El llenguatge SQL

Carme Martín Escofet

PID_00201445

Universitat Oberta de Catalunya

Índex

Introducció

SQL és el llenguatge estàndard ANSI/ISO de definició, manipulació i control de bases de dades (BD) relacionals. És un llenguatge declaratiu, només s'ha de dir què es vol fer. En canvi, en els llenguatges procedimentals cal especificar com s'ha de fer qualsevol cosa sobre la BD. SQL és un llenguatge molt semblant al llenguatge natural, concretament s'assembla a l'anglès, i és molt expressiu. Per aquestes raons, i com a llenguatge estàndard, SQL és un llenguatge amb què es pot accedir a tots els sistemes de gestió de bases de dades (SGBD) relacionals comercials.

Comencem amb una breu explicació de la manera com SQL ha arribat a ser el llenguatge estàndard de les BD relacionals:

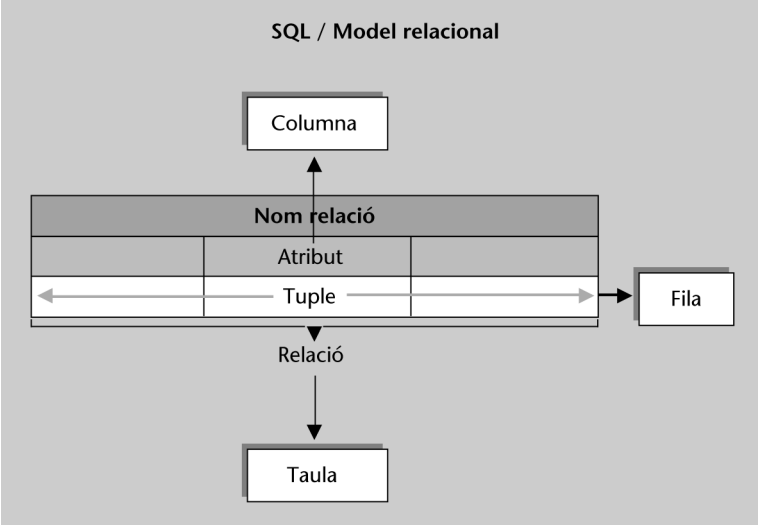
1. Al principi dels anys setanta els laboratoris d'investigació Santa Teresa d'IBM van començar a treballar en el projecte System R. L'objectiu d'aquest projecte era implementar un prototipus de SGBD relacional i, per tant, necessitaven també investigar en el camp dels llenguatges de BD relacionals. A mitjan anys setanta, el projecte d'IBM va donar un primer llenguatge anomenat *SEQUEL* (*Structured English Query Language*), que per raons legals més endavant es va denominar *SQL* (*Structured Query Language*). Al final de la dècada dels setanta i al principi de la dels vuitanta, un cop finalitzat el projecte System R, IBM i altres empreses van començar a utilitzar SQL en els seus SGBD relacionals, i així aquest llenguatge va adquirir una gran popularitat.

2. El 1982 l'ANSI (American National Standards Institute) va encarregar a un dels seus comitès (X3H2) la definició d'un llenguatge de BD relacionals. Aquest comitè, després d'avaluar diferents llenguatges, i davant l'acceptació comercial de SQL, va escollir com a llenguatge estàndard un llenguatge basat en SQL gairebé en la seva totalitat. SQL va esdevenir oficialment el llenguatge estàndard d'ANSI l'any 1986, i d'ISO (International Standards Organization) el 1987. També ha estat adoptat com a llenguatge estàndard per FIPS (Federal Information Processing Standard), Unix X/Open i SAA (Systems Application Architecture) d'IBM.

3. L'any 1989, l'estàndard fou objecte d'una revisió i una ampliació que van donar lloc al llenguatge que es coneix amb el nom de SQL1 o SQL:1989. L'any 1992, l'estàndard va tornar a ser revisat i ampliat considerablement per a cobrir mancances de la versió anterior. Aquesta nova versió de SQL es coneix amb el nom de SQL2 o SQL:1992. L'any 1999, va aparèixer l'esperada versió SLQ3 o SQL:1999, en què, entre altres coses, es definia formalment un dels components lògics de control més útils: els disparadors.

En aquest mòdul didàctic, encara que aparegui només la sigla SQL, sempre ens referirem a la **darrera versió**, ja que té com a subconjunt totes les anteriors i, per tant, tot el que era vàlid a l'anterior ho continuarà essent a la següent. Només especificarem l'any d'una versió de SQL quan, per exemple, vulguem emfatitzar que va ser concretament en aquesta en què es va fer una aportació determinada.

El model relacional té com a estructura d'emmagatzemament de les dades les relacions. La intensió o esquema d'una relació consisteix en el nom que hem donat a la relació i un conjunt d'atributs. L'extensió d'una relació és un conjunt de tuples. En treballar amb SQL aquesta nomenclatura canvia, com podem veure en la figura següent:



- Parlarem de **taules** en lloc de relacions.
- Parlarem de **columnes** en lloc d'atributs.
- Parlarem de **files** en lloc de tuples.

Però, malgrat que la nomenclatura utilitzada sigui diferent, els conceptes són els mateixos.

Amb SQL es pot definir i manipular una BD relacional. Tot seguit veurem, encara que només de manera introductòria, com es podria fer això:

1. Caldria crear una taula que contingués les dades dels productes de la nostra empresa:

```
CREATE TABLE productes ——— Nom de la taula
  (codi_producte INTEGER,
  nom_producte CHAR(20),
  tipus CHAR(20),          } Nom de les columnes i tipus
  descripcio CHAR(50),
  preu REAL,
  PRIMARY KEY (codi_producte));——— Clau primària
```

Notació

Encara que no és una convenció de SQL estàndard, la majoria de sistemes relacionals comercials requereixen que una sentència finalitzi amb ;

2. Inserir un producte en la taula creada anteriorment:

```
INSERT INTO productes ——— Nom de la taula
VALUES (1250, "LENA", "Taula", "Disseny Joan Pi. Any 1920.", 2500.0);
       └──────────────────────────────────────────────────────┘
                        Valors de la fila
```

3. Consultar quins productes de la nostra empresa són cadires:

```
            Columnes seleccionades
          ┌──────────┴──────────┐
SELECT codi_producte, nom_producte
FROM productes ——— Nom de la taula
WHERE tipus = "Cadira";
      └──────┬──────┘
       Files seleccionades
```

I moltes més coses que anirem veient punt per punt en els apartats següents.

Fixem-nos en l'estructura de tot el que hem fet fins ara amb SQL. Les operacions de SQL s'anomenen **sentències** i estan formades per diferents parts que denominem **clàusules**, tal com podem veure en l'exemple següent:

```
Sentència {
  SELECT codi_producte, nom_producte, tipus ——— Clàusula
  FROM productes ——— Clàusula
  WHERE preu > 1000.0; ——— Clàusula
}
```

Aquesta consulta mostra el codi, el nom i el tipus dels productes que costen més de 1.000 euros.

Els dos primers apartats d'aquest mòdul tracten d'un tipus de SQL anomenat **SQL interactiu**, que permet accedir directament a una BD relacional. En el primer apartat definirem les anomenades **sentències de definició**, en què crearem la BD, les taules que la compondran, els dominis i les assercions que vulguem. En el segon aprendrem a **manipular** la BD, introduint-hi valors a les files, modificant-los o esborrant-los, o fent-hi consultes.

Però moltes vegades voldrem accedir a la BD des d'una aplicació feta en un llenguatge de programació qualsevol, que ens ofereixi molta més potència fora de l'entorn de les BD. Per a utilitzar SQL des d'un llenguatge de programació, necessitarem sentències especials que ens permetin distingir entre les instruccions del llenguatge de programació i les sentències de SQL. La idea és que treballant bàsicament amb un llenguatge de programació amfitrió es pot acollir SQL com si fos un hoste. Per aquest motiu, aquest tipus de SQL es coneix amb el nom de SQL **hostatjat**. Per a treballar amb SQL hostatjat necessitem un precompilador que separi les sentències del llenguatge de programació de les del llenguatge de BD. Una alternativa a aquesta manera

de treballar són les rutines **SQL/CLI*** (SQL/*Call-Level Interface*), que resolen també el problema d'accedir a SQL des d'un llenguatge de programació i no necessiten precompilador.

*** Les rutines SQL/CLI es van afegir a l'estàndard SQL:1992 el 1995.**

Abans de començar a conèixer el llenguatge, convé afegir un últim comentari. Encara que SQL sigui el llenguatge estàndard per a BD relacionals i hagi estat àmpliament acceptat pels sistemes relacionals comercials, no és capaç de reflectir tota la teoria del model relacional establerta per E. F. Codd, com anirem veient a mesura que aprofundim en el llenguatge.

Els sistemes relacionals comercials i els investigadors de BD són una referència molt important per a mantenir l'estàndard actualitzat. En aquests moments la darrera versió de que es disposa és SQL: 2008. Però abans havien aparegut, respectivament, SQL: 2003 i SQL: 2006. Les principals novetats d'aquests darrers estàndards tenen a veure en com utilitzar SQL amb conjunció amb XML. I ben segur que continuaran apareixent versions noves a mesura que es vagin incorporant millores sobre el llenguatge SQL. Per això, en informàtica en general, i particularment en BD, cal estar sempre al dia, i és molt important tenir l'hàbit de llegir publicacions periòdiques que ens informin i ens mantinguin al corrent de les novetats.

Objectius

Un cop finalitzat l'estudi dels materials didàctics d'aquest mòdul, tindreu les eines indispensables per a assolir els objectius següents:

1. Conèixer els conceptes bàsics del llenguatge estàndard SQL.

2. Definir una BD relacional, incloent-hi dominis.

3. Saber introduir, esborrar i modificar dades.

4. Ser capaç de plantejar qualsevol tipus de consulta a la BD.

5. Saber utilitzar sentències de control.

6. Conèixer els principis bàsics de la utilització de l'SQL des d'un llenguatge de programació.

1. Sentències de definició de dades

Per a poder treballar amb BD relacionals, el primer que hem de fer és definir-les. Veurem les sentències de l'estàndard SQL per a crear i esborrar una BD relacional i per a crear, esborrar i modificar les diferents taules que la componen. En aquest apartat també veurem com es defineixen els dominis i les assercions (restriccions).

La senzillesa i l'homogeneïtat de SQL fan que:

1. Per a crear BD, taules, dominis i assercions s'utilitzi la **sentència `CREATE`**.
2. Per a modificar taules i dominis es faci servir la **sentència `ALTER`**.
3. Per a esborrar BD, taules, dominis i assercions s'empri la **sentència `DROP`**.

L'adequació d'aquestes sentències a cada cas ens donarà diferències que anirem perfilant en fer la descripció individual de cadascuna.

Per a il·lustrar l'aplicació de les sentències de SQL que anirem veient, utilitzarem **una BD d'exemple** molt senzilla d'una petita empresa amb seu a Barcelona, Girona, Lleida i Tarragona, que s'encarrega de desenvolupar projectes informàtics. La informació que ens interessarà emmagatzemar d'aquesta empresa, que anomenarem BDUOC, serà la següent:

1. Dels empleats que treballen a l'empresa, en voldrem saber el codi d'empleat, el nom i cognom, el sou, el nom i la ciutat del seu departament i el número de projecte al qual estan assignats.

2. Dels diferents departaments en què està estructurada l'empresa, ens n'interessa conèixer el nom, la ciutat on es troben i el telèfon. Caldrà tenir en compte que un departament amb el mateix nom pot estar a ciutats diferents i que en una mateixa ciutat hi poden haver departaments amb noms diferents.

3. Dels projectes informàtics que es desenvolupin, en voldrem saber el codi, el nom, el preu, la data d'inici, la data prevista de finalització, la data real de finalització i el codi de client per a qui es desenvolupa.

4. Dels clients per a qui treballa l'empresa, en voldrem saber el codi de client, el nom, el NIF, l'adreça, la ciutat i el telèfon.

1.1. Creació i esborrament d'una BD relacional

SQL estàndard no disposa de cap sentència de creació de BD. La idea és que una BD no és res més que un conjunt de taules i, per tant, les sentències que ens ofereix SQL es concentren en la creació, la modificació i l'esborrament d'aquestes taules.

La instrucció `CREATE DATABASE`

Molts dels sistemes relacionals comercials (com és el cas d'Informix, DB2, SQL Server i d'altres) han incorporat sentències de creació de BD amb la sintaxi següent:

```
CREATE DATABASE
<nom_BD>;
```

En canvi, disposem d'una sentència més potent que la de creació de BD: la **sentència de creació d'esquemes** anomenada **`CREATE SCHEMA`**.

Amb la creació d'esquemes podem agrupar un conjunt d'elements de la BD que són propietat d'un usuari. Aquesta sentència s'ofereix des de la primera versió de l'estàndard però no amb la sintaxi que veurem en aquest mòdul. La sintaxi de SQL:1992 d'aquesta sentència és la que teniu a continuació:

```
CREATE SCHEMA {<nom_esquema>|AUTHORIZATION <usuari>}
[<llista_elements_esquema>];
```

La **nomenclatura utilitzada en aquesta sentència** i d'ara en endavant és la següent:

- Les paraules en negreta són paraules reservades del llenguatge.
- La notació [...] vol dir que el que hi ha entre els claudàtors es podria posar o no.
- La notació {A|...|B} vol dir que hem d'escollir entre totes les opcions que hi ha entre les claus. Però n'hem de posar una obligatòriament.
- La notació <A> vol dir que A és un element pendent de definició.

La sentència de creació d'esquemes permet que un conjunt de taules (que denoten `llista_elements_esquema`) es pugui agrupar sota un mateix nom (`nom_esquema`) i que tinguin un propietari (`usuari`). Encara que els paràmetres de la sentència `CREATE SCHEMA` siguin opcionals, com a mínim s'ha de donar o bé el nom de l'esquema, o bé el nom de l'usuari propietari de la BD.

Si només n'especifiquem l'usuari, aquest serà l'escollit com a nom de l'esquema.

La creació d'esquemes pot fer molt més que agrupar taules i cal tenir present que, perquè `llista_elements_esquema` pot ser, a més de taules i dominis, altres components.

Per a esborrar una BD trobem el mateix problema que per a crear-la. SQL ens ofereix només la **sentència d'esborrament d'esquemes `DROP SCHEMA`**, que té la sintaxi següent:

```
DROP SCHEMA <nom_esquema> {RESTRICT|CASCADE};
```

La sentència `DROP DATABASE`

Molts dels sistemes relacionals comercials (com ara Informix, DB2, SQL Server i d'altres) han incorporat sentències d'esborrament de BD amb la sintaxi següent:

```
DROP DATABASE
<nom_BD>;
```

En què tenim el següent:

- L'opció d'esborrament d'esquemes **`RESTRICT`** fa que l'esquema només es pugui esborrar si no conté cap element.
- L'opció **`CASCADE`** esborra l'esquema encara que no estigui completament buit.

1.2. Creació de taules

Com ja hem vist, l'estructura d'emmagatzemament de les dades del model relacional són les taules. Per a **crear una taula** cal fer servir la **sentència `CREATE TABLE`**. Vegem-ne el format:

```
CREATE TABLE <nom_taula>
   ( <definició_columna>
   [, <definició_columna>...]
   [, <restriccions_taula>]
   );
```

En què **`definició_columna`** és:

```
<nom_columna> {<tipus_dades>|<domini>} [<def_defecte>] [<restriccions_columna>]
```

El procés que cal seguir per a crear una taula és el següent:

1) Decidir el nom de la taula (`nom_taula`).

2) Donar nom a cadascun dels atributs que formaran les columnes de la taula (`nom_columna`).

3) Assignar a cadascuna de les columnes un tipus de dades predefinit o bé un domini definit per l'usuari. També es pot donar definicions per defecte i restriccions de columna.

4) Un cop definides les columnes, només caldrà donar les restriccions de taula.

1.2.1. Tipus de dades

Per a cada columna hem d'escollir entre algun domini definit per l'usuari o algun dels tipus de dades predefinits que es descriuen a continuació:

Tipus de dades predefinits	
Tipus de dades	**Descripció**
`CHARACTER (longitud)`	Cadenes de caràcters de longitud fixa
`CHARACTER VARYING (longitud)`	Cadenes de caràcters de longitud variable
`BIT (longitud)`	Cadenes de bits de longitud fixa
`BIT VARYING (longitud)`	Cadenes de bits de longitud variable
`NUMERIC (precisió, escala)`	Nombres decimals amb tants dígits com indiqui la precisió i tants decimals com indiqui l'escala
`DECIMAL (precisió, escala)`	Nombres decimals amb tants dígits com indiqui la precisió i tants decimals com indiqui l'escala
`INTEGER`	Nombres enters
`SMALLINT`	Nombres enters petits
`REAL`	Nombres amb coma flotant amb precisió predefinida
`FLOAT (precisió)`	Nombres amb coma flotant amb la precisió especificada
`DOUBLE PRECISION`	Nombres amb coma flotant amb més precisió predefinida que la del tipus `REAL`
`DATE`	Dates. Estan compostes de: `YEAR any, MONTH mes, DAY dia`
`TIME`	Hores. Estan compostes de: `HOUR hora, MINUT minuts, SECOND segons`
`TIMESTAMP`	Dates i hores. Estan compostes de: `YEAR any, MONTH mes, DAY dia, HOUR hora, MINUT minuts, SECOND segons`

Els tipus de dades `NUMERIC` i `DECIMAL`

`NUMERIC` i `DECIMAL` es descriuen igual i es poden utilitzar tant l'un com l'altre per a definir nombres decimals.

El tractament del temps

SQL defineix la nomenclatura següent per a treballar amb el temps:

`YEAR`	(0001..9999)
`MONTH`	(01..12)
`DAY`	(01..31)
`HOUR`	(00..23)
`MINUT`	(00..59)
`SECOND`	(00..59.precisió)

De totes maneres, els SGBD comercials disposen de diferents formats, entre els quals podem escollir quan hem de treballar amb columnes temporals.

Exemples d'assignacions de columnes

Vegem alguns exemples d'assignacions de columnes als tipus de dades predefinits `DATE`, `TIME` i `TIMESTAMP`:

- La columna `data_naixement` podria ser de tipus `DATE` i podria tenir com a valor '1978-12-25'.
- La columna `inici_partit` podria ser de tipus `TIME` i podria tenir com a valor '17:15:00.000000'.
- La columna `entrada_feina` podria ser de tipus `TIMESTAMP` i podria tenir com a valor '1998-7-8 9:30:05'.

Dominis definits per l'usuari

La sentència `CREATE DOMAIN` de SQL:1992 hi ha pocs SGBD comercials que permetin usar-la. De tota manera, SQL:1999 ofereix els `USER DEFINED TYPES` com una nova possibilitat per a la creació de dominis definits per l'usuari més potent que la proposada per SQL:1992.

1.2.2. Creació, modificació i esborrament de dominis

A més dels dominis donats pels tipus de dades predefinits, SQL ens ofereix la possibilitat de treballar amb dominis definits per l'usuari.

Per a **crear un domini** cal fer servir la sentència **`CREATE DOMAIN`**:

```
CREATE DOMAIN <nom_domini> [AS] <tipus_dades>
            [<def_defecte>] [<restriccions_domini>];
```

en què **`restriccions_domini`** té el format següent:

```
CONSTRAINT [<nom_restricció>] CHECK (<condicions>)
```

Explicarem la construcció de condicions més endavant, en el subapartat 2.5, quan parlem de com es fan consultes en una BD. Veurem `def_defecte` en el subapartat 1.2.3 d'aquest mòdul. Encara que donar nom de la restricció sigui opcional, normalment se li dóna per a poder-la referenciar posteriorment.

Creació d'un domini a BDUOC

Si volguéssim definir un domini per a les ciutats on es troben els departaments de l'empresa BDUOC, faríem:

```
CREATE DOMAIN dom_ciutats AS CHAR(20)
  CONSTRAINT ciutats_valides
  CHECK (VALUE IN ("Barcelona", "Tarragona", "Lleida", "Girona"));
```

D'aquesta manera, quan definim la columna `ciutat` dins de la taula `departaments` no s'haurà de dir que és de tipus `CHAR(20)`, sinó de tipus `dom_ciutats`. Això ens hauria d'assegurar, segons el model relacional, que només farem operacions sobre la columna `ciutat` amb altres columnes que tinguin aquest mateix domini definit per l'usuari, però SQL no ens ofereix eines per a assegurar que les comparacions que fem siguin entre els mateixos dominis definits per l'usuari.

Per exemple, si tenim una columna amb els noms dels empleats definida sobre el tipus de dades `CHAR(20)`, SQL ens permet comparar-la amb la columna `ciutat`, encara que semànticament no tingui sentit. En canvi, segons el model relacional, aquesta comparació no s'hauria d'haver permès.

Per a **esborrar un domini definit per l'usuari** cal fer servir la **sentència `DROP DOMAIN`**, que té aquest format:

```
DROP DOMAIN <nom_domini> {RESTRICT|CASCADE};
```

En aquest cas, tenim que:

- L'opció d'esborrament de dominis **`RESTRICT`** fa que el domini només es pugui esborrar si no s'utilitza enlloc.
- L'opció **`CASCADE`** esborra el domini encara que estigui referenciat i posa el tipus de dades del domini allà on s'emprava.

Esborrar un domini de BDUOC

Si volguéssim esborrar el domini que hem creat abans per a les ciutats on es troben els departaments de l'empresa BDUOC, faríem:

```
DROP DOMAIN dom_ciutats RESTRICT;
```

En aquest cas caldria assegurar-se que cap columna no està definida sobre `dom_ciutats` abans d'esborrar el domini.

Per a **modificar un domini semàntic** s'utilitza la **sentència `ALTER DOMAIN`**. Vegem-ne el format:

```
ALTER DOMAIN <nom_domini> {<acció_modificar_domini>|
                        <acció_modificar_restricció_domini>};
```

En què tenim el següent:

- **`acció_modificar_domini`** pot ser:

```
SET {<def_defecte>|DROP DEFAULT}
```

- **`acció_modificar_restricció_domini`** pot ser:

```
ADD {<restricció_domini>|DROP CONSTRAINT <nom_restricció>}
```

Modificar un domini a BDUOC

Si volguéssim afegir una ciutat nova (Mataró) en el domini que hem creat abans per a les ciutats on es troben els departaments de l'empresa BDUOC, faríem:

```
ALTER DOMAIN dom_ciutats DROP CONSTRAINT ciutats_valides;
```

Amb això hem eliminat la restricció de domini antiga. I ara hem d'introduir la nova restricció:

```
ALTER DOMAIN dom_ciutats ADD CONSTRAINT ciutats_valides
  CHECK (VALUE IN ("Barcelona","Tarragona", "Lleida",
                  "Girona", "Mataro"));
```

1.2.3. Definicions per defecte

Ja hem vist la importància dels valors nuls i la seva inevitable aparició com a valors de les BD.

L'opció **`def_defecte`** ens permet especificar quina nomenclatura volem donar als nostres valors per omissió.

Exemple

Per a un empleat que encara no s'ha decidit quant guanyarà, es pot escollir que, de moment, tingui un sou de 0 euros (`DEFAULT 0.0`), o bé que tingui un sou amb un valor nul (`DEFAULT NULL`).

Cal tenir en compte, però, que si escollim l'opció **DEFAULT NULL**, la columna per a la qual donarem la definició per defecte de valor nul hauria d'admetre valors nuls.

L'opció **DEFAULT** té el format següent:

```
DEFAULT {<literal>|<funció>|NULL}
```

La possibilitat més utilitzada i l'opció per defecte si no especifiquem res és la paraula reservada **NULL**. Però també podem definir el nostre propi literal, o bé recórrer a una de les funcions que apareixen en la taula següent:

Funció	Descripció
{USER|CURRENT_USER}	Identificador de l'usuari actual
SESSION_USER	Identificador de l'usuari d'aquesta sessió
SYSTEM_USER	Identificador de l'usuari del sistema operatiu
CURRENT_DATE	Data actual
CURRENT_TIME	Hora actual
CURRENT_TIMESTAMP	Data i hora actuals

1.2.4. Restriccions de columna

En cadascuna de les columnes de la taula, un cop els hem donat un nom i n'hem definit el domini, podem imposar certes restriccions que sempre s'hauran de complir. Les restriccions que es poden donar són les que apareixen en la taula que teniu a continuació:

Restriccions de columna	
Restricció	**Descripció**
NOT NULL	La columna no pot tenir valors nuls
UNIQUE	La columna no pot tenir valors repetits. És una clau alternativa
PRIMARY KEY	La columna no pot tenir valors repetits ni nuls. És la clau primària
REFERENCES <nom_taula> [(<nom_columna>)]	La columna és la clau forana de la columna de la taula especificada
CONSTRAINT [<nom_restricció>] CHECK (<condicions>)	La columna ha de complir les condicions especificades

1.2.5. Restriccions de taula

Un cop hem donat un nom, hem definit un domini i hem imposat certes restriccions per a cadascuna de les columnes, podem aplicar restriccions sobre

tota la taula, les quals sempre s'hauran de complir. Les restriccions que es poden donar són les següents:

Restriccions de taula	
Restricció	**Descripció**
`UNIQUE (<nom_columna> [, <nom_columna>...])`	El conjunt de les columnes especificades no pot tenir valors repetits. És una clau alternativa
`PRIMARY KEY (<nom_columna> [, <nom_columna>...])`	El conjunt de les columnes especificades no pot tenir valors nuls ni repetits. És una clau primària
`FOREIGN KEY (<nom_columna> [, <nom_columna>...]) REFERENCES <nom_taula> [(<nom_columna2> [, <nom_columna2>...])]`	El conjunt de les columnes especificades és una clau forana que referencia la clau primària formada pel conjunt de les columnes2 de la taula donada. Si les columnes i les columnes2 s'anomenen exactament igual, llavors no caldria posar columnes2
`CONSTRAINT [<nom_restricció>] CHECK (<condicions>)`	La taula ha de complir les condicions especificades

1.2.6. Modificació i esborrament de claus primàries amb claus foranes que hi fan referència

Hem vist tres polítiques aplicables als casos d'esborrament i modificació de files que tenen una clau primària referenciada per claus foranes. Aquestes polítiques eren la restricció, l'actualització en cascada i l'anul·lació.

SQL ens ofereix la possibilitat d'especificar, en definir una clau forana, quina política volem seguir. Vegem-ne el format:

```
CREATE TABLE <nom_taula>
  ( <definició_columna>
    [, <definició_columna>...]
    [, <restriccions_taula>]
  );
```

En què una de les restriccions de taula era **la definició de claus foranes**, que té el format següent:

```
FOREIGN KEY <clau_forana> REFERENCES <nom_taula> [(<clau_primària>)]
[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
```

En què `NO ACTION` correspon a la política de restricció; `CASCADE`, a l'actualització en cascada, i `SET NULL` seria l'anul·lació. `SET DEFAULT` es podria considerar una variant de `SET NULL`, en què en lloc de valors nuls es pot posar el valor especificat per defecte.

1.2.7. Assercions

Una asserció és una restricció general que fa referència a una o més columnes d'una o més taules. Per a **definir una asserció** s'usa la sentència **CREATE ASSERTION** i té el format següent:

```
CREATE ASSERTION <nom_asserció> CHECK (<condicions>);
```

Crear una asserció a BDUOC

Creem una asserció sobre la BD BDUOC que ens asseguri que no hi ha cap empleat amb un `sou` superior a `80000` assignat al projecte `SALSA`:

```
CREATE ASSERTION restriccio1 CHECK (NOT EXISTS (SELECT *
                        FROM projectes p, empleats e
                        WHERE p.codi_proj = e.num_proj
                             and e.sou > 80000.0
                             and p.nom_proj = "SALSA"));
```

Per a **esborrar una asserció** cal fer servir la sentència **DROP ASSERTION**, que presenta aquest format:

```
DROP ASSERTION <nom_asserció>;
```

Esborrar una asserció a BDUOC

Per a esborrar l'asserció `restriccio1`, faríem servir la sentència `DROP ASSERTION` de la manera següent:

```
DROP ASSERTION restriccio1;
```

Assercions

Tot i que SQL ofereixi la sentència `CREATE ASSERTION`, no existeixen productes comercials que permetin usar-la. L'alternativa a les assercions són els disparadors (*triggers*). Els disparadors van ser introduïts amb SQL:1999 i a diferència de les assercions la gran majoria de productes comercials els ofereixen.

