TEMA 4: SQL

1. Introducció

Moltes vegades hem sentit estadístiques de futbol que diuen coses com:

- ✓ *Esta jornada s'han tret 7 targetes roges, que és rècord esta temporada i igualen el rècord de la temporada passada, que es va fer en la jornada 5.*
- ✓ *El partit d'esta jornada amb més expulsions ha sigut el Sevilla-València, amb 3. En total han sigut expulsats 11 jugadors, però no supera el rècord de la temporada passada, que eren 12 expulsions, en les jornades 6 i 33.*
- ✓ *En el que va de campionat, Messi ja porta més gols que 15 equips de primera. Només 3 clubs superen els 8 gols de Messi: el Betis amb 9, el Reial Madrid amb 16 i, és clar, el Barça, amb 22. Messi acabà la lliga passada amb 31 gols. De moment, en la present temporada la seua mitjana és 1 gol cada 48 minuts.*

Com es fan totes eixes estadístiques? Hi ha algú repassant totes les dades una a una de totes les jornades i va veient rècords, mitjanes, etc? És evident que no. Per a fer això, caldrà fer, amb el llenguatge SQL:

- Crear l'estructura idònia de base de dades (només una vegada):

  CREATE TABLE partits (...

- Introduir les dades dels resultats dels partits (després de cada jornada):

  INSERT INTO partits VALUES (...

  UPDATE partits SET gols_casa = gols_casa + 1 WHERE ...

- Fer les consultes que necessitem (per a cada informació estadística):

```
SELECT SUM(targetes)
  FROM partits
  WHERE jornada = 7;
```

Per a fer eixes i altres operacions cal un tipus de programa anomenat Sistema de Gestió de Bases de Dades (SGBD).

Un SGBD és un conjunt de programes que permeten crear BD, emmagatzemar dades, modificar-les, esborrar-les i consultar-les.

El SGBD que usarem nosaltres és el MySQL.

Este SGBD, com molts altres, per a interactuar amb les BD usa el llenguatge **SQL**: *Structured Query Language* (Llenguatge de consultes estructurat). Les instruccions de SQL (sentències) es poden classificar en 5 grups:

SQL = DDL + DML + DQL + DCL + TCL

- **DDL (Llenguatge de Definició de Dades)**. Crear taules amb els camps, claus, etc.

- **DML (Llenguatge de Manipulació de Dades)**. Inserir dades en les taules, esborrar-les o modificar-les.

- **DQL (Llenguatge de Consulta de Dades)**. Consultar dades guardades en la BD.

- **DCL (Llenguatge de Control de Dades)**. Restringir permisos d'accés a les taules a determinats usuaris.

- **TCL (Llenguatge de Control de Transaccions)**. Permetre accés concurrent (simultani) de diversos usuaris a les dades.

SQL és un llenguatge informàtic per a realitzar tasques de gestió amb una BD. Les instruccions (ordres o sentències) poden executar-se de forma independent (consola, etc) o bé poden incorporar-se a un altre llenguatge (C, Java...) per a permetre'ns accedir a una BD des d'un programa.

Nota: és obligat acabar la sentència amb ";" si la llancem des de consola.

A continuació vorem un resum de les sentències més importants.

SENTÈNCIES SQL		
SUBLLENGUATGE	SENTÈNCIA	DESCRIPCIÓ
DDL (Definició de dades)	CREATE DATABASE	Crea una BD
	USE	Selecciona una BD per a usar-la
	DROP DATABASE	Elimina una BD
	CREATE TABLE	Crea una taula
	DROP TABLE	Elimina una taula
	ALTER TABLE	Modifica l'estructura d'una taula
	CREATE INDEX	Crea un índex per a una columna
	DROP INDEX	Elimina un índex d'una columna
	CREATE VIEW	Crea una vista
	DROP VIEW	Elimina una vista
DML (Manipulació de dades)	INSERT	Afig files a una taula
	DELETE	Suprimix files d'una taula
	UPDATE	Modifica dades d'una taula
DQL (Consultes)	SELECT	Mostra dades d'una BD
DCL (Control de dades)	GRANT	Dóna permisos d'accés a usuaris
	REVOKE	Lleva permisos d'accés a usuaris
TCL (Control de transaccions)	BEGIN	Inicia una transacció
	COMMIT	Finalitza una transacció
	ROLLBACK	Avorta la transacció
	SAVEPOINT	Per a avortar parcialment la transacció

Totes les sentències SQL comencen amb una paraula clau que descriu el que fa la sentència (SELECT, INSERT, DELETE...). La sentència continua amb una o més clàusules, que també comencen amb una paraula clau (WHERE, FROM, ORDER BY...).

Manual MySQL: http://dev.mysql.com/doc/refman/5.0/en/

2. DDL

El Llenguatge de Definició de Dades (DDL) de SQL permet definir els objectes de les nostres bases de dades. Amb ell podrem crear bases de dades, taules, claus, etc, així com crear vistes parcials de les taules i gestionar els diferents permisos per a cada usuari o grup d'usuaris, etc.

Potser varia un poc d'un SGBD a altre (tipus de dades, etc). Nosaltres ens basarem en el MySQL.

2.1 Instruccions sobre bases de dades

- Per a mostrar les bases de dades que tenim:

```
SHOW DATABASES;
```

- Per a crear una base de dades nova:

```
CREATE DATABASE nom_base_de_dades;
```

- Per a treballar amb una base de dades (que ja estiga creada):

```
USE nom_base_de_dades;
```

- Per a consultar amb quina base de dades estem treballant:

```
SELECT database();
```

- Per a eliminar una base de dades que ja no volem:

```
DROP DATABASE nom_base_de_dades;
```

Abans de vore les instruccions sobre les taules, vorem primer els tipus de dades que accepta MySQL per a crear les columnes de les taules.

2.2 Tipus de dades

Alguns dels tipus de dades de MySQL són:

CLASSIFICACIÓ	TIPUS DE DADES	DESCRIPCIÓ
ENTERS	TINYINT	Enter entre -127 i 127
	BOOL	Igual que TINYINT. True=1. False=0
	SMALLINT	Fins 65.535
	MEDIUMINT	Fins 8 milions (+-).
	INT o INTEGER	Fins 4000 milions (+-)
	BIGINT	Fins 18 trilions (+-)
DECIMALS	DECIMAL(m, d)	
	FLOAT	
	DOUBLE	
	REAL	
DATA I HORA	DATE	Guarda dates en any-mes-dia (0000-00-00)
	TIME	Guarda les hores en hora:minuts:segons (00:00:00). Amb la funció DATE_FORMAT() es pot mostrar en AM i PM.
	DATETIME o TIMESTAMP	Data i hora. Format: any-mes-dia hora:minuts:segons (0000-00-00 00:00:00).
	YEAR	4 dígits
TEXT	CHAR(n)	Fins 255 caràcters abans de versió 5.03 Fins 65.000 en versions posteriors.
	VARCHAR(n)	Ocupa menys que CHAR però les consultes són més lentes.
ALTRES	BLOB	Informació binària (imatges, sons...)

2.3 Instruccions sobre taules

- ✓ Per a mostrar informació de les taules existents:

```
SHOW TABLES;                    // Mostra les taules de la BD en ús
DESCRIBE nom_taula; // Mostra els camps d'una taula (o DESC o EXPLAIN)
SHOW CREATE TABLE nom_taula; // Mostra el CREATE TABLE de la taula
SHOW INDEX FROM nom_taula;    // Mostra els índexs d'una taula
```

- ✓ Per a crear/modificar/eliminar les taules:

```
CREATE TABLE nom_taula...       // Crear una taula
CREATE TERMPORARY TABLE nom_taula...   // Crear una taula temporal
ALTER TABLE nom_taula...        // Modificar estructura d'una taula
DROP TABLE nom_taula;           // Eliminar una taula
```

Ara vorem en detall estes últimes.

2.3.1. CREATE TABLE

Creació de taules sense claus

```
CREATE TABLE alumnes(
        codi INT,
        nom VARCHAR (20)
);
```

Els noms (de taules, camps, etc) han de començar per lletra i no poden contindre espais o caràcters de puntuació especials.

