o als *dbspace* de l'Informix.

Pel fet que cada taula és físicament un o més fitxers, no cal supervisar l'ocupació d'aquestes zones (no hi ha la possibilitat d'exhaurir l'espai intern del clúster), ja que els fitxers augmentaran de mida a mesura que s'hi insereixin dades.

Malgrat aquesta diferència de concepte amb altres SGBD, hi ha la possibilitat d'indicar la ubicació física dels fitxers, a l'efecte d'optimitzar la concurrència en l'accés als disc on s'emmagatzemen les dades.

4.7 Monitoratge

Una de les tasques més importants d'un administrador de bases de dades és monitorar els sistemes a càrrec seu per saber com estan funcionant i planejar modificacions i actualitzacions futures.

En el nostre cas, monitorar significa vigilar el funcionament d'un sistema, servei o activitat. Hi ha dos tipus de monitoratge:

- ***Ad hoc***. Monitoratge específic en cas de problemes o proves. S'utilitza generalment per investigar una situació puntual en què intentem trobar una explicació a un succés, canvi o problema.

- **Preventiu**. Detecta interrupcions de serveis, alerta sobre possibles problemes i crea gràfics amb tendències i dades històriques sobre els nostres sistemes. Aquest tipus de monitoratge està automatitzat i ens ajuda a descobrir canvis en els nostres sistemes que provoquen o poden provocar problemes en un futur proper.