1.3. Modificació i esborrament de taules

Per a **modificar una taula** cal fer servir la sentència **ALTER TABLE**. Vegem-ne el format:

```
ALTER TABLE <nom_taula> {<acció_modificar_columna>|
                     <acció_modificar_restricció_taula>};
```

En aquest cas, tenim que:

- **acció_modificar_columna** pot ser:

```
{ADD [COLUMN] <nom_columna> <def_columna> |
  ALTER [COLUMN] <nom_columna> {SET <def_defecte>|DROP DEFAULT}|
  DROP [COLUMN] <nom_columna> {RESTRICT|CASCADE}}
```

- **acció_modificar_restricció_taula** pot ser:

```
{ADD <def_restricció>|
  DROP CONSTRAINT <nom_restricció> {RESTRICT|CASCADE}}
```

Si volem modificar una taula és que volem fer una de les operacions següents:

1. Afegir-hi una columna (`ADD <nom_columna>`).
2. Modificar les definicions per defecte de la columna (`ALTER <nom_columna>`).
3. Esborrar la columna (`DROP <nom_columna>`).
4. Afegir alguna nova restricció de taula (`ADD <def_restricció>`).
5. Esborrar alguna restricció de taula (`DROP CONSTRAINT <nom_restricció>`).

Per a **esborrar una taula** cal fer servir la sentència **DROP TABLE**:

```
DROP TABLE <nom_taula> {RESTRICT|CASCADE};
```

En aquest cas tenim que:

- Si utilitzem l'opció **RESTRICT** la taula no s'esborrarà si està referenciada, per exemple, des d'alguna altra taula o vista.

- Si usem l'opció **CASCADE**, tot el que referencií a la taula s'esborrarà amb ella.

1.4. Creació i esborrament de vistes

L'arquitectura ANSI/SPARC distingeix tres nivells que es descriuen en l'esquema conceptual, l'esquema intern i els esquemes externs. Fins ara, creant les taules de la BD, anàvem descrivint l'esquema conceptual. Per a descriure els diferents esquemes externs usem el concepte de vista de SQL.

Per a **crear una vista** cal fer servir la **sentència CREATE VIEW**. Vegem-ne el format:

```
CREATE VIEW <nom_vista> [(llista_columnes)] AS (consulta)
[WITH CHECK OPTION];
```

El primer que hem de fer per a crear una vista és decidir quin nom li volem posar (`nom_vista`). Si volem canviar el nom de les columnes, o bé posar nom a alguna que en principi no en tenia, ho podem fer a `llista_columnes`. I ja només ens restarà definir la consulta que formarà la nostra vista.

Les vistes no existeixen realment com un conjunt de valors emmagatzemats a la BD, sinó que són taules fictícies, anomenades *derivades* (no materialitzades). Es construeixen a partir de taules reals (materialitzades) emmagatzemades a la BD. Aquestes darreres es coneixen amb el nom de *taules bàsiques* o *taules de base*. La no-existència real de les vistes fa que puguin ser actualitzables o no.

Creació d'una vista a BDUOC

Creem una vista sobre la BD BDUOC que ens doni per a cada client el nombre de projectes que té encarregats el client en qüestió.

```
CREATE VIEW projectes_per_client (codi_cli, nombre_projectes) AS
(SELECT c.codi_cli, COUNT(*)
 FROM projectes p, clients
 c WHERE p.codi_client = c.codi_cli
 GROUP BY c.codi_cli)
```

Si tinguéssim les extensions següents:

- Taula `clients`:

clients					
codi_cli	**nom_cli**	**nif**	**adreça**	**ciutat**	**telefon**
10	ECIGSA	38.567.893-C	Aragó 11	Barcelona	NULL
20	CME	38.123.898-E	València 22	Girona	972.23.57.21
30	ACME	36.432.127-A	Mallorca 33	Lleida	973.23.45.67

- Taula `projectes`:

projectes						
codi_proj	**nom_proj**	**preu**	**data_inici**	**data_prev_fi**	**data_fi**	**codi_client**
1	GESCOM	1,0E+6	1-1-98	1-1-99	NULL	10
2	PESCI	2,0E+6	1-10-96	31-3-98	1-5-98	10
3	SALSA	1,0E+6	10-2-98	1-2-99	NULL	20
4	TINELL	4,0E+6	1-1-97	1-12-99	NULL	30

I miréssim l'extensió de la vista `projectes_per_clients`, veuríem el que trobem en el marge.

projectes_per_clients	
codi_cli	**nombre_projectes**
10	2
20	1
30	1

A les vistes, a més de fer-hi consultes, podem inserir, modificar i esborrar files.

Actualització de vistes a BDUOC

Si algú inserís a la vista `projectes_per_client` els valors per a un nou client 60 amb 3 projectes encarregats, trobaríem que aquests 3 projectes haurien de figurar re-

alment a la taula `projectes` i, per tant, l'SGBD els hauria d'inserir amb la informació que tenim, que és gairebé inexistent. Vegem gràficament com quedarien les taules després d'aquesta hipotètica actualització que no arribarem a fer mai, ja que contradiria la teoria del model relacional:

- Taula `clients`:

clients					
codi_cli	**nom_cli**	**nif**	**adreça**	**ciutat**	**telefon**
10	ECIGSA	38.567.893-C	Aragó 11	Barcelona	NULL
20	CME	38.123.898-E	València 22	Girona	972.23.57.21
30	ACME	36.432.127-A	Mallorca 33	Lleida	973.23.45.67
60	NULL	NULL	NULL	NULL	NULL

- Taula `projectes`:

projectes						
codi_proj	**nom_proj**	**preu**	**data_inici**	**data_prev_fi**	**data_fi**	**codi_client**
1	GESCOM	1,0E+6	1-1-98	1-1-99	NULL	10
2	PESCI	2,0E+6	1-10-96	31-3-98	1-5-98	10
3	SALSA	1,0E+6	10-2-98	1-2-99	NULL	20
NULL	NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

L'SGBD no pot actualitzar la taula bàsica `clients`, sabent només la clau primària i encara menys la taula bàsica `projectes`, sense la clau primària i, per tant, aquesta vista no seria actualitzable.

Si definim, en canvi, una vista per a saber els clients que tenim a Barcelona o a Girona, faríem:

```
CREATE VIEW clients_Barcelona_Girona AS
(SELECT *
 FROM clients
 WHERE ciutat IN ('Barcelona', 'Girona'))
WITH CHECK OPTION;
```

Si volem assegurar-nos que es compleixi la condició de la clàusula `WHERE`, hem de posar l'opció `WITH CHECK OPTION`. Si no ho féssim, podria passar que algú inserís a la vista `clients_Barcelona_Girona` un client nou amb el codi 70, de nom JMB, amb el NIF 36.788.224-C, l'adreça a NULL, la ciutat Lleida i el telèfon NULL.

Si consultéssim l'extensió de la vista `clients_Barcelona_Girona`, veuríem:

clients_Barcelona_Girona					
codi_cli	**nom_cli**	**nif**	**adreça**	**ciutat**	**telefon**
10	ECIGSA	38.567.893-C	Aragó 11	Barcelona	NULL
20	CME	38.123.898-E	València 22	Girona	972.23.57.21

Aquesta vista sí que podria ser actualitzable. Podríem inserir un nou client amb codi 50, de nom CEA, amb el NIF 38.226.777-D, amb l'adreça París 44, la ciutat Barcelona

i el telèfon 93.422.60.77. Després d'aquesta actualització, a la taula bàsica `clients` trobaríem, efectivament:

clients					
codi_cli	**nom_cli**	**nif**	**adreça**	**ciutat**	**telefon**
10	ECIGSA	38.567.893-C	Aragó 11	Barcelona	NULL
20	CME	38.123.898-E	València 22	Girona	972.23.57.21
30	ACME	36.432.127-A	Mallorca 33	Lleida	973.23.45.67
50	CEA	38.226.777-D	París 44	Barcelona	93.422.60.77

Per a **esborrar una vista** cal fer servir la **sentència `DROP VIEW`**, que presenta el format:

```
DROP VIEW <nom_vista> {RESTRICT|CASCADE};
```

Si emprem l'opció **`RESTRICT`** la vista no s'esborrarà si està referenciada, per exemple, per alguna altra vista. En canvi, si posem l'opció **`CASCADE`**, tot el que referenciï a la vista s'esborrarà amb aquesta.

Esborrar una vista a BDUOC

Per a esborrar la vista `clients_Barcelona_Girona`, faríem el següent:

```
DROP VIEW clients_Barcelona_Girona RESTRICT;
```

1.5. Definició de la BD relacional BDUOC

Vegem com es crearien les taules de la BD BDUOC:

Ordre de creació

Abans de crear una taula amb una o més claus foranes, s'han d'haver creat les taules que tenen com a clau primària les referenciades per les foranes.

```
CREATE TABLE clients (codi_cli INTEGER,
  nom_cli CHAR(30) NOT NULL,
  nif CHAR(12),
  adreça CHAR(30),*
  ciutat CHAR(20),
  telefon CHAR(12) DEFAULT NULL,
  PRIMARY KEY(codi_cli), UNIQUE(nif)
);

CREATE TABLE departaments
  (nom_dpt CHAR(20) PRIMARY KEY*,
  ciutat_dpt CHAR(20),
  telefon CHAR(12) DEFAULT NULL,
  PRIMARY KEY (nom_dpt, ciutat_dpt)
);
```

*** La majoria de productes comercials no accepten ni la "ç" ni els accents. Aquí els deixem per a facilitar la legibilitat.**

*** Hem d'escollir restricció de taula perquè la clau primària està composta per més d'un atribut.**

```
CREATE TABLE projectes
  (codi_proj INTEGER,
  nom_proj CHAR(20),
  preu REAL,
  data_inici DATE,
  data_prev_fi DATE,
  data_fi DATE DEFAULT NULL,
  codi_client INTEGER,
  PRIMARY KEY (codi_proj),
  FOREIGN KEY codi_client REFERENCES clients(codi_cli),
  CHECK (data_inici < data_prev_fi),
  CHECK (data_inici < data_fi)
);

CREATE TABLE empleats
  (codi_empl INTEGER,
  nom_empl CHAR(20),
  cognom_empl CHAR(20),
  sou REAL CHECK(sou > 7000.0),
  nom_dpt CHAR(20),
  ciutat_dpt CHAR(20),
  num_proj INTEGER,
  PRIMARY KEY (codi_empl),
  FOREIGN KEY (nom_dpt, ciutat_dpt)
    REFERENCES departaments(nom_dpt, ciutat_dpt),
  FOREIGN KEY (num_proj) REFERENCES projectes(codi_proj)
);
```

En crear una taula veiem que moltes restriccions es poden imposar de dues maneres: com a restriccions de columna o com a restriccions de taula. Per exemple, quan volem dir quina és la clau primària d'una taula tenim ambdues possibilitats. Això és degut a la flexibilitat de SQL:

- En el cas que la restricció faci referència a un sol atribut, podem escollir la possibilitat que ens agradi més.

- En el cas de la taula departaments, hem d'escollir forçosament l'opció de restriccions de taula, perquè la clau primària està composta per més d'un atribut.

En general, per homogeneïtat, ho posarem tot com a restriccions de taula, excepte `NOT NULL` i `CHECK` quan faci referència a una sola columna.

2. Sentències de manipulació de dades

Un cop creada la BD amb les seves taules, cal poder-hi inserir, modificar i esborrar els valors de les files de les taules. Per a poder fer això, SQL ens ofereix les sentències següents: **INSERT**, per a inserir; **UPDATE**, per a modificar, i **DELETE**, per a esborrar. Un cop hem inserit valors a les taules, hem de poder consultar-los. La sentència per a fer consultes a una BD amb SQL és **SELECT FROM**. Vegem tot seguit aquestes sentències.

2.1. Inserció de files en una taula

Abans de poder consultar les dades d'una BD, cal **introduir les dades** amb la sentència **INSERT INTO VALUES**, que té el format següent:

```
INSERT INTO <nom_taula> [(<columnes>)]
  {VALUES ({<v1>|DEFAULT|NULL}, ..., {<vn>|DEFAULT|NULL}) |<consulta>};
```

Inserció de files múltiples

Per a inserir més d'una fila amb una sola sentència hem d'obtenir els valors com a resultat d'una consulta feta a una o més taules.

Els valors `v1, v2, ..., vn` s'han de correspondre exactament amb les columnes que hem dit que tindríem amb el `CREATE TABLE` i han d'estar en el mateix ordre, tret que les tornem a posar a continuació del nom de la taula. En aquest darrer cas els valors s'han de disposar de manera coherent amb el nou ordre que hem imposat. Podria ser que volguéssim que alguns valors a inserir fossin valors per omissió, definits prèviament amb l'opció `DEFAULT`. Llavors posaríem el mot reservat `DEFAULT`. Si es tracta d'introduir valors nuls també podem usar el mot reservat `NULL`.

Inserció d'una fila a BDUOC

La manera d'inserir un client en la taula clients de la BD BDUOC és:

```
INSERT INTO clients
VALUES (10, "ECIGSA", "37.248.573-C", "ARAGO 242", "Barcelona",
       DEFAULT);
```

o bé:

```
INSERT INTO clients(nif, nom_cli, codi_cli, telefon, adreca, ciutat)
VALUES("37.248.573-C", "ECIGSA", 10, DEFAULT,
       "ARAGO 242", "Barcelona");
```

2.2. Esborrament de files d'una taula

Per a **esborrar valors d'algunes files d'una taula** podem fer servir la **sentència `DELETE FROM WHERE`**. El seu format és el següent:

```
DELETE FROM <nom_taula>
[WHERE <condicions>];
```

En canvi, si el que volguéssim aconseguir fos **esborrar totes les files d'una taula**, aleshores només hauríem de posar la sentència **`DELETE FROM`**, sense `WHERE`.

Esborrar totes les files d'una taula a BDUOC

Podem deixar la taula `projectes` sense cap fila:

```
DELETE FROM projectes;
```

En la nostra BD, esborrar els projectes del client amb `codiClient = 2` es faria de la manera que mostrem a continuació:

```
DELETE FROM projectes WHERE codi_client = 2;
```

Esborrament de files múltiples

Fixem-nos que el client amb `codi_client = 2` podria tenir més d'un projecte contractat i, per tant, s'esborraria més d'una fila amb una sola sentència.

2.3. Modificació de files d'una taula

Si volguéssim **modificar els valors d'algunes files d'una taula** hauríem de fer servir la sentència **`UPDATE SET WHERE`**. A continuació en presentem el format:

```
UPDATE <nom_taula>
SET <nom_columna> = {<expressió>|DEFAULT|NULL}
    [, <nom_columna> = {<expressió>|DEFAULT|NULL} ...]
WHERE <condicions>;
```

Modificació dels valors d'algunes files a BDUOC

Suposem que els empleats del projecte amb `num_proj = 2` passen a guanyar un sou més alt. La modificació d'aquesta situació seria:

```
UPDATE empleats
SET sou = sou + 1000.0
WHERE num_proj = 2;
```

Modificació de files múltiples

Fixem-nos que el projecte amb `num_proj = 2` podria tenir més d'un empleat assignat i, per tant, es modificaria la columna sou, de més d'una fila amb una sola sentència.

2.4. Introducció de files en la BD relacional BDUOC

Abans de començar a fer consultes a la BD BDUOC haurem introduït unes quantes files en les seves taules amb la sentència `INSERT INTO`. D'aquesta

manera podrem veure reflectit el resultat de les consultes que anirem fent, a partir d'aquest moment. A continuació presentem l'extensió de les diferents taules:

- Taula `departaments`:

departaments		
nom_dpt	**ciutat_dpt**	**telefon**
DIR	Barcelona	93.422.60.70
DIR	Girona	972.23.89.70
DISS	Lleida	973.23.50.40
DISS	Barcelona	93.224.85.23
PROG	Tarragona	977.33.38.52
PROG	Girona	972.23.50.91

- Taula `clients`:

clients					
codi_cli	**nom_cli**	**nif**	**adreça**	**ciutat**	**telefon**
10	ECIGSA	38.567.893-C	Arago 11	Barcelona	NULL
20	CME	38.123.898-E	Valencia 22	Girona	972.23.57.21
30	ACME	36.432.127-A	Mallorca 33	Lleida	973.23.45.67
40	JGM	38.782.345-B	Rossello 44	Tarragona	977.33.71.43

- Taula `empleats`:

empleats						
codi_empl	**nom_empl**	**cognom_empl**	**sou**	**nom_dpt**	**ciutat_dpt**	**num_proj**
1	Maria	Puig	100000.0	DIR	Girona	1
2	Pere	Mas	90000.0	DIR	Barcelona	4
3	Anna	Ros	70000.0	DISS	Lleida	3
4	Jordi	Roca	70000.0	DISS	Barcelona	4
5	Clara	Blanc	40000.0	PROG	Tarragona	1
6	Laura	Tort	30000.0	PROG	Tarragona	3
7	Roger	Salt	40000.0	NULL	NULL	4
8	Sergi	Grau	30000.0	PROG	Tarragona	NULL

- Taula `projectes`:

empleats						
codi_proj	**nom_proj**	**preu**	**data_inici**	**data_prev_fi**	**data_fi**	**codi_client**
1	GESCOM	1000000.0	1-1-98	1-1-99	NULL	10
2	PESCI	2000000.0	1-10-96	31-3-98	1-5-98	10
3	SALSA	1000000.0	10-2-98	1-2-99	NULL	20
4	TINELL	4000000.0	1-1-97	1-12-99	NULL	30

2.5. Consultes a una BD relacional

Per a fer **consultes** sobre una taula amb SQL cal utilitzar la **sentència `SELECT FROM`**, que té el format següent:

```
SELECT <nom_columna_seleccionar> [[AS] <col_reanomenada>]
[,<nom_columna_seleccionar> [[AS] <col_reanomenada>]...]
FROM <taula_consultar> [[AS] <taula_reanomenada>];
```

L'opció **`AS`** ens permet reanomenar les columnes que volem seleccionar o les taules que volem consultar, en aquest cas només una. Dit d'una altra manera, ens permet la definició d'àlies. Fixem-nos que la paraula clau `AS` és opcional, i és força habitual, en definir àlies en la clàusula `FROM`, posar-hi només un espai en blanc en lloc de tota la paraula.

Consultes a BDUOC

A continuació presentem un exemple de consulta amb la BD BDUOC per a conèixer totes les dades que hi ha en la taula `clients`:

```
SELECT * FROM clients;
```

L'asterisc (*) després de `SELECT` indica que volem veure tots els atributs que hi ha a la taula.

La resposta a aquesta consulta seria:

codi_cli	nom_cli	nif	adreça	ciutat	telefon
10	ECIGSA	38.567.893-C	Arago 11	Barcelona	NULL
20	CME	38.123.898-E	Valencia 22	Girona	972.23.57.21
30	ACME	36.432.127-A	Mallorca 33	Lleida	973.23.45.67
40	JGM	38.782.345-B	Rossello 44	Tarragona	977.33.71.43

Si haguéssim volgut veure només el codi, el nom, l'adreça i la ciutat, hauríem escrit el següent:

```
SELECT codi_cli, nom_cli, adreca, ciutat
FROM clients;
```

El resultat de la consulta anterior seria el següent:

codi_cli	nom_cli	adreça	ciutat
10	ECIGSA	Arago 11	Barcelona
20	CME	Valencia 22	Girona
30	ACME	Mallorca 33	Lleida
40	JGM	Rossello 44	Tarragona

Amb la comanda `SELECT FROM` podem seleccionar columnes d'una taula, però per a seleccionar files d'una taula hi cal afegir la clàusula `WHERE`. El format és el següent:

```
SELECT <nom_columnes_seleccionar>
FROM <taula_consultar>
WHERE <condicions>;
```

La clàusula `WHERE` ens permet obtenir les files que compleixen la condició especificada a la consulta.

Consultes a BDUOC seleccionant files

Vegem un exemple en què demanem "els codis dels empleats que treballen en el projecte número 4":

```
SELECT codi_empl
FROM empleats
WHERE num_proj = 4;
```

La resposta a aquesta consulta seria la següent:

codi_empl
2
4
7

Per a definir les condicions de la clàusula `WHERE`, podem fer servir algun dels operadors de què disposa SQL, que són els següents:

Operadors de comparació	
=	Igual
<	Més petit
>	Més gran
<=	Més petit o igual
>=	Més gran o igual
<>	Diferent

Operadors lògics	
NOT	Per a la negació de condicions
AND	Per a la conjunció de condicions
OR	Per a la disjunció de condicions

Si volem que en una consulta ens apareguin les files resultants sense repeticions, cal posar la paraula clau **`DISTINCT`** immediatament després del `SELECT`.

També podríem explicitar que ho volem tot, fins i tot amb repeticions, posant ALL (opció per defecte) en lloc de DISTINCT. El format de DISTINCT és el següent:

```
SELECT DISTINCT <nom_columnes_seleccionar>
FROM <taula_consultar>
[WHERE <condicions>];
```

Consulta a BDUOC seleccionant files sense repeticions

Si volguéssim veure quins sous es paguen en la nostra empresa, podríem fer:

```
SELECT DISTINCT sou
FROM empleats;
```

La resposta a aquesta consulta, sense repeticions, seria la següent:

sou
30000.0
40000.0
70000.0
90000.0
100000.0

2.5.1. Funcions d'agregació

SQL ofereix les **funcions d'agregació** següents per a efectuar diverses operacions amb les dades d'una BD:

Funcions d'agregació	
Funció	**Descripció**
COUNT	Ens dóna el nombre total de files seleccionades
SUM	Suma els valors d'una columna
MIN	Ens dóna el valor mínim d'una columna
MAX	Ens dóna el valor màxim d'una columna
AVG	Calcula el valor mitjà d'una columna

En general, les funcions d'agregació s'apliquen a una columna, excepte la funció d'agregació COUNT, que normalment s'aplica a totes les columnes de la taula o taules seleccionades. Per tant, COUNT(*) comptarà totes les files de la taula o taules que compleixin les condicions. Si s'utilitzés COUNT(DISTINCT <nom_columna>), només comptaria els valors que no fossin nuls ni repetits, i si s'emprés COUNT(<nom_columna>), només comptaria els valors que no fossin nuls.

Exemple d'utilització de la funció `COUNT(*)`

Vegem un exemple d'ús de la funció `COUNT`, que apareix en la clàusula `SELECT`, per a fer la consulta "Quants departaments estan ubicats a la ciutat de Lleida?":

```
SELECT COUNT(*) AS nombre_dpt FROM departaments
WHERE ciutat_dpt = "Lleida";
```

La resposta a aquesta consulta és la següent:

Veurem exemples de les altres funcions d'agregació en els apartats següents.

nombre_dept
1

2.5.2. Subconsultes

Una subconsulta és una consulta inclosa dins una clàusula `WHERE` o `HAVING` d'una altra consulta. De vegades, per a expressar certes condicions no hi ha altre remei que obtenir el valor que busquem com a resultat d'una consulta.

Veurem la clàusula `HAVING` en el subapartat 2.5.5 d'aquest mòdul didàctic.

Subconsulta a BDUOC

Si volguéssim saber els codis i els noms dels projectes de preu més alt, en primer lloc hauríem de trobar els projectes que tenen el preu més alt. Ho faríem de la manera següent:

```
SELECT codi_proj, nom_proj FROM projectes
WHERE preu = (SELECT MAX(preu)
              FROM projectes);
```

El resultat de la consulta anterior seria el següent:

codi_proj	nom_proj
4	TINELL

Els projectes de preu més baix

Si en lloc dels codis i els noms de projectes de preu més alt haguéssim volgut saber els de preu més baix, hauríem aplicat la funció d'agregació `MIN`.

2.5.3. Altres predicats

1. Predicat `BETWEEN`

Per a expressar una condició que vol trobar un valor entre uns límits concrets podem fer servir `BETWEEN`:

```
SELECT <nom_columnes_seleccionar>
FROM <taula_consultar>
WHERE <columna> BETWEEN <límit1> AND <límit2>;
```

Exemple d'ús del predicat `BETWEEN`

Es demana "Els codis dels empleats que guanyen entre 20.000 i 50.000 euros anuals".

```
SELECT codi_empl
FROM empleats
WHERE sou BETWEEN 20000.0 and 50000.0;
```

La resposta a aquesta consulta seria la següent:

codi_empl
5
6
7
8

2. Predicat `IN`

Per a veure si un valor coincideix amb els elements d'una llista utilitzarem `IN`, i per a veure si no hi coincideix, `NOT IN`:

```
SELECT <nom_columnes_seleccionar>
FROM <taula_consultar>
WHERE <columna> [NOT] IN (<valor1>, ..., <valorN>);
```

Exemple d'ús del predicat `IN`

"Volem saber el nom de tots els departaments que es troben a les ciutats de Lleida o Tarragona."

```
SELECT nom_dpt, ciutat_dpt
FROM departaments
WHERE ciutat_dpt IN ("Lleida", "Tarragona");
```

La resposta seria la següent:

nom_dpt	ciutat_dpt
DISS	PROG
Lleida	Tarragona

3. Predicat `LIKE`

Per a veure si una columna de tipus caràcter compleix alguna característica determinada podem fer servir `LIKE`:

```
SELECT <nom_columnes_seleccionar> FROM <taula_consultar>
WHERE <columna> LIKE <característica>;
```

Els **patrons de SQL per a expressar característiques** són els següents:

Altres patrons

Encara que _ i % són els caràcters escollits per l'estàndard, cada SGBD comercial ofereix diverses variants.

a) Es posa un caràcter _ per a cada caràcter individual que es vulgui considerar.

b) Es posa un caràcter % per a expressar una seqüència de caràcters, que pot ser cap.

Exemple d'ús del predicat `LIKE`

Atributs afegits

Encara que a la consulta es demani només el nom dels empleats, hi afegim el codi per a poder diferenciar dos empleats amb el mateix nom.

Es busquen els noms dels empleats que comencen amb la lletra J i els projectes que comencen per S i tenen cinc lletres:

a) Noms d'empleats que comencen amb la lletra J:

```
SELECT codi_empl, nom_empl
FROM empleats
WHERE nom_empl LIKE "J%";
```

La resposta a aquesta consulta seria la següent:

codi_empl	nom_empl
4	Jordi

b) Projectes que comencen per S i tenen cinc lletres:

```
SELECT codi_proj
FROM projectes
WHERE nom_proj LIKE "S_ _ _ _";
```

I la resposta a aquesta altra consulta seria la següent:

codi_proj
3

4. Predicat `IS NULL`

Per a veure si un valor és nul utilitzarem `IS NULL`, i per a veure si no ho és, `IS NOT NULL`. El format és:

```
SELECT <nom_columnes_seleccionar>
FROM <taula_consultar>
WHERE <columna> IS [NOT] NULL;
```

Exemple d'ús del predicat `IS NULL`

"Volem saber el codi i el nom de tots els empleats que no estan assignats a cap projecte."

```
SELECT codi_empl, nom_empl FROM empleats
WHERE num_proj IS NULL;
```

Obtindríem la resposta següent:

codi_empl	nom_empl
8	Sergi

5. Predicats ANY/SOME i ALL

Per a veure si una columna compleix que totes les seves files (ALL) o alguna de les seves files (ANY/SOME) satisfacin una condició, podem fer:

Els predicats ANY/SOME

Podem escollir qualsevol dels dos predicats per a demanar que alguna fila satisfaci una condició.

```
SELECT <nom_columnes_seleccionar>
FROM <taula_consultar>
WHERE <nom_columna> <operador_comparació> {ALL|ANY|SOME}
      <subconsulta>;
```

Exemple d'ús dels predicats ALL i ANY/SOME

a) Un exemple d'utilització d'ALL per a trobar els codis i els noms dels projectes en què els sous de tots els empleats assignats són més petits que el preu del projecte.

```
SELECT codi_proj, nom_proj FROM projectes
WHERE preu > ALL(SELECT sou
                 FROM empleats
                 WHERE codi_proj = num_proj);
```

Fixem-nos en la condició de WHERE de la subconsulta, que ens assegura que els sous que mirem són els dels empleats assignats al projecte de la consulta. La resposta a aquesta consulta seria la següent:

codi_proj	nom_proj
1	GESCOM
2	PESCI
3	SALSA
4	TINELL

b) Un exemple d'utilització d'ANY/SOME per a trobar els codis i els noms dels projectes que tenen algun empleat que guanya un sou més elevat que el preu del projecte en el qual treballa.

```
SELECT codi_proj, nom_proj FROM projectes
WHERE preu < ANY(SELECT sou
                 FROM empleats
                 WHERE codi_proj = num_proj);
```

La resposta a aquesta consulta és buida, com es veu a continuació:

codi_proj	nom_proj

6. Predicat EXISTS

Per a veure si una subconsulta produeix alguna fila de resultats, podem fer servir la sentència anomenada *test d'existència*: EXISTS. Per a comprovar si una subconsulta no produeix cap fila de resultats, podem emprar NOT EXISTS.

```
SELECT <nom_columnes_seleccionar>
FROM <taula_consultar>
WHERE [NOT] EXISTS <subconsulta>;
```

Exemple d'ús del predicat EXISTS

Es busquen els codis i els noms dels empleats que estan assignats a algun projecte.

```
SELECT codi_empl, nom_empl FROM empleats
WHERE EXISTS (SELECT *
              FROM projectes
              WHERE codi_proj = num_proj);
```

La resposta a aquesta consulta seria la següent:

codi_empl	nom_empl
1	Maria
2	Pere
3	Anna
4	Jordi
5	Clara
6	Laura
7	Roger

2.5.4. Ordenació de les dades obtingudes en respostes a consultes

Si es vol que, en fer una consulta, les dades apareguin en un ordre determinat, cal utilitzar la clàusula **ORDER BY** en la sentència SELECT, que té el format següent:

```
SELECT <nom_columnes_seleccionar>
FROM <taula_consultar>
[WHERE <condicions>]
ORDER BY <nom_columna_segons_la_que_ordenar> [DESC]
         [, <nom_columna_segons_la_que_ordenar> [DESC]...];
```

Consulta amb resposta ordenada a BDUOC

Es vol consultar els noms dels empleats ordenats segons el sou que guanyin, i si guanyen el mateix sou, ordenats alfabèticament pel nom:

```
SELECT codi_empl, nom_empl, cognom_empl, sou
FROM empleats
ORDER BY sou, nom_empl;
```

Aquesta consulta donaria la resposta següent:

codi_empl	nom_empl	cognom_empl	sou
6	Laura	Tort	30000.0
8	Sergi	Grau	30000.0
5	Clara	Blanc	40000.0
7	Roger	Salt	40000.0
3	Anna	Ros	70000.0
4	Jordi	Roca	70000.0
2	Pere	Mas	90000.0
1	Maria	Puig	10000.0

Si no s'especifica res més, l'ordre que se seguirà serà ascendent, però si es vol seguir un ordre descendent cal afegir **DESC** darrere de cada factor d'ordenació expressat en la clàusula ORDER BY:

```
ORDER BY <nom_columna> [DESC][, <nom_columna> [DESC]...];
```

També es pot explicitar un ordre ascendent posant la paraula clau **ASC** (opció per defecte).

2.5.5. Consultes amb agrupació de files d'una taula

Les clàusules següents, afegides a la comanda SELECT FROM, permeten organitzar les files per grups:

a) La clàusula **GROUP BY** ens serveix per a agrupar files segons les columnes que indiqui aquesta clàusula.

b) La clàusula **HAVING** especifica condicions de cerca per a grups de files; porta a terme la mateixa funció que abans feia la clàusula WHERE per a les files de tota la taula, però ara les condicions s'apliquen als grups obtinguts.

Presenta el format següent:

```
SELECT <nom_columnes_seleccionar>
FROM <taula_consultar>
[WHERE <condicions>]
```

```
GROUP BY <columnes_segons_les_que_agrupar>
[HAVING <condicions_per_grups>]
[ORDER BY <nom_columna> [DESC][, <nom_columna> [DESC]...]];
```

Fixem-nos que en les sentències SQL es van afegint clàusules a mesura que la dificultat o l'exigència de la consulta ho requereix.

Consulta amb agrupació de files a BDUOC

Es vol saber el sou mitjà què guanyen els empleats de cada departament.

```
SELECT nom_dpt, ciutat_dpt, AVG(sou) AS sou_mitja
FROM empleats
GROUP BY nom_dpt, ciutat_dpt;
```

Factors d'agrupació

Els factors d'agrupació de la clàusula `GROUP BY` han de ser, com a mínim, les columnes que figuren en `SELECT`, exceptuant les columnes afectades per les funcions d'agregació.

La clàusula `GROUP BY` podríem dir que empaqueta els empleats de la taula `empleats` segons el departament en què estan assignats. En la figura següent podem veure els grups que es formarien després d'agrupar:

empleats

codiEmpl	nom_empl	cognom_empl	sou	nom_dpt	ciutat_dpt	num_proj
1	Maria	Puig	100000.0	DIR	Girona	1
2	Pere	Mas	90000.0	DIR	Barcelona	4
3	Anna	Ros	70000.0	DISS	Lleida	3
4	Jordi	Roca	70000.0	DISS	Barcelona	4
5	Clara	Blanc	40000.0	PROG	Tarragona	1
6	Laura	Tort	30000.0	PROG	Tarragona	3
8	Sergi	Grau	30000.0	PROG	Tarragona	NULL
7	Roger	Salt	40000.0	NULL	NULL	4

I així, amb l'agrupació anterior, és senzill obtenir el resultat d'aquesta consulta:

nom_dpt	ciutat_dpt	sou_mitja
DIR	Girona	100000.0
DIR	Barcelona	90000.0
DISS	Lleida	70000.0
DISS	Barcelona	70000.0
PROG	Tarragona	33000.0
NULL	NULL	40000.0

Exemple d'ús de la funció d'agregació `SUM`

Vegem un exemple d'ús d'una funció d'agregació `SUM` de SQL que apareix en la clàusula `HAVING` de `GROUP BY`: "Volem saber els codis dels projectes en els quals la suma dels sous dels empleats és més gran de 180.000 euros":

```
SELECT num_proj
FROM empleats
GROUP BY num_proj
HAVING SUM(sou)>180000.0;
```

DISTINCT i GROUP BY

En aquest exemple no cal posar `DISTINCT`, tot i que la columna `num_proj` no és atribut identificador. Fixem-nos que en la taula `empleats` hem posat tots els projectes que tenen el mateix número junts en un mateix grup i no pot ser que surtin repetits.

Novament, abans de mostrar el resultat d'aquesta consulta, vegem gràficament quins grups es formarien en aquest cas:

empleats

codiEmpl	nom_empl	cognom_empl	sou	nom_dpt	ciutat_dpt	num_proj
1	Maria	Puig	100000.0	DIR	Girona	1
5	Clara	Blanc	40000.0	PROG	Tarragona	1
3	Anna	Ros	70000.0	DISS	Lleida	3
6	Laura	Tort	30000.0	PROG	Tarragona	3
2	Pere	Mas	90000.0	DIR	Barcelona	4
4	Jordi	Roca	70000.0	DISS	Barcelona	4
7	Roger	Salt	40000.0	NULL	NULL	4
8	Sergi	Grau	30000.0	PROG	Tarragona	NULL

El resultat de la consulta seria el següent:

num_proj
4

2.5.6. Consultes a més d'una taula

Moltes vegades volem consultar dades de més d'una taula fent combinacions de columnes de taules diferents. En SQL és possible llistar més d'una taula especificant-ho en la clàusula `FROM`.

1. Combinació

La combinació aconsegueix crear una sola taula a partir de les taules especificades en la clàusula `FROM`, fent coincidir els valors de les columnes relacionades d'aquestes taules.

Exemple de combinació a BDUOC

Es vol saber el NIF del client i el codi i el preu del projecte que es desenvolupa per al client número 20:

```
SELECT projectes.codi_proj, projectes.preu, clients.nif
FROM clients, projectes
WHERE clients.codi_cli = projectes.codi_client AND
      clients.codi_cli = 20;
```

El resultat seria el següent:

projectes.codi_proj	projectes.preu	clients.nif
3	1000000.0	38.123.898-E

Si treballem amb més d'una taula pot passar que la taula resultant tingui dues columnes amb el mateix nom. Per això és obligatori especificar a quina taula corresponen les columnes a què ens referim, anomenant la taula a la qual per-

tanyen, abans de posar-les (per exemple, `clients.codi_cli`). Per a simplificar-ho s'utilitzen els àlies, que, en aquest cas, es defineixen en la clàusula `FROM`.

Exemple d'àlies a BDUOC

c podria ser l'àlies de la taula clients. D'aquesta manera, per a indicar a quina taula pertany `codi_cli` només caldria posar `c.codi_cli`.

Vegem com quedaria la consulta anterior expressada mitjançant àlies, encara que en aquest exemple no serien necessaris, perquè totes les columnes de les dues taules tenen noms diferents. Demanarem, a més, les columnes `c.codi_cli` i `p.codi_client`.

```
SELECT p.codi_proj, p.preu, c.nif, p.codi_client, c.codi_cli
FROM clients c, projectes p
WHERE c.codi_cli = p.codi_client AND c.codi_cli = 20;
```

El resultat seria el següent:

p.codi_proj	p.preu	c.nif	p.codi_client	c.codi_cli
3	1000000.0	38.123.898-E	20	20

Notem que en el `WHERE` necessitem expressar el lligam que hi ha entre les dues taules, en aquest cas `codi_cli` de clients i `codi_client` de projectes. Expressat en operacions de l'àlgebra relacional, això vol dir que fem una combinació en lloc d'un producte cartesià.

Fixem-nos que, igual que en àlgebra relacional, l'operació que acabem de fer és una equicombinació (*equi-join*) i, per tant, ens apareixen dues columnes idèntiques: `c.codi_cli` i `p.codi_client`.

La manera d'expressar la combinació que acabem de veure pertany a SQL:1989. Una manera alternativa de fer l'equicombinació d'abans, utilitzant SQL:1992, seria la següent:

```
SELECT <nom_columnes_seleccionar>
FROM <taula1> JOIN <taula2>
       {ON <condicions>|
       USING (<nom_columna> [,<nom_columna>...])}
[WHERE <condicions>];
```

Exemple anterior amb SQL:1992

L'exemple que hem fet abans emprant SQL:1992 seria:

```
SELECT p.codi_proj, p.preu, c.nif, p.codi_client, c.codi_cli
FROM clients c JOIN projectes p ON c.codi_cli = p.codi_client
WHERE c.codi_cli = 20;
```

Amb aquesta operació s'obtindria el mateix resultat que amb el mètode anterior.

L'opció `ON`, a més d'expressar condicions amb la igualtat, en el cas que les columnes que volem lligar tinguin noms diferents, ens ofereix la possibilitat d'expressar condicions amb els altres operadors de comparació que no siguin

el d'igualtat. Seria l'equivalent a l'operació que en àlgebra relacional hem anomenat *θ-combinació* (*θ-join*).

També podem fer servir una mateixa taula dues vegades amb àlies diferents, per a poder-les distingir.

Dos àlies per una mateixa taula a BDUOC

Es demanen els codis i els cognoms dels empleats que guanyen més que l'empleat que té per codi el número 5.

```
SELECT e1.codi_empl, e1.cognom_empl
FROM empleats e1 JOIN empleats e2 ON e1.sou > e2.sou
WHERE e2.codi_empl = 5;
```

S'ha pres la taula `e2` per a fixar la fila de l'empleat amb codi número 5, de manera que es pugui comparar el sou de la taula `e1`, que conté tots els empleats, amb el sou de la taula `e2`, que conté només l'empleat 5.

La resposta a aquesta consulta seria la següent:

e1.codi_empl	e1.cognom_empl
1	Puig
2	Mas
3	Ros
4	Roca

2. Combinació natural

La combinació natural (*natural join*) de dues taules consisteix bàsicament, igual que en l'àlgebra relacional, a fer una equicombinació entre columnes del mateix nom i eliminar les columnes repetides. La combinació natural, utilitzant SQL:1992, es faria de la manera següent:

```
SELECT <nom_columnes_seleccionar>
FROM <taula1> NATURAL JOIN <taula2>
[WHERE <condicions>];
```

Combinació natural a BDUOC

Un cas en què les columnes per a les quals es faria la combinació natural s'anomenen igual en totes dues taules. Es vol saber el codi i el nom dels empleats que estan assignats al departament que té per telèfon 977.333.852.

```
SELECT codi_empl, nom_empl
FROM empleats NATURAL JOIN departaments
WHERE telefon = "977.333.852";
```

La combinació natural també es podria fer amb la clàusula `USING`, només aplicant la paraula reservada `JOIN`:

```
SELECT codi_empl, nom_empl
FROM empleats JOIN departaments USING (nom_dpt,ciutat_dpt)
WHERE telefon = "977.333.852";
```

La resposta a aquesta consulta seria la següent:

empleats.codi_empl	empleats.nom_empl
5	Clara
6	Laura
8	Sergi

3. Combinació interna i externa

Qualsevol combinació pot ser interna o externa.

La **combinació interna** (*inner join*) només es queda amb les files que tenen valors idèntics en les columnes de les taules que compara. Això pot fer que es perdi alguna fila interessant d'alguna de les dues taules, per exemple, perquè es troba a `NULL` en el moment de fer la combinació. El seu format és el següent:

```
SELECT <nom_columnes_seleccionar>
FROM <taula1> [NATURAL] [INNER] JOIN <taula2>
       {ON <condicions>|
       USING (<nom_columna> [,<nom_columna>...])}
[WHERE <condicions>];
```

Si no es vol perdre cap fila d'una taula determinada, es pot fer una **combinació externa** (*outer join*), que permet obtenir tots els valors de la taula que hem posat a la dreta, o els valors de la que hem posat a l'esquerra, o tots els valors d'ambdues taules. El seu format és el següent:

```
SELECT <nom_columnes_seleccionar>
FROM <taula1> [NATURAL] {LEFT|RIGHT|FULL} [OUTER] JOIN <taula2>
     {ON <condicions>|
     USING (<nom_columna> [,<nom_columna>...])}
[WHERE <condicions>];
```

Combinació natural interna a BDUOC

Si es volgués lligar les taules `empleats` i `departaments` amb una combinació natural interna per a saber el codi i el nom de tots els empleats i el nom, la ciutat i el telèfon de tots els departaments, s'hauria de fer el següent:

```
SELECT e.codi_empl, e.nom_empl, e.nom_dpt, e.ciutat_dpt, d.telefon
FROM empleats e NATURAL JOIN departaments d;
```

Combinació interna

Encara que en l'exemple fem una combinació natural interna, no cal posar la paraula `INNER`, ja que és l'opció per defecte.

Amb aquesta combinació s'obtindria el resultat següent:

e.codi_empl	e.nom_empl	e.nom_dpt	e.ciutat_dpt	d.telefon
1	Maria	DIR	Girona	972.23.89.70
2	Pere	DIR	Barcelona	93.422.60.70
3	Anna	DISS	Lleida	973.23.50.40
4	Jordi	DISS	Barcelona	93.224.85.23
5	Clara	PROG	Tarragona	977.33.38.52
6	Laura	PROG	Tarragona	977.33.38.52
8	Sergi	PROG	Tarragona	977.33.38.52

En el resultat no surt l'empleat número 7, que no està assignat a cap departament, ni el departament de programació de Girona, que no té cap empleat assignat.

Combinació natural externa a BDUOC

En els exemples següents veurem com varien els resultats que anirem obtenint segons els tipus de combinació externa:

a) Combinació externa esquerra

```
SELECT e.codi_empl, e.nom_empl, e.nom_dpt, e.ciutat_dpt, d.telefon
FROM empleats e NATURAL LEFT OUTER JOIN departaments d;
```

Combinació externa esquerra

Hi figura l'empleat 7.

Amb aquesta combinació s'obtindria el resultat següent:

e.codi_empl	e.nom_empl	nom_dpt	ciutat_dpt	d.telefon
1	Maria	DIR	Girona	972.23.89.70
2	Pere	DIR	Barcelona	93.422.60.70
3	Anna	DISS	Lleida	973.23.50.40
4	Jordi	DISS	Barcelona	93.224.85.23
5	Clara	PROG	Tarragona	977.33.38.52
6	Laura	PROG	Tarragona	977.33.38.52
7	Roger	NULL	NULL	NULL
8	Sergi	PROG	Tarragona	977.33.38.52

b) Combinació externa dreta

```
SELECT e.codi_empl, e.nom_empl, e.nom_dpt, e.ciutat_dpt, d.telefon
FROM empleats e NATURAL RIGHT OUTER JOIN departaments d;
```

Combinació externa dreta

Hi figura el departament de programació de Girona.

Amb aquesta combinació s'obtindria el resultat següent:

e.codi_empl	e.nom_empl	e.nom_dpt	e.ciutat_dpt	d.telefon
1	Maria	DIR	Girona	972.23.89.70
2	Pere	DIR	Barcelona	93.422.60.70
3	Anna	DISS	Lleida	973.23.50.40
4	Jordi	DISS	Barcelona	93.224.85.23

e.codi_empl	e.nom_empl	e.nom_dpt	e.ciutat_dpt	d.telefon
5	Clara	PROG	Tarragona	977.33.38.52
6	Laura	PROG	Tarragona	977.33.38.52
8	Sergi	PROG	Tarragona	977.33.38.52
NULL	NULL	PROG	Girona	972.23.50.91

c) Combinació externa plena

```
SELECT e.codi_empl, e.nom_empl, e.nom_dpt, e.ciutat_dpt, d.telefon
FROM empleats e NATURAL FULL OUTER JOIN departaments d;
```

Combinació externa plena

Hi figura l'empleat 7 i el departament de programació de Girona.

Amb aquesta combinació s'obtindria el resultat següent:

e.codi_empl	e.nom_empl	e.nom_dpt	e.ciutat_dpt	d.telefon
1	Maria	DIR	Girona	972.23.89.70
2	Pere	DIR	Barcelona	93.422.60.70
3	Anna	DISS	Lleida	973.23.50.40
4	Jordi	DISS	Barcelona	93.224.85.23
5	Clara	PROG	Tarragona	977.33.38.52
6	Laura	PROG	Tarragona	977.33.38.52
7	Roger	NULL	NULL	NULL
8	Sergi	PROG	Tarragona	977.33.38.52
NULL	NULL	PROG	Girona	972.23.50.91

4. Combinacions amb més de dues taules

Si es vol combinar tres taules o més amb SQL:1989 només cal afegir totes les taules en el `FROM` i els lligams necessaris en el `WHERE`. Si es volen combinar amb SQL:1992 cal anar fent combinacions per parelles de taules i la taula resultant es convertirà en la primera parella de la següent.

Combinacions amb més de dues taules a BDUOC

Vegem exemples de tots dos casos, suposant que es vulgui combinar les taules `empleats`, `projectes` i `clients`:

```
SELECT *
FROM empleats, projectes, clients
WHERE num_proj = codi_proj AND codi_client = codi_cli;
```

o bé:

```
SELECT *
FROM (empleats JOIN projectes ON num_proj = codi_proj)
JOIN clients ON codi_client = codi_cli;
```

2.5.7. La unió

La clàusula **UNION** permet unir consultes de dues o més sentències SELECT FROM. El seu format és el següent:

```
SELECT <nom_columnes>
FROM <taula>
[WHERE <condicions>]
UNION [ALL]
SELECT <nom_columnes>
FROM <taula>
[WHERE <condicions>];
```

Si s'utilitza l'opció ALL apareixen totes les files obtingudes en fer la unió. No s'escriurà aquesta opció si es volen eliminar les files repetides. El més important de la unió és que som nosaltres els que hem de vigilar que es faci entre columnes definides sobre dominis compatibles; és a dir, que tinguin la mateixa interpretació semàntica. Com ja s'ha comentat, SQL no ofereix eines per a assegurar la compatibilitat semàntica entre columnes.

Utilització de la unió a BDUOC

Si es volen saber totes les ciutats que hi ha a la BD es pot fer el següent:

```
SELECT ciutat
FROM clients
UNION
SELECT ciutat_dpt
FROM departaments;
```

El resultat d'aquesta consulta seria el següent:

ciutat
Barcelona
Girona
Lleida
Tarragona

2.5.8. La intersecció

Per a fer la intersecció entre dues o més sentències SELECT FROM es pot utilitzar la clàusula **INTERSECT**, el format de la qual és el següent:

```
SELECT <nom_columnes>
FROM <taula>
[WHERE <condicions>]
INTERSECT [ALL]
SELECT <nom_columnes>
FROM <taula>
[WHERE <condicions>];
```

Si s'utilitza l'opció `ALL` apareixen totes les files obtingudes en fer la intersecció. No s'escriurà aquesta opció si es volen eliminar les files repetides. El més important de la intersecció és que som nosaltres els que hem de vigilar que es faci entre columnes definides sobre dominis compatibles; és a dir, que tinguin la mateixa interpretació semàntica.

Utilització de la intersecció a BDUOC

Es volen saber totes les ciutats on hi ha departaments en els quals es pot trobar algun client.

```
SELECT ciutat
FROM clients
INTERSECT
SELECT ciutat_dpt
FROM departaments;
```

El resultat d'aquesta consulta seria el següent:

ciutat
Barcelona
Girona
Lleida
Tarragona

La intersecció és una de les operacions de SQL que es pot fer de més maneres diferents:

- Intersecció utilitzant `IN`:

```
SELECT <nom_columnes>
FROM <taula>
WHERE <nom_columna> IN (SELECT <nom_columna>
                        FROM <taula>
                        [WHERE <condicions>]);
```

- Intersecció utilitzant `EXISTS`:

```
SELECT <nom_columnes>
FROM <taula>
WHERE EXISTS (SELECT *
              FROM <taula>
              WHERE <condicions>);
```

Exemple anterior expressat amb `IN` i amb `EXISTS`

L'exemple anterior es podria expressar amb `IN`:

```
SELECT ciutat
FROM clients
WHERE ciutat IN (SELECT d.ciutat_dpt
                 FROM departaments d);
```

o també amb EXISTS:

```
SELECT c.ciutat
FROM clients c
WHERE EXISTS (SELECT *
              FROM departaments d
              WHERE c.ciutat = d.ciutat_dpt);
```

2.5.9. La diferència

Per a trobar la diferència entre dues o més sentències SELECT FROM podem fer servir la clàusula **EXCEPT**, que té aquest format:

```
SELECT <nom_columnes>
FROM <taula>
[WHERE <condicions>]
EXCEPT [ALL]
SELECT <nom_columnes>
FROM <taula>
[WHERE <condicions>];
```

Si s'utilitza l'opció ALL apareixen totes les files obtingudes en fer la diferència. No s'escriurà aquesta opció si es vol eliminar les files repetides. El més important de la diferència és que som nosaltres els que hem de vigilar que es faci entre columnes definides sobre dominis compatibles.

Utilització de la diferència a BDUOC

Es volen saber els clients que no han contractat cap projecte.

```
SELECT codi_cli FROM clients EXCEPT
SELECT codi_client FROM projectes;
```

El resultat d'aquesta consulta seria el següent:

codi_cli
40

La diferència és, juntament amb la intersecció, una de les operacions de SQL que es pot fer de més maneres diferents:

- Diferència utilitzant NOT IN:

```
SELECT <nom_columnes>
FROM <taula>
WHERE <nom_columna> NOT IN (SELECT <nom_columna>
                            FROM <taula>
                            [WHERE <condicions>]);
```

- Diferència utilitzant `NOT EXISTS`:

```
SELECT <nom_columnes>
FROM <taula>
WHERE NOT EXISTS (SELECT *
                  FROM <taula>
                  WHERE <condicions>);
```

Exemple anterior expressat amb `NOT IN` i amb `NOT EXISTS`

L'exemple anterior es podria expressar amb `NOT IN`:

```
SELECT codi_cli
FROM clients
WHERE codi_cli NOT IN (SELECT codi_client FROM projectes);
```

o també amb `NOT EXISTS`:

```
SELECT c.codi_cli
FROM clients c
WHERE NOT EXISTS (SELECT *
                  FROM projectes p
                  WHERE c.codi_cli = p.codi_client);
```

3. Sentències de control

A més de definir i manipular una BD relacional, és important que s'estableixin **mecanismes de control** per a resoldre problemes de concurrència d'usuaris i garantir la seguretat de les dades. Per a la concurrència d'usuaris farem servir el concepte de *transacció* i per a la seguretat veurem com es poden autoritzar i desautoritzar usuaris a accedir a la BD.

3.1. Les transaccions

Una transacció és una unitat lògica de treball. O, informalment, i treballant amb SQL, un conjunt de sentències que s'executen com si fossin una de sola. En general, les sentències que formen part d'una transacció s'interrelacionen entre si, i no té sentit que se n'executi una sense que s'executin les altres.

La majoria de transaccions s'inicien de manera implícita en utilitzar alguna sentència que comença amb `CREATE`, `ALTER`, `DROP`, `SET`, `DECLARE`, `GRANT` o `REVOKE`, encara que existeix la sentència SQL per a **iniciar transaccions**, que és la següent:

```
SET TRANSACTION {READ ONLY|READ WRITE};
```

Si volem actualitzar la BD usarem l'opció `READ WRITE`, mentre que si no la volem actualitzar triarem l'opció `READ ONLY`.

Però, en canvi, una transacció sempre ha d'acabar explícitament amb alguna de les següents sentències:

```
{COMMIT|ROLLBACK} [WORK];
```

La diferència entre `COMMIT` i `ROLLBACK` és que mentre la sentència `COMMIT` confirma tots els canvis produïts contra la BD durant l'execució de la transacció, la sentència `ROLLBACK` desfà tots els canvis que s'hagin produït a la BD i la deixa com estava abans d'iniciar la nostra transacció.

La paraula reservada `WORK` només serveix per a aclarir el que fa la sentència, i és totalment opcional.

Exemple de transacció

Tot seguit proposem un exemple de transacció en què es vol disminuir el sou dels empleats que han treballat en el projecte 3 en 1.000 euros i augmentar el sou dels empleats que han treballat en el projecte 1 també en 1.000 euros.

```
SET TRANSACTION READ WRITE
UPDATE empleats SET sou = sou - 1000 WHERE num_proj = 3
UPDATE empleats SET sou = sou + 1000 WHERE num_proj = 1
COMMIT
```

3.2. Les autoritzacions i desautoritzacions

El propietari de la BD és qui té tots els privilegis sobre aquesta, però no és l'únic que hi accedeix. Per això, SQL ens ofereix sentències per a autoritzar i desautoritzar altres usuaris.

1) Autoritzacions

Per a autoritzar SQL disposa de la sentència següent:

```
GRANT <privilegis> ON <objecte> TO <usuaris>
[WITH GRANT OPTION];
```

a) Exemples de **`privilegis`** poden ser:

- `ALL PRIVILEGES`: tots els privilegis sobre l'objecte especificat.
- `SELECT`: [(`columnes`)]: consultes. Es pot concretar a quines `columnes`.
- `INSERT`: insercions.
- `UPDATE` [(`columnes`)]: modificacions. Es pot concretar a quines `columnes`.
- `DELETE`: esborrats.
- `REFERENCES` [(`columna`)]: referència de l'objecte en restriccions d'integritat. Es pot concretar a quines `columnes`.

b) Exemples d'**`objecte`** poden ser:

- `TABLE`: taula.
- `Vista`.

c) Usuaris pot ser tothom: `PUBLIC`, o bé una llista dels identificadors dels usuaris que volem autoritzar.

d) L'opció **`WITH GRANT OPTION`** permet que l'usuari que autoritzem pugui, al seu torn, autoritzar d'altres usuaris a accedir a l'objecte amb els mateixos privilegis amb els quals ha estat autoritzat.

2) Desautoritzacions

Per a desautoritzar, SQL disposa de la sentència següent:

```
REVOKE [GRANT OPTION FOR] <privilegis> ON <objecte> FROM
<usuaris> {RESTRICT|CASCADE};
```

On tenim que:

a) `privilegis`, `objecte` i `usuaris` són els mateixos que per a la sentència `GRANT`.

b) L'opció **`GRANT OPTION FOR`** s'usaria en el cas que volguéssim desautoritzar el dret a autoritzar (`WITH GRANT OPTION`).

c) L'opció **`CASCADE`** fa que, si un usuari que hem autoritzat, n'ha autoritzat al seu torn d'altres, que alhora poden haver fet més autoritzacions, quedin desautoritzats tots de cop.

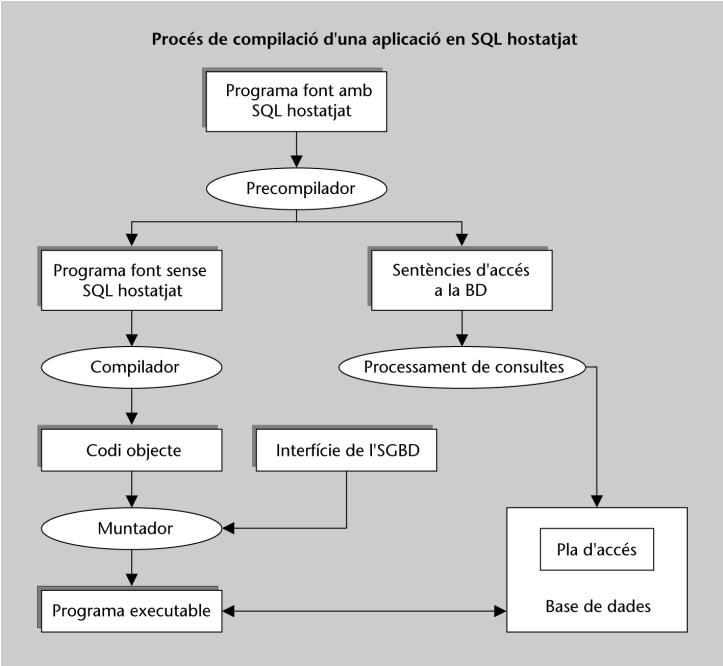
d) L'opció **`RESTRICT`** no ens permet desautoritzar un usuari si aquest n'ha autoritzat d'altres.

4. Subllenguatges especialitzats

Moltes vegades voldrem accedir a la BD des d'una aplicació feta en un llenguatge de programació qualsevol. Per a utilitzar SQL des d'un llenguatge de programació podem fer servir **SQL hostatjat**, i per a treballar amb SQL hostatjat necessitem un precompilador que separi les sentències del llenguatge de programació de les del llenguatge de BD. Una alternativa molt interessant a aquesta manera de treballar són les **rutines SQL/CLI**.

L'objectiu d'aquest apartat no és explicar en profunditat ni SQL hostatjat ni, encara menys, les rutines SQL/CLI. Només introduirem les idees bàsiques del funcionament d'ambdós i presentarem un exemple de combinació de SQL amb el llenguatge de programació Java.

4.1. SQL hostatjat



Per a crear i manipular una BD relacional necessitem SQL. A més, si la feina que volem fer requereix el poder de processament d'un llenguatge de programació com Java, C++, Cobol, Fortran, Pascal, etc., podem utilitzar SQL hostatjat en el llenguatge de programació escollit, que l'anomenarem *amfitrió*. Per a poder compilar aquest batibull de crides de SQL i sentències de programació, abans hem de fer servir un precompilador. Un precompilador és una eina que separa les sentències de SQL i les sentències de programació. Allà on en el programa font hi hagi una sentència d'accés a la BD, marcada segons l'estàndard amb la clàusula `EXEC SQL`*, s'hi ha d'inserir una crida a la interfície de l'SGBD. El programa font resultant de la precompilació ja està tot en llenguatge de programació, preparat per a ser compilat, muntat i executat.

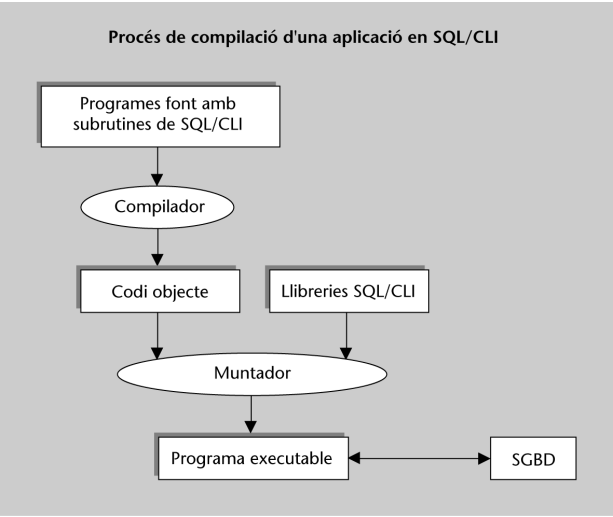
*** Hi pot haver petites diferències depenent del llenguatge de programació concret que considerem. Per exemple, si utilitzem SQLJ (SQL hostatjat en Java) la paraula reservada és `#sql`.**

Totes les sentències de definició i manipulació que hem vist per a SQL es poden utilitzar en SQL hostatjat. Tenint present ara que necessitarem un nou tipus de variables anomenades **variables pont** que serviran perquè el programa pugui recollir i utilitzar els valors d'una fila de la BD. Una altra cosa a tenir en compte és que quan el resultat d'una sentència SQL obtingui més d'una fila, llavors haurem de treballar amb el **concepte de cursor**.

Un cursor s'ha d'haver declarat abans de fer-lo servir (`EXEC SQL DECLARE <nom_cursor> CURSOR FOR`). Per a usar-lo, s'ha d'obrir (`EXEC SQL OPEN <nom_cursor>`), anar agafant-ne les dades una a una, (`EXEC SQL FETCH <nom_cursor> INTO`), tractar-les i, finalment, tancar-lo (`EXEC SQL CLOSE <nom_cursor>`).

4.2. Les SQL/CLI

Les SQL/CLI, permeten que aplicacions desenvolupades en un cert llenguatge de programació (amb només les eines disponibles per a aquest llenguatge i sense l'ús d'un precompilador) hi puguin incloure sentències SQL mitjançant crides a rutines predefinides disponibles en llibreries. Aquestes sentències SQL s'han d'interpretar en temps d'execució. A diferència de SQL hostatjat que requeria l'ús d'un precompilador, les SQL/CLI no en necessiten. Però en generar el pla d'accés de les sentències SQL en temps d'execució es produeix una disminució global del rendiment respecte a SQL hostatjat que troba el pla d'accés de les consultes en temps de compilació. Si s'utilitzen les SQL/CLI el que cal fer és simplement muntar l'aplicació juntament amb la llibreria de rutines tal com es mostra en la figura següent:



La **interfície ODBC** (*Open Database Connectivity*) defineix una llibreria de funcions que permet a les aplicacions accedir a l'SGBD utilitzant SQL. Les SQL/CLI es basen fortament en les característiques de la interfície ODBC i, gràcies al treball desenvolupat per SAG-X/Open (SQL Access Group-X/Open), van ser afegides a l'estàndard SQL:1992 l'any 1995.

Les SQL/CLI són rutines que criden l'SGBD per a interpretar les sentències SQL que demana l'aplicació. Des del punt de vista de l'SGBD, les SQL/CLI es poden veure, simplement, com altres aplicacions. Això fa que el codi de les aplicacions només contingui instruccions del llenguatge de programació. Totes les sentències SQL queden amagades dins les rutines de les SQL/CLI. A més, les aplicacions que es desenvolupin no necessiten conèixer les sentències SQL abans de ser executades i poden ser independents d'un SGBD concret.

El nom de les rutines SQL/CLI consta de dues parts: un prefix (`SQL`) i el nom de la feina que fa la subrutina, per exemple: `Fetch`. Així, aquesta subrutina s'anomenaria `SQLFetch`.

4.3. SQL i Java

Per a il·lustrar les dues tècniques anteriors de SQL programat hem escollit el llenguatge Java. En el cas de SQL hostatjat en Java, **SQLJ**, segueix força fidelment el que hem explicat per a aquesta tècnica. En canvi, les SQL/CLI de Java, anomenades **JDBC**, són una solució basada en la filosofia de la tècnica que hem presentat però que es deslliga de les fortes característiques de la interfície ODBC. Per aquest motiu les presentarem en primer lloc.

4.3.1. JDBC

JDBC és un API (*Application Programming Interface*) estàndard de Java que permet tenir interacció amb les dades de la BD des d'un programa escrit en Java. Es basa en la tècnica SQL/CLI i consisteix en una sèrie de classes i interfícies escrites en Java que pot fer servir un programador per a accedir a la BD.

Exemple

Execució d'una consulta amb el seu corresponent tractament d'errors sobre la taula `empleats` per a obtenir el nom de l'empleat amb codi 1.

```
try
{
  Statement stmt = conn.createStatement();
  ResultSet rs = stmt.executeQuery("SELECT nom_empl FROM empleats
  WHERE codi_empl=1");
  rs.next();
  String qNom = rs.getString("nom_empl");
  System.out.println(qNom);
  rs.close();
  stmt.close();
}
catch (SQLException e)
{
  if (e.getSQLState().equals("02000"))
  {
     System.out.println("No existeix cap tuple que compleixi la
                         condició");
  }
}
```

Nota

Abans d'executar aquesta consulta, caldria haver localitzat i carregat el *driver* JDBC i haver-se connectat a la BD. Després de l'execució, cal fer la desconnexió de la BD.

Analitzem la consulta:

- Se sol·licita la creació d'objecte `Statement` (`stmt`) associat a un objecte de connexió actiu (`conn`). Com que l'objecte de la classe `Statement` del JDBC es crea a partir d'una connexió activa, també disposa del seu propi entorn d'execució.
- Com que el que es vol és fer una consulta, el mètode que s'ha d'utilitzar a l'objecte `Statement` és `executeQuery` que torna un objecte JDBC anomenat `ResultSet`.
- Per a poder recuperar el resultat de la consulta (en aquest cas es tracta d'un únic valor), cal executar el mètode `next`.
- Per a visualitzar el resultat que interessi del `ResultSet` es pot usar el mètode `get<Tipus>`, en què `<Tipus>` es correspondrà amb el tipus de dades de la columna que es vulgui consultar.
- Si ja no s'ha d'utilitzar més, es pot tancar el `ResultSet` explícitament mitjançant el mètode `close`. L'objecte de la classe `Statement` també es pot tancar explícitament mitjançant el mètode `close`.

A més, s'ha fet un senzill tractament d'excepcions utilitzant un objecte `SQLException` del JDBC que ofereix mètodes per a obtenir informació dels errors. En aquest cas, per a obtenir un codi estàndard d'error s'ha utilitzat `getSQLState`. El codi estàndard d'error `02000` vol dir que la consulta no ha retornat cap dada.

4.3.2. SQLJ

SQLJ és una extensió/abstracció de l'API JDBC que permet introduir sentències SQL directament en el codi de Java. Les instruccions de SQLJ dins d'un programa Java es denoten, a diferència del que diu l'estàndard, amb la paraula reservada **`#sql`**.

Exemple

Vegem l'execució de la mateixa consulta de l'apartat anterior sobre la taula `empleats` amb el seu tractament d'errors corresponent.

```
try
{
  String qNom = null;
  #sql {SELECT nom_empl INTO qNom FROM empleats WHERE
  codi_empl=1};
  System.out.println(qNom);

}
catch (SQLException e)
{
  if (e.getSQLState().equals("02000"))
  {
      System.out.println("No existeix cap tuple que compleixi la
                          condició");
  }
}
```

A diferència de l'exemple anterior, en què la consulta era un paràmetre d'un mètode, en el cas de SQLJ es demana directament la consulta amb la paraula reservada `#sql`. Per tant, per a recollir el resultat de la consulta cal utilitzar una variable pont, la variable `qNom`.

Resum

En aquest mòdul hem presentat les sentències més utilitzades del llenguatge estàndard SQL de definició i manipulació de BD relacionals. Com ja hem comentat en la introducció, SQL és un llenguatge molt potent i això fa que hi hagi més sentències i opcions de les que hem explicat en aquest mòdul. També és cert, però, que hem vist més sentències de les que alguns sistemes relacionals comercials ofereixen actualment. Hem intentat seguir amb la major fidelitat l'estàndard, incloent-hi comentaris només quan en la majoria de SGBD comercials alguna operació es feia de manera diferent.

Si es coneix SQL es pot treballar amb qualsevol SGBD comercial; només caldrà dedicar unes quantes hores a identificar-ne les variacions respecte de l'estàndard.

Recordem com serà **la creació d'una BD amb SQL**:

1. Donar nom a la BD, amb la sentència `CREATE DATABASE`, si n'hi ha, o amb `CREATE SCHEMA`.

2. Definir les taules, els dominis, les assercions i les vistes que formaran la BD.

3. Emplenar les taules amb la sentència `INSERT INTO`.

Quan la BD tingui un conjunt de files, es podrà **manipular**; és a dir, actualitzar-ne les files o fer-hi consultes directament amb SQL interactiu o SQL programat.

A més, podem fer servir les sentències de control que hem explicat. També hem plantejat el problema de l'**actualització de les vistes**.

Activitat

1. De segur que sempre heu volgut saber on teníeu aquells apunts que mai no trobeu. Us proposem crear una BD per a organitzar els apunts i localitzar-los ràpidament quan us vingui de gust fer-los servir. Haureu de crear la BD i les taules; decidir les claus primàries i les foranes, i inserir-hi files.

Per a emmagatzemar els apunts haurem de crear les taules següents:

- Els apunts: voldrem saber-ne el codi, la prestatgeria on es troben, el prestatge i la carpeta, suposant que en un prestatge hi cap més d'una carpeta. El codi dels apunts l'haurem d'escriure al llom de cada carpeta física.

- Els temes: en voldrem saber el codi i el nom. El codi dels temes l'haurem d'escriure a cada carpeta física per a distingir temes que tenen el mateix nom.

- Els autors: només en voldrem saber un codi, el nom i els cognoms. El codi dels autors, que inventarem nosaltres, permetrà distingir entre autors que es diuen igual.

- Temes que hi ha a cada carpeta d'apunts: en aquesta taula introduirem el codi dels apunts i el codi del tema. En uns apunts hi pot haver més d'un tema i un tema pot aparèixer repetit en més d'una carpeta d'apunts, i s'ha de tenir en compte aquest fet a l'hora d'escollir la clau primària.

- Autors dels temes: en aquesta taula introduirem el codi del tema i el codi de l'autor. En un tema hi pot haver més d'un autor i un autor pot fer més d'un tema, i això s'ha de tenir present quan es triï la clau primària.

Esperem que, a més de practicar sentències de definició i manipulació de SQL, aquesta activitat us resulti útil.

Exercicis d'autoavaluació

Amb l'activitat hem inserit files i, si ens haguéssim equivocat, també hauríem esborrat i modificat alguna fila. Amb els exercicis d'autoavaluació practicarem la part de sentències de manipulació de què encara no hem tractat: les consultes. Els exercicis que proposem es faran sobre la BD relacional BDUOC que ha anat sortint en tot aquest mòdul.

1. Obtingueu els codis i els noms i cognoms dels empleats ordenats alfabèticament de manera descendent per cognom i, en cas de repeticions, per nom.

2. Consulteu el codi i el nom dels projectes dels clients que són de Barcelona.

3. Obtingueu els noms i les ciutats dels departaments que treballen en els projectes número 3 i número 4.

4. De tots els empleats que perceben un sou d'entre 50.000 i 80.000 euros, busqueu-ne els codis d'empleat i els noms dels projectes que tenen assignats.

5. Busqueu el nom, la ciutat i el telèfon dels departaments on treballen els empleats del projecte GESCOM.

6. Obtingueu els codis i els noms i cognoms dels empleats que treballen en els projectes de preu més alt.

7. Esbrineu quin és el sou més alt de cada departament. Concretament, cal donar el nom i la ciutat del departament i el sou més gran.

8. Obtingueu els codis i els noms dels clients que tenen més d'un projecte contractat.

9. Esbrineu els codis i els noms dels projectes en què tots els empleats que hi estan assignats tenen un sou superior a 30.000 euros.

10. Busqueu els noms i les ciutats dels departaments que no tenen cap empleat assignat.

Solucionari

1.

```
SELECT cognom_empl, nom_empl, codi_empl
FROM empleats
ORDER BY cognom_empl DESC, nom_empl DESC;
```

2. Amb SQL:1989, la solució seria:

```
SELECT p.codi_proj, p.nom_proj
FROM projectes p, clients c
WHERE c.ciutat = "Barcelona" and c.codi_cli = p.codi_client;
```

Amb SQL:1992, la solució seria:

```
SELECT p.codi_proj, p.nom_proj
FROM projectes p JOIN clients c ON c.codi_cli = p.codi_client
WHERE c.ciutat = "Barcelona";
```

3.

```
SELECT DISTINCT e.nom_dpt, e.ciutat_dpt
FROM empleats e
WHERE e.num_proj IN (3,4);
```

4. Amb SQL:1989, la solució seria:

```
SELECT e.codi_empl, p.nom_proj
FROM empleats e, projectes p
WHERE e.sou BETWEEN 50000.0 AND 80000.0 and e.num_proj = p.codi_proj;
```

Amb SQL:1992, la solució seria:

```
SELECT e.codi_empl, p.nom_proj
FROM empleats e JOIN projectes p ON e.num_proj = p.codi_proj
WHERE e.sou BETWEEN 50000.0 AND 80000.0;
```

5. Amb SQL:1989, la solució seria:

```
SELECT DISTINCT d.*
FROM departaments d, empleats e, projectes p
WHERE p.nom_proj = "GESCOM" and d.nom_dpt = e.nom_dpt AND
                  d.ciutat_dpt = e.ciutat_dpt and e.num_proj = p.codi_proj;
```

Amb SQL:1992, la solució seria:

```
SELECT DISTINCT d.nom_dpt, d.ciutat_dpt, d.telefon
FROM (departaments d NATURAL JOIN empleats e) JOIN projectes p ON
e.num_proj = p.codi_proj
WHERE nom_proj = "GESCOM";
```

6. Amb SQL:1989, la solució seria:

```
SELECT e.codi_empl, e.nom_empl, e.cognom_empl
FROM projectes p, empleats e
WHERE e.num_proj = p.codi_proj and p.preu = (SELECT MAX(p1.preu)
                                             FROM projectes p1);
```

Amb SQL:1992, la solució seria:

```
SELECT e.codi_empl, e.nom_empl, e.cognom_empl
FROM empleats e JOIN projectes p ON e.num_proj = p.codi_proj
WHERE p.preu = (SELECT MAX(p1.preu)
         FROM projectes p1);
```

7.

```
SELECT nom_dpt, ciutat_dpt, MAX(sou) AS sou_maxim
FROM empleats
GROUP BY nom_dpt, ciutat_dpt;
```

8. Amb SQL:1989, la solució seria:

```
SELECT c.codi_cli, c.nom_cli
FROM projectes p, clients c
WHERE c.codi_cli = p.codi_client
GROUP BY c.codi_cli, c.nom_cli
HAVING COUNT(*) > 1;
```

Amb SQL:1992, la solució seria:

```
SELECT c.codi_cli, c.nom_cli
FROM projectes p JOIN clients c ON c.codi_cli = p.codi_client
GROUP BY c.codi_cli, c.nom_cli
HAVING COUNT(*) > 1;
```

9. Amb SQL:1989, la solució seria:

```
SELECT p.codi_proj, p.nom_proj FROM projectes p, empleats e
WHERE e.num_proj = p.codi_proj
GROUP BY p.codi_proj, p.nom_proj
HAVING MIN(e.sou) > 30000.0;
```

Amb SQL:1992, la solució seria:

```
SELECT p.codi_proj, p.nom_proj
FROM empleats e JOIN projectes p ON e.num_proj = p.codi_proj
GROUP BY p.codi_proj, p.nom_proj
HAVING MIN(e.sou)>30000.0;
```

10.

```
SELECT d.nom_dpt, d.ciutat_dpt FROM departaments d
WHERE NOT EXISTS (SELECT *
```

```
                FROM empleats e
                WHERE e.nom_dpt = d.nom_dpt AND e.ciutat_dpt =
                      d.ciutat_dpt);
```

o bé:

```
SELECT nom_dpt, ciutat_dpt
FROM departaments
EXCEPT
SELECT nom_dpt, ciutat_dpt
FROM empleats;
```

Bibliografia

L'SLQ:1992 es defineix, segons si el cerqueu a ISO o a ANSI, en qualsevol dels dos documents següents:

- Database Language SQL (1992). Document ISO/IEC 9075:1992. International Organization for Standardization (ISO).

- Database Language SQL (1992). Document ANSI/X3.135-1992. American National Standards Institute (ANSI).

Date, C. J. (2001). *Introducción a los sistemas de bases de datos* (7a. ed.). Madrid: Prentice-Hall.

Teniu encara una versió més resumida d'un dels mateixos autors del llibre anterior en el capítol 4 d'aquest llibre. A més en l'apèndix B podeu trobar una panoràmica de SQL: 1999.

Date, C. J.; Darwen, H. (1997). *A Guide to the SQL Standard* (4a. ed.). Reading, Massachusetts: Addison-Wesley.

Els llibres que contenen la descripció de l'estàndard ANSI/ISO SQL: 1992 són força gruixuts i feixucs de llegir. Aquest llibre és un resum de l'oficial.

Altres llibres traduïts al castellà de SQL: 1992 que us recomanem són els següents:

Groff, J. R.; Weinberg, P. N. (1998). *LAN Times. Guía de SQL*. Osborne: McGraw-Hill.

Us en recomanem la consulta per la seva claredat i pels comentaris sobre la manera com s'utilitza l'estàndard en els diferents sistemes relacionals comercials.

Silberschatz, A.; Korth, H. F.; Sudarshan, S. (1998). *Fundamentos de bases de datos*. (3a. ed.). Madrid: McGraw-Hill.

Podeu trobar una lectura ràpida, resumida, però força completa de SQL en el capítol 4 d'aquest llibre.

Per a aprofundir en l'estudi de SQL: 1999 recomanem el llibre següent:

Melton, J.; Simon, A. R. (2002). *SQL: 1999. Understanding Relational Language Components*. San Francisco: Morgan Kaufmann.

Per a mantenir-vos informats de les publicacions de noves versions de l'estàndard, podeu consultar l'adreça web d'ANSI:

http://www.ansi.org/ i d'ISO: http://www.iso.org/

Annexos

Annex 1. Sentències de definició de dades

1. Creació d'esquemes:

```
CREATE SCHEMA {<nom_esquema>|AUTHORIZATION <usuari>}
              [<llista_elements_esquema>];
```

2. Esborrament d'esquemes:

```
DROP SCHEMA <nom_esquema> {RESTRICT|CASCADE};
```

3. Creació de BD (aquesta sentència no forma part de SQL estàndard):

```
CREATE DATABASE <nom_BD>;
```

4. Esborrament de BD (aquesta sentència no forma part de SQL estàndard):

```
DROP DATABASE <nom_BD>;
```

5. Creació de taules:

```
CREATE TABLE <nom_taula>
   ( <definició_columna>
   [, <definició_columna>...]
   [, <restriccions_taula>]
);
```

En què tenim el següent:

- **definició_columna** és:

```
<nom_columna> {<tipus_dades>|<domini>} [<def_defecte>]
              [<restriccions_columna>]
```

- Una de les restriccions de taula era la definició de claus foranes:

```
FOREIGN KEY <clau_forana> REFERENCES <taula> [(<clau_primària>)]
[ON DELETE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
[ON UPDATE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
```

6. Modificació d'una taula:

```
ALTER TABLE <nom_taula> {<acció_modificar_columna>|
                    <acció_modificar_restricció_taula>}
```

En què tenim el següent:

- **acció_modificar_columna** pot ser:

```
{ADD [COLUMN] <nom_columna> <def_columna> |
ALTER [COLUMN] <nom_columna> {SET <def_defecte>|DROP DEFAULT}|
DROP [COLUMN] <nom_columna> {RESTRICT|CASCADE}}
```

- **acció_modificar_restricció_taula** pot ser:

```
ADD {<def_restricció>|
DROP CONSTRAINT <nom_restricció> {RESTRICT|CASCADE}}
```

7. Esborrament de taules:

```
DROP TABLE <nom_taula> {RESTRICT|CASCADE};
```

8. Creació de dominis:

```
CREATE DOMAIN <nom_domini> [AS] <tipus_dades> [<def_defecte>]
[<restriccions_domini>];
```

En què tenim el següent:

- **def_defecte** té el format següent:

```
DEFAULT {<literal>|<funció>|NULL};
```

- **restriccions_domini** té el format següent:

```
CONSTRAINT [<nom_restricció>] CHECK (<condicions>)
```

9. Modificació d'un domini:

```
ALTER DOMAIN <nom_domini> {<acció_modificar_domini>|
<acció_modificar_restricció_domini>};
```

En què tenim el següent:

- **acció_modificar_domini** pot ser:

```
SET {<def_defecte>|DROP DEFAULT}
```

- **acció_modificar_restricció_domini** pot ser:

```
{ADD <restriccions_domini>|DROP CONSTRAINT <nom_restricció>}
```

10. Esborrament de dominis creats per l'usuari:

```
DROP DOMAIN <nom_domini> {RESTRICT|CASCADE};
```

11. Definició d'una asserció:

```
CREATE ASSERTION <nom_asserció> CHECK (<condicions>);
```

12. Esborrament d'una asserció:

```
DROP ASSERTION <nom_asserció>;
```

13. Creació d'una vista:

```
CREATE VIEW <nom_vista> [(llista_columnes)] AS (consulta)
[WITH CHECK OPTION];
```

14. Esborrament d'una vista:

```
DROP VIEW <nom_vista> {RESTRICT|CASCADE};
```

Annex 2. Sentències de manipulació de dades

1. Inserció de files en una taula:

```
INSERT INTO <nom_taula> [(<columnes>)]
{VALUES ({<v1>|DEFAULT|NULL}, ..., {<vn>|DEFAULT|NULL})|
<consulta>};
```

2. Esborrament de files d'una taula:

```
DELETE FROM <nom_taula> [WHERE <condicions>];
```

3. Modificació de files d'una taula:

```
UPDATE <nom_taula>
SET <nom_columna> = {<expressió>|DEFAULT|NULL}
    [, <nom_columna> = {<expressió>|DEFAULT|NULL} ...]
WHERE <condicions>;
```

4. Consultes d'una BD relacional:

```
SELECT [DISTINCT] <nom_columnes_seleccionar>
FROM <taules_consultar>
[WHERE <condicions>]
[GROUP BY <atributs_segons_els_que_agrupar>]
[HAVING <condicions_per_grups>]
[ORDER BY <nom_columna_ord> [DESC][, <nom_columna_ord> [DESC]...]]
```

Annex 3. Sentències de control

1. Iniciació de transaccions:

```
SET TRANSACTION {READ ONLY|READ WRITE}
```

2. Finalització de transaccions:

```
{COMMIT|ROLLBACK} [WORK];
```

3. Autoritzacions:

```
GRANT <privilegis> ON <objecte> TO <usuaris>
[WITH GRANT OPTION];
```

4. Desautoritzacions:

```
REVOKE [GRANT OPTION FOR] <privilegis> ON <objecte> FROM <usuaris>
{RESTRICT|CASCADE};
```