Creació de taules amb claus primàries i alternatives

Quan creem una taula, si volem indicar que un camp és clau primària utilitzarem PRIMARY KEY. Per a indicar que és clau alternativa, usarem: UNIQUE.

```
CREATE TABLE persones (
   codi      INT             PRIMARY KEY AUTO_INCREMENT,
   dni       VARCHAR(10)     UNIQUE,
   nom       VARCHAR (40)    NOT NULL,
   deute     INT             DEFAULT 0      );
```

La clau primària i/o les claus alternatives també es poden indicar després de l'última columna:

```
CREATE TABLE persones (
    codi    INT             AUTO_INCREMENT,
    dni     VARCHAR(10),
    nom     VARCHAR (40)    NOT NULL,
    PRIMARY KEY(codi),
    UNIQUE (dni)
);
```

En eixe cas, també podem posar noms a les restriccions (que han de ser únics en tota la base de dades). La utilitat és poder esborrar després una restricció indicant el seu nom.

```
CREATE TABLE persones (
    codi    INT             AUTO_INCREMENT,
    dni     VARCHAR(10),
    nom     VARCHAR (40)    NOT NULL,
    CONSTRAINT cpri_persones PRIMARY KEY(codi),
    CONSTRAINT calt_persones UNIQUE(dni)
);
...
```

Després podríem esborrar una restricció amb:

DROP CONSTRAINT "calt_persones";

Esta forma d'indicar-ho (al final de la taula) és necessària si volem indicar que una clau és composta. Per exemple:

```
CREATE TABLE liniesFactura (
        num     INT,
        linia   INT,
        ...,
        PRIMARY KEY(num, linia)
);
```

Notes:

- AUTO_INCREMENT: si no indiquem el codi quan inserim una persona, automàticament li assignarà un més del màxim codi dels alumnes inserits.
- NOT NULL: per a impedir que un camp tinga valors nuls.
- DEFAULT: per a posar un valor per defecte.

Creació de taules amb claus alienes

Per a usar-les en MySQL cal crear les taules amb el motor InnoDB.

Motors d'emmagatzematge de MySQL

El motor d'emmagatzematge és la forma en què MySQL emmagatzema les taules. Els motors més importants són MyISAM i InnoDB (cadascuna amb els seus avantatges i inconvenients).

Nosaltres utilitzarem InnoDB perquè suporta claus alienes i transaccions.

Per a especificar el motor s'indica al final de la creació de cada taula, amb la paraula ENGINE (o bé TYPE, és el mateix).

```
CREATE TABLE persones (
    ...
) ENGINE = InnoDB;
```

Si no s'especifica el motor en una taula, es crea amb MyISAM en versions de MySQL anteriors a la 5.5. En versions posteriors, el motor per defecte és InnoDB. Per a saber la versió de MySQL que tenim, podem vore-ho amb: *show variables like 'version'*

Una sentència SELECT pot barrejar diferents tipus de taules.

```
CREATE TABLE pobles (
    cpo INT PRIMARY KEY,
    nom CHAR(30)
) ENGINE = InnoDB;

CREATE TABLE clients (
    codi INT PRIMARY KEY,
    nom VARCHAR (40) NOT NULL,
    poble INT,
    tel VARCHAR(15),
    FOREIGN KEY (poble) REFERENCES pobles(cpo)
) ENGINE = InnoDB;
```

En este exemple veiem com seria la definició d'una clau aliena composta:

```
CREATE TABLE cintes (
    cod_pel         INT,
    num_copia       INT,
    rebobinada      BOOL,
    PRIMARY KEY(cod_pel, num_copia)
) ENGINE = InnoDB;

CREATE TABLE prestecs
    pel             INT,
    copia           INT,
    data            DATE,
    soci            INT,
    PRIMARY KEY(pel, copia, data),
    FOREIGN KEY(pel, copia) REERENCES cintes(cod_pel, num_copia)
) ENGINE = InnoDB;
```

Notes:

- També es pot posar un nom a la clau aliena, amb el CONSTRAINT.
- És opcional indicar la columna de la taula referenciada (cpo) a no ser que en la taula pare fóra clau alternativa (en compte de clau principal).
  FOREIGN KEY(poble) REFERENCES **pobles ~~(cpo)~~**
- El motor MyISAM també permet establir les claus alienes, però no serveix de res: no fa control d'existència de les referències.
- La clau aliena també es pot posar en la pròpia definició del camp en la taula, però no fa absolutament res (almenys en MySQL):

```
CREATE TABLE clients (
        ...
        poble INT REFERENCES pobles (cpo),
        ...
)
```

  Només es posa com a "comentari per al programador" de que eixa columna fa referència a altra columna d'altra taula.

Més endavant vorem sentències d'insert, delete, etc. però veiem ara uns exemples per vore el funcionament de claus primàries i alienes:

Omplim la taula pobles:

```
INSERT INTO pobles VALUES (46410, 'Sueca');
INSERT INTO pobles VALUES (55555, 'Ciutat de prova');
```

Omplim la taula clients:

```
INSERT INTO clients VALUES (1, 'Pep', 46410);   // No dóna problemes
INSERT INTO clients VALUES (1, 'Pepa', 46410); // Error per clau primària
INSERT INTO clients VALUES (2, 'Pepa', 77777); // Error per clau aliena
```

Esborrem registre de la taula de pobles:

```
DELETE FROM pobles WHERE cpo = 55555;      // No dóna problemes
DELETE FROM pobles WHERE cpo = 46410;      // Error per la clau aliena
```

Accions associades a la clau aliena

Quan definim una clau aliena podem especificar que, si s'esborra un registre de la taula pare (la de pobles), que s'esborren també, automàticament, els registres relacionats de la taula filla (és a dir: que s'eliminen també els clients d'eixe poble).

O bé, podem indicar que, quan modifiquem un codi postal en la taula de pobles, que també es modifique eixe codi en els corresponents clients que tenien eixe poble.

Eixes dos coses (i més) es poden fer posant-ho en la restricció de clau aliena de la taula. Per exemple:

```
CREATE TABLE clients (
    ...
    FOREIGN KEY (poble) REFERENCES pobles(cpo)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

Estes són les accions que podem fer en un ON DELETE o en un ON UPDATE:

- **CASCADE**: Els registres dependents també s'esborren o es modifiquen.
- **SET NULL**: Els registres dependents es posen a null (si no tenien la restricció de NOT NULL, clar).
- **SET DEFAULT**: Els registres dependents es posen al valor per defecte (si el tenien, clar). El motor InnoDB ho accepta, però sense cap efecte.
- **RESTRICT**: No deixa esborrar/modificar de la taula pare si té registres dependents. És l'opció per defecte (si no posem ON UPDATE o ON DELETE).
- **NO ACTION**: En MySQL fa exactament el mateix que RESTRICT. Ara bé, en altres SGBD com Oracle o PostgreSQL, el NO ACTION deixa esborrar/modificar de la taula pare però no fa res en els registres dependents. Això trencaria la definició de clau aliena, però el que fa realment és endarrerir les comprovacions de clau aliena per a quan acabe la transacció (ja vorem les transaccions més endavant).

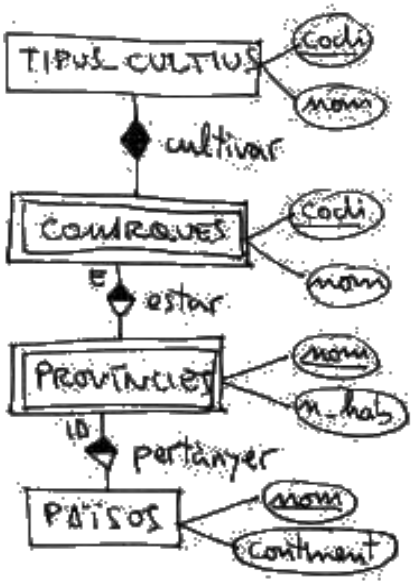
EXERCICIS DE CREACIÓ DE BD I TAULES (BD cultius)

1) Crea la base de dades de nom "cultius" i, dins d'ella, escriu les instruccions DDL necessàries per a crear les següents taules en la base de dades que has creat. Els codis seran un varchar(3). Els noms varchar(30) i la quantitat d'habitants (hab) serà integer.

Tin en compte que han de coincidir els tipus dels camps que enllacen les taules amb les claus alienes.

Tin en compte que abans de crear una taula amb claus alienes, caldrà crear primer les taules de les qual depén.

Nota: Els camps de l'E/R no són exactament igual als de l'esquema relacional adjunt.



TIPUS_CULTIUS = cod + nom

CULTIVAR = cul + com
C. Ali: cult→ TIPUS_CULTIUS (cod)
com → COMARQUES (cod)

COMARQUES = cod + nom + pro + pai
C. Ali: (pro + pai) → PROVÍNCIES (cod + pai)
VNN (pro)
VNN (pai)

PROVÍNCIES = cod + pai + nom + hab
C. Ali: pai → PAÏSOS (cod)

PAÏSOS = cod + nom + con

Cal crear les taules de forma que:

- si modifiquem el codi d'un país o província en la taula de províncies, també haurà de canviar en la taula de comarques.

- si modifiquem un codi de cultiu en la taula tipus_cultius, també s'haurà de modificar en la taula cultivar.

- si esborrem un tipus de cultiu, també s'hauran d'esborrar els registres de cultivar que tinguen eixe cultiu.

2.3.2. ALTER TABLE

S'utilitza per a **modificar l'estructura** (no les dades) d'una taula ja existent.

a) Canviar el nom d'una taula:

ALTER TABLE nom_antic **RENAME TO** nom_nou;

b) Afegir un camp

ALTER TABLE taula **ADD COLUMN** camp INT NOT NULL;

ALTER TABLE taula **ADD COLUMN** camp VARCHAR (20) NOT NULL **FIRST**;

ALTER TABLE taula **ADD COLUMN** camp INT NOT NULL **AFTER** camp_existent

Nota: el COLUMN és opcional en ADD, DROP, CHANGE i ALTER

c) Eliminar un camp

ALTER TABLE taula **DROP COLUMN** camp_a_esborrar;

d) Modificar un camp

- Canvi de nom (i/o del tipus de dades, de posició...):

ALTER TABLE taula **CHANGE COLUMN** camp_actual
camp_nou INT NOT NULL AFTER camp_exi;

- Posar o llevar el valor per defecte:

ALTER TABLE taula **ALTER COLUMN** camp
[SET DEFAULT valor | DROP DEFAULT];

e) Afegir claus o índexos

ALTER TABLE taula **ADD PRIMARY KEY** (camps); // 1 o més camps separats per comes

ALTER TABLE taula **ADD FOREIGN KEY** (camps) **REFERENCES** taula_pare (camps);

ALTER TABLE taula **ADD UNIQUE** [nom_index] (camps); // Clau alternativa

ALTER TABLE taula **ADD INDEX** [nom_index] (camps); // Per a recerques més ràpides

Nota: també podem posar el CONSTRAINT per a posar nom a les restriccions.

f) Esborrar claus o índexos

ALTER TABLE taula **DROP PRIMARY KEY**; // Esborra clau primària

ALTER TABLE taula **DROP FOREIGN KEY** nom_clau_ali; // Esborra clau aliena

ALTER TABLE taula **DROP INDEX** nom_index; // Esborra clau alternativa o índex

2.3.3. DROP TABLE

DROP TABLE [IF EXISTS] nom_taula; // Elimina una taula

EXERCICIS D'ALTERAR L'ESTRUCTURA DE TAULES (BD empresa)

2) Crea la base de dades de nom *empresa* amb les taules següents:

   a. Taula d'empleats (*emp*) amb les columnes:

num	enter
nom	10 caràcters, camp obligatori
cap	enter
comissio	enter
dept	enter, camp obligatori

   b. Taula de departaments (*dept*) amb les columnes:

codi	enter, clau primària
nom	15 caràcters, camp oblibagori, únic
poble	20 caràcters

3) Fes que el camp *num* de la taula *emp* siga la clau principal (ALTER TABLE).

4) Afegeix una nova columna anomenada *salari* a la taula *emp*, de tipus enter i amb la condició de no poder prendre valors nuls. No anomenes la restricció.

5) Incorpora la condició de que la columna *dept* de la taula *emp* es clau aliena respecte a la taula *dept* (camp *codi*). Fes que quan s'esborre un departament, també s'esborren els empleats associats a eixe departament.

6) Fes que la columna *cap* de la taula *emp* siga clau aliena, referenciant a la seua pròpia taula.

7) Intenta eliminar la columna *cap* de la taula *emp*.

8) Elimina la condició de que la columna *codi* de *dept* es clau primària.

3. DML

3.1. Inserir dades (INSERT)

- ✓ Inserir una fila indicant quines columnes i en quin ordre es van a posar els valors:

```
INSERT INTO alumnes(codi, nom) VALUES(25, "Pep");
```

- ✓ Inserir una fila sense indicar columnes. S'introduiran tots els valors i en l'ordre en què estan les columnes en la taula:

```
INSERT INTO alumnes VALUES(25, "Pep", "Garcia");
```

- ✓ Inserir diverses files en una sola sentència:

```
INSERT INTO alumnes(codi, nom) VALUES
    (25, "Pep"),
    (26, "Pepa"),
    (27, "Pepet");
```

- ✓ Inserir diverses files a partir d'un fitxer (per defecte, en la carpeta de la BD):

```
LOAD DATA INFILE 'fitxer_alumnes.txt' INTO TABLE alumnes;
```

En el nom del fitxer pots posar ruta però només s'admet el directori on estan les BD o algun subdirectori amb permisos d'escriptura per a tots (o /tmp en Ubuntu, etc). Està fet així per motius de seguretat.

Per a més informació:

http://dev.mysql.com/doc/refman/5.0/es/load-data.html

L'operació inversa (de taula a fitxer) seria:

```
SELECT ...
    INTO OUTFILE 'fitxer_alumnes.txt'
    FROM...
    WHERE...
```

- ✓ Inserir files en una taula a partir de la SELECT d'una altra taula;

```
INSERT INTO alumnes_cicles (cod, nom)
    SELECT ...
    FROM alumnes
    WHERE alumnes.cicle = 's';
```

Nota: per a vore els registres d'una taula (i així comprovar que s'han fet els inserts, etc) ja vorem que ho podem fer amb la següent instrucció:

```
SELECT * FROM nom_taula;
```

3.2. Esborrar dades (DELETE)

```
DELETE FROM alumnes
    WHERE codi = 25;
```

Veiem que en el DELETE no s'especifiquen els camps a esborrar, ja que s'esborra tota la fila.

Nota: més endavant comentarem més coses de la condició del WHERE (quan veiem la clàusula WHERE de la sentència SELECT).

3.3. Modificar dades (UPDATE)

```
UPDATE alumnes
    SET nom = "Josep",
        poble = 46410
    WHERE nom = "Pep";
```

EXERCICIS DE DML (BD cultius)

9) Inserix un nou tipus de cultiu de codi ARR i nom Arròs

10) Inserix un nou país de nom Espanya del continent Europa.

11) Inserix les següents 4 províncies, en una sola sentència:

    València, amb 1 milió d'habitants
    Alacant, amb 700000 d'habitants
    Castelló, amb mig milió d'habitants
    Madrid, amb 2 milions d'habitants

12) Inserix la comarca "Ribera Baixa", de codi RB

13) Esborra la província de Madrid

14) Augmenta un 10% la població de totes les províncies

15) Modifica la comarca RB fent que el nom comence per "La". Utilitza l'operador CONCAT (consulta la sintaxi a Internet).

16) Guarda les dades de les provincies al fitxer "provincies.txt" del directori "/tmp".

    Nota: Caldrà tindre permisos.

17) Esborra el contingut de la taula províncies i torna a carregar les dades a partir del fitxer que has creat a l'exercici anterior.

4. DQL (SELECT)

4.1. Consultes simples

4.1.1. Introducció

La sentència SELECT, que s'utilitza per a expressar consultes SQL, és la més potent i complexa de les sentències de SQL. Este capítol discutix les consultes SQL més simples: aquelles que recuperen dades d'una única taula de la BD.

La sentència SELECT consta de 6 clàusules:

- La clàusula **SELECT** llista els elements a recuperar. Poden ser columnes de la BD o expressions a partir de les columnes, etc.

- La clàusula **FROM** llista les taules on estan els elements a recuperar per la consulta. Les consultes que extrauen les seues dades d'una única taula es descriuen en este capítol. Les de més de 2 taules les vorem més avant.

- La clàusula **WHERE** diu a SQL que incloga només certes files (registres) de dades en els resultats de la consulta. S'utilitza una *condició de recerca* per a especificar les files desitjades.

- La clàusula **GROUP BY** especifica una consulta sumària. En comptes de produir una fila de resultats per cada fila de dades, una consulta sumària agrupa totes les files semblants i després produïx una fila de resultats per cada grup.

- La clàusula **HAVING** diu a SQL que incloga només certs grups produïts per la clàusula GROUP BY en els resultats de la consulta. Igual que la clàusula WHERE, utilitza una condició de recerca per a especificar els grups desitjats.

- La clàusula **ORDER BY** ordena els resultats de la consulta en base a les dades d'una o més columnes. Si s'omet, els resultats de la consulta no apareixen ordenats.

4.1.2. La clàusula SELECT

Esta clàusula consta de la paraula SELECT, seguida per la llista d'elements que volem obtindre (separats per comes). Cada element de la llista pot ser: un nom de columna, una constant, una expressió, el símbol * o funcions internes:

a) Un nom de columna

```
SELECT cod AS 'CODI_ART', preu
    FROM articles
```

Àlies que volem que aparega en la visualització de les dades en compte del nom de la columna.

CODI_ART	preu
Tomaques	2.50
Creïlles	1.25
Pomes	1.90
Cebes	1.50

b) Una constant

✓ **Constants numèriques**

```
SELECT codi, preu, 21 AS 'IVA'
    FROM articles
```

codi	preu	IVA
Tomaques	2.50	21
Creïlles	1.25	21
Pomes	1.90	21
Cebes	1.50	21

Van sense cometes. El separador decimal és el punt. Poden portar els signes + o -

21 -375 2000.00 +497500.8778

✓ **Constants de cadena**

```
SELECT codi, "del Mareny" AS 'origen', preu
    FROM articles
```

codi	origen	preu
Tomaques	del Mareny	2.50
Creïlles	del Mareny	1.25
Pomes	del Mareny	1.90
Cebes	del Mareny	1.50

Entre cometes simples o dobles.

Si volem posar una cadena que conté un apòstrof, ho tancarem entre cometes simples. Si volem posar dins unes cometes dobles, ho posarem entre cometes simples:

'Del "Marenyet" ' "L'Alcúdia"

✓ **Constants de data i hora**

Entre cometes simples o dobles. S'indica any, mes i dia separats per la barra (/), guió (-) o dos punts (:). No té molt de sentit posar-les en la clàusula SELECT però sí en la WHERE:

```
SELECT codi, preu, data
    FROM articles
    WHERE data > '2015/7/8' ;  -- o bé: "2015-7-8"  o  '2015:07:08'
```

codi	*preu*	*data*
Tomaques	2.50	2015/7/9
Creïlles	1.25	2015/9/4
Pomes	1.90	2015/9/2
Cebes	1.50	2016/1/9

Investiga: esbrina si la data es pot posar en altre format, i la forma de posar les hores.

✓ **Constants simbòliques**

SQL inclou constants simbòliques especials que tornen valors de dades mantingudes pel propi SGBD. Algunes són: CURRENT_DATE (per a la data), CURRENT_TIME (per a l'hora), CURRENT_TIMESTAMP (per a data i hora). No requerixen la clàusula FROM.

```
SELECT CURRENT_DATE, CURRENT_TIME;
```

CURRENT_DATE	*CURRENT_TIME*
2016-01-25	19:32:34

c) Una expressió

Les expressions s'utilitzen en el llenguatge SQL per a calcular valors derivats dels que hi ha en la BD (i per a realitzar recerques en la BD). Els operadors matemàtics són els normals: suma (+), resta (–), multiplicació (*) i divisió (/).

```
SELECT  quant, preu, (quant * preu) * 1.21
        FROM  vendes
        WHERE (quant * preu) > 100
```

quant	preu	(quant * preu) * 1.21
3	3.00	10.89
2	4.00	9.68
3	2.00	7.26
4	5.00	24.20

d) El símbol *

Serveix per a mostrar totes les columnes d'una taula:

```
SELECT *
    FROM autors;
```

codi	nom	pais
1	Umberto Eco	Itàlia
3	Camilo J. Cela	Espanya
4	Homer	Grècia
5	M. Cervantes	Espanya

e) Funcions internes

MySQL inclou algunes funcions internes. Per exemple:

✓ **Sobre dates:**

```
SELECT data_naix, DAY(data_naix), MONTH(data_naix),
                  YEAR(data_naix), DAYNAME(data_naix)
    FROM alumnes
    WHERE YEAR(data_naix) > 1960;
```

data_naix	DAY(data_naix)	MONTH(data_naix)	YEAR(data_naix)	DAYNAME(data_naix)
1970-12-30	30	12	1970	Wednesday
1975-11-03	3	11	1975	Monday
1985-10-18	18	10	1985	Friday

✓ **Sobre cadenes:**

```
SELECT nom, CHAR_LENGTH(nom) AS longitud,
            CONCAT(cog1, " ", cog, ", ", nom) AS 'Nom complet',
            LOWER(grup) AS Minúscules, UPPER(nom_cicle) AS Majúscules
    FROM alumnes
```

nom	longitud	Nom complet	Minúscules	Majúscules
Pep	3	Garcia Garcia, Pep	pep	PEP
Jaume	5	Peris Martí, Jaume	jaume	JAUME
Sebastià	8	Pi Garcia, Sebastià	sebastià	SEBASTIÀ

Investiga: mira en Internet altres funcions per a treballar en les dates, cadenes, etc.

4.1.3. La clàusula FROM

Esta clàusula consta de la paraula clau FROM, seguida d'una llista de taules separades per comes. Posarem les taules necessàries per a obtindre els elements que hem posat en la clàusula SELECT.

```
SELECT nom_proveïdor, decrip_article
        FROM proveidors, proveir, articles
        WHERE proveidors.codi = proveir.pro
           AND proveir.art = articles.codi
```

nom_proveïdor	descrip_article
ANECOOP	tomaques
COPSEMAR	arròs
UNIANA	taronges

Ja veurem més endavant com utilitzar estes consultes amb més d'una taula en el FROM.

EXERCICIS SOBRE LA CLÀUSULA SELECT I FROM (BD empresa)

18) Afig la data de naixement (d_naix) a cada empleat.

19) Posa dates de naixement als empleats existents.

20) Mostra totes les dades de la taula d'empleats i, a més, de cadascun d'ells:

a. la longitud del seu nom

b. la comissió en tant per 1 (en la taula està com a percentatge)

c. el dia de la setmana en què nasqueren

d. la data de naixement expressat amb el format: 19 del 3 del 1970

21) Mostra els noms dels departaments en minúscula i en majúscula.

4.1.4. Resultats de consultes

El resultat d'una consulta SQL és semblant a una taula de la BD, amb files i columnes. En compte de mostrar eixe resultat per pantalla, el podem aprofitar per a diverses coses:

a) Inserir-ho en altra taula existent (ja ho hem vist en l'apartat de l'insert):

```
INSERT INTO vendes_febrer   -- Taula ja existent
    SELECT *
        FROM vendes
        WHERE mes = 2;
```

Estem afegint en la taula *vendes_febrer* (que ja existia) els registres de la taula *vendes* que complien una determinada condició. Els camps de la taula *vendes_febrer* hauran de coincidir amb quantitat i tipus de les columnes de la clàusula *select*.

b) Crear una altra taula amb les files i columnes seleccionades:

```
CREATE TABLE socis_sense_tel AS  -- Taula que creem ara
    SELECT codi_soci, nom_soci
        FROM socis
        WHERE telefon IS NULL
```

Estem creant la taula *socis_sense_tel* amb 2 camps (*codi_soci* i *nom_soci*) i amb les dades resultants d'aplicar la "select" a la taula *socis*:

c) Crear una vista amb les files i columnes seleccionