Llenguatge SQL. Consultes.

3

Isidre Guixà i Miranda
IES SEP Milà i Fontanals, d'Igualada

Sistemes gestors de bases de dades relacionals

Novembre del 2008
© Isidre Guixà i Miranda
IES SEP Milà i Fontanals
C/. Emili Vallès, 4
08700 - Igualada

En cas de suggeriment i/o detecció d'error podeu posar-vos en contacte via el correu electrònic `iguixa@xtec.cat`

Índex

Introducció

Les aplicacions informàtiques utilitzades en l'actualitat per a la gestió de qualsevol organització, mouen una quantitat considerable de dades que s'emmagatzemen en bases de dades gestionades per sistemes gestors de bases de dades (SGBD).

En la unitat didàctica precedent us heu iniciat en el model relacional en el que es basen els sistemes gestors de bases de dades actuals i heu pogut dur iniciar-vos també en la gestió de bases de dades ofimàtiques.

Les actuals bases de dades ofimàtiques (*MsAccess*, *OpenOffice*,...) donen prous prestacions per a emmagatzemar informació que podríem anomenar "domèstica". En podem trobar molts exemples: base de dades per organitzar els nostres volums musicals (discos, CD...), base de dades per portar la comptabilitat domèstica, base de dades per gestionar les fotos que tenim a casa, base de dades per gestionar els llibres de la nostra biblioteca... I, també, per què no, gestionar les dades de petites organitzacions empresarials (botigues, tallers,...)

Però les bases de dades ofimàtiques acostumen a no tenir suficients recursos quan es tracta de gestionar grans volums d'informació a la qual han de poder accedir molts usuaris simultàniament des de la xarxa local i també des de llocs de treball remots i, per aquest motiu, apareixen les bases de dades corporatives.

Tots els SGBD (ofimàtiques i corporatives) incorporen el llenguatge SQL (***s****tructured* ***q****uery* ***l****anguage*) per a poder donar instruccions al sistema gestor i així poder efectuar altes, baixes, consultes, modificacions, crear les estructures de dades (taules i índexs) i els usuaris per accedir a les bases de dades, concedir i revocar permisos d'accés...

El treball en SGBD ofimàtiques s'acostuma a efectuar sense la utilització d'aquest llenguatge, donat que la interfície gràfica que aporta l'entorn acostuma a permetre tot tipus d'operació. Els SGBD corporatives també aporten (cada vegada més) interfícies gràfiques potents que permeten efectuar moltes operacions però, tot i així, es fa necessari el coneixement del llenguatge SQL per a efectuar múltiples tasques.

En el nucli d'activitat "Llenguatge SQL. Tipus de dades. Consultes simples" farem les primeres passes en el coneixement del llenguatge SQL, introduint-nos en els tipus de dades que pot gestionar i en el disseny de consultes senzilles a la base de dades.

En el nucli d'activitat "Llenguatge SQL. Consultes complexes" ampliarem el nostre coneixement sobre el llenguatge SQL per tal d'aprofitar tota la potència que dona en l'àmbit de la consulta d'informació. Així, per exemple, aprendrem a ordenar la informació, a agrupar-la efectuant-hi filtratges per reduir el nombre de resultats, a combinar resultats de diferents consultes,... vaja, que aquest llenguatge és una meravella que possibilita efectuar qualsevol tipus de consulta sobre una base de dades per a obtenir la informació desitjada.

Per aconseguir un bon coneixement del llenguatge SQL és necessari que aneu reproduint en el vostre ordinador tots els exemples incorporats en el text així com les activitats i els exercicis d'autoavaluació. I per a poder-ho fer utilitzarem el SGBD Oracle 11g i les eines adequades seguint les instruccions del material web.

Objectius

En acabar la unitat didàctica heu de ser capaços del següent:

1) Identificar les característiques dels sistemes gestors de bases de dades relacionals (SGBDR), les prestacions dels productes existents en l'actualitat i les tendències.

2) Relacionar les operacions bàsiques de l'àlgebra i del càlcul relacionals amb els conceptes associats a la representació de la informació.

3) Identificar les funcions, la sintaxi i les ordres bàsiques del llenguatge SQL per a la consulta de dades, segons el sistema gestor de bases de dades relacional.

4) Elaborar la guia d'usuari i la documentació completa relativa a les taules i als atributs de la base de dades relacional, de manera estructurada i clara.

5) Identificar els tipus de dades en les bases de dades corporatives, segons el sistema gestor de bases de dades relacionals.

1. Llenguatge SQL. Tipus de dades. Consultes simples.

Per iniciar l'estudi del llenguatge SQL, la millor manera és executar consultes senzilles en la base de dades. Abans, però, ens convé conèixer els orígens i evolució que ha tingut aquest llenguatge, els diferents tipus de sentències SQL existents (subdivisió del llenguatge SQL), així com els diferents tipus de dades (numèriques, dates, cadenes...) que ens podem trobar emmagatzemades en les bases de dades. I llavors ja podrem iniciar-nos en l'execució de consultes simples.

1.1. Orígens i evolució del llenguatge SQL de la ma dels SGBD.

El model relacional en el que es basen els actuals SGBD va ser presentat en 1970 pel matemàtic Edgar Frank Codd, que treballava en els laboratoris d'investigació de l'empresa d'informàtica IBM. Un dels primers SGBD relacionals en aparèixer va ser System R d'IBM que es va desenvolupar com a prototip per a provar la funcionalitat del model relacional i que anava acompanyat del llenguatge *SEQUEL* (acrònim de ***S****tructured* ***E****nglish* ***Que****ry* ***L****anguage*) per manipular i accedir a les dades emmagatzemades en el System R. Posteriorment el mot *SEQUEL* es va condensar en *SQL* (acrònim de ***S****tructured* ***Q****uery* ***L****anguage*).

Per què SQL enlloc de SEQUEL?

S'utilitza l'acrònim *SQL* enlloc de *SEQUEL* per què el mot *SEQUEL* ja estava registrat per la companyia anglesa d'avions Hawker-Siddeley.

Una vegada comprovada l'eficiència dels model relacional i del llenguatge SQL, es va iniciar una dura cursa entre diferents marques comercials. Així, tenim:

- IBM comercialitza diversos productes relacionals amb el llenguatge SQL: System/38 en 1979, SQL/DS en 1981 i DB2 en 1983.

- Relational Software, Inc. (actualment *Oracle* Corporation) desenvolupa la seva pròpia versió de SGBD relacional per la Marina dels E.E.U.U., la C.I.A. i d'altres i l'estiu del 1979 allibera *Oracle V2* (Versió 2) per a les computadores VAX (les grans competidores de l'època amb les computadores d'IBM) .

Pronunciació de SQL

ANSI ha definit que la pronunciació anglesa de l'acrònim SQL és /ɛs kju: ɛl/ i la corresponent catalana seria /ésə ku élə/. Avui en dia es troba molts professionals que erròniament pronuncien sequel.

El llenguatge SQL va anar evolucionant (cada marca comercial seguia el seu propi criteri) fins que els principals organismes d'estandardització hi van ficar ma per tal d'obligar que els diferents SGBD relacionals implementessin una versió comuna del llenguatge i així, l'ANSI (**A**merican **N**ational **S**tandards **I**nstitute) publica en 1986 l'estàndard SQL-86, que en 1987 és ratificat per l'ISO (Organització Internacional per

a la Normalització o **I**nternational **O**rganization for **S**tandardization en anglès).

La taula 1 presenta les diferents revisions que han anat apareixent de l'estàndard SQL des de 1986.

Taula 1. Revisions de l'estàndard SQL

Any	Revisió	Àlies	Comentaris
1986	SQL-86	SQL-87 / SQL1	Publicat per ANSI en 1986 i ratificat per ISO en 1987
1989	SQL-89		Petita revisió
1992	SQL-92	SQL2	Gran revisió
1999	SQL:1999	SQL3	Introdueix consultes recursives, disparadors,...
2003	SQL:2003		Introdueix temes d'XML, funcions windows,...
2006	SQL:2006		

No ens cal saber què ha aportat cada revisió, sinó que han existit aquestes revisions

1.2. Tipus de sentències SQL

Els SGBD relacionals incorporen el llenguatge SQL per executar diferents tipus de tasques en les bases de dades: definició de dades, consulta de da des, actualització de dades, definició d'usuaris, concessió de privilegis... Per aquest motiu, les sentències que aporta el llenguatge SQL s'acostumen a agrupar en:

a) Sentències destinades a la definició de les dades (LDD), que permeten definir els objectes (taules, camps, valors possibles, regles d'integritat referencial, restriccions,...).

b) Sentències destinades al control sobre les dades (LCD), que permeten concedir i retirar permisos sobre els diferents objectes de la base de dades.

c) Sentències destinades a la consulta de les dades (LC), que permeten accedir a les dades en mode consulta.

d) Sentències destinades a la manipulació de les dades (LMD), que permeten actualitzar la base de dades (altes, baixes i modificacions).

En alguns SGBD no hi ha distinció entre LC i LMD i es parla únicament de LMD per a les consultes i actualitzacions. De la mateixa manera, a vegades s'inclouen les sentències de control (LCD) juntament amb les de definició de dades (LDD). No té cap importància que s'incloguin en un grup o que siguin un grup propi, és una simple classificació.

Termes en anglès

En l'argot informàtic s'utilitza els següents termes:
- *Data Definition Language*, abreujat per DDL
- *Data Control Language*, abreujat per DCL
- *Query Language*, abreujat per QL
- *Data Manipulation Language*, abreujat per DML

Tots aquests llenguatges acostumen a tenir una sintaxi senzilla, similar a les ordres de consola per a un sistema operatiu, anomenada **sintaxi autosuficient**. (!!)

SQL hostatjat

Les sentències SQL poden presentar, però, una segona sintaxi, sintaxi hostatjada, consistent en un conjunt de sentències que són admeses dins d'un llenguatge de programació, anomenat llenguatge amfitrió.

Així ens podem trobar LC i LMD que es poden hostatjar en llenguatges de tercera generació com C, Cobol, Fortran,... i en llenguatges de quarta generació.

Els SGBD acostumen a incloure un llenguatge de tercera generació que permet hostatjar sentències SQL en petites unitats de programació (funcions i/o procediments). Així, el SGBD *Oracle* incorpora el llenguatge PL/SQL, el SGBD *SQLServer* incorpora el llenguatge Transact-SQL, el SGBD *MySQL* 5.x segueix la sintaxi SQL 2003 per la definició de rutines de la mateixa manera que el SGBD DB2 d'IBM.

1.3. Tipus de dades

L'evolució anàrquica que ha seguit el llenguatge SQL ha fet que cada SGBD hagi pres les seves decisions quant als tipus de dades permeses. Certament els diferents estàndards SQL que han anat apareixent han marcat una certa línia i els SGBD s'hi apropen, però tampoc poden deixar de donar suport als tipus de dades que han anat proporcionant al llarg de la seva existència, ja que hi ha moltes bases de dades repartides pel món que les estan utilitzant.

De tot això hem de deduir que per treballar amb un SGBD hem de conèixer els principals tipus de dades que facilita (numèriques, alfanumèriques, moments temporals,...) i ho hem de fer centrant-nos en un SGBD en concret (la nostra elecció ha estat *Oracle* en la versió *11g Release 1*, la més actual en el moment de la redacció d'aquests materials) tenint en compte que la resta de SGBD incorporen també tipus de dades similars i, en cas d'haver-hi de treballar, ens caldrà efectuar sempre una ullada a la documentació que cada SGBD facilita.

Cada valor manipulat per *Oracle* correspon a un tipus de dada el qual associa un conjunt de propietats al valor. Les propietats associades a cada tipus de dada provoquen que *Oracle* tracti de forma diferent els valors de diferents tipus de dades. Així, per exemple, es pot sumar valors del tipus de dada `NUMBER`, però no es pot sumar valors del tipus de dada `RAW`.

En el moment de creació d'una taula o *cluster* cal especificar un tipus de dada per a cadascuna de les columnes. En la creació d'una acció o funció emmagatzemada a la base de dades, cal especificar un tipus de dada per a cadascun dels arguments. La correcta assignació del tipus de dada és fonamental, doncs:

Cluster

Conjunt de taules relacionades entre sí.

– D'una banda, els tipus de dades defineixen el domini de valors que cada columna o argument pot contenir. Així, per exemple, les columnes de tipus `DATE` no podran acceptar el valor '30 de febrer' ni el valor 2 ni la cadena 'Hola'.

– Per altra banda, cada valor col·locat en una columna assumeix el tipus de dada de la columna. Per exemple, si inserim la cadena '01-JAN-08' en una columna `DATE`, el valor de tipus cadena passa a ser un valor de tipus `DATE` després de verificar que es pot traduir per una correcta data.

Oracle permet gestionar diferents tipus de dades que podem classificar en:

a) Tipus de dades natius d'*Oracle* anomenats tipus de dades incorporats (*built-in datatypes*).
b) Tipus de dades reconeguts per ANSI i pels productes SQL/DS i DB2 de l'empresa IBM.
c) Tipus de dades definibles per l'usuari.
d) Tipus de dades implementats en els llenguatges C/C++, Java o PL/SQL.

Es convenient conèixer de l'existència dels dos darrers tipus de dades en *Oracle*, però per a l'aprenentatge del llenguatge SQL en tenim prou amb conèixer les possibilitats que faciliten els tipus de dades natius d'*Oracle* i els tipus de dades reconeguts per ANSI.

1.3.1. Tipus de dades incorporats (*built-in datatypes*)

Dins els tipus de dades natius d'*Oracle* podem distingir:

– Tipus de dades per a gestionar informació alfanumèrica.
– Tipus de dades per a gestionar informació numèrica.
– Tipus de dades per a gestionar moments temporals (dates i temps).
– Tipus de dades per a gestionar grans volums d'informació.
– Tipus de dada `ROWID`.

Tipus de dades caràcter

Els tipus de dades caràcter emmagatzemen dades alfanumèriques en el conjunt de caràcters de la base de dades o en el conjunt de caràcters nacional. Aquests tipus són menys restrictius que altres tipus de dades i, en conseqüència, tenen menys propietats. Així, per exemple, les columnes de tipus caràcter poden emmagatzemar valors alfanumèrics

però les columnes de tipus numèric només poden emmagatzemar valors numèrics.

Les dades caràcters s'emmagatzemen en cadenes de bytes corresponents al conjunt de caràcters de la base de dades (*database character set*), el qual s'indica en el moment de creació de la base de dades. *Oracle* suporta conjunts de caràcters d'1 byte i conjunts de caràcters multibyte.

Distinció en *Oracle* entre *Database character set* i *National character set*

L'SGBD *Oracle* utilitza dos conjunts de caràcters en la gestió d'una base de dades:

El conjunt de caràcters de la base de dades (*database character set*) que s'utilitza en:
L'emmagatzematge de dades en columnes SQL de tipus CHAR, VARCHAR2, CLOB i LONG
Els identificadors com noms de taules, noms de columnes i PL/SQL variables
Introducció i emmagatzematge de sentències SQL i codi font PL/SQL.
Aquest conjunt de caràcters determina quins caràcters poden ser utilitzats per anomenar els objectes de la base de dades.

Oracle proporciona diverses possibilitats per a la definició d'aquest conjunt de caràcters i correspon a l'administrador de la base de dades prendre la decisió adequada en funció dels llenguatges que utilitzin els clients que hagin d'establir connexió amb la base de dades. L'opció per defecte depèn de la configuració del sistema operatiu en la que resideix el sistema gestor de bases de dades.

El conjunt de caràcters nacional (*national character set*) que s'utilitza en l'emmagatzematge de les columnes SQL de tipus NCHAR, NVARCHAR2 i NCLOB.

Només hi ha dues possibilitats: AL16UTF16 (per defecte) i UTF8.

Els dos conjunts de caràcters es decideixen en el moment de creació de la base de dades. *Oracle* facilita algun mecanisme per a poder canviar els conjunts de caràcters una vegada la base de dades ja està creada, però és un mecanisme perillós que pot produir l'aparició de dades corruptes.

Oracle proporciona i recomana 4 tipus de dades per gestionar dades alfanumèriques: CHAR, NCHAR, VARCHAR2 i NVARCHAR2. També existeix el tipus LONG, però *Oracle* recomana la substitució de les columnes LONG per columnes LOB.

a) El tipus CHAR[(llargada)][BYTE|CHAR]

Aquest tipus especifica una cadena de longitud fixa (indicada per llargada) i, per tant, *Oracle* assegura que tots els valors emmagatzemats a la columna tenen la longitud especificada. Si s'insereix una cadena de longitud menor, *Oracle* l'omple amb espais en blanc fins la llargada indicada. Si s'intenta inserir una cadena de longitud major, *Oracle* retorna un error.

La llargada mínima i per defecte (no és obligatòria) per a una columna CHAR és 1 byte i la llargada màxima permesa és de 2000 bytes.

La llargada es pot indicar de dues maneres:

– Amb un número, que indica el número de bytes: `CHAR(10)`. Seria equivalent a l'expressió `CHAR(10 BYTE)`.

– Amb un número seguit de la paraula `CHAR`, per indicar la llargada amb caràcters, a partir de la qual *Oracle* calcula la llargada en bytes segons el número de bytes que ocupi un caràcter en el conjunt de caràcters de la base de dades (d'1 a 4): `CHAR(10 CHAR)`.

b) El tipus `NCHAR[(llargada)]`

Aquest tipus de dada és similar al tipus `CHAR` però emmagatzema caràcters `UNICODE`. En aquest cas, la llargada sempre s'indica en caràcters. La llargada mínima i llargada per defecte (no és obligatòria) per a una columna `NCHAR` és 1 byte. La màxima llargada permesa dependrà del conjunt de caràcters nacional i no pot superar els 2000 bytes.

c) El tipus `VARCHAR2(llargada)[BYTE|CHAR]`

Aquest tipus especifica una cadena de longitud variable que pot ser, com a molt, la indicada per `llargada`, valor que és obligatori introduir. *Oracle* emmagatzema, per tant, el valor exacte que indica l'usuari sense afegir espais en blanc. Si s'intenta inserir una cadena de longitud major, *Oracle* retorna un error.

Tipus de dada VARCHAR

En les versions actuals d'*Oracle*, també existeix el tipus VARCHAR que és un sinònim del tipus de dada VARCHAR2. *Oracle* recomana, però, la utilització de VARCHAR2 donat que en el futur, VARCHAR pot esdevenir un nou tipus de dada amb funcionament diferenciat.

El valor mínim per `llargada` és 1 byte o 1 caràcter i el valor màxim és de 4000 bytes o 4000 caràcters.

La llargada es pot indicar de dues maneres:

– Amb un número, que indica el número de bytes: `VARCHAR2(10)`. Seria equivalent a l'expressió `VARCHAR2(10 BYTE)`.

– Amb un número seguit de la paraula `CHAR`, per indicar la llargada amb caràcters, a partir de la qual *Oracle* calcula la llargada en bytes segons el número de bytes que ocupi un caràcter en el conjunt de caràcters de la base de dades (d'1 a 4): `VARCHAR2(10 CHAR)`.

d) El tipus `NVARCHAR2(llargada)`

Aquest tipus de dada és similar al tipus `VARCHAR2` però emmagatzema caràcters `UNICODE`. En aquest cas, la llargada sempre s'indica en caràcters. La llargada mínima per a una columna `NVARCHAR2` és 1 byte. La màxima llargada permesa dependrà del conjunt de caràcters nacional i no pot superar els 4000 bytes.

e) El tipus LONG

Les columnes LONG emmagatzemen textos de longitud variable que poden arribar fins a 2 GB-1 o 2^{31}-1 bytes. Les columnes LONG s'assemblen a les columnes VARCHAR2 però en realitat tenen moltes limitacions en la seva utilització (només hi pot haver una columna LONG per taula, no es poden utilitzar per a filtrar informació, no poden ser indexades,...). La longitud dels valors LONG pot quedar limitada per la memòria disponible a l'ordinador.

Tipus de dades numèriques

Tradicionalment el SGBD *Oracle* havia incorporat un únic tipus de dada numèric natiu amb el qual es pot emmagatzemar qualsevol tipus de dada numèrica: el tipus NUMBER. Aquesta és una característica força diferent de la resta de SGBD. A partir de la versió 10g, però, *Oracle* incorpora dos tipus de dades específics per al tractament de nombres en punt flotant: BINARY_FLOAT i BINARY_DOUBLE.

a) El tipus NUMBER.

El tipus NUMBER permet emmagatzemar nombres en punt fix i nombres en punt flotant.

Representació de nombres reals en punt fix i en punt flotant

La representació d'un nombre real en punt fix consisteix en que el punt decimal ocupa sempre la mateixa posició (punt fix) i aquest fet fixa la potència de la base per la que s'ha de multiplicar cada dígit de la representació. Així, per exemple, en una representació binària de 8 bits, amb 5 bits per a la part entera i 3 per a la part fraccionària, podríem tenir números com:

$10101.101 = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 21.625_{10}$
$10001.010 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} = 17.25_{10}$

Si anomenem E al número de dígits de la part entera i F al nombre de dígits de la part decimal, és clar que no podem representar nombres enters majors o iguals que 2^E ($2^5 = 32$ en l'exemple anterior) ni nombres menors que 2^{-F} ($2^{-3} = 0.125$ en l'exemple anterior) exceptuant el zero.

En canvi, la representació d'un nombre real en punt flotant permet que la representació s'adapti a l'ordre de magnitud del nombre a representar, fet que s'assoleix desplaçant el punt decimal cap a la primera xifra significativa del valor i guardant en un exponent la posició real.

En la representació en punt flotant es distingeix sempre:
la mantissa, que correspon als dígits significatius del valor a representar; té una grandària normalment fixada i limitada, anomenada precisió, i acostuma a estar normalitzada de manera que la part entera consta d'un dígit que és la primera xifra significativa a representar,
la base del sistema de representació,
l'exponent, que indica l'ordre de magnitud de la mantissa.

Així, per exemple, observem la representació dels següents valors en punt flotant de mantissa amb precisió 5, els quals no serien representables, de cap de les maneres, en punt fix (5,3):

$0.02421 = 2.421 \times 10^{-2}$
$422{,}421 = 4.2242 \times 10^{2}$
$422{,}428 = 4.2243 \times 10^{2}$

Els següents valors poden ser emmagatzemats en una columna `NUMBER`:

- Valors positius entre 1×10^{-130} i $9.99\ldots9 \times 10^{125}$ amb 38 dígits significatius com a molt.
- Valors negatius entre -1×10^{-130} i $-9.99\ldots9 \times 10^{125}$ amb 38 dígits significatius com a molt.
- Zero

La definició és `NUMBER[(p[,s)]]` on:

- El paràmetre `p`, entre 1 i 38, és la precisió i indica el màxim nombre de dígits decimals significatius, on el dígit més significatiu és el dígit situat més a l'esquerra que no sigui zero i el dígit menys significatiu és el dígit situat més a la dreta.

- El paràmetre s, entre -84 i 127, és l'**escala** i indica el nombre total de dígits decimals que hi pot haver a la dreta del punt decimal.

El valor per defecte de l'escala és zero (i indica valors enters) i el valor per defecte de la precisió és 38.

Un valor negatiu de l'escala implica l'arrodoniment al número de llocs especificats a l'esquerra del punt decimal. Per exemple, una escala -2 implica arrodoniment a les centenes.

En cas que el valor de l'escala sigui superior a la precisió, *Oracle* interpreta la precisió com el màxim número de dígits significatius (diferents de zero) a la dreta del punt decimal.

Si un valor supera la precisió, *Oracle* retorna un error. Si un valor supera l'escala indicada, *Oracle* l'arrodoneix.

El tipus `NUMBER` també permet emmagatzemar valors reals en punt flotant, és a dir, valors que poden tenir un punt decimal en qualsevol posició des del primer dígit al darrer o no tenir punt decimal i, que a més, poden anar acompanyats d'un exponent (amb la sintaxis `{e|E}[+|-]` precedint el valor numèric de l'exponent.

Oracle emmagatzema els valors `NUMBER` en format decimal de longitud variable. Cada valor és emmagatzemat en notació científica amb 1 byte utilitzat per a emmagatzemar l'exponent i fins a 20 bytes per emmagatzemar la mantissa. Això porta a que el valor resultant tingui la limitació de 38 dígits de precisió. Així, per exemple, el nombre 412 és

Oracle emmagatzema els valors NUMBER en format decimal, és a dir, emprant els dígits 0,1,2,... i 9, fet que garanteix l'exactitud del valor emmagatzemat sempre i quan no s'hagi excedit la màxima precisió.

emmagatzemat en un format similar a 4.12 x 10^2, amb 1 byte per emmagatzemar l'exponent (2) i 2 bytes per emmagatzemar les 3 xifres significatives de la mantissa (4, 1, 2). Els números negatius inclouen també el signe en la seva longitud. Així doncs, *Oracle* pot necessitar d'1 a 22 bytes per a emmagatzemar un valor.

La taula 2 exemplifica valors emmagatzemats en columnes `NUMBER`.

Taula 2. Exemples d'emmagatzematge en columnes `NUMBER`.

Valor numèric	Tipus de dada	Valor emmagatzemat
`7456123.89`	`NUMBER`	`7456123.89`
`7456123.89`	`NUMBER(9)`	`7456124` (s'ha arrodonit)
`7456123.89`	`NUMBER(9,2)`	`7456123.89`
`7456123.89`	`NUMBER(9,1)`	`7456123.9` (s'ha arrodonit)
`7456123.89`	`NUMBER(6)`	Error: Precisió excedida
`7456123.89`	`NUMBER(7,-2)`	`7456100` (s'ha arrodonit)
`7456123.89`	`NUMBER(7,2)`	Error: Precisió excedida
`.01234`	`NUMBER(4,5)`	`.01234`
`.00012`	`NUMBER(4,5)`	`.00012`
`.000127`	`NUMBER(4,5)`	`.00013` (s'ha arrodonit)
`.0000012`	`NUMBER(2,7)`	`.0000012`
`.00000123`	`NUMBER(2,7)`	`.0000012` (s'ha arrodonit)
`1.777e-20`	`NUMBER`	1.777e-20
`1.777e-20`	`NUMBER(5,10)`	`0` (en superar l'escala, s'ha arrodonit)
`1.777e-10`	`NUMBER(5,10)`	`2.0000E-10` (s'ha arrodonit)
`1.777e-9`	`NUMBER(5,10)`	`1.8000E-09` (s'ha arrodonit)
`1.777e-8`	`NUMBER(5,10)`	`1.7800E-08` (s'ha arrodonit)

b) El subtipus `FLOAT` de `NUMBER`

Oracle també facilita el tipus `FLOAT` com a un subtipus de `NUMBER` que pot ser especificat amb o sense precisió (binària). Es tracta d'un `NUMBER` i, per tant, la seva precisió (binària) pot estar entre 1 i 126. No s'hi pot especificar l'escala i *Oracle* la interpreta segons la dada que li arriba. A l'igual que el tipus `NUMBER`, es pot necessitar d'1 a 22 bytes per emmagatzemar un valor.

Per a convertir una precisió binària en decimal, cal multiplicar per *log 2* (és a dir, 0.30103) i per a convertir una precisió decimal en binària cal dividir per *log 2*. Fent aquests càlculs s'observa que la màxima precisió binària (126) coincideix amb la màxima precisió decimal (38).

A diferència del tipus NUMBER, el tipus FLOAT no es queixa si no pot emmagatzemar el valor a causa d'una precisió superior i arrodoneix el valor a la màxima precisió possible.

La taula 3 il·lustra els comportaments dels tipus NUMBER i FLOAT. El tipus FLOAT(5) no pot emmagatzemar més de 5 dígits binaris o, equivalentment, 2 dígits decimals. Per això, quan la precisió supera els 2 dígits decimals, procedeix a arrodonir a 2 dígits decimals.

Taula 3. Comportaments dels tipus NUMBER i FLOAT

Valor numèric	**Tipus NUMBER(5,2)**	**Tipus FLOAT(5)**
1.23	1.23	1.2
7.89	7.89	7.9
12.79	12.79	13
123.45	123.45	120

Oracle utilitza el tipus FLOAT de forma interna quan efectua conversions del tipus de dada ANSI FLOAT. És lícit declarar columnes amb aquest tipus de dada, però *Oracle* ho desaconsella des de que ha incorporat, a partir de la versió 10g, els tipus BINARY_FLOAT i BINARY_DOUBLE.

c) Tipus de dades per a nombres en punt flotant

Els valors reals en punt flotant poden tenir un punt decimal en qualsevol posició des del primer dígit al darrer o no tenir punt decimal i, a més, poden anar acompanyats d'un exponent (amb la sintaxis {e|E}[+|-] precedint el valor numèric de l'exponent.

Oracle facilita, a partir de la versió 10g, dos tipus de dades específics per al tractament de nombres en punt flotant: BINARY_FLOAT i BINARY_DOUBLE. Els valors *Oracle* d'aquests tipus es diferencien dels valors NUMBER en la forma en que estan emmagatzemats internament a la base de dades: els valors NUMBER són emmagatzemats amb precisió decimal (emprant els dígits 0,1,2,... i 9) mentre que els valors en punt flotant s'emmagatzemen amb precisió binària (emprant 0 i 1).

El tipus BINARY_FLOAT permet emmagatzemar nombres en punt flotant de precisió simple (32 bits). Cada valor BINARY_FLOAT requereix 5 bytes.

El tipus BINARY_DOUBLE permet emmagatzemar nombres en punt flotant de precisió doble (62 bits). Cada valor BINARY_DOUBLE requereix 9 bytes.

La taula 4 mostra els diversos valors a emmagatzemar en tipus BINARY.

Taula 4. Valors que pot emmagatzemar els tipus BINARY

Valor	BINARY_FLOAT	BINARY_DOUBLE
Major valor positiu	3.40282E+38	1.79769313486231E+308
Mínim valor positiu	1.17549E-38	2.22507485850720E-308
Menor valor negatiu	-3.40282E+38	-1.79769313486231E+308
Major valor negatiu	-1.17549E-38	-2.22507485850720E-308
Zero		
Infinit	BINARY_FLOAT_INFINITY	BINARY_DOUBLE_INFINITY
-Infinit	-BINARY_FLOAT_INFINITY	-BINARY_DOUBLE_INFINITY
NaN	BINARY_FLOAT_NAN	BINARY_DOUBLA_NAN

El valor NaN (*Not a Number*) és per a indicar valors no vàlids com a BINARY obtinguts, normalment, com a resultats d'operacions.

Tipus de dades per a moments temporals

Oracle proporciona diversos tipus de dades per a gestionar moments temporals, els quals es poden classificar en dos grans grups segons permetin emmagatzemar un moment temporal concret (grup DATETIME) o permetin emmagatzemar un interval entre valors temporals concrets (grup INTERVAL).

Els tipus de dades d'ambdós grups es gestionen per camps (dies, mesos, anys, hores, minuts, segons,...) La taula 5 en mostra els principals.

Taula 5. Alguns dels camps que formen els tipus de dades DATETIME i INTERVAL d'*Oracle*

Camp	Valors vàlids per a tipus del grup DATETIME	Valors vàlids per a tipus del grup INTERVAL
YEAR	-4712 to 9999 (exceptuant any 0)	Qualsevol enter
MONTH	De 01 a 12	De 0 a 11
DAY	De 01 a 31 (limitat pels valors de MONTH i YEAR)	Qualsevol enter
HOUR	De 00 a 23	De 0 a 23
MINUTE	De 00 a 59	De 0 a 59
SECOND	De 00 a 59.9(n), on "9(n)" és la precisió de la fracció de segons. No aplicable pel tipus DATE.	De 0 a 59.9(n), on "9(n)" és la precisió de la fracció de segons

Tenint en compte aquests camps ja podem presentar els diversos tipus de dades dels grups DATETIME i INTERVAL.

1) Tipus de dades del grup DATETIME

Són els tipus de dades per a emmagatzemar moments temporals concrets i hi distingim, entre d'altres, els tipus DATE i TIMESTAMP.

a) Tipus de dada DATE

El tipus de dada DATE emmagatzema informació sobre la data i el temps, la qual és representable en caràcters i en nombres. Per a cada valor DATE, *Oracle* emmagatzema el segle, l'any, el mes, el dia, l'hora, el minut i el segon.

Hi ha diferents maneres d'indicar un valor de tipus DATE:

– Seguint l'especificació ANSI: `DATE 'YYYY-MM-DD'`.

  Per exemple: `DATE '2008-02-29'`.

– Indicant directament una cadena que segueixi l'especificació del paràmetre NLS_DATE_FORMAT actiu a la sessió connectada a la base de dades.

  Així, si NLS_DATE_FORMAT conté el format `'DD/MM/YYYY'`, es pot especificar una data a partir d'una cadena amb el format indicat.

  Per exemple: `'29/02/2008'`.

– També es pot emprar la funció incorporada d'*Oracle* TO_DATE per a introduir una data en el format desitjat a partir d'una cadena.

  Per exemple:

```
TO_DATE('02/29/2008:17:30:45','MM/DD/YYYY:HH24:MI:SS')
```

  S'observa de forma intuïtiva que MM indica el mes, DD indica el dia dins el mes, YYYY indica l'any, HH24 indica l'hora en format de 24 hores, MI indica el minut i SS indica el segon.

Funcions incorporades

Els SGBD incorporen funcions per a facilitar la gestió de les seves dades. En general no són funcions que formin part del llenguatge SQL i, per tant, no hi ha uniformitat entre els diferents SGBD.

En especificar un valor DATE sense indicar els valors corresponents als apartats hora – minut – segon, *Oracle* hi assigna per defecte el valor 12:00:00 AM. Per altra banda, en cas d'utilitzar un valor DATE per a enregistrar una hora – minut – segon sense indicar el dia – mes – any (cosa gens normal), *Oracle* hi assigna per defecte el dia 1 del mes en curs.

Oracle permet efectuar certes operacions amb els tipus de dades DATE. Així, les operacions permeses són:

– Sumar/restar un número enter a una data, que s'interpreta com sumar/restar un número de dies a una data i el resultat és la corresponent data.

– Restar dues dates, que dona per resultat un número enter que indica els dies que separaven les dues dates.

Així mateix *Oracle* facilita un munt de funcions per a la gestió de valors dels tipus de dada `DATE`.

b) Tipus de dada `TIMESTAMP[(precisió_fracció_segons)]`

El tipus de dada `TIMESTAMP` és una extensió del tipus de dada `DATE` que permet emmagatzemar valors de temps molt precisos, permetent indicar la precisió de la fracció de segons.

En efectuar la declaració de la columna es pot indicar la precisió per als segons (número de decimals de segon a emmagatzemar), el qual pot ser un valor de 0 a 9, sent 6 el valor per defecte.

Es pot emprar la funció incorporada d'*Oracle* `TO_TIMESTAMP` per a introduir un valor de tipus `TIMESTAMP` en el format desitjat a partir d'una cadena. Exemples:

– `TO_TIMESTAMP('17:30:45.432','HH24:MI:SS.FF4')`

  Oracle interpreta la data 1 del més en curs (doncs no s'ha indicat data) amb l'hora indicada (15:30:45) i 432 mil·lèsimes de segons.

  S'observa de manera intuïtiva que `FF[n]` indica la precisió de segons fins a `n` decimals.

– `TO_TIMESTAMP('17:30:45.43245','HH24:MI:SS.FF4')`

  Oracle retorna un error doncs s'està indicant una precisió de segons superior a la indicada en la cadena de format.

2) Tipus de dades del grup `INTERVAL`

Són els tipus de dades per a emmagatzemar intervals entre moments temporals concrets i en distingim dos: `INTERVAL YEAR TO MONTH` i `INTERVAL DAY TO SECOND`.

a) Tipus de dada `INTERVAL YEAR [(precisió_anys)] TO MONTH`

Aquest tipus de dada permet emmagatzemar un període de temps en termes d'anys i mesos. El valor optatiu `precisió_anys` permet indicar el número de dígits per a representar els anys, sent 2 el valor per defecte.

Es pot emprar la funció incorporada d'*Oracle* `TO_YMINTERVAL` per a introduir un valor de tipus `INTERVAL YEAR TO MONTH` a partir d'una cadena que ha de tenir el format `'YY-MM'`.

Així, per exemple, `TO_YMINTERVAL ('01-02')`, indica l'interval corresponent a 1 any i 2 mesos.

b) Tipus de dada `INTERVAL DAY [(precisió_dies)] TO SECOND [(precisió_fracció_segons)]`

Aquest tipus de dada permet emmagatzemar un període de temps en termes de dies, hores, minuts i segons. El valor optatiu `precisió_dies` permet indicar el número de dígits per a representar els anys, sent 2 el valor per defecte. El valor optatiu `precisió_fracció_segons` permet indicar la precisió de la fracció de segons (número de decimals de segon a emmagatzemar), el qual pot ser un valor de 0 a 9, sent 6 el valor per defecte.

Es pot emprar la funció incorporada d'*Oracle* `TO_DSINTERVAL` per a introduir un valor de tipus `INTERVAL DAY TO SECOND` a partir d'una cadena que ha de tenir el format `'nnn HH:MI:SS'`.

Així, per exemple, `TO_DSINTERVAL ('100 10:30:25.425')`, indica l'interval corresponent a 100 dies, 10 hores, 30 minuts, i 25.425 segons.

3) Operacions aritmètiques entre valors `DATETIME` i valors `INTERVAL`

Oracle permet efectuar certes operacions entre els tipus de dades `DATETIME` i `INTERVAL`. Així, les operacions permeses són:

- Sumar/restar un interval a una data, que dona per resultat una nova data.
- Sumar una data a un interval, que dona per resultat una nova data.
- Restar dues dates que no siguin de tipus `DATE`, que dona per resultat un interval. La resta de dues dates de tipus `DATE` ja sabem que retorna un enter que indica el número de dies entre les dates.
- Sumar/restar un interval a un interval, que dona per resultat un nou interval.
- Multiplicar/Dividir un interval per un valor numèric, que dona per resultat un nou interval.

– Multiplicar un valor numèric per un interval, que dona un nou interval.

Tipus de dades per a grans volums d'informació binària

Oracle facilita diferents tipus de dades per a gestionar grans volums d'informació (textual i/o multimèdia): `RAW`, `LONG RAW` i `LOB`.

1) Tipus de dades `RAW` i `LONG RAW`

Els tipus de dades `RAW` i `LONG RAW` emmagatzemen dades que no són interpretades per *Oracle* (i, per tant, no sofreixen cap tipus de conversió quan es traspassen entre diferents sistemes). Aquests tipus de dades es tracten de forma binària.

Oracle recomana que les columnes `LONG RAW` siguin reconvertides a columnes `LOB` (`BLOB`) les quals estan subjectes a menys restriccions que les columnes `LONG RAW`.

El tipus `RAW` és un tipus de dada de longitud variable similar a `VARCHAR2` però cal tenir en compte que en les sessions connectades a la base de dades des de sistemes que no tinguin el mateix conjunt de caràcters (paràmetre `NLS_LANGUAGE` de la sessió) no s'efectua cap tipus de conversió, per la qual cosa es poden produir resultats inesperats.

2) Tipus de dades `LOB`

Oracle facilita els tipus de dades `LOB` (*Large Object*) per emmagatzemar grans volums d'informació fins a 4 GB. Hi ha 4 tipus de dades `LOB`: `BLOB`, `CLOB` i `NCLOB` (emmagatzemades dins la base de dades) i `BFILE` (emmagatzemades fora la base de dades).

a) Tipus de dades `BFILE`

El tipus de dada `BFILE` activa l'accés a un fitxer binari que està emmagatzemat en el sistema de fitxers extern a la base de dades. Un valor `BFILE` emmagatzema un localitzador que actua com un punter a un fitxer binari en el sistema de fitxers del sistema operatiu. El localitzador manté el nom del directori i el nom del fitxer.

Els tipus de dades BFILE no es tenen en compte en les transaccions i no són recuperables.

Transacció

Conjunt d'operacions sobre una base de dades pel que es pot assegurar que s'executen totes les operacions amb èxit o no se n'executa cap.

Oracle facilita la funció `BFILENAME` per a permetre el canvi del nom del fitxer i el camí d'una dada `BFILE` sense afectar la taula que conté aquesta informació.

L'administrador de la base de dades ha d'assegurar que el fitxer existeix i que els processos d'*Oracle* tenen permisos de lectura sobre el fitxer. El tipus de dada `BFILE` activa suport de només lectura per fitxers binaris. No és possible modificar o replicar un fitxer del sistema accedit per `BFILE`. *Oracle* facilita interfícies per accedir a les dades del fitxer.

b) Tipus de dades `BLOB – CLOB - NCLOB`

Els tres tipus de dades emmagatzemen grans volums d'informació dins la pròpia base de dades. Els tres tipus de dades participen a les transaccions. Es diferencien en:

- El tipus `BLOB` emmagatzema informació binària.
- El tipus `CLOB` permet emmagatzemar dades caràcter utilitzant el conjunt de caràcters de la base de dades (*database character set*).
- El tipus `NCLOB` emmagatzema dades `UNICODE` utilitzant el conjunt de caràcters nacional (*national character set*).

Tipus de dada ROWID

Cada fila en una base de dades *Oracle* té una adreça i aquesta resideix en una pseudocolumna anomenada `ROWID`.

Els valors d'aquesta columna són cadenes que representen l'adreça de cada fila. Aquestes cadenes són de tipus `ROWID`. Es pot crear taules amb columnes de tipus `ROWID`, però *Oracle* no garanteix que els valors emmagatzemats en tals columnes siguin valors `ROWID` vàlids.

1.3.2. Tipus de dades reconeguts per ANSI i pels productes SQL/DS i DB2 d'IBM

Les instruccions SQL que permeten la creació de taules i *clusters* poden utilitzar els tipus de dades ANSI i els tipus de dades reconeguts pels productes SQL/DS i DB2 d'IBM. *Oracle* reconeix els noms dels tipus de dades ANSI i IBM que difereixen del nom dels tipus de dades natius d'*Oracle* i enregistra el nom del tipus de dada de la columna, però emmagatzema la dada en un tipus de dada natiu d'*Oracle* segons les conversions de les taules 6 i 7.

Taula 6. Conversió dels tipus de dada ANSI SQL a tipus de dada *Oracle*

Tipus de dada ANSI SQL	**Tipus de dada *Oracle***
CHARACTER(n) CHAR(n)	CHAR(n)
CHARACTER VARYING(n) CHAR VARYING(n)	VARCHAR(n)
NATIONAL CHARACTER(n) NATIONAL CHAR(n) NCHAR(n)	NCHAR(n)
NATIONAL CHARACTER VARYING(n) NATIONAL CHAR VARYING(n) NCHAR VARYING(n)	NVARCHAR2(n)
NUMERIC(p,s) DECIMAL(p,s)	NUMBER(p,s)
INTEGER INT SMALLINT	NUMBER(38)
FLOAT(b) DOUBLE PRECISION REAL	NUMBER

Els tipus NUMERIC i DECIMAL només poden especificar nombres en punt fix. Per tant, el valor per defecte de s és 0.

El tipus FLOAT és per a nombres en punt flotant amb precisió binària b. La precisió per defecte per aquet tipus de dada és 126 binària o 38 decimal.

El tipus de dada DOUBLE PRECISION és per a nombres en punt flotant amb precisió 126 binària

El tipus de dada REAL és per a nombres en punt flotant amb precisió 63 binària o 18 decimal.

Taula 7. Conversió dels tipus de dada SQL/DS o DB2 d'IBM a tipus de dada *Oracle*

Tipus de dada SQL/DS o DB2	**Tipus de dada *Oracle***
CHARACTER(n)	CHAR(n)
VARCHAR(n)	VARCHAR(n)
LONG VARCHAR(n)	LONG
DECIMAL(p,s)	NUMBER(p,s)
INTEGER SMALLINT	NUMBER(38)
FLOAT(b)	NUMBER

Els tipus DECIMAL només pot especificar nombres en punt fix. Per tant, el valor per defecte de s és 0.

El tipus FLOAT és per a nombres en punt flotant amb precisió binària b. La precisió per defecte per aquet tipus de dada és 126 binària o 38 decimal.

1.4. Consultes simples

Ara que ja som coneixedors dels diferents tipus de dades que ens podem trobar emmagatzemats en una base de dades (nosaltres ens hem centrat en *Oracle* 11g, però en la resta de SGBD és similar), ja estem en condicions d'iniciar l'explotació de la base de dades, és a dir, començar la gestió de les dades. Evidentment, per poder gestionar dades, cal prèviament haver definit les taules que han de contenir les dades i per a

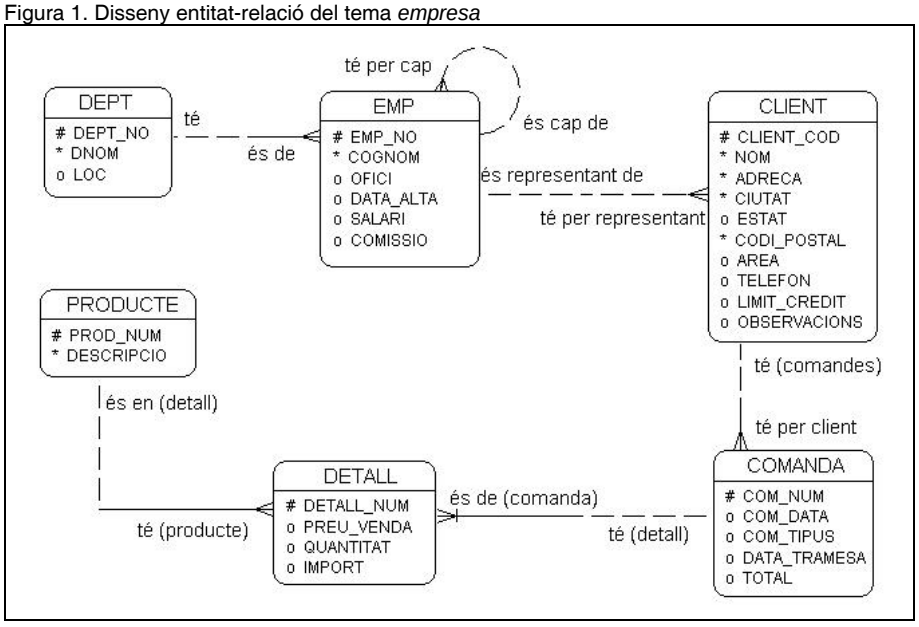
poder consultar dades cal abans haver-les introduït. L'aprenentatge del llenguatge SQL s'efectua, però, en sentit invers; és a dir, anem a començar per a conèixer les possibilitats de consulta de dades sobre taules ja creades i amb dades ja introduïdes. Necessitem, però, conèixer l'estructura de les taules a gestionar i les relacions existents entre elles.

Les taules que anem a gestionar formen part de dos dissenys diferents, és a dir, són taules de temes disjunts. Vegem-les.

a) Temàtica *empresa*.

La figura 1 mostra el disseny entitat-relació del tema *empresa* les dades del qual gestionarem al llarg d'aquest crèdit. Aquest disseny està efectuat amb l'eina d'anàlisi i disseny *Designer* d'*Oracle*.

Figura 1. Disseny entitat-relació del tema *empresa*



Els dissenys entitat-relació, altrament anomenats dissenys CHEN, s'estudien a fons en el crèdit "Anàlisi i disseny de bases de dades".

Tinguis present que la figura 1 mostra el diagrama entitat-relació, altrament anomenat disseny CHEN, i no pas el diagrama relacional. Fent una ràpida ullada a aquest disseny, hem d'interpretar:

- Tenim sis entitats diferents: departaments (`DEPT`), empleats (`EMP`), clients (`CLIENT`), productes (`PRODUCTE`), comandes (`COMANDA`) i detall de les comandes (`DETALL`).

- Entre les sis entitats s'estableixen relacions:

  – Entre `DEPT` i `EMP` (relació `1:N`) doncs un empleat és assignat obligatòriament a un departament i un departament té assignats zero o varis empleats.

  - Entre `EMP` i `EMP` (relació reflexiva `1:N`) doncs un empleat pot tenir per cap un altre empleat de l'empresa i un empleat pot ser cap de zero o varis empleats.

  - Entre `EMP` i `CLIENT` (relació `1:N`) doncs un empleat pot ser el representant de zero o varis clients i un client pot tenir assignat un representant que ha de ser un empleat de l'empresa.

  - Entre `CLIENT` i `COMANDA` (relació `1:N`) doncs un client pot tenir zero o vàries comandes a l'empresa i una comanda és obligatòriament d'un client.

  - Entre `COMANDA` i `DETALL` (relació forta-feble `1:N`) doncs la comanda està formada per diverses línies, anomenades detall de la comanda.

  - Entre `DETALL` i `PRODUCTE` (relació `N:1`) doncs cada línia de detall correspon a un producte.

- En ocasions, algun alumne no expert en dissenys entitat-relació es pregunta el per què de l'entitat `DETALL` i pensa que no hi hauria de ser, substituint-la per una relació `N:N` entre les entitats `COMANDA` i `PRODUCTE`. Gran error!

  L'error resideix en que en una relació `N:N` entre `COMANDA` i `PRODUCTE`, un mateix producte no pot estar més d'una vegada a la mateixa comanda. En certs negocis això pot ser una decisió encertada, però no és sempre així, doncs poden donar-se situacions similars a:

  - Per raons comercials o d'altra índole, en una mateixa comanda hi ha certa quantitat d'un producte amb un preu i descomptes determinats i una altra quantitat del mateix producte amb unes condicions de venda (preu i/o descomptes) diferents.

  - Pot ser que una quantitat de producte calgui lliurar-la en una data i una altra quantitat del mateix producte en una altra data. En aquesta situació, la data de tramesa hauria de residir a cada línia de detall.

La corresponent traducció al model relacional és:

DEPT (#dept_no, dnom, loc$_{(VN)}$)
EMP (#emp_no, cognom, ofici$_{(VN)}$, cap$_{(VN)}$, data_alta$_{(VN)}$, salari$_{(VN)}$, comissió$_{(VN)}$, dept_no$_{DEPT}$)
CLIENT (#client_cod, nom, adreça, ciutat, estat$_{(VN)}$, codi_postal, àrea$_{(VN)}$, telèfon$_{(VN)}$, repr_cod$_{(VN)\ EMP}$, límit_crèdit$_{(VN)}$, observacions$_{(VN)}$)

PRODUCTE (#prod_num, descripció)
COMANDA (#com_num, com_data$_{(VN)}$, com_tipus$_{(VN)}$, client_cod$_{CLIENT}$, data_tramesa$_{(VN)}$, total$_{(VN)}$)
DETALL (#com_num$_{COMANDA}$, #detall_num, prod_num$_{PRODUCTE}$, preu_venda$_{(VN)}$, quantitat$_{(VN)}$, import$_{(VN)}$)

La implementació d'aquest model relacional en *Oracle* ha provocat les següents taules:

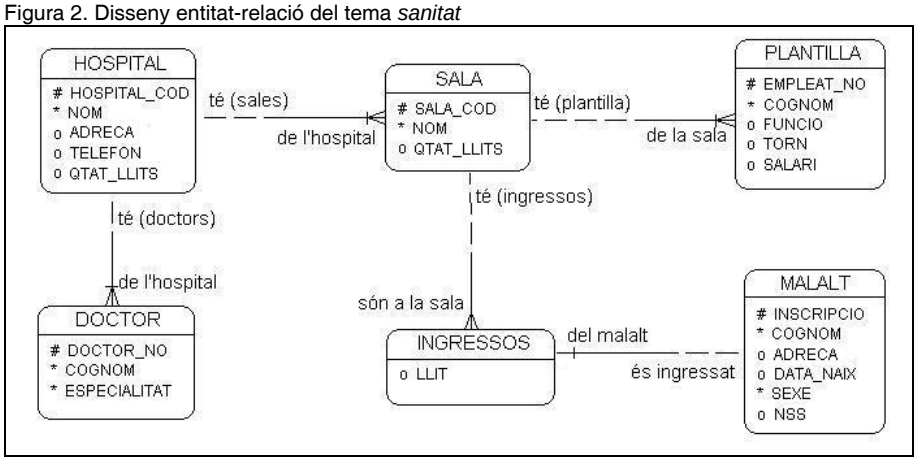
```
>> Taula DEPT, que conté els departaments de l'empresa

Nom          Nul?     Tipus         Descripció
------------ -------- ------------- ------------------------------------------------------
DEPT_NO      NOT NULL NUMBER(2)     Número de departament de l'empresa
DNOM         NOT NULL VARCHAR2(14)  Descripció del departament
LOC                   VARCHAR2(14)  Localitat del departament

>> Taula EMP, que conté els empleats de l'empresa

Nom          Nul?     Tipus         Descripció
------------ -------- ------------- ------------------------------------------------------
EMP_NO       NOT NULL NUMBER(4)     Número d'empleat de l'empresa
COGNOM       NOT NULL VARCHAR2(10)  Cognom de l'empleat
OFICI                 VARCHAR2(10)  Ofici de l'empleat
CAP                   NUMBER(4)     Número de l'empleat que és el cap directe (taula EMP)
DATA_ALTA             DATE          Data d'alta
SALARI                NUMBER(10)    Salari mensual
COMISSIO              NUMBER(10)    Import de les comissions
DEPT_NO      NOT NULL NUMBER(2)     Departament al qual pertany (taula DEPT)

>> Taula CLIENT, que conté els clients de l'empresa

Nom          Nul?     Tipus         Descripció
------------ -------- ------------- ------------------------------------------------------
CLIENT_COD   NOT NULL NUMBER(6)     Codi de client
NOM          NOT NULL VARCHAR2(45)  Nom del client
ADRECA       NOT NULL VARCHAR2(40)  Adreça del client
CIUTAT       NOT NULL VARCHAR2(30)  Ciutat del client
ESTAT                 VARCHAR2(2)   País del client
CODI_POSTAL  NOT NULL VARCHAR2(9)   Codi postal del client
AREA                  NUMBER(3)     Àrea telefònica
TELEFON               VARCHAR2(9)   Telèfon del client
REPR_COD              NUMBER(4)     Codi del representant del client
                                    És un dels empleats de l'empresa (taula EMP)
LIMIT_CREDIT          NUMBER(9,2)   Límit de crèdit que disposa el client
OBSERVACIONS          LONG          Observacions

>> Taula PRODUCTE, que conté els productes a vendre

Nom          Nul?     Tipus         Descripció
------------ -------- ------------- ------------------------------------------------------
PROD_NUM     NOT NULL NUMBER(6)     Codi de producte
DESCRIPCIO   NOT NULL VARCHAR2(30)  Descripció del producte

>> Taula COMANDA, que conté les comandes de venda

Nom          Nul?     Tipus         Descripció
------------ -------- ------------- ------------------------------------------------------
```

```
COM_NUM       NOT NULL NUMBER(4)     Número de comanda de venda
COM_DATA               DATE          Data de la comanda de venda
COM_TIPUS              VARCHAR2(1)   Tipus de comanda – Valors vàlids: A, B, C
CLIENT_COD    NOT NULL NUMBER(6)     Codi del client que efectua la comanda (taula CLIENT)
DATA_TRAMESA           DATE          Data d'enviament de la comanda
TOTAL                  NUMBER(8,2)   Import total de la comanda

>> Taula DETALL, que conté el detall de les comandes de venda

Nom           Nul?     Tipus         Descripció
------------ -------- ------------- ------------------------------------------------------
COM_NUM       NOT NULL NUMBER(4)     Número de comanda (taula COMANDA)
DETALL_NUM    NOT NULL NUMBER(4)     Número de línia per cada comanda
PROD_NUM      NOT NULL NUMBER(6)     Codi del producte de la línia (taula PRODUCTE)
PREU_VENDA             NUMBER(8,2)   Preu de venda del producte
QUANTITAT              NUMBER(8)     Quantitat de producte a vendre
IMPORT                 NUMBER(8,2)   Import total de la línia
```

b) Temàtica *sanitat*.

La figura 2 mostra el disseny entitat-relació del tema *sanitat* les dades del qual també gestionarem al llarg d'aquest crèdit.

Figura 2. Disseny entitat-relació del tema *sanitat*



Fent una ràpida ullada a aquest disseny, hem d'interpretar:

- Tenim sis entitats diferents: hospitals (`HOSPITAL`), sales dels hospitals (`SALA`), doctors dels hospitals (`DOCTOR`), empleats de les sales dels hospitals (`PLANTILLA`), malalts (`MALALT`) i malalts ingressats actualment (`INGRESSOS`).

- Entre les sis entitats s'estableixen relacions:

  – Entre `HOSPITAL` i `SALA` (relació forta-feble `1:N`) doncs les sales s'identifiquen amb un codi de sala dins cada hospital; és a dir, podem tenir una sala identificada amb el codi 1 a l'hospital X i una sala identificada també amb el codi 1 a l'hospital Y.

  - Entre HOSPITAL i DOCTOR (relació forta-feble 1:N) doncs els doctors s'identifiquen amb un codi de doctor dins cada hospital; és a dir, podem tenir un doctor identificat amb el codi 10 a l'hospital X i un doctor identificat també amb el codi 10 a l'hospital Y.

  - Entre SALA i PLANTILLA (relació forta-feble 1:N) doncs els empleats s'identifiquen amb un codi dins de cada sala; és a dir, podem tenir un empleat identificat amb el codi 55 a la sala 10 de l'hospital X i un empleat identificat també amb el codi 55 en una altra sala de qualsevol hospital.

  - Entre MALALT i INGRESSOS (relació forta-feble 1:1) doncs un malalt pot o no estar ingressat.

  - Entre SALA i INGRESSOS (relació 1:N) doncs en una sala hi pot haver zero o varis malalts ingressats i un malalt ingressat només ho pot estar en una única sala.

- Ben segur no és el millor disseny per a una correcta gestió d'hospitals, però ens interessa mantenir aquest disseny per a les possibilitats que ens donarà de cara a l'aprenentatge del llenguatge SQL. Aprofitem, però, l'ocasió per a comentar els punts foscos en el disseny:

  - Potser no és massa normal que els empleats d'un hospital s'identifiquin dins de cada sala. És a dir, en el disseny, l'entitat PLANTILLA és feble de l'entitat SALA i, potser, seria més lògic que fos feble de l'entitat HOSPITAL de manera similar a l'entitat DOCTOR.

  - Per a poder gestionar els pacients (MALALT) que actualment estan ingressats, és necessari establir una relació entre MALALT i SALA, la qual seria d'ordre N:1 doncs en una sala hi pot haver varis pacients ingressats i un pacient, si està ingressat, ho està en una sala. La corresponent traducció al model relacional provocaria:

```
SALA (#hospital_cod, #sala_cod, nom, qtat_llits(VN))
        HOSPITAL
MALALT (#inscripció, cognom, adreça(VN), data_naix(VN), sexe, nss(VN),
        hosp_ingres(VN), sala_ingres(VN))
        **************SALA*****************
```

    Fixem-nos que la relació (taula) MALALT conté la parella d'atributs (hosp_ingres, sala_ingres) que conjuntament són clau forana de la relació (taula) SALA i que poden tenir valors nuls (VN) donat que un pacient no té per què estar ingressat. Si pensem una mica en la gestió real d'aquestes taules, ens trobarem que la taula

MALALT normalment contindrà moltes files i que, per sort per als pacients, moltes d'elles tindran vuits els camps hosp_ingres i sala_ingres, ja que del total de pacients que passen per un hospital, un conjunt molt petit hi està ingressat en un moment donat. Això pot arribar a provocar una greu pèrdua d'espai a la base de dades.

En aquestes situacions és lícit pensar en una entitat que aglutini els pacients que estan ingressats actualment (INGRESSOS), la qual ha de ser feble de l'entitat que engloba tots els pacients (MALALT). Aquesta és l'opció adoptada en aquest disseny.

– També seria adequat disposar d'una entitat que aglutinés les diferents especialitats mèdiques existents, de manera que poguéssim establir una relació entre aquesta entitat i l'entitat DOCTOR. No és el cas i, per tant, l'especialitat de cada doctor s'introdueix com un valor alfanumèric.

– Així mateix, de forma similar, seria adequat disposar d'una entitat que aglutinés les diferents funcions que pot desenvolupar el personal de la plantilla, de manera que poguéssim establir una relació entre aquesta entitat i l'entitat PLANTILLA. Tampoc és el cas i, per tant, la funció que cada empleat desenvolupa s'introdueix com un valor alfanumèric.

La corresponent traducció al model relacional és:

```
HOSPITAL (#hospital_cod, nom, adreça(VN), telèfon(VN), qtat_llits(VN))
SALA (#hospital_cod, #sala_cod, nom, qtat_llits(VN))
        HOSPITAL
PLANTILLA (#hospital_cod, #sala_cod, #empleat_no, cognom, funció(VN), torn(VN), salari(VN))
            **********SALA*********
MALALT (#inscripció, cognom, adreça(VN), data_naix(VN), sexe, nss(VN))
INGRESSOS (#inscripció, hospital_cod, sala_cod, llit(VN))
             MALALT     *********SALA**********
DOCTOR (#hospital_cod, #doctor_no, cognom, especialitat)
          HOSPITAL
```

La implementació d'aquest model relacional en *Oracle* ha provocat les següents taules:

```
 >> Taula HOSPITAL, que conté l'enumeració dels hospitals

Nom          Nul?     Tipus         Descripció
------------ -------- ------------- ------------------------------------------------------
HOSPITAL_COD NOT NULL NUMBER(2)     Codi de l'hospital
NOM          NOT NULL VARCHAR2(10)  Nom de l'hospital
ADRECA                VARCHAR2(20)  Adreça de l'hospital
TELEFON               VARCHAR2(8)   Telèfon de l'hospital
NUM_LLITS             NUMBER(3)     Número de llits de l'hospital
```

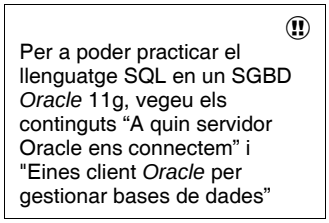
```
>> Taula SALA, que conté les sales de cada hospital

Nom          Nul?     Tipus         Descripció
------------ -------- ------------- ------------------------------------------------------
HOSPITAL_COD NOT NULL NUMBER(2)     Codi de l'hospital (taula HOSPITAL)
SALA_COD     NOT NULL NUMBER(2)     Codi de la sala dins cada hospital
NOM          NOT NULL VARCHAR2(20)  Nom de la sala
NUM_LLITS             NUMBER(3)     Número de llits de la sala

>> Taula DOCTOR, que conté els doctors dels diferents hospitals

Nom          Nul?     Tipus         Descripció
------------ -------- ------------- ------------------------------------------------------
HOSPITAL_COD NOT NULL NUMBER(2)     Codi de l'hospital (taula HOSPITAL)
DOCTOR_NO    NOT NULL NUMBER(3)     Codi de doctor dins cada hospital
COGNOM       NOT NULL VARCHAR2(13)  Cognom del doctor
ESPECIALITAT NOT NULL VARCHAR2(16)  Especialitat del doctor

>> Taula PLANTILLA, que conté els treballadors no doctors de les sales dels hospitals

Nom          Nul?     Tipus         Descripció
------------ -------- ------------- ------------------------------------------------------
HOSPITAL_COD NOT NULL NUMBER(2)     Codi de l'hospital
SALA_COD     NOT NULL NUMBER(2)     Codi de la sala dins cada hospital
                                    La parella (HOSPITAL_COD, SALA_COD) és clau forana de
                                    la taula SALA
EMPLEAT_NO   NOT NULL NUMBER(4)     Codi de l'empleat (independent d'hospital i sala)
COGNOM       NOT NULL VARCHAR2(15)  Cognom de l'empleat
FUNCIO                VARCHAR2(10)  Tasca de l'empleat
TORN                  VARCHAR2(1)   Torn de l'empleat
                                    Valors possibles: (M)atí – (T)arda – (N)it
SALARI                NUMBER(10)    Salari anual de l'empleat

>> Taula MALALT, que conté els malalts

Nom          Nul?     Tipus         Descripció
------------ -------- ------------- ------------------------------------------------------
INSCRIPCIO   NOT NULL NUMBER(5)     Identificació del malalt
COGNOM       NOT NULL VARCHAR2(15)  Cognom del malalt
ADRECA                VARCHAR2(20)  Adreça del malalt
DATA_NAC              DATE          Data de naixement del malalt
SEXE         NOT NULL VARCHAR2(1)   Sexe del malalt
                                    Valors possibles: (H)ome – (D)ona
NSS                   NUMBER(9)     Número de Seguretat Social del malalt

>> Taula INGRESSOS, que conté els malalts ingressats en els hospitals

Nom          Nul?     Tipus         Descripció
------------ -------- ------------- ------------------------------------------------------
INSCRIPCIO   NOT NULL NUMBER(5)     Codi de malalt
HOSPITAL_COD NOT NULL NUMBER(2)     Codi d'hospital
SALA_COD     NOT NULL NUMBER(2)     Codi de sala d'hospital
LLIT                  NUMBER(4)     Número de llit que ocupa dins la sala
```

Bé, doncs ja estem en condicions d'introduir-nos en l'aprenentatge de les instruccions de consulta, les quals practicarem en les taules dels temes empresa i sanitat presentats.

Un parell de comentaris inicials:

- Les instruccions SQL finalitzen, obligatòriament, amb un punt i coma.

- Es pot intercalar en mig de les sentències SQL, comentaris tipus llenguatge C, és a dir, entre els símbols `/*` i `*/`.



Totes les consultes en el llenguatge SQL són realitzades amb una única sentència, anomenada `SELECT`, que es pot utilitzar amb diferents nivells de complexitat.

Com el seu nom indica, aquesta sentència permet **seleccionar** allò que l'usuari demana, el qual no ha d'indicar on anar-ho a cercar ni com cercar-ho.

La sentència `SELECT` es composa de diferents apartats que s'acostumen a anomenar clàusules. Dos d'aquests apartats són sempre obligatoris i són els primers que anem a presentar. La resta de clàusules cal utilitzar-les segons els resultats a obtenir.

1.4.1. Clàusules `select` i `from`.

La sintaxis més simple de la sentència `SELECT` utilitza aquestes dues clàusules de forma obligatòria:

```
select <expressió/columna>, <expressió/columna>,...
from <taula>, <taula>,...;
```

La clàusula `select` permet escollir columnes i/o valors (resultats de les expressions) derivats d'elles. En terminologia d'àlgebra relacional es correspon amb l'operació **projecció**.

La clàusula `from` permet especificar les taules on cal anar a cercar les columnes o sobre les que es calcularan els valors resultats de les expressions.

Una sentència SQL pot escriure's en una única línia, però per a fer més llegible la sentència s'acostumen a utilitzar diferents línies per les diferents clàusules.

Exemple d'utilització simple de les clàusules `select` i `from`.

Es desitja, en el tema *empresa*, mostrar els codis, cognoms i oficis dels empleats.

Aquest és un clar exemple de les consultes més simples: cal indicar les columnes a visualitzar i la taula d'on visualitzar-les. La sentència és:

```
select emp_no, cognom, ofici
from emp;
```

El resultat que s'obté, via consola *SQL*Plus* o via *SQL Developer*, és:

```
EMP_NO               COGNOM     OFICI
-------------------- ---------- ----------
7369                 SÁNCHEZ    EMPLEAT
7499                 ARROYO     VENEDOR
7521                 SALA       VENEDOR
7566                 JIMÉNEZ    DIRECTOR
7654                 MARTÍN     VENEDOR
7698                 NEGRO      DIRECTOR
7782                 CEREZO     DIRECTOR
7788                 GIL        ANALISTA
7839                 REY        PRESIDENT
7844                 TOVAR      VENEDOR
7876                 ALONSO     EMPLEAT
7900                 JIMENO     EMPLEAT
7902                 FERNÁNDEZ  ANALISTA
7934                 MUÑOZ      EMPLEAT

14 rows selected
```

Exemple d'utilització d'expressions en la clàusula `select`.

Es desitja, en el tema *empresa*, mostrar els codis, cognoms i salari anual dels empleats.

Com que coneixem que a la taula `EMP` hi consta el salari mensual de cada empleat, sabem calcular via producte pel número de pagues mensuals en un any (12, 14, 15,...?) el seu salari anual. Suposarem que l'empleat té 14 pagues mensuals, com la majoria dels mortals! Per tant, en aquest cas, alguna de les columnes a visualitzar és el resultat d'una expressió:

```
select emp_no, cognom, salari*14
from emp;
```

El resultat que s'obté, via consola *SQL*Plus* o via *SQL Developer*, és:

```
EMP_NO               COGNOM     SALARI*14
-------------------- ---------- ----------
7369                 SÁNCHEZ    1456000
7499                 ARROYO     2912000
7521                 SALA       2275000
7566                 JIMÉNEZ    5414500
7654                 MARTÍN     2275000
7698                 NEGRO      5187000
7782                 CEREZO     4459000
7788                 GIL        5460000
7839                 REY        9100000
7844                 TOVAR      2730000
7876                 ALONSO     2002000
7900                 JIMENO     1729000
7902                 FERNÁNDEZ  5460000
7934                 MUÑOZ      2366000

14 rows selected
```

Observem que el llenguatge SQL utilitza els noms reals de les columnes com a títols en la presentació del resultat i, en cas de columnes que corresponguin a expressions, ens mostra l'expressió com a títol.

El llenguatge SQL permet donar un nom alternatiu (àlies) a cada columna. Per aconseguir-ho, es pot emprar la següent sintaxis: ‼

```
select <expressió/columna> [as àlies], <expressió/columna> [as àlies],...
from <taula>, <taula>,...;
```

Tinguis en compte que:

- Si l'àlies és format per vàries paraules, cal tancar-lo entre cometes dobles.

- Hi ha alguns SGBD que permeten la no utilització de la partícula `as` (com *Oracle* i *MySQL*) però en d'altres és obligatòria (com *MsAccess*)

Exemple d'utilització d'àlies en la clàusula `select`.

Es desitja, en el tema *empresa*, mostrar els codis, cognoms i salari anual dels empleats.

La instrucció per assolir l'objectiu pot ser, amb la utilització d'àlies:

```
select emp_no, cognom, salari*14 as "Salari Anual"
from emp;
```

El resultat que s'obté en aquest cas és:

```
EMP_NO                 COGNOM     Salari Anual
---------------------- ---------- ----------------------
7369                   SÁNCHEZ    1456000
7499                   ARROYO     2912000
7521                   SALA       2275000
7566                   JIMÉNEZ    5414500
7654                   MARTÍN     2275000
7698                   NEGRO      5187000
7782                   CEREZO     4459000
7788                   GIL        5460000
7839                   REY        9100000
7844                   TOVAR      2730000
7876                   ALONSO     2002000
7900                   JIMENO     1729000
7902                   FERNÁNDEZ  5460000
7934                   MUÑOZ      2366000

14 rows selected
```

El llenguatge SQL facilita una forma senzilla de mostrar totes les columnes de les taules seleccionades en la clàusula `from` (perdent la possibilitat d'emprar un àlies) i consisteix en utilitzar un asterisc en la clàusula `select`. (!!)

Tot i que disposem de l'asterisc per a visualitzar totes les columnes de les taules de la clàusula `from`, en ocasions ens interessarà conèixer les columnes d'una taula per a dissenyar una sentència `SELECT` adequada a les necessitats i no utilitzar l'asterisc per a visualitzar totes les columnes.

Les consoles dels SGBD acostumen a facilitar mecanismes per a visualitzar un breu descriptor d'una taula. En *Oracle* (i també *MySQL*), disposem del comandament `desc` (no és sentència del llenguatge SQL) per a tal menester. Cal emprar-la acompanyant el nom de la taula.

Exemple d'utilització d'asterisc en la clàusula `select` i d'obtenció del descriptor d'una taula

Se'ns demana de mostrar, en el tema *empresa*, tota la informació existent a la taula que conté els departaments.

La instrucció que ens permet assolir l'objectiu és:

```
select * from dept;
```

I obtenim el resultat esperat:

```
DEPT_NO    DNOM           LOC
---------- -------------- --------------
10         COMPTABILITAT  SEVILLA
20         INVESTIGACIÓ   MADRID
30         VENDES         BARCELONA
40         PRODUCCIÓ      BILBAO
```

Observem que la instrucció `SELECT` amb asterisc ens mostra les dades de totes les columnes de la taula. Si únicament necessitem conèixer les columnes que formen la taula (i llurs característiques bàsiques), podem obtenir-ne el descriptor:

```
SQL> desc dept;

Nom                                       Nul?     Tipus
----------------------------------------- -------- ------------
DEPT_NO                                   NOT NULL NUMBER(2)
DNOM                                      NOT NULL VARCHAR2(14)
LOC                                               VARCHAR2(14)
```

Suposem que estem connectats amb el SGBD *Oracle* en un esquema que conté les taules corresponents al tema *empresa* i que necessitem accedir a taules d'un esquema que conté les taules corresponents a l'esquema *sanitat*. Podem aconseguir-ho?

La clàusula `from` pot fer referència a taules d'un altre esquema. En tal situació cal notar la taula com: `<nom_esquema>.<nom_taula>`. ‼

En SGBD on no hi ha el concepte d'esquema però sí de diverses bases de dades gestionades per la mateixa instància de l'SGBD (*MySQL*, *SQLServer*, *PostgreSQL*), s'acostuma a poder fer referència a taules d'una altra base de dades com: `<nom_base_de_dades>.<nom_taula>`.

L'accés a objectes d'altres esquemes (bases de dades) per a un usuari connectat a un esquema, només és possible si té concedits els corresponents permisos d'accés.

Bases de dades gestionades per una instància d'un SGBD corporatiu.

Un SGBD és un conjunt de programes encarregats de gestionar bases de dades.

En els SGBD ofimàtiques (*MsAccess*) podem posar en marxa el SGBD sense la obligació d'obrir (engegar) cap base de dades. Podem, en un moment en concret, tenir diferents execucions de l'SGBD (instàncies), cadascuna de les quals pot donar servei a una única base de dades.

Els SGBD corporatius (*Oracle*, *MsSQLServer*, *MySQL*, *PostgreSQL*,...) obliguen, en posar-los en marxa, a tenir definit el conjunt de bases de dades al que donen servei, conjunt que acostuma anomenar-se *cluster database*. Podem, en una mateixa màquina, tenir diferents execucions d'un mateix SGBD (instàncies), cadascuna de les quals dona servei a un

conjunt diferent de bases de dades (*cluster database*). Així doncs, cada instància d'un SGBD permet gestionar un *cluster database*.

En les màquines on hi ha SGBD corporatius instal·lats s'acostuma a configurar un servei de sistema operatiu per a cada instància configurada per a ser gestionada pel SGBD. Així, en màquines amb sistema operatiu MsWindows, aquesta situació es pot constatar ràpidament fent una ullada als serveis instal·lats (Tauler de control | Eines administratives | Serveis), on podríem trobar diversos serveis *d'Oracle*, *MySQL*, *SQLServer*,... identificats per noms que es decideixen en el moment de creació de la instància (*cluster database*).

El concepte de *cluster database* (conjunt de bases de dades gestionades per una instància) utilitzat per la majoria de SGBD corporatius (*MySQL*, *PostgreSQL*, *SQLServer*) no existeix en els SGBD *Oracle*, degut a que en *Oracle*, una instància dona servei a una única base de dades i, per tant, no té sentit parlar de *cluster database* (conjunt de bases de dades). Ara bé, *Oracle* organitza aquesta única base de dades en compartiments, anomenats esquemes, que podem assimilar a les diferents bases de dades d'un *cluster database*.

Així doncs, per exemple, en una empresa en la que és molt usual tenir una base de dades per a la gestió comercial, una base de dades per al control de la producció, una base de dades per a la gestió de personal, una base de dades per a la gestió financera,... i que en SGBD com *MySQL*, *PostgreSQL* i *SQLServer* podrien ser diferents bases de dades gestionades per una mateixa instància de l'SGBD, en *Oracle* haurien de ser diferents esquemes dins la única base de dades gestionada per una instància de l'SGBD.

Exemple d'accés a taules d'altres esquemes

Si estant connectats a l'esquema (base de dades) *empresa* volem mostrar els hospitals existents a l'esquema *sanitat*, caldrà fer:

```
select * from sanitat.hospital;
```

El resultat que s'obté, via consola *SQL*Plus* o via *SQL Developer*, és:

```
HOSPITAL_COD  NOM        ADRECA                TELEFON  QTAT_LLITS
------------ ---------- -------------------- -------- -----------
13           Provincial O Donell 50          964-4264 88
18           General    Atocha s/n           595-3111 63
22           La Paz     Castellana 1000      923-5411 162
45           San Carlos Ciudad Universitaria 597-1500 92

4 rows selected
```

El llenguatge SQL efectua el producte cartesià de totes les taules que troba a la clàusula `from`. En aquest cas, hi pot haver columnes amb mateix nom a diferents taules i si és així i cal seleccionar-ne una d'elles, cal utilitzar obligatòriament la sintaxis `<nom_taula>.<nom_columna>` i fins i tot la sintaxis `<nom_esquema>.<nom_taula>.<nom_columna>` si s'està accedint a una taula d'un altre esquema.

Exemple de sentència SELECT amb vàries taules i coincidència en noms de columnes

Si des de l'esquema *empresa* volem mostrar el producte cartesià de totes les files de la taula `DEPT` amb totes les files de la taula `SALA` de l'esquema *sanitat* (visualització que no té cap sentit, però que ho fem a tall d'exemple), mostrant únicament les columnes que formen les respectives claus primàries, executaríem:

```
select dept.dept_no, sanitat.sala.hospital_cod,
       sanitat.sala.sala_cod
from dept, sanitat.sala;
```

El resultat obtingut és format per 40 files. En mostrem només algunes:

```
DEPT_NO              HOSPITAL_COD         SALA_COD
-------------------- -------------------- ----------------
10                   13                   3
10                   13                   6
10                   18                   3
```

```
...
40                    45                    1
40                    45                    2
40                    45                    4

40 rows selected
```

En aquest cas, la sentència s'hagués pogut escriure sense utilitzar el prefix `sanitat` en les columnes de la taula SALA en la clàusula `select`, donat que a la clàusula `from` no apareix més d'una taula anomenada SALA i, per tant, no hi ha problemes d'ambigüitat. Haguéssim pogut escriure:

```
select dept.dept_no, sala.hospital_cod, sala.sala_cod
from dept, sanitat.sala;
```

El llenguatge SQL permet definir **àlies** per a una taula. Per aconseguir-ho, cal escriure l'àlies en la clàusula `from` després del nom de la taula i abans de la coma que la separa amb la següent taula (si existeix) de la clàusula `from`. (!!)

Exemple d'utilització d'àlies per a noms de taules

La instrucció per a obtenir, estant connectats a l'esquema *empresa*, el producte cartesià de totes les files de la taula DEPT amb totes les files de la taula SALA de l'esquema *sanitat*, mostrant únicament les columnes que formen les respectives claus primàries, podríem executar:

```
select d.dept_no, s.hospital_cod, s.sala_cod
from dept d, sanitat.sala s;
```

El llenguatge SQL permet utilitzar el resultat d'una sentència SELECT com a taula dins la clàusula `from` d'una altra sentència SELECT. (!!)

Exemple de sentència SELECT com a taula en una clàusula `from`.

Així doncs, una altra forma d'obtenir, estant connectats a l'esquema *empresa*, el producte cartesià de totes les files de la taula DEPT amb totes les files de la taula SALA de l'esquema *sanitat*, mostrant únicament les columnes que formen les respectives claus primàries, seria:

```
select dept_no, hospital_cod, sala_cod
from dept, (select hospital_cod, sala_cod from sanitat.sala);
```

A més de les columnes de les taules implicades en la sentència SELECT, *Oracle* permet seleccionar unes pseudocolumnes entre les que cal destacar `rownum` que serveix per enumerar les files seleccionades (provinguin d'una o vàries taules) i `rowid` que és l'adreça física (en nomenclatura d'*Oracle*) de cada fila per cada taula dins la base de dades.

Pseudocolumnes en SGBD

El concepte de pseudocolumnes no està estanditzat en tots els SGBD

`Rowid` en *Oracle*

El `rowid` (identificador de fila) és l'adreça física de la base de dades per cada fila de cada taula. Una vegada assignat (quan la fila és inserida per primera vegada en la base de dades), no canvia mai fins que s'elimina la fila o s'elimina la taula.

El `rowid` és utilitzat internament en índexs com a forma ràpida de recuperar files amb un valor clau particular. Els desenvolupadors d'aplicacions l'utilitzen en instruccions SQL com a forma ràpida d'accedir a una fila una vegada conegut el seu `rowid`.

En les sentències SELECT es pot fer referència al valor `rowid` com si d'una columna es tractés, però cal tenir sempre present que no és una columna real de la taula. Així doncs,

no es pot modificar el valor de pseudocolumna `rowid`, així com tampoc es pot utilitzar aquesta pseudocolumna com a clau primària de la taula.

En eliminar una fila, *Oracle* pot utilitzar el rowid que tenia la fila eliminada per a una nova fila inserida amb posterioritat.

En cas de processos d'exportació i importació de dades amb les utilitats *Export* i *Import* que facilita *Oracle*, no es mantenen els `rowid` de les files exportades i importades.

Exemple d'utilització de les pseudocolumnes `rownum` i `rowid`.

Per a mostrar, en el tema *empresa*, els departaments, enumerats i amb l'adreça física de les files, escriuríem:

```
select rownum "Fila", dept_no "Codi", dnom "Departament", rowid "Adr.Física"
from dept;
```

El resultat obtingut seria similar a:

```
Fila Codi departament    Adr.Física
---- ---- -------------- ------------------
   1   10 COMPTABILITAT  AAAC4jAACAAAANVAAA
   2   20 INVESTIGACIÓ   AAAC4jAACAAAANVAAB
   3   30 VENDES         AAAC4jAACAAAANVAAC
   4   40 PRODUCCIÓ      AAAC4jAACAAAANVAAD
```

Ordenació de les files d'una sentència `SELECT`

La sentència `SELECT` té més clàusules a banda de les conegudes `select` i `from`. Així, té una clàusula `order by` que permet ordenar el resultat de la consulta.

La columna `rownum` simplement enumera les files seleccionades en l'ordre en què es visualitzen el qual pot ser indicat en la sentència `SELECT`. Per tant, el seu contingut depèn de cada execució i no té res a veure amb l'antiguitat de les files a la taula.

Oracle incorpora una taula fictícia, anomenada **`dual`**, per a efectuar càlculs independents de cap taula de la base de dades aprofitant la potència de la sentència `SELECT`. ‼

Així, doncs, podem emprar aquesta taula per a:

a) Efectuar càlculs matemàtics

```
SQL> select 4 * 3 - 8 / 2 as "Resultat" from dual;

RESULTAT
----------------------
8

1 rows selected
```

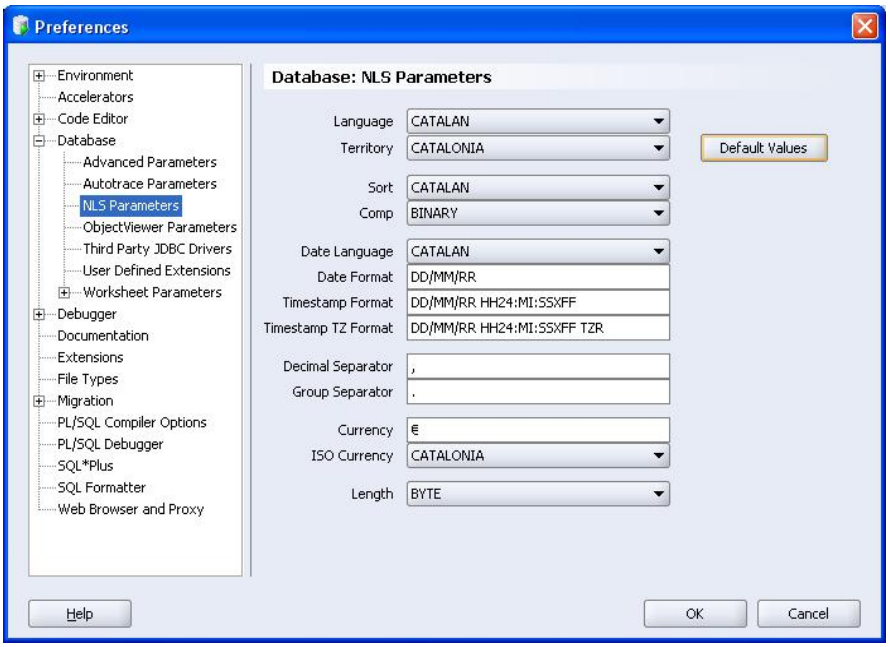
b) Obtenir la data del sistema, sabent que està emmagatzemada en una variable de l'SGBD anomenada `sysdate`

```
SQL> select sysdate from dual;

SYSDATE
-------------------------
09/02/08

1 rows selected
```

El fet que ens mostri la data en format dia/mes/any(2 dígits) és degut a la configuració del paràmetre `NLS_DATE_FORMAT` de l'eina client

que estem utilitzant per a comunicar-nos amb la base de dades. Si estem utilitzant l'eina *SQL Developer*, disposem de l'opció *Tools* | *Preferences* | *Database* | *NLS Parameters* (figura 3) per a definir les preferències de treball.

Figura 3. Pantalla de *SQL Developer* per a la configuració de preferències de treball



En canvi, si estem utilitzant una consola *SQL*Plus*, podem tenir definit el paràmetre `NLS_DATE_FORMAT` com a variable d'entorn del sistema operatiu (sigui MsWindows o Linux). En cas de MsWindows també pot estar definida en la branca del registre de MsWindows on sigui registrada la consola *SQL*Plus* que s'utilitza, però només la utilitzarà si no està carregada com a variable d'entorn.

En qualsevol cas, el paràmetre `NLS_DATE_FORMAT` ha d'estar correctament configurat i per aconseguir-ho cal fer una ullada a la documentació d'*Oracle* per a conèixer les diverses possibilitats.

Consideracions similars hem de tenir en compte per a altres paràmetres com `NLS_DATE_LANGUAGE`, `NLS_LANGUAGE`, `NLS_TIMESTAMP_FORMAT`, `NLS_TIMESTAMP_TZ_FORMAT`, ...

c) Comprovar el funcionament de les operacions entre dades corresponents a moments temporals. Així, suposant que tenim els paràmetres de configuració com es veu a la figura 3, podem comprovar:

- Comunicació d'una data al SGBD en format ANSI (yyyy-mm-dd)

```
SQL> select DATE '2008-02-29' as Exemple1 from dual;

EXEMPLE1
--------
29/02/08
```

```
SQL> select DATE '2008/02/29' as Exemple2 from dual;
select DATE '2008/02/29' as Exemple2 from dual
            *
ERROR a la línia 1:
ORA-01861: el literal no coincideix amb la cadena del format
```

Observem que la primera sentència és acceptada per *Oracle* però en canvi la segona no ho és doncs el format de data utilitzat no segueix l'especificació ANSI.

- Per una persona que ha nascut el 4 d'abril del 1962, càlcul del número de dies que porta de vida:

```
SQL> select sysdate - DATE '1962-04-04' as DiesDeVida from dual;

DIESDEVIDA
----------
 16747,517
```

Com be sabem, els tipus de dades `DATE` emmagatzemen dia i hora. Per aquest motiu, el valor que emmagatzema la variable `sysdate` conté el dia i l'hora i la diferència en dies ha de portar part decimal. Recordem, que a les dates introduïdes sense indicar els valors corresponents als apartats hora – minut – segon, *Oracle* hi assigna el valor 12:00:00 AM.

Si volem saber els dies sencers (sense cap decimal), caldrà utilitzar algunes de les funcions incorporades que facilita *Oracle*: `round()` per arrodonir o `trunc()` per a truncar. Fixem-nos els resultats que haguéssim obtingut en el mateix moment en que hem executat la darrera instrucció en cas d'haver utilitzat aquestes funcions:

```
SQL> select round(sysdate - DATE '1962-04-04') as DiesDeVida from dual;

DIESDEVIDA
----------
     16748

SQL> select trunc(sysdate - DATE '1962-04-04') as DiesDeVida from dual;

DIESDEVIDA
----------
     16747
```

Observem que el resultat inicial 16.747,517 indica que la instrucció s'ha executat passades les 12 del migdia (doncs la part decimal és superior a 0,5). Per això, el dia en el que s'ha executat la instrucció es comptabilitza dins els dies de vida que porta la persona quan s'efectua l'arrodoniment via `round()` mentre que no es comptabilitza en efectuar el truncament via `trunc()`.

- Mostrar, en el tema *empresa*, l'antiguitat en dies que porten tots els empleats:

```
select emp_no as "Codi", cognom as "Empleat",
       round(sysdate-data_alta) as "Dies Contracte"
from emp;
```

Obtenim el resultat (instrucció executada el dia 9 de febrer del 2008):

```
Codi                 Empleat    Dies Contracte
-------------------- ---------- --------------------
7369                 SÁNCHEZ    9916
7499                 ARROYO     10217
7521                 SALA       9849
7566                 JIMÉNEZ    9810
7654                 MARTÍN     9630
7698                 NEGRO      9781
7782                 CEREZO     9742
7788                 GIL        9589
7839                 REY        9581
7844                 TOVAR      9651
7876                 ALONSO     9636
7900                 JIMENO     9565
7902                 FERNÁNDEZ  9565
7934                 MUÑOZ      9514

14 rows selected
```

1.4.2. Clàusula `where`

Aquesta clàusula s'afegeix darrera de la clàusula `from` de manera que ampliem la sintaxis de la sentència `SELECT`:

```
select <expressió/columna>, <expressió/columna>,...
from <taula>, <taula>,...
[where <condició_de_recerca>];
```

La clàusula `where` permet establir els criteris de recerca sobre les files generades per la clàusula `from`. En terminologia d'àlgebra relacional es correspon amb l'operació **selecció**.

La complexitat de la clàusula `where` és pràcticament il·limitada gràcies a l'abundància d'operadors disponibles per a efectuar operacions.

1) Operadors aritmètics

Són els típics operadors `+, -, *, /` utilitzables per formar expressions amb constants, valors de columnes i funcions de valors de columnes.

2) Operadors de dates

Disposem de:

– Operador + per a sumar un determinat número de dies a una data i obtenir una nova data.

– Operador - per a restar un determinat número de dies a una data i obtenir una nova data

– Operador - per a restar dues dates obtenint el número de dies que les separen.

3) Operadors per cadenes de caràcters

L'operador || per concatenar cadenes.

4) Operadors de comparació

Disposem de diferents operadors per efectuar comparacions:

- Els típics operadors `=, !=, >, <, >=, <=` per efectuar comparacions entre dades obtenint un resultat booleà: cert o fals.

- L'operador `[NOT] LIKE` per comparar una cadena (part esquerra de l'operador) amb una cadena patró (part dreta de l'operador) que pot contenir els següents caràcters especials:

  '%' per indicar qualsevol cadena de 0 o més caràcters
  '_' per indicar qualsevol caràcter

  Així:

`LIKE 'Torres'`	Compara amb la cadena 'Torres'
`LIKE 'Torr%'`	Compara amb qualsevol cadena iniciada per 'Torr'
`LIKE '%S%'`	Compara amb qualsevol cadena que contingui 'S'
`LIKE '_o%'`	Compara amb qualsevol cadena que tingui per segon caràcter una 'o'
`LIKE '__'`	Compara amb qualsevol cadena de dos caràcters

  Per finalitzar, remarcar que en el treball amb cadenes en *Oracle* cal tenir present:

  – Les majúscules i minúscules són significatives.

  – Les constants alfanumèriques sempre van **tancades entre cometes simples**.

- Un últim conjunt d'operadors lògics:

  `[NOT] BETWEEN valor_1 AND valor_2`
  que permet efectuar la comparació entre dos valors

`[NOT] IN (llista_valors_separats_per_comes)`
que permet comparar amb una llista de valors

`IS [NOT] NULL`
que permet reconèixer si estem davant d'un valor `NULL`

`<comparador genèric> ANY (llista_valors)`
que permet efectuar una comparació genèrica (`=, !=, >, <, >=, <=`) amb QUALSEVOL dels valors de la dreta

`<comparador genèric> ALL (llista_valors)`
que permet efectuar una comparació genèrica (`=, !=, >, <, >=, <=`) amb TOTS els valors de la dreta

Observem que:

`=ANY`	és equivalent a `IN`
`!=ALL`	és equivalent a `NOT IN`
`=ALL`	és sempre fals si la llista té més d'un element diferent
`!=ANY`	és sempre cert si la llista té més d'un element diferent

Exemple de filtratge simple en la clàusula `where`.

Es desitja mostrar, en el tema *empresa*, els empleats (codi i cognom) que tenen un salari mensual igual o superior a 200.000 així com el seu salari anual (suposem que en un any hi ha 14 pagues mensuals).

La instrucció que permet assolir l'objectiu és:

```
select emp_no as "Codi", cognom as "Empleat", salari*14 as "Salari anual"
from emp
where salari >= 200000;
```

El resultat obtingut és:

```
Codi                 Empleat    Salari anual
-------------------- ---------- --------------------
7499                 ARROYO     2912000
7566                 JIMÉNEZ    5414500
7698                 NEGRO      5187000
7782                 CEREZO     4459000
7788                 GIL        5460000
7839                 REY        9100000
7902                 FERNÁNDEZ  5460000

7 rows selected
```

Exemple de filtratge de dates utilitzant l'especificació ANSI per indicar una data.

Es desitja mostrar, en el tema *empresa*, els empleats (codi, cognom i data de contractació) contractats a partir del mes de juny del 1981.

La instrucció que permet assolir l'objectiu és:

```
select emp_no as "Codi", cognom as "Empleat", data_alta as "Contracte"
from emp
where data_alta >= DATE '1981-06-01');
```

El resultat obtingut és:

```
Codi                 Empleat    Contracte
-------------------- ---------- -------------------------
7654                 MARTÍN     29/09/81
7782                 CEREZO     09/06/81
7788                 GIL        09/11/81
7839                 REY        17/11/81
7844                 TOVAR      08/09/81
7876                 ALONSO     23/09/81
7900                 JIMENO     03/12/81
7902                 FERNÁNDEZ  03/12/81
7934                 MUÑOZ      23/01/82

9 rows selected
```

Exemple d'utilització d'operacions lògiques en la clàusula `where`

Es desitja mostrar, en el tema *empresa*, els empleats (codi, cognom) resultat de la intersecció dels dos darrers exemples, és a dir, empleats que tenen un sou mensual igual o superior a 200.000 i contractats a partir del mes de juny del 1981.

La instrucció per aconseguir el que se'ns demana és:

```
select emp_no as "Codi", cognom as "Empleat"
from emp
where data_alta >= to_date ('01/06/1981','dd/mm/yyyy')
  and salari >= 200000;
```

El resultat obtingut és:

```
Codi                 Empleat
-------------------- ----------
7782                 CEREZO
7788                 GIL
7839                 REY
7902                 FERNÁNDEZ

4 rows selected
```

Exemple 1 d'utilització de l'operador `like`

Es desitja mostrar, en el tema *empresa*, els empleats que tenen per inicial del seu cognom una 'S'.

La instrucció demanada pot ser:

```
select cognom as "Empleat"
from emp
where cognom like 'S%';
```

Aquesta instrucció mostra els empleats que el seu cognom comença per la lletra 'S' majúscula i és de suposar que els cognoms estan introduïts amb la inicial en majúscula, però per assegurar la solució a l'enunciat, és millor utilitzar la funció incorporada `upper()` que retorna una cadena en majúscules:

```
select cognom as "Empleat"
from emp
where upper(cognom) like 'S%';
```

Exemple 2 d'utilització de l'operador `like`

Es desitja mostrar, en el tema *empresa*, els empleats que tenen alguna 'S' en el seu cognom.

La instrucció demanada pot ser:

```
select cognom as "Empleat"
from emp
where upper(cognom) like '%S%';
```

Exemple 3 d'utilització de l'operador `like`

Es desitja mostrar, en el tema *empresa*, els empleats que no tenen la 'R' com a tercera lletra del seu cognom.

La instrucció demanada pot ser:

```
select cognom as "Empleat"
from emp
where upper(cognom) not like '__R%';
```

Exemple d'utilització de l'operador `between`

Es desitja mostrar, en el tema *empresa*, els empleats que tenen un salari mensual entre 100.000 i 200.000.

La instrucció demanada pot ser:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari"
from emp
where salari >= 100000 and salari <= 200000;
```

Podem, però, utilitzar l'operador `between`:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari"
from emp
where salari between 100000 and 200000;
```

Exemple d'utilització dels operadors `in` o `=any`

Es desitja mostrar, en el tema *empresa*, els empleats dels departaments 10 i 30.

La instrucció demanada pot ser:

```
select emp_no as "Codi", cognom as "Empleat", dept_no as "Departament"
from emp
where dept_no = 10 or dept_no = 30;
```

Podem, però, utilitzar els operadors `in` o `=any`:

```
select emp_no as "Codi", cognom as "Empleat", dept_no as "Departament"
from emp
where dept_no in (10,30);

select emp_no as "Codi", cognom as "Empleat", dept_no as "Departament"
from emp
where dept_no =any (10,30);
```

Exemple d'utilització dels operadors `not in` o `!=all`

Es desitja mostrar, en el tema *empresa*, els empleats que no treballen en els departaments 10 i 30.

La instrucció demanada pot ser:

```
select emp_no as "Codi", cognom as "Empleat", dept_no as "Departament"
from emp
where dept_no !=10 and dept_no != 30;
```

Podem, però, utilitzar els operadors `not in` o `!=all`:

```
select emp_no as "Codi", cognom as "Empleat", dept_no as "Departament"
from emp
where dept_no not in (10,30);

select emp_no as "Codi", cognom as "Empleat", dept_no as "Departament"
from emp
where dept_no !=all (10,30);
```

2. Llenguatge SQL. Consultes complexes.

Una vegada coneixem els tipus de dades que facilita el SGBD i sabem utilitzar la sentència `SELECT` per a l'obtenció d'informació de la base de dades amb la utilització de les clàusules `select` (selecció de columnes i/o expressions), `from` (selecció de les taules corresponents) i `where` (filtrat adequat de les files que interessen), estem en condicions d'aprofundir en les possibilitats que té la sentència `SELECT`, doncs només en coneixem les clàusules bàsiques.

A l'hora d'obtenir informació de la base de dades, ens interessa poder incorporar, en les expressions de les clàusules `select` i `where`, càlculs genèrics que els SGBD faciliten amb funcions incorporades (càlculs com el valor absolut, arrodoniments, truncaments, extracció de subcadenes en cadenes de caràcters, extracció de l'any, mes o dia en dates,...) així com poder efectuar consultes més complexes que permetin classificar la informació, efectuar agrupaments de files, realitzar combinacions entre diferents taules i incloure els resultats de consultes dins altres consultes. Anem, doncs, a aconseguir-ho a continuació.

2.1. Funcions incorporades per *Oracle*

Els SGBD acostumen a incorporar funcions utilitzables des del llenguatge SQL. El llenguatge SQL, per sí mateix, no incorpora funcions genèriques, a excepció de les anomenades funcions d'agrupament.

Agrupació de files d'una consulta SELECT

La sentència `SELECT` té més clàusules a banda de les conegudes `select`, `from`.i `where`. Així, té una clàusula `group by` que permet agrupar les files resultants de la consulta i aplicar-hi funcions d'agrupament (`max( )` per al càlcul del major valor de cada grup, `min( )`per al càlcul del menor valor de cada grup,...)

> Les funcions incorporades facilitades pels SGBD es poden utilitzar dins d'expressions i actuen amb els valors de les columnes, variables o constants en les clàusules `select`, `where` i `order by` (clàusula que permetrà l'ordenació de les files).

També és possible utilitzar el resultat d'ona funció com a valor per a utilitzar en una altra funció.

Vegem, a continuació, un recull de les principals funcions facilitades per *Oracle* (matemàtiques, de cadenes de caràcters, de gestió de moments temporals i de conversió de dades) tot i que no són totes les funcions de què disposem. Sempre caldrà consultar la documentació d'*Oracle*.

1) Matemàtiques

La taula 8 ens presenta les principals funcions matemàtiques facilitades pel SGBD *Oracle*.

Taula 8. Principals funcions matemàttiques facilitades pel SGBD *Oracle*.

Funció	Descripció	Exemples
`ABS(n)`	Retorna el valor absolut de `n`	`ABS(-10) retorna 10`
`CEIL (n)`	Retorna el valor enter immediatament superior o igual a `n`	`CEIL (10.6) retorna 10`
`FLOOR (n)`	Retorna el valor enter immediatament inferior o igual a `n`	`FLOOR (10.6) retorna 10`
`MOD (m,n)`	Retorna el residu de la divisió entera entre `m` i `n`	`MOD (15,2) retorna 1`
`POWER (m,n)`	Retorna el valor m elevat a la potència `n`	`POWER (4,2) retorna 16`

`ROUND (n[,m])`	Retorna el valor `n` arrodonit a `m` decimals.
	Si `m` no existeix, es suposa zero.
	Si `m` és negatiu, l'arrodoniment de dígits s'efectua per l'esquera del punt decimal.
		`ROUND (1.6724,1) retorna 1.7`
		`ROUND (1.6724) retorna 2`
		`ROUND (5462.67,-2) retorna 5500`

`SQRT (n)`	Retorna l'arrel quadrada de `n`, que no pot ser negatiu	`SQRT (5) retorna 2.23606798`

`TRUNC (n[,m])`	Retorna `n` truncat a `m` decimals.
	Si `m` no existeix, es suposa zero.
	Si `m` és negatiu, el truncament de dígits s'efectua per l'esquerra del punt decimal.
		`TRUNC (1.6724,1) retorna 1.6`
		`TRUNC (1.6724) retorna 1`
		`TRUNC (5462.67,-2) retorna 5400`

2) De cadenes de caràcters

La taula 9 ens presenta les principals funcions per a la gestió de cadenes de caràcters facilitades pel SGBD *Oracle*.

Taula 9. Principals funcions de gestió de cadenes de caràcters facilitades pel SGBD *Oracle*.

Funció	Descripció	Exemples
`CHR (n)`	Retorna el caràcter que ocupa el lloc `n` a la taula ASCCI	`CHR(75) retorna 'K'`
`CONCAT (cad1,cad2)`	Retorna la concatenació de les cadenes `cad1` i `cad2`. És equivalent a l'operador `||`	`CONCAT ('Avui és','diumenge') retorna 'Avui és diumenge'`
`INITCAP (cad)`	Retorna la cadena amb la inicial amb majúscula i la resta en minúscules	`INITCAP ('ORIOL') retorna 'Oriol'` `INITCAP ('guifré') retorna 'Guifré'`
`LOWER (cad)`	Retorna la cadena `cad` amb totes les lletres convertides a minúscules	`LOWER ('Avui') retorna 'avui'`
`LPAD (cad1, n [,cad2])`	Retorna `cad1` amb longitud `n` i justificada per la dreta. La cadena `cad2` s'utilitza per omplir per l'esquerra.	`LPAD ('R',7,'xs') retorna 'xsxsxsR'` `LPAD ('R',3,'Hola') retorna 'HoR'`
`LTRIM (cad [,sub])`	Retorna `cad` amb el grup de caràcters `sub` eliminats de l'esquerra de la cadena tantes vegades com aparegui.	`LTRIM ('Hola','Ho') retorna 'la'` `LTRIM ('lalala','la') retorna ''`
`REPLACE (cad, tros [,subst])`	Retorna `cad` amb cada aparició de la cadena `tros` substituïda per la cadena `subst`. Si no hi ha cadena substituta, es procedeix a eliminar cada aparició de la cadena tros.	`REPLACE ('glugluglú','glu','x') retorna 'xxglú'` `REPLACE ('glugluglú','glu') retorna 'glú'`
`RPAD (cad1, n [,cad2])`	Retorna `cad1` amb longitud `n` i justificada per l'esquerra. La cadena `cad2` s'utilitza per omplir per la dreta.	`RPAD ('R',7,'xs') retorna 'Rxsxsxs'` `RPAD ('R',3,'Hola') retorna 'RHo'`

Funció	Descripció	Exemples
RTRIM (cad [,sub])	Retorna cad amb el grup de caràcters sub eliminats de la dreta de la cadena tantes vegades com aparegui.	RTRIM ('Hola','la') retorna 'Ho' RTRIM ('lalala','la') retorna ''
SUBSTR (cad,m[,n])	Retorna la subcadena de la cadena cad a partir de a la posició m i fins a n caràcters. El valor de n no pot ser inferior a 1. El valor de m pot ser negatiu i s'interpreta com a la posició començant pel final.	SUBSTR ('Catalunya',3,3) retorna 'tal' SUBSTR ('Catalunya',-5,2) retorna 'lu'
TRANSLATE (cad,in, fi)	Retorna la cadena cad traduïda segons la correspondència de caràcters de les cadenes in i fi	TRANSLATE ('Hola','aeiou','zyxwv') retorna 'Hwlz'
UPPER (cad)	Retorna la cadena cad amb totes les lletres convertides a majúscules	UPPER ('Avui') retorna 'AVUI'
ASCII (cad)	Retorna el valor ASCII del primer caràcter de cad	ASCII ('Hola') retorna 72
INSTR (cad1, cad2 [,n,[m]])	Retorna la posició de la m ocurrència de cad2 dins cad1, començant la recerca a la posició n	INSTR('Catalunya','a',1,1) retorna 2 INSTR('Catalunya','a',2,3) retorna 9
LENGTH (cad)	Retorna la longitud de la cadena cad	LENGTH ('Catalunya') retorna 9

3) De gestió de dates

La taula 10 ens presenta algunes de les funcions per a la gestió de dades DATE facilitades pel SGBD *Oracle*.

Taula 10. Funcions per a la gestió de dades DATE facilitades pel SGBD *Oracle*

Funció	Descripció	Exemples
ADD_MONTHS (d,n)	Retorna la data d incrementada en n mesos	ADD_MONTHS(TO_DATE('01-09-2000','dd-mm-yyyy'),5) retorna '01-02-2001'
LAST_DAY (d)	Retorna la data del darrer dia del mes de la data d	LAST_DAY(TO_DATE('01-09-2000','dd-mm-yyyy')) retorna '30-09-2000'
MONTHS_BETWEEN(d1, d2)	Retorna la diferència en mesos entre les dates d1 i d2	MONTHS_BETWEEN(TO_DATE('01-09-2000','dd-mm-yyyy'), TO_DATE('31-12-1985','dd-mm-yyyy')) retorna 176.032258 (els decimals corresponen als dies que no arriben a un mes)
NEXT_DAY(d,cad)	Retorna la data del primer dia de la setmana cad posterior a la data d	NEXT_DAY(TO_DATE('11-09-2000','dd-mm-yyyy'),'dissabte') retorna '16-09-00'

4) De conversió de tipus de dades

La taula 11 presenta les funcions més utilitzades per a la conversió de dades de diferents tipus. En la declaració de totes elles s'observa un tercer paràmetre opcional, nlspar, que s'utilitza per:

a) Especificar el llenguatge en el que s'escriuen els noms i les abreviatures dels mesos i dies de la setmana. El format d'aquest paràmetre és:

```
'NLS_DATE_LANGUAJE = llenguatge'
```

Si no s'utilitza aquest paràmetre, el llenguatge emprat és el que té parametritzat l'eina client des de la que s'executen les instruccions SQL

(cas d'utilitzar *SQL Developer*) o la sessió de sistema des de la que s'executa l'eina client (cas d'utilitzar *SQL*Plus*).

Algunes de les cadenes corresponents al paràmetre `llenguatge` que *Oracle* accepta són: `catalan`, `spanish`, `french`, `english`, `german`, `portuguese`,...

b) Especificar paràmetres numèrics com els caràcters que representen els milers i els decimals, la simbologia per les monedes,...

Suposem que els productes han estat instal·lats de manera que el caràcter que representa el punt decimal és la coma i el que representa els milers és el punt. Si no és així o se'n vol especificar un altre es disposa del paràmetre:

```
' NLS_NUMERIC_CHARACTERS=''.,'' '
```

que tal com està escrit indica que el punt és el símbol pels decimals i la coma és el símbol pels milers.

Taula 11. Funcions facilitades pel SGBD *Oracle* per a la conversió de dades de diferents tipus.

Funció	Descripció	Exemples

`TO_NUMBER ( cad [,fmt[,nlspar]])`	Retorna la cadena `cad`, que es suposa que conté un número en format `fmt`, com a número.	`TO_NUMBER('12345') retorna 12345`
		`TO_NUMBER('123,45','99999D99') retorna 123,45`
		`TO_NUMBER('123,123.45','999G999D99','NLS_NUMERIC_CHARACTERS =''.,'' ') retorna 123123,45`

`TO_CHAR ( X [,fmt[,nlspar]])`	Retorna X, que es suposa que és un número o una data, com a cadena	`TO_CHAR(12345) retorna '12345'`
		`TO_CHAR(123.45) retorna '123,45'`
		`TO_CHAR(DATE '2008-09-11','day, dd-month-yyyy') retorna 'dilluns, 11-setembre-2000'`

`TO_DATE ( cad [,fmt[,nlspar]])`	Retorna la cadena `cad`, que es suposa que conté una data en format `fmt`, com a data	`TO_DATE('31-12-1999','dd-mm-yyyy') retorna '31-12-1999'.`
		`TO_DATE('31-12-99','dd-mm-yy') retorna la data '31-12-####' on l'any #### coincideix amb l'any de la data de l'SGBD.`
		`TO_DATE('31-12-99','dd-mm-RR') retorna la data '31-12-1999' si la data de l'SGBD està entre els anys 2000 i 2049, però en canvi retorna la data '31-12-2099' si la data de l'SGBD està entre els anys 2050 i 2099.`
		`TO_DATE('31-12-40','dd-mm-yy') retorna la data '31-12-####' on l'any #### coincideix amb l'any de la data de l'SGBD.`
		`TO_DATE('31-12-40','dd-mm-RR') retorna la data '31-12-1940' si la data de l'SGBD està entre els anys 1900 i 1949, però en canvi retorna la data '31-12-2040' si la data de l'SGBD està entre els anys 1950 i 1999.`

El format per controlar les dades `DATETIME` no pot excedir de 22 caràcters i pot estar format per diferents combinacions de les màscares presentades a la taula 12.

La màscara `RR` proporciona una flexibilitat per emmagatzemar valors de dates en altres segles. Actualment, estem a cavall dels segles XX i XXI,

és una eina per a ser utilitzada, però la seva utilitat no està únicament en aquest canvi de segle.

La funcionalitat de la màscara RR depèn dels dos darrera dígits de l'any especificat i dels dos darrers dígits de l'any actual, segons la taula 13.

Taula 12. Màscares utilitzables per a controlar el format de les dades DATETIME.

Màscares de format numèric		mi	Minuts
cc o scc	Valor del segle	Ss	Segons
y,yyy o sy,yyy	Any amb coma, amb o sense signe	Sssss	Segons transcorreguts des de mitja nit
yyyy	Any sense signe	**Màscares de format de caràcters**	
yyy	Darrers tres dígits de l'any	syear o year	Any sempre en anglès
yy o RR	Darrers dos dígits de l'any	month	Nom del mes
y	Darrer dígit de l'any	mon	Abreviatura de tres lletres del nom del mes
q	Número del trimestre	day	Nom del dia de la setmana
ww	Numero de setmana de l'any	dy	Abreviatura de tres lletres del nom del dia de la setmana
w	Número de setmana del mes	a.m. o p.m.	Horari del matí o de la tarda
mm	Número del mes	b.c. o a.d.	Abans (before) de Crist o després (after) de Crist
ddd	Número del dia de l'any	th	
dd	Número del dia del mes	**Sufixos de màscares de format**	
d	Número del dia de la setmana	th	Cardinal del número (st, nd, rd, th)
hh o hh12	Hora (1-12)	Sp	Escriu el text, en anglès, corresponent al valor d'un número
hh24	Hora (1-24)	Spth o thsp	Combina les dues accions anteriors

Taula 13. Funcionalitat de la màscara RR segons els dígits de l'any especificat i de l'any actual

		Dígits de l'any especificat	
		0 – 49	50 - 99
Dígits de l'any actual	0 - 49	La data retornada es troba en el segle actual	La data retornada es troba en el segle anterior a l'actual
	50 - 99	La data retornada es troba en el segle següent a l'actual	La data retornada es troba en el segle actual

La taula 14 mostra els formats possibles per controlar les dades numèriques.

Taula 14. Màscares utilitzables per a controlar el format de les dades numèriques.

, (coma)	Correspon a una coma de milers en la posició especificada.	L	Col·loca en la posició especificada l'abreviatura local de la moneda activa (definida en el paràmetre NLS_CURRENCY). En el nostre cas correspon €.
. (punt)	Correspon al punt decimal en la posició especificada.	MI	Per a mostrar els valors negatius amb un signe - per la dreta i els positius amb un espai en blanc per la dreta.
$	Afegeix el símbol $ al davant del número	PR	Per a mostrar els valors negatius entre símbols < > i els valors positius entre dos espais en blanc

0	Afegeix un zero per l'esquerra o per la dreta (segons es situï). Per la dreta només té sentit si està darrera del punt decimal.	RN	Numeració romana en majúscules
9	Reserva un espai per un dígit afegint un espai en blanc per l'esquerra si és positiu i un signe - si és negatiu.	rn	Numeració romana en minúscules
C	Col·loca en la posició especificada l'abreviatura ISO de la moneda activa (definida en el paràmetre NLS_ISO_CURRENCY). En el nostre cas correspon EUR.	S	Per a mostrar els valors negatius amb un signe - per l'esquerra i els positius amb un signe + per l'esquerra.
D	Col·loca en la posició especificada el símbol corresponent al punt decimal.	TM	Retorna, en sortida decimal, el menor número de caràcters possible.
EEEE	Notació científica	U	Col·loca en la posició especificada l'abreviatura local de la moneda alternativa a l'activa (definida en el paràmetre NLS_DUAL_CURRENCY).
FM	Número sense espais en blanc per l'esquerra i per la dreta.	V	Correspon a un valor multiplicat per la 10^n on n és el número de 9 després de V.
G	Col·loca en la posició especificada el símbol corresponent a la separació de milers.	X	Retorna el valor hexadecimal del número

5) Altres funcions genèriques

La taula 15 mostra unes funcions genèriques incorporades pel SGBD *Oracle* que no es poden catalogar en cap dels apartats anteriors però que també són molt utilitzades en les sentències SQL.

Taula 15. Recull d'altres funcions genèriques incorporades pel SGBD *Oracle*

Funció	Descripció	Exemples
DECODE (var, val1, cod1, val2, cod2,...,valor_per_defecte)	Si var és igual a qualsevol valor de la llista, retorna el corresponent codi; en cas contrari, s'obté el valor assignat com a valor per defecte	DECODE ('X','B','Barcelona','L','Lleida','T', 'Tarragona','GI','Girona','Error') retorna 'Error'
GREATEST (e1, e2, e3...)	Retorna el major valor de la llista	GREATEST (10,20,30) retorna 30
LEAST (e1, e2, e3...)	Retorna el menor valor de la llista	LEAST (10,20,30) retorna 10
NVL (x, exp)	Si x és NULL retorna exp; si no ho és, retorna x. El valor d'x i exp pot ser de qualsevol tipus.	Vegeu observacions i exemple a continuació.

La funció NVL és molt important, ja que s'utilitza per evitar els valors nuls en expressions aritmètiques i funcions. Cal tenir present que:

- Els valors nuls no intervenen en les funcions d'agrupament (max(), min(),...). Si ens interessa que hi intervinguin, amb aquesta funció podem aconseguir que el valor NULL sigui considerat un valor en concret.

- Els valors nuls en les expressions sempre provocarien un valor nul, i amb la utilització de la funció NVL podem estalviar-nos aquest problema.

Exemple de la necessitat de disposar de la funció NVL.

Es desitja mostrar, en l'esquema *empresa*, els empleats (codi, cognom, salari i comissió) acompanyats d'una columna que contingui la suma del salari i la comissió.

En un principi consideraríem la sentència:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari",
       comissio as "Comissió", salari+comissio as "Suma"
from emp;
```

El resultat obtingut és:

```
Codi    Empleat    Salari    Comissió     Suma
------- ---------- --------- ------------ -----------------
7369    SÁNCHEZ    104000
7499    ARROYO     208000    39000        247000
7521    SALA       162500    65000        227500
7566    JIMÉNEZ    386750
7654    MARTÍN     162500    182000       344500
7698    NEGRO      370500
7782    CEREZO     318500
7788    GIL        390000
7839    REY        650000
7844    TOVAR      195000    0            195000
7876    ALONSO     143000
7900    JIMENO     123500
7902    FERNÁNDEZ  390000
7934    MUÑOZ      169000

14 rows selected
```

El resultat no és el desitjat ja que tots els empleats tenen salari però no tots tenen comissió assignada (pels que no tenen comissió hi ha un valor `NULL` a la corresponent columna) i aquest fet motiva que la suma només es calculi pel que tenen comissió. Possiblement esperaríem que el SGBD considerés que un `NULL` en la comissió dels empleats és equivalent a zero. Li ho hem d'especificar, tal i com es veu en la següent sentència:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari",
       comissio as "Comissió", salari+nvl(comissio,0) as "Suma"
from emp;
```

Ara sí que el resultat és el desitjat:

```
Codi   Empleat    Salari    Comissió     Suma
------ ---------- --------- ------------ ----------------------
7369   SÁNCHEZ    104000                 104000
7499   ARROYO     208000    39000        247000
7521   SALA       162500    65000        227500
7566   JIMÉNEZ    386750                 386750
7654   MARTÍN     162500    182000       344500
7698   NEGRO      370500                 370500
7782   CEREZO     318500                 318500
7788   GIL        390000                 390000
7839   REY        650000                 650000
7844   TOVAR      195000    0            195000
7876   ALONSO     143000                 143000
7900   JIMENO     123500                 123500
7902   FERNÁNDEZ  390000                 390000
7934   MUÑOZ      169000                 169000

14 rows selected
```

2.2. Classificació de files. Clàusula `order by`.

La clàusula `select` permet decidir quines columnes seleccionar del producte cartesià de les taules especificades en la clàusula `from` i la clàusula `where` en filtra les files corresponents. No es pot assegurar, però, l'ordre en que el SGBD facilitarà el resultat.

La clàusula `order by` permet especificar el criteri de classificació del resultat de la consulta.

Aquesta clàusula s'afegeix darrera de la clàusula `where` si n'hi ha, de manera que ampliem la sintaxis de la sentència `SELECT`:

```
select <expressió/columna>, <expressió/columna>,...
from <taula>, <taula>,...
[where <condició_de_recerca>]
[order by <expressió/columna> [ASC|desc], <expressió/columna> [asc|desc],...];
```

Com es pot observar, la clàusula `order by` permet ordenar la sortida segons diferents expressions i/o columnes, que han de ser calculables a partir dels valors de les columnes de les taules de la clàusula `from` encara que no apareguin en les columnes de la clàusula `select`.

Les expressions i/o columnes de la clàusula `order by` que apareixen a la clàusula `select` es poden referenciar pel número ordinal de la posició que ocupen a la clàusula `select` enlloc d'escriure'n el seu nom.

El criteri d'ordenació depèn del tipus de dada de l'expressió o columna i, per tant, podrà ser numèric o lexicogràfic.

Quan hi ha més d'un criteri d'ordenació (vàries expressions i/o columnes), es classifiquen d'esquerra a dreta.

La seqüència d'ordenació per defecte és ascendent per cada criteri. Es pot, però, especificar que la seqüència d'ordenació per un criteri sigui descendent amb la partícula `desc` a continuació del corresponent criteri. També es pot especificar la partícula `asc` per a indicar una seqüència d'ordenació ascendent, però és innecessari per ser la seqüència d'ordenació per defecte.

Exemple de classificació de resultats segons vàries columnes

Es desitja mostrar, en l'esquema *empresa*, els empleats ordenats de forma ascendent pel seu salari mensual i, ordenats pel cognom quan tinguin el mateix salari.

La instrucció per assolir l'objectiu és:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari"
from emp
order by salari, cognom;
```

En cas que hi hagi criteris d'ordenació que també apareixen en la clàusula `select` i tinguin un àlies definit, es pot utilitzar aquest àlies en la clàusula `order by`. Així doncs, tindríem:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari"
from emp
order by "Salari", "Empleat";
```

I, com hem dit més amunt, també podríem utilitzar l'ordinal:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari"
from emp
order by 3, 2;
```

Exemple de classificació de resultats segons expressions

Es desitja, en l'esquema *empresa*, mostrar els empleats amb el seu salari i comissió ordenats, de forma descendent, pel sou total mensual (salari+comissió).

La instrucció per assolir l'objectiu és:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari",
       comissio as "Comissió"
from emp
order by salari+NVL(comissio,0) desc;
```

Observis que si no utilitzem la funció `NVL`, també apareixen tots els empleats però tots els que no tenen comissió assignada apareixen agrupats a l'inici doncs el valor `NULL` és considerat superior a tots els valors i el resultat de la suma `salari+comissio` és `NULL` per les files que tenen `NULL` en la columna `comissio`.

2.3. Exclusió de files repetides. Opció `distinct` o `unique`.

La clàusula `select` permet decidir quines columnes seleccionar del producte cartesià de les taules especificades en la clàusula `from` i la clàusula `where` en filtra les files corresponents. El resultat, però, pot tenir files repetides, per les quals pot interessar tenir només un exemplar.

L'opció `distinct` o `unique` acompanyant a la clàusula `select` permet especificar que es vol un únic exemplar per les files repetides.

La sintaxis és:

```
select [distinct|unique] <expressió/columna>, <expressió/columna>,...
from <taula>, <taula>,...
[where <condició_de_recerca>]
[order by <expressió/columna> [asc|desc], <expressió/columna> [asc|desc],...];
```

La utilització de l'opció `distinct` o `unique` implica que el SGBD executi obligatòriament un `order by` sobre totes les columnes seleccionades (encara que no s'especifiqui la clàusula `order by`), fet que implica un cost d'execució addicional. Per tant, l'opció `distinct` o `unique` caldria utilitzar-la en cas que hi pugui haver files repetides i interessi un únic exemplar i s'hauria d'evitar en estar segur que no hi pot haver files repetides.

Exemple de la necessitat d'utilitzar l'opció `distinct` o `unique`

Com a exemple en el que cal utilitzar l'opció `distinct` o `unique` vegem com mostrar, en l'esquema *empresa*, els departaments (només el codi) en els que hi ha algun empleat.

La instrucció per assolir l'objectiu és:

```
select distinct dept_no as "Codi"
from emp;
```

Evidentment, la consulta no es pot efectuar sobre la taula dels departaments, ja que hi pot haver algun departament que no tingui cap empleat. Per aquest motiu l'executem sobre la taula dels empleats i hem d'utilitzar l'opció `distinct` ja que del contrari un mateix departament apareixeria tantes vegades com empleats tingués assignats.

2.4. Agrupaments de files. Clàusules `group by` i `having`.

Fins el moment hem vist com seleccionar files d'una taula o d'un producte cartesià de taules (clàusula `where`) i com quedar-nos amb les columnes interessants (clàusula `select`). Anem a veure com agrupar les files seleccionades i com filtrar per condicions sobre els grups.

La clàusula `group by` permet agrupar les files resultat de les clàusules `select`, `from` i `where` segons una o vàries de les columnes seleccionades.

La clàusula `having` permet especificar condicions de filtrat sobre els grups assolits per la clàusula `group by`.

Aquestes clàusules s'afegeixen darrera la clàusula `where` si n'hi ha i abans de la clàusula `order by` (si n'hi ha), de manera que ampliem la sintaxis de la sentència `SELECT`:

```
select [distinct] <expressió/columna>, <expressió/columna>,...
from <taula>, <taula>,...
[where <condició_de_recerca>]
[group by <àlies/columna>, <àlies/columna>,...]
[having <condició_sobre_grups>]
[order by <expressió/columna> [asc|desc], <expressió/columna> [asc|desc],...];
```

La taula 16 ens mostra les funcions d'agrupament que es pot utilitzar en les sentències `SELECT` de selecció de conjunts.

Taula 16. Funcions d'agrupament que es pot utilitzar en les sentències SELECT d'agrupament de files

Funció	Descripció	Exemples
`AVG (n)`	Retorna el valor mig de la columna `n` ignorant els valors nuls.	`AVG (salari)` retorna el salari mitjà de tots els empleats seleccionats que tenen salari (els nuls s'ignoren).

Funció	Descripció	Exemples
`COUNT ( [*|expr])`	Retorna el número de vegades que `expr` avalua alguna dada amb valor no nul. L'opció * comptabilitza totes les files seleccionades.	`COUNT (dept_no) (sobre la taula d'empleats) compta quants empleats estan assignats a algun departament.`
`MAX (expr)`	Retorna el valor màxim d'`expr`.	`MAX (salari) retorna el salari més alt.`
`MIN (expr)`	Retorna el valor mínim d'`expr`	`MIN (salari) retorna el salari més baix.`
`STDDEV (expr)`	Retorna la desviació típica d'`expr` sense tenir en compte els valors nuls.	`STDDEV (salari) retorna la desviació típica dels salaris.`
`SUM (expr)`	Retorna la suma dels valors d'`expr` sense tenir en compte els valors nuls	`SUM (salari) retorna la suma de tots els salaris.`
`VARIANCE (expr)`	Retorna la variància d'`expr` sense tenir en compte els valors nuls.	`VARIANCE (salari) retorna la variància dels salaris.`

L'expressió sobre la que es calculen les funcions d'agrupament pot anar precedida de l'opció `disctinct` per a indicar que s'avaluï sobre els valors diferents de l'expressió o de l'opció `all` per a indicar que s'avaluï sobre tots els valors de l'expressió. El valor per defecte és `all`.

Una sentència `SELECT` és una sentència de selecció de conjunts quan apareix la clàusula `group by` o la clàusula `having` o una funció d'agrupament, és a dir, una sentència `SELECT` pot ser sentència de selecció de conjunts encara que no hi hagi clàusula `group by`, cas en què es considera que hi ha un únic conjunt format per totes les files seleccionades.

Les columnes i/o expressions que no són funcions d'agrupament i que apareixen en una clàusula `select` d'una sentència `SELECT` de selecció de conjunts han d'aparèixer obligatòriament en la clàusula `group by` de la sentència. Ara bé, no totes les columnes i expressions de la clàusula `group by` tenen per què aparèixer en la clàusula `select`.

Exemple d'utilització de la funció `count()` sobre tota la consulta.

Es desitja, en l'esquema *empresa*, comptar quants empleats hi ha.

La instrucció per assolir l'objectiu és:

```
select count(*) as "Quants empleats" from emp;
```

Aquesta sentència `SELECT` és una sentència de selecció de conjunts malgrat que no aparegui la clàusula `group by`. En aquest cas, el SGBD ha agrupat totes les files en un únic conjunt per tal de poder-les comptar.

Exemple d'utilització de l'opció `distinct` en una funció d'agrupament

Es desitja, en l'esquema *empresa*, comptar quants oficis diferents hi ha.
La instrucció per assolir l'objectiu és:

```
select count (distinct ofici) as "Quants oficis" from emp;
```

En aquest cas és necessari indicar l'opció `distinct` doncs del contrari comptaria totes les files que tenen algun valor a la columna ofici, sense descartar els valors repetits.

Exemple d'utilització de la funció `count()` sobre una consulta amb grups

Es desitja, en l'esquema *empresa*, mostrar quants empleats hi ha de cada ofici.

La instrucció per assolir l'objectiu és:

```
select ofici as "Ofici", count (*) as "Quants empleats"
from emp
group by ofici;
```

El resultat obtingut és:

```
Ofici      Quants empleats
---------- ---------------------
EMPLEAT    4
VENEDOR    4
ANALISTA   2
PRESIDENT  1
DIRECTOR   3

5 rows selected
```

Exemple de coexistència de les clàusules `group by` i `order by`

Es desitja mostrar, en l'esquema *empresa*, els departaments que tenen empleats, acompanyats del salari més alt dels seus empleats i ordenats de forma ascendent pel màxim salari.

La instrucció per assolir l'objectiu és:

```
select dept_no, max (salari)
from emp
group by dept_no
order by max(salari);
```

O també:

```
select dept_no as "Codi", max (salari) as "Màxim salari"
from emp
group by dept_no
order by "Màxim salari";
```

O també:

```
select dept_no as "Codi", max (all salari) as "Màxim salari"
from emp
group by dept_no
order by "Màxim salari";
```

Exemple de coexistència de les clàusules `group by` i `order by`

Es desitja, en l'esquema *empresa*, comptar quants empleats hi ha de cada ofici en cada departament, veient el resultats ordenats per departament de forma ascendent i per nombre d'empleats de forma descendent.

La instrucció per assolir l'objectiu és:

```
select dept_no as "Codi", ofici as "Ofici",
       count (*) as "Quants empleats"
from emp
group by dept_no, ofici
order by dept_no, 3 desc;
```

Exemple de coexistència de les clàusules `group by` i `where`

Es desitja, en l'esquema *empresa*, mostrar quants empleats hi ha de cada ofici en el departament 20.

La instrucció per assolir l'objectiu és:

```
select ofici as "Ofici", count (*) as "Quants empleats"
from emp
where dept_no=20
group by ofici;
```

Exemple d'utilització de la clàusula `having`

Es desitja, en l'esquema *empresa*, mostrar el número d'empleats que hi ha de cada ofici pels oficis que tenen més d'un empleat.

La instrucció per assolir l'objectiu és:

```
select ofici as "Ofici", count (*) as "Quants empleats"
from emp
group by ofici
having count(*)>1;
```

2.5. Expressions amb sentències SELECT

El llenguatge SQL permet efectuar operacions sobre els resultats de les sentències `SELECT` per tal d'obtenir un nou resultat.

Tenim tres possibles operacions: *unió*, *intersecció* i *diferència*. A l'igual que en l'àlgebra relacional, els conjunts a unir, intersecar o restar han de ser compatibles (igual quantitat de columnes i columnes compatibles -tipus de dades equivalents- dos a dos).

2.5.1. Unió de SELECTs

El llenguatge SQL proporciona l'operador `union` per combinar totes les files del resultat d'una sentència `SELECT` amb totes les files del resultat d'una altra sentència `SELECT`, eliminant qualsevol duplicació de files que es pogués produir en el conjunt resultant.

La sintaxis és:

```
sentència_select_sense_order_by
union
sentència_select_sense_order_by
[order by ...]
```

El resultat final mostrarà, com a títols, els corresponents a les columnes de la primera sentència `SELECT`. Així doncs, en cas de voler assignar àlies a les columnes, només cal definir-los en la primera sentència `SELECT`.

Exemple de l'operació `union` entre sentències SQL

Es desitja, en l'esquema *sanitat*, presentar el personal que treballa en cada hospital, incloent el personal del plantilla i els doctors, i mostrant l'ofici que hi desenvolupen.

Una possible instrucció per assolir l'objectiu és:

```
select nom as "Hospital", 'Doctor' as "Ofici", doctor_no as "Codi",
       cognom "Empleat"
from hospital, doctor
where hospital.hospital_cod=doctor.hospital_cod
union
select nom, funcio, empleat_no, cognom
from hospital, plantilla
where hospital.hospital_cod=plantilla.hospital_cod
order by 1,2;
```

Observem que als doctors els hi hem assignat com a ofici la constant `'Doctor'`.

El resultat obtingut és:

```
Hospital   Ofici      Codi       Empleat
---------- ---------- ---------- --------------
General    Doctor     585        Miller G.
General    Doctor     982        Cajal R.
General    Intern     6357       Karplus W.
La Paz     Doctor     386        Cabeza D.
La Paz     Doctor     398        Best K.
La Paz     Doctor     453        Galo D.
La Paz     Infermer   8422       Bocina G.
La Paz     Infermera  1009       Higueras D.
La Paz     Infermera  6065       Rivera G.
La Paz     Infermera  7379       Carlos R.
La Paz     Intern     9901       Adams C.
Provincial Doctor     435        López A.
Provincial Infermer   3106       Hernández J.
Provincial Infermera  3754       Díaz B.
San Carlos Doctor     522        Adams C.
San Carlos Doctor     607        Nico P.
San Carlos Infermera  8526       Frank H.
San Carlos Intern     1280       Amigó R.

18 rows selected
```

2.5.2. Intersecció de SELECTs

El llenguatge SQL proporciona l'operador `intersect` per presentar les files que són, simultàniament, en dos conjunts resultats de sentències `SELECT`, files que són presentades una única vegada en el resultat final.

La sintaxis és:

```
sentència_select_sense_order_by
intersect
sentència_select_sense_order_by
[order by ...]
```

Exemple de l'operació `intersect` entre sentències SQL

Es desitja, en l'esquema *sanitat*, detectar la coincidència de cognoms entre doctors i empleats que treballin com a interns.

Una possible instrucció per assolir l'objectiu és:

```
select cognom as "Doctor-Intern" from doctor
intersect
select cognom from plantilla where funcio='Intern';
```

El resultat obtingut és:

```
Doctor-Intern
---------------
Adams C.

1 rows selected
```

2.5.3. Diferència de SELECTs

El llenguatge SQL proporciona l'operador `minus` per presentar les files que es troben en el primer i en canvi no es troben en el segon.

La sintaxis és:

```
sentència_select_sense_order_by
minus
sentència_select_sense_order_by
[order by ...]
```

Exemple de l'operació `minus` entre sentències SQL

Es desitja, en l'esquema *sanitat*, mostrar els empleats que treballen en qualsevol hospital com a *Intern* i que no tenen el torn del matí.

Una possible instrucció per assolir l'objectiu és:

```
select cognom as "Intern NO Matí" from plantilla
where funcio='Intern'
minus
select cognom from plantilla
where torn='M';
```

El resultat obtingut és:

```
Intern NO Matí
---------------
Amigó R.
Karplus W.

2 rows selected
```

Evidentment, la solució també s'obté amb la sentència:

```
select cognom as "Intern NO Matí" from plantilla
where funcio='Intern' and torn!='M';
```

2.6. Combinacions entre taules

La clàusula `from` efectua el producte cartesià de totes les taules que apareixen a la clàusula. La majoria de les ocasions no ens interessarà el producte cartesià, sinó únicament un subconjunt d'aquest.

Els tipus de subconjunts que ens poden interessar coincideixen amb els resultat de les combinacions *θ-join*, l'*equi-join*, el *natural-join* i *l'outer join* existents en l'àlgebra relacional.

Recordatori de les operacions en àlgebra relacional per combinar taules

Donades dues **relacions R i S**, es defineix el **θ-join de R segons l'atribut A i de S segons l'atribut Z**, i es nota per **R[AθZ]S** com el subconjunt de files del producte cartesià R x S que verifiquen A θ Z on θ és qualsevol dels operadors relacionals (`>, >=, <, <=, ==, !=`).

L'**equi-join** és un **θ-join** en el que l'operador θ és la igualtat. Es nota com **R[A=Z]S.**

El **natural-join** és un equi-join en el que l'atribut pel que s'executa la combinació només apareix una vegada en el resultat. Es nota com **R[A*Z]S.**

La notació **R*S** indica el natural-join per tots els atributs del mateix nom en les dues relacions.

En certes ocasions és menester tenir el resultat de l'**equi-join** ampliat amb totes les files d'una de les relacions que no tenen corresponent tupla en l'altre relació. Estem davant un **outer join** i tenim dues possibilitats (**left** o **right**) segons on es trobi (esquerra o dreta) la taula per la que han d'aparèixer totes les files encara que no tinguin correspondència en l'altre taula.

Donades dues **relacions R i S**, es defineix el **left-outer-join de R segons l'atribut A i de S segons l'atribut Z**, i es nota per **R[•A=Z]S,** com el subconjunt de files del producte cartesià R x S que verifiquen A = Z (resultat de R[A=Z]S) més les files de R que no tenen, per l'atribut A correspondència amb cap tupla de S segons l'atribut Z, les quals presenten valors NULL en els atributs provinents de S.

Donades dues **relacions R i S**, es defineix el **right-outer-join de R segons l'atribut A i de S segons l'atribut Z**, i es nota per **R[A=Z•]S,** com el subconjunt de files del producte cartesià R x S que verifiquen A = Z (resultat de R[A=Z]S) més les files de S que no tenen, per l'atribut Z correspondència amb cap tupla de R segons l'atribut A, les quals presenten valors NULL en els atributs provinents de R.

Podem considerar també, el **full-outer-join de R segons l'atribut A i de S segons l'atribut Z**, i es nota per **R[•A=Z•]S**, com la unió d'un **right-outer-join** i d'un **left-outer-join**. Recordem que en àlgebra relacional, la unió de conjunts no té en compte les files repetides. És a dir, amb un **full-outer-join** aconseguiríem tenir totes les files de les dues taules: les files que tenen correspondència pels atributs de la combinació i les files que no hi tenen correspondència.

Actualment tenim diverses formes d'efectuar combinacions entre taules, producte de l'evolució dels estàndards SQL i dels diversos SGBD comercials existents: les combinacions segons la norma SQL-87 i les combinacions segons la norma SQL-92.

2.6.1. Combinacions entre taules segons la norma SQL-87 (SQL-ISO)

El llenguatge SQL-86 ratificat per ISO en el 1987 establia que els diferents tipus de combinacions es podien assolir afegint, a la clàusula `where`, els filtres corresponents a les combinacions entre les columnes de les taules a combinar.

Exemple de combinació entre 2 taules segons SQL-87

Es desitja mostrar, en l'esquema *empresa*, els empleats (codi i cognom) juntament amb el codi i nom del departament al que pertanyen.

La instrucció per assolir l'objectiu és:

```
select emp.emp_no as "Codi empleat", emp.cognom as "Empleat",
       emp.dept_no as "Codi departament", dnom as "Descripció"
from emp, dept
where emp.dept_no = dept.dept_no;
```

Observem que l'accés a la taula `DEPT` és necessari per aconseguir el nom del departament. Tinguem en compte, també, que com que el camp `dept_no` de la taula `EMP` no permet valors NULL tots els empleats tindran valor en aquest camp i, a causa de la integritat referencial, no poden tenir un valor que no existeixi a la taula `DEPT`. Així doncs, una vegada executat el filtre de la clàusula `where`, totes les files de la taula `EMP` estaran combinades amb una fila de la taula `DEPT`.

El resultat obtingut és:

```
Codi empleat       Empleat    Codi departament       Descripció
------------------ ---------- ---------------------- --------------
7369               SÁNCHEZ    20                     INVESTIGACIÓ
7499               ARROYO     30                     VENDES
7521               SALA       30                     VENDES
7566               JIMÉNEZ    20                     INVESTIGACIÓ
7654               MARTÍN     30                     VENDES
7698               NEGRO      30                     VENDES
7782               CEREZO     10                     COMPTABILITAT
7788               GIL        20                     INVESTIGACIÓ
7839               REY        10                     COMPTABILITAT
7844               TOVAR      30                     VENDES
7876               ALONSO     20                     INVESTIGACIÓ
7900               JIMENO     30                     VENDES
7902               FERNÁNDEZ  20                     INVESTIGACIÓ
7934               MUÑOZ      10                     COMPTABILITAT

14 rows selected
```

Aplicant els filtres adequats en la clàusula `where` aconseguim implementar el *θ-join*, l'*equi-join* i el *natural-join* però no arribem a poder implementar els *outer-join*, doncs el producte cartesià no es pot "inventar" files amb valors nuls. Com solucionem aquest problema?

L'estàndard SQL-ISO (any 1987) va proposar (potser pressionat per *Oracle* que ja tenia implementada la solució) la utilització d'una marca, en la condició de combinació entre les columnes de les taules R i S a combinar, que indiqués la necessitat de fer aparèixer totes les files d'una taula (per exemple R), malgrat que per la columna de combinació no hi hagués correspondència en l'altra taula (S) , fent aparèixer valors nuls en les columnes de la taula S indicades en la clàusula `select`. La marca adoptada va ser el símbol `(+)` enganxat a la dreta de la taula (S) per la que cal generar valors nuls.

És a dir, un *left-outer-join* de la taula R amb la taula S segons les respectives columnes A (taula R) i Z (taula S), que en àlgebra relacional es notaria com **R[•A=Z]S,** es convertiria en SQL en una condició com:

```
where R.A = S.Z(+)
```

De la mateixa manera, un *right-outer-join* de la taula R amb la taula S segons les respectives columnes A (taula R) i Z (taula S), que en àlgebra relacional es notaria com **R[A=Z•]S,** es convertiria en SQL en una condició com:

```
where R.A(+) = S.Z
```

L'estàndard SQL-ISO no proporciona cap instrucció específica per assolir un *full-outer-join* entre dues taules i **NO** és permès escriure:

```
where R.A(+) = S.Z(+)
```

La solució en SQL-ISO per aconseguir un *full-outer-join* passa per a una operació `union` entre una sentència `SELECT` amb el *left-outer-join* i una sentència `SELECT` amb el *right-outer-join*. ‼

Exemple 1 de *right-outer-join* i *left-outer-join* entre taules segons SQL-ISO

Es desitja, en l'esquema *empresa*, mostrar tots els departaments (codi i descripció) acompanyats del salari més alt d'entre els seus empleats.

La instrucció per assolir l'objectiu és:

```
select d.dept_no as "Codi", dnom as "Departament",
       max(salari) as "Major salari"
from dept d, emp e
where d.dept_no = e.dept_no(+)
group by d.dept_no, dnom
order by 1;
```

L'*outer-join* en aquest cas és necessari ja que hi pot haver (i de fet hi ha) departaments sense empleat assignat. Si no es considera un *outer-join*, els departaments sense cap empleat no apareixerien. En canvi, amb l'*outer-join* apareixen i les dades corresponents a la taula dels empleats prenen valors `NULL`.

El resultat obtingut és:

```
Codi                 Departament    Major salari
-------------------- -------------- ----------------------
10                   COMPTABILITAT  650000
20                   INVESTIGACIÓ   390000
30                   VENDES         370500
40                   PRODUCCIÓ

4 rows selected
```

En la sentència anterior hem utilitzat un *left-outer-join*, el qual es pot convertir en un *right-outer-join* si canviem l'ordre de les taules a la clàusula `from`:

```
select d.dept_no as "Codi", dnom as "Departament",
       max(salari) as "Major salari"
from emp e, dept d
where e.dept_no(+) = d.dept_no
group by d.dept_no, dnom
order by 1;
```

El resultat obtingut en aquest cas és el mateix que en la sentència anterior.

És molt important copsar la necessitat d'utilitzar un *outer-join* en funció de les possibles dades que poden tenir les taules en qualsevol moment, segons la seva definició, i no pas en funció de les dades existents en un moment donat. ‼

Exemple 2 de *right-outer-join* i *left-outer-join* entre taules segons SQL-ISO

Es desitja, en l'esquema *empresa*, mostrar **tots** els empleats acompanyats dels clients dels quals són representants.

La instrucció per assolir l'objectiu és:

```
select emp_no as "Codi", cognom as "Empleat", client_cod as "Client", nom as "Raó social"
from emp, client
where emp_no = repr_cod (+)
order by 1,2;
```

O també:

```
select emp_no as "Codi", cognom as "Empleat", client_cod as "Client", nom as "Raó social"
from emp, client
where repr_cod (+) = emp_no
order by 1,2;
```

El resultat que s'obté, en ambdós casos, és:

```
Codi       Empleat    Client     Raó social
---------- ---------- ---------- ---------------------------------------
7369       SÁNCHEZ
7499       ARROYO     107        WOMEN SPORTS
7499       ARROYO     104        EVERY MOUNTAIN
7521       SALA       101        TKB SPORT SHOP
7521       SALA       103        JUST TENNIS
7521       SALA       106        SHAPE UP
7566       JIMÉNEZ
7654       MARTÍN     102        VOLLYRITE
7698       NEGRO
7782       CEREZO
7788       GIL
7839       REY
7844       TOVAR      108        NORTH WOODS HEALTH AND FITNESS SUPPLY CENTER
7844       TOVAR      105        K + T SPORTS
7844       TOVAR      100        JOCKSPORTS
7876       ALONSO
7900       JIMENO
7902       FERNÁNDEZ
7934       MUÑOZ

19 rows selected
```

Observem que per assegurar l'aparició de tots els empleats, cal utilitzar un *outer join*, doncs del contrari els empleats que no tenen assignat cap client, no apareixerien.

Exemple 3 de *right-outer-join* i *left-outer-join* entre taules segons SQL-ISO

Es desitja, en l'esquema *empresa*, mostrar **tots** els clients acompanyats de l'empleat que tenen com a representant.

La instrucció per assolir l'objectiu és:

```
select client_cod as "Client", nom as "Raó social", emp_no as "Codi", cognom as "Empleat"
from client, emp
where repr_cod = emp_no (+);
```

O també

```
select client_cod as "Client", nom as "Raó social", emp_no as "Codi", cognom as "Empleat"
from client, emp
where emp_no (+) = repr_cod;
```

El resultat que s'obté és:

```
Client     Raó social                                 Codi       Empleat
---------- ------------------------------------------ ---------- ----------
107        WOMEN SPORTS                               7499       ARROYO
104        EVERY MOUNTAIN                             7499       ARROYO
106        SHAPE UP                                   7521       SALA
103        JUST TENNIS                                7521       SALA
101        TKB SPORT SHOP                             7521       SALA
102        VOLLYRITE                                  7654       MARTÍN
108        NORTH WOODS HEALTH AND FITNESS SUPPLY CENTER 7844     TOVAR
105        K + T SPORTS                               7844       TOVAR
100        JOCKSPORTS                                 7844       TOVAR
109        SPRINGFIELD NUCLEAR POWER PLANT

10 rows selected
```

Observem que per assegurar l'aparició de tots els clients, cal utilitzar un *outer join*, doncs del contrari els clients que no tenen assignat representant, no apareixerien.

Exemple de *full-outer-join* entre taules segons SQL-ISO

Es desitja, en l'esquema *empresa*, mostrar **tots** els clients i **tots** els empleats relacionant cada client amb el seu representant (i, de retruc, cada empleat amb els seus clients).
La instrucció per assolir l'objectiu és:

```
select client_cod as "Client", nom as "Raó social", emp_no as "Codi", cognom as "Empleat"
from client, emp
where emp_no (+) = repr_cod
union
```

```
select client_cod as "Client", nom as "Raó social", emp_no as "Codi", cognom as "Empleat"
from client, emp
where emp_no = repr_cod (+);
```

El resultat obtingut és:

```
Client     Raó social                                    Codi       Empleat
---------- -------------------------------------------- ---------- ----------
100        JOCKSPORTS                                    7844       TOVAR
101        TKB SPORT SHOP                                7521       SALA
102        VOLLYRITE                                     7654       MARTÍN
103        JUST TENNIS                                   7521       SALA
104        EVERY MOUNTAIN                                7499       ARROYO
105        K + T SPORTS                                  7844       TOVAR
106        SHAPE UP                                      7521       SALA
107        WOMEN SPORTS                                  7499       ARROYO
108        NORTH WOODS HEALTH AND FITNESS SUPPLY CENTER  7844       TOVAR
109        SPRINGFIELD NUCLEAR POWER PLANT
                                                         7369       SÁNCHEZ
                                                         7566       JIMÉNEZ
                                                         7698       NEGRO
                                                         7782       CEREZO
                                                         7788       GIL
                                                         7839       REY
                                                         7876       ALONSO
                                                         7900       JIMENO
                                                         7902       FERNÁNDEZ
                                                         7934       MUÑOZ
20 rows selected
```

Oracle és dels pocs SGBD (potser l'únic) que permet la utilització del símbol (+) de l'estàndard SQL-ISO per als *outer-joins*. De fet, abans de la versió 9 d'*Oracle*, era la única forma de gestionar *outer-joins*. (!!)

2.6.2. Combinacions entre taules segons la norma SQL-92

No tots els SGBD van seguir la modalitat de la marca (+) per implementar les dues variants d'*outer-join*. I és comprensible donat que hi ha una forma més entenedora d'explicar el per què de les combinacions entre vàries taules.

En moltes ocasions, en la combinació entre dues taules hi ha una taula que es pot considerar la taula principal (on hem d'anar a cercar la informació) i una altra taula que es pot considerar secundària (on hem d'anar a cercar informació que complementi la informació cercada a la taula principal).

Per aconseguir el nostre propòsit, el llenguatge SQL facilita, des de la revisió del 92 (SQL-92), les operacions `join` en la clàusula `from` indicant la condició de combinació, la qual no s'haurà d'indicar ja (excepte en un cas) dins la clàusula `where`.

És a dir, passem d'una sentència `SELECT` que inclou una part similar a:

```
...
from taula1, taula2
where <condició combinació entre taula1 i taula2>
...
```

a una sentència `SELECT` on la condició de combinació entre taules ja no s'indica a la clàusula `where` sinó que acompanya a la clàusula `from`:

```
...
from taula1 [ cross | inner | left | right | full ] join taula2
on <condició combinació entre taula1 i taula2>
where ...
```

Com s'observa a la sintaxis anterior, hi ha diferents opcions de `join`: `cross`, `inner`, `left`, `right` i `full`.

Els actuals SGBD relacionals (*Oracle*, *MySQL*, *SQLServer*, *PostgreSQL*, *MsAccess*,...) incorporen les combinacions `inner`, `left` i `right` amb les que es pot aconseguir tots els tipus de combinacions entre taules. La resta d'opcions són suportades per alguns SGBD però no sempre seguint idèntica sintaxis. La sintaxis aquí presentada és la facilitada pel SGBD *Oracle* a partir de la versió 9 i fins l'actual 11g.

La combinació `cross` coincideix amb la combinació existent en SQL-87 en la que cal continuar indicant la condició de combinació en la clàusula `where`. (!!)

Exemple de combinació `cross` entre 2 taules

Es desitja mostrar, en l'esquema *empresa*, els empleats (codi i cognom) juntament amb el codi i nom del departament al que pertanyen.

```
select emp.emp_no as "Codi empleat", emp.cognom as "Empleat",
       emp.dept_no as "Codi departament", dnom as "Descripció"
from emp cross join dept
where emp.dept_no = dept.dept_no;
```

Observem que la columna corresponent al codi de departament només apareix una vegada ja que només l'hem indicat una vegada en la clàusula `select`.

La combinació `inner` és la més comuna i de fet és la que s'executa si no s'indica cap de les opcions. S'anomena *combinació interna* i combina files de dues taules sempre que hi hagi valors coincidents en el(s) camp(s) de combinació. (!!)

Exemple de combinació `inner` entre 2 taules

Es desitja mostrar, en l'esquema *empresa*, els empleats (codi i cognom) juntament amb el codi i nom del departament al que pertanyen.

```
select emp.emp_no as "Codi empleat", emp.cognom as "Empleat",
       emp.dept_no as "Codi departament", dnom as "Descripció"
from emp inner join dept on emp.dept_no = dept.dept_no;
```

Observem que la columna corresponent al codi de departament només apareix una vegada ja que només l'hem indicat una vegada en la clàusula `select`. Recordem també que no és obligatori utilitzar la paraula `inner`.

Les combinacions `left join`, `right join` i `full join` (anomenades *combinacions externes*) són les opcions que facilita SQL-92 per assolir les diverses opcions d'*outer-join*. (!!)

La combinació `left join` permet combinar *totes* les files de la taula de l'esquerra del `join` amb les files amb valors coincidents de la taula de la dreta, facilitant valors nuls per a les columnes de la taula de la dreta quan no hi ha files amb valors coincidents.

La combinació `right join` permet combinar *totes* les files de la taula de la dreta del `join` amb les files amb valors coincidents de la taula de l'esquerra, facilitant valors nuls per a les columnes de la taula de l'esquerra quan no hi ha files amb valors coincidents.

La combinació `full join` és la unió del `left join` i `right join` eliminant la duplicitat de files a causa de les files d'ambdues taules que tenen valors coincidents.

Exemple 1 de *right-outer-join* i *left-outer-join* entre taules segons SQL-92

Es desitja, en l'esquema empresa, mostrar tots els departaments (codi i descripció) acompanyats del salari més alt d'entre els seus empleats.

```
select d.dept_no as "Codi", dnom as "Departament",
       max(salari) as "Major salari"
from dept d left join emp e on d.dept_no = e.dept_no
group by d.dept_no, dnom
order by 1;
```

El resultat obtingut és:

```
Codi       Departament    Major salari
---------- -------------- ----------------------
10         COMPTABILITAT  650000
20         INVESTIGACIÓ   390000
30         VENDES         370500
40         PRODUCCIÓ

4 rows selected
```

En la sentència anterior hem utilitzat un `left join`, el qual es pot convertir en un `right join` si canviem l'ordre de les taules a la clàusula `from`:

```
select d.dept_no as "Codi", dnom as "Departament",
       max(salari) as "Major salari"
from emp e right join dept d on e.dept_no = d.dept_no
group by d.dept_no, dnom
order by 1;
```

El resultat obtingut en aquest cas és el mateix que en la sentència anterior.

Exemple 2 de *right-outer-join* i *left-outer-join* entre taules segons SQL-92

Es desitja, en l'esquema *empresa*, mostrar **tots** els empleats acompanyats dels clients dels quals són representants.

La instrucció per assolir l'objectiu és:

```
select emp_no as "Codi", cognom as "Empleat", client_cod as "Client", nom as "Raó social"
from emp left join client on emp_no = repr_cod
order by 1,2;
```

O també:

```
select emp_no as "Codi", cognom as "Empleat", client_cod as "Client", nom as "Raó social"
from client right join emp on repr_cod = emp_no
order by 1,2;
```

El resultat que s'obté, en ambdós casos, és:

```
Codi       Empleat    Client                Raó social
---------- ---------- ---------- ---------------------------------------
7369       SÁNCHEZ
7499       ARROYO     107        WOMEN SPORTS
7499       ARROYO     104        EVERY MOUNTAIN
7521       SALA       101        TKB SPORT SHOP
7521       SALA       103        JUST TENNIS
7521       SALA       106        SHAPE UP
7566       JIMÉNEZ
7654       MARTÍN     102        VOLLYRITE
7698       NEGRO
7782       CEREZO
7788       GIL
7839       REY
7844       TOVAR      108        NORTH WOODS HEALTH AND FITNESS SUPPLY CENTER
7844       TOVAR      105        K + T SPORTS
7844       TOVAR      100        JOCKSPORTS
7876       ALONSO
7900       JIMENO
7902       FERNÁNDEZ
7934       MUÑOZ

19 rows selected
```

Observem que per assegurar l'aparició de tots els empleats, cal utilitzar un *outer join*, doncs del contrari els empleats que no tenen assignat cap client, no apareixerien.

Exemple 3 de *right-outer-join* i *left-outer-join* entre taules segons SQL-92

Es desitja, en l'esquema *empresa*, mostrar **tots** els clients acompanyats de l'empleat que tenen com a representant.

La instrucció per assolir l'objectiu és:

```
select client_cod as "Client", nom as "Raó social", emp_no as "Codi", cognom as "Empleat"
from client left join emp on repr_cod = emp_no;
```

O també:

```
select client_cod as "Client", nom as "Raó social", emp_no as "Codi", cognom as "Empleat"
from emp right join client on emp_no = repr_cod;
```

El resultat que s'obté és:

```
Client     Raó social                                    Codi       Empleat
---------- --------------------------------------------- ---------- ----------
107        WOMEN SPORTS                                  7499       ARROYO
104        EVERY MOUNTAIN                                7499       ARROYO
106        SHAPE UP                                      7521       SALA
103        JUST TENNIS                                   7521       SALA
101        TKB SPORT SHOP                                7521       SALA
102        VOLLYRITE                                     7654       MARTÍN
108        NORTH WOODS HEALTH AND FITNESS SUPPLY CENTER  7844       TOVAR
105        K + T SPORTS                                  7844       TOVAR
100        JOCKSPORTS                                    7844       TOVAR
109        SPRINGFIELD NUCLEAR POWER PLANT

10 rows selected
```

Observem que per assegurar l'aparició de tots els clients, cal utilitzar un *outer join*, doncs del contrari els clients que no tenen assignat representant, no apareixerien.

Exemple 4 de *full outer-join* entre taules segons SQL-92

Es desitja, en l'esquema *empresa*, mostrar **tots** els clients i **tots** els empleats relacionant cada client amb el seu representant (i, de retruc, cada empleat amb els seus clients).
La instrucció per assolir l'objectiu és:

```
select client_cod as "Client", nom as "Raó social", emp_no as "Codi", cognom as "Empleat"
from client full join emp on emp_no = repr_cod
order by 1,3;
```

El resultat obtingut és:

```
Client     Raó social                                    Codi       Empleat
---------- -------------------------------------------- ---------- ----------
100        JOCKSPORTS                                   7844       TOVAR
101        TKB SPORT SHOP                               7521       SALA
102        VOLLYRITE                                    7654       MARTÍN
103        JUST TENNIS                                  7521       SALA
104        EVERY MOUNTAIN                               7499       ARROYO
105        K + T SPORTS                                 7844       TOVAR
106        SHAPE UP                                     7521       SALA
107        WOMEN SPORTS                                 7499       ARROYO
108        NORTH WOODS HEALTH AND FITNESS SUPPLY CENTER 7844       TOVAR
109        SPRINGFIELD NUCLEAR POWER PLANT
                                                        7369       SÁNCHEZ
                                                        7566       JIMÉNEZ
                                                        7698       NEGRO
                                                        7782       CEREZO
                                                        7788       GIL
                                                        7839       REY
                                                        7876       ALONSO
                                                        7900       JIMENO
                                                        7902       FERNÁNDEZ
                                                        7934       MUÑOZ
20 rows selected
```

El llenguatge SQL facilita dues simplificacions en l'escriptura de les combinacions `join` que precisen de l'opció `on` (totes menys en el `cross join`), pels casos en que les columnes a combinar tinguin coincidència de noms:

– Si la combinació es vol efectuar per a *totes* les columnes que tinguin noms coincidents en les taules a combinar, disposem de les combinacions `natural join`. (!!)

  En aquest cas, la paraula `natural` davant el tipus de combinació `join` (`natural inner join`, `natural left join`, `natural right join`, `natural full join`) provoca que s'efectuï la combinació entre les taules per a *totes* les columnes que tenen coincidència de nom, sense haver d'indicar la condició de combinació.

  El resultat dels `natural join` facilita una única columna per aquelles columnes de les taules combinades que tenen el mateix nom i, per tant, en cas d'haver de fer referència a aquesta columna en alguna clàusula de la sentència `SELECT`, no s'ha d'indicar el nom de la taula a la que pertany, doncs pertany simultàniament a vàries taules.

– Si la combinació es vol efectuar per a algunes de les columnes que tinguin noms coincidents en les taules a combinar, disposem de les combinacions `join` amb l'opció `using (col1, col2...)`. (!!)
  En aquest cas, l'opció `using (col1, col2...)` provoca que s'efectuï la combinació entre les taules per a les columnes indicades, sense haver d'indicar la condició de combinació. Com en els `natural join`, també dona com a resultat una única columna per a les columnes coincidents.

Exemple de simplificació d'una combinació `inner join`

La sentència següent, corresponent a l'esquema *empresa*, mostra els empleats (codi i cognom) juntament amb el codi i nom del departament al que pertanyen.

```
select emp.emp_no as "Codi empleat", emp.cognom as "Empleat",
       emp.dept_no as "Codi departament", dnom as "Descripció"
from emp inner join dept on emp.dept_no = dept.dept_no;
```

Com que a les taules `EMP` i `DEPT` hi ha coincidència de nom per a la columna de combinació `dept_no` i no hi ha altres columnes amb noms coincidents, podem utilitzar un `natural join`:

```
select emp.emp_no as "Codi empleat", emp.cognom as "Empleat",
       dept_no as "Codi departament", dnom as "Descripció"
from emp natural inner join dept;
```

Observem que en la clàusula `select` s'ha hagut de suprimir, en visualitzar la columna `dept_no`, la referència a la taula a la que pertany, doncs el `natural join` només facilita una de les dues columnes amb coincidència de noms.

Però també s'hagués pogut utilitzar l'opció `using`:

```
select emp.emp_no as "Codi empleat", emp.cognom as "Empleat",
       dept_no as "Codi departament", dnom as "Descripció"
from emp inner join dept using (dept_no);
```

Així mateix, en ambdós casos, en tractar-se d'una combinació `inner join`, ens podem estalviar el mot `inner`, i tindríem:

```
select emp.emp_no as "Codi empleat", emp.cognom as "Empleat",
       dept_no as "Codi departament", dnom as "Descripció"
from emp natural join dept;

select emp.emp_no as "Codi empleat", emp.cognom as "Empleat",
       dept_no as "Codi departament", dnom as "Descripció"
from emp join dept using (dept_no);
```

Exemple de simplificació de combinacions `left join` i `right join`

Les sentències següents, corresponent a l'esquema empresa, mostren tots els departaments (codi i descripció) acompanyats del salari més alt d'entre els seus empleats.

```
select d.dept_no as "Codi", dnom as "Departament",
       max(salari) as "Major salari"
from dept d left join emp e on d.dept_no = e.dept_no
group by d.dept_no, dnom
order by 1;

select d.dept_no as "Codi", dnom as "Departament",
       max(salari) as "Major salari"
from emp e right join dept d on e.dept_no = d.dept_no
group by d.dept_no, dnom
order by 1;
```

Donat que la columna de combinació té el mateix nom i no hi ha altres columnes amb coincidència de noms, haguéssim pogut emprar un `natural join`:

```
select dept_no as "Codi", dnom as "Departament",
       max(salari) as "Major salari"
from dept natural left join emp e
group by dept_no, dnom
order by 1;
select dept_no as "Codi", dnom as "Departament",
       max(salari) as "Major salari"
from emp e natural right join dept d
group by dept_no, dnom
order by 1;
```

I també, utilitzant l'opció `using`:

```
select dept_no as "Codi", dnom as "Departament",
       max(salari) as "Major salari"
from dept left join emp e using (dept_no)
group by dept_no, dnom
order by 1;

select dept_no as "Codi", dnom as "Departament",
       max(salari) as "Major salari"
from emp e right join dept d using (dept_no)
group by dept_no, dnom
order by 1;
```

2.7. Subconsultes

En certes ocasions és necessari executar una sentència `SELECT` per aconseguir un resultat que cal utilitzar com a part de la condició de filtrat d'una altra sentència `SELECT`. El llenguatge SQL ens facilita efectuar aquest tipus d'operacions amb la utilització de les **subconsultes**.

Una subconsulta és una sentència `SELECT` que s'inclou en la clàusula `where` d'una altra sentència `SELECT`. La subconsulta es tanca entre parèntesis i no inclou el punt i coma finals. (!!)

Una subconsulta pot contenir, a la vegada, altres subconsultes.

Exemple de subconsulta que calcula un resultat a utilitzar en una clàusula `where`

Es demana, en l'esquema *empresa*, mostrar els empleats que tenen salari igual o superior al salari mig de l'empresa.

La instrucció per assolir l'objectiu és:

```
select emp_no as "Codi", cognom as "Empleat", salari as "Salari"
from emp
where salari >= (select avg(salari) from emp)
order by 3 desc,1;
```

En certes situacions pot ser necessari accedir des de la subconsulta als valors de les columnes seleccionades en la consulta. El llenguatge SQL ho permet sense problemes i, en cas que els noms de les columnes coincideixen, es poden utilitzar àlies.

Els noms de columnes que apareixen en les clàusules d'una subconsulta s'intenten avaluar, en primer lloc, com a columnes de les taules definides en la clàusula `from` de la subconsulta, a no ser que vagin acompanyades d'àlies que les identifiquin com a columnes d'una taula en la consulta contenidora.

Exemple de subconsulta que fa referència a columnes de la consulta contenidora

Es demana, en l'esquema *empresa*, mostrar els empleats de cada departament que tenen salari menor que el salari mig del mateix departament.

La instrucció per assolir l'objectiu és:

```
select dept_no as "Dept.", emp_no as "Codi", cognom as "Empleat",
       salari as "Salari"
from emp e1
where salari >= (select avg(salari)
                 from emp e2
                 where dept_no=e1.dept_no
                )
order by 1, 4 desc,2;
```

Els valors retornats per les subconsultes s'utilitzen en les clàusules `where` com a part dreta d'operacions de comparacions en les que intervenen els operadors `=, !=, <, <=, >, >=, [not] in, <op> any` i `<op> all`.

Les subconsultes també poden vincular-se a la consulta contenidora per la partícula `[not] exists:`

```
...
where [not] exists (subconsulta)
```

En tal cas, la subconsulta acostuma a fer referència a valors de les taules de la consulta contenidora. S'anomenen **subconsultes sincronitzades.**

Les consultes que poden donar com a resultat un únic valor o cap, poden actuar com a subconsultes en expressions on el valor resultat es compara amb qualsevol operador de comparació.

Les consultes que poden donar com a resultat més d'un valor (encara que en execucions concretes només en donin un) mai poden actuar com a subconsultes en expressions on els valors resultats es comparen amb l'operador =, ja que el SGBDR no sabria amb quin dels resultats efectuar la comparació d'igualtat i es produiria un error similar a:

```
ORA-01427: una subconsulta d'única fila en retorna més d'una
```

Si cal aprofitar els resultats de més d'una columna de la subconsulta, aquesta es col·loca a la dreta de l'operació de comparació i a la part esquerra es col·loquen els valors a comparar, en el mateix ordre que els valors retornats per la subconsulta, separats per comes i tancats entre parèntesi:

```
...
where (valor1, valor2...) <op> (select col1, col2...)
```

Exemple d'utilització de l'operador = per a comparar amb el resultat d'una subconsulta

Es desitja, en l'esquema empresa, mostrar els empleats que tenen el mateix ofici que l'ofici que té l'empleat de cognom 'ALONSO'.

La instrucció per assolir l'objectiu sembla que podria ser:

```
select cognom as "Empleat"
from emp
where ofici = (select ofici
               from emp
               where upper(cognom)='ALONSO')
  and upper(cognom)!='ALONSO';
```

En aquesta sentència hem utilitzat l'operador = de manera errònia, ja que no podem estar segurs que no hi hagi dos empleats amb cognom 'ALONSO'. Com que només n'hi ha un, la sentència s'executa correctament, però en cas que n'hi hagués més d'un, fet que pot succeir en qualsevol moment, l'execució de la sentència provocaria l'error abans esmentat.

Per tant, hauríem de cercar un altre operador de comparació per evitar aquest problema:

```
select cognom as "Empleat"
from emp
where ofici in (select ofici
                from emp
                where upper(cognom)='ALONSO')
  and upper(cognom)!='ALONSO';
```

O també:

```
select cognom as "Empleat"
from emp e
where exists (select *
              from emp
              where upper(cognom)='ALONSO'
                and ofici=e.ofici
             )
  and upper(cognom)!='ALONSO';
```

Exemple d'utilització dels operadors `ANY` i `EXISTS`

Es demana, en l'esquema *empresa*, mostrar els noms i oficis dels empleats del departament 20 que la seva feina coincideixi amb la d'algun empleats del departament de 'VENDES'.

La instrucció per assolir l'objectiu pot ser:

```
select cognom as "Empleat", ofici as "Ofici"
from emp
where dept_no=20
  and ofici =ANY (select ofici
                  from emp
                  where dept_no =ANY (select dept_no
                                      from dept
                                      where upper(dnom)='VENDES'
                                     )
                 );
```

Aquesta instrucció està pensada per a que el resultat sigui correcte en cas que hi pugui haver varis departaments amb nom 'VENDES'. En cas que la columna `dnom` taula `DEPT` tingués definida la restricció d'unicitat també seria correcta la instrucció:

```
select cognom as "Empleat", ofici as "Ofici"
from emp
where dept_no=20
  and ofici =ANY (select ofici
                  from emp
                  where dept_no = (select dept_no
                                   from dept
                                   where upper(dnom)='VENDES'
                                  )
                 );
```

Una altra manera de resoldre el mateix problema és amb la uitilització de l'operador `EXISTS`:

```
select cognom as "Empleat", ofici as "Ofici"
from emp e
where dept_no=20
  and EXISTS (select *
```

```
              from emp, dept
              where emp.dept_no=dept.dept_no
                and upper(dnom)='VENDES'
                and ofici=e.ofici
             );
```

Exemple d'utilització de l'operador `IN`

Es demana, en l'esquema *empresa*, mostrar els empleats amb el mateix ofici i salari que 'JIMÉNEZ'.

La instrucció per assolir l'objectiu pot ser:

```
select emp_no "Codi", cognom "Empleat"
from emp
where (ofici,salari) IN (select ofici, salari
                         from emp
                         where upper(cognom)='JIMÉNEZ'
                        )
  and upper(cognom)!='JIMÉNEZ';
```

Exemple de condició complexa de filtrat amb diverses subconsultes i operacions

Es demana, en l'esquema *empresa*, mostrar els empleats que efectuïn la mateixa feina que 'JIMÉNEZ' o que tinguin un salari igual o superior al de 'FERNÁNDEZ'.

```
select emp_no "Codi", cognom "Empleat"
from emp
where (ofici IN (select ofici
                 from emp
                 where upper(cognom)='JIMÉNEZ'
                )
       and upper(cognom)!='JIMÉNEZ'
      )
   or (salari>= (select salari
                 from emp
                 where upper(cognom)='FERNÁNDEZ'
                )
       and upper(cognom)!='FERNÁNDEZ'
      );
```

2.8. Recuperació jeràrquica

Situem-nos, per un moment, en la fase de disseny d'una base de dades. En certes ocasions tenim entitats en les que s'estableix una relació reflexiva: *ser fill de*, *ser pare de*, *ser mare de*, *ser cap de*, *ser subordinat de*,... Les instàncies concretes d'aquestes relacions provoquen l'aparició d'una estructura jeràrquica (com un arbre genealògic). Aquests dissenys, transferits al model relacional, provoquen l'aparició de taules en les que un o varis camps són clau forana de la pròpia taula.

En certes ocasions i davant la situació anterior, ens podem trobar en la necessitat d'efectuar una recuperació de les files seguint un recorregut en preordre per l'arbre equivalent. El llenguatge SQL estàndard no facilita aquesta recuperació, però *Oracle* sí la incorpora.

La **recuperació jeràrquica** que facilita el SGBD *Oracle* precisa de la clàusula `connect by` que es situa immediatament després de la clàusula `where`:

```
select <expressió/columna>, <expressió/columna>,...
from <taula>, <taula>,...
where <condició_de_recerca>
connect by <condició_de_connexió> start with <condició_inicial>
```

La `<condició_de_connexió>` defineix la relació entre els camps que són clau forana i els camps que són clau primària. Aquests últims han de portar la partícula `prior` al davant per indicar que són la clau primària.

La `<condició_inicial>` defineix el node de l'arbre a partir del qual s'efectua el recorregut. Si l'apartat `start with` no s'indica, s'efectua el recorregut per cada tupla normalment seleccionada.

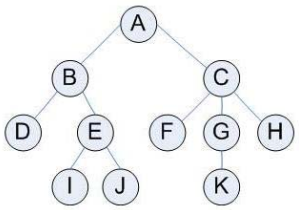
El SGBD *Oracle* gestiona automàticament la variable `level` per indicar el nivell de les files recuperades seguint el recorregut en preordre.

La recuperació jeràrquica és possible sempre i quan les dades de la taula segueixin una relació reflexiva tot i que la integritat referencial no hagi estat definida.

Recorregut en preordre per un arbre

El recorregut en preordre per un arbre consisteix a visitar, en primer lloc, l'arrel i, posteriorment, tots subarbres fills de l'arrel, també en preordre.

Així, el recorregut en preordre de l'arbre següent seria A, B, D, E, I, J, C, F, G, K, H.



Exemple de recuperació jeràrquica

Es demana, en l'esquema *empresa*, mostrar la jerarquia d'empleats que penja del 'PRESIDENT' de l'empresa (empleat amb ofici='PRESIDENT').

La instrucció adequada per aconseguir l'objectiu és:

```
select emp_no as "Codi", cognom as "Empleat", ofici as "Ofici",
       dept_no as "Dept.", level as "Nivell", cap as "Superior"
from emp
connect by prior emp_no = cap
start with upper(ofici)='PRESIDENT';
```

O també:

```
select emp_no as "Codi", cognom as "Empleat", ofici as "Ofici",
       dept_no as "Dept.", level as "Nivell", cap as "Superior"
from emp
connect by cap = prior emp_no
start with upper(ofici)='PRESIDENT';
```

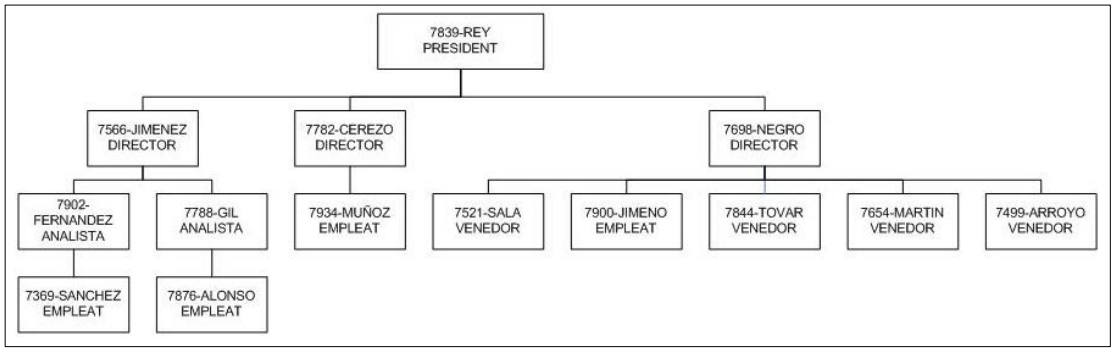
El resultat obtingut és:

```
Codi Empleat    Ofici      Dept.      Nivell     Superior
---- ---------- ---------- ---------- ---------- ----------------
7839 REY        PRESIDENT  10         1
7566 JIMÉNEZ    DIRECTOR   20         2          7839
7788 GIL        ANALISTA   20         3          7566
7876 ALONSO     EMPLEAT    20         4          7788
7902 FERNÁNDEZ  ANALISTA   20         3          7566
7369 SÁNCHEZ    EMPLEAT    20         4          7902
7698 NEGRO      DIRECTOR   30         2          7839
7499 ARROYO     VENEDOR    30         3          7698
7521 SALA       VENEDOR    30         3          7698
7654 MARTÍN     VENEDOR    30         3          7698
7844 TOVAR      VENEDOR    30         3          7698
7900 JIMENO     EMPLEAT    30         3          7698
7782 CEREZO     DIRECTOR   10         2          7839
```

```
7934 MUÑOZ       EMPLEAT     10          3           7782

14 rows selected
```

Fixem-nos que el resultat obtingut ens permet efectuar un gràfic de la jerarquia existent a l'empresa, com ens mostra la figura 4.

Figura 4. Organigrama dels empleats l'esquema empresa construït amb la informació obtinguda per una recuperació jeràrquica



En cas de no indicar l'opció `start with`, *Oracle* facilita la recuperació jeràrquica per a cada fila seleccionada:

```
Codi Empleat    Ofici       Dept.       Nivell      Superior
---- ---------- ---------- ---------- ---------- ----------------------
7788 GIL        ANALISTA   20          1          7566
7876 ALONSO     EMPLEAT    20          2          7788
7902 FERNÁNDEZ  ANALISTA   20          1          7566
7369 SÁNCHEZ    EMPLEAT    20          2          7902
7499 ARROYO     VENEDOR    30          1          7698
7521 SALA       VENEDOR    30          1          7698
7654 MARTÍN     VENEDOR    30          1          7698
7844 TOVAR      VENEDOR    30          1          7698
7900 JIMENO     EMPLEAT    30          1          7698
7934 MUÑOZ      EMPLEAT    10          1          7782
7876 ALONSO     EMPLEAT    20          1          7788
7566 JIMÉNEZ    DIRECTOR   20          1          7839
7788 GIL        ANALISTA   20          2          7566
7876 ALONSO     EMPLEAT    20          3          7788
7902 FERNÁNDEZ  ANALISTA   20          2          7566
7369 SÁNCHEZ    EMPLEAT    20          3          7902
7698 NEGRO      DIRECTOR   30          1          7839
7499 ARROYO     VENEDOR    30          2          7698
7521 SALA       VENEDOR    30          2          7698
7654 MARTÍN     VENEDOR    30          2          7698
7844 TOVAR      VENEDOR    30          2          7698
7900 JIMENO     EMPLEAT    30          2          7698
7782 CEREZO     DIRECTOR   10          1          7839
7934 MUÑOZ      EMPLEAT    10          2          7782
7369 SÁNCHEZ    EMPLEAT    20          1          7902
7839 REY        PRESIDENT  10          1
7566 JIMÉNEZ    DIRECTOR   20          2          7839
7788 GIL        ANALISTA   20          3          7566
7876 ALONSO     EMPLEAT    20          4          7788
7902 FERNÁNDEZ  ANALISTA   20          3          7566
7369 SÁNCHEZ    EMPLEAT    20          4          7902
7698 NEGRO      DIRECTOR   30          2          7839
7499 ARROYO     VENEDOR    30          3          7698
7521 SALA       VENEDOR    30          3          7698
7654 MARTÍN     VENEDOR    30          3          7698
7844 TOVAR      VENEDOR    30          3          7698
7900 JIMENO     EMPLEAT    30          3          7698
7782 CEREZO     DIRECTOR   10          2          7839
7934 MUÑOZ      EMPLEAT    10          3          7782

39 rows selected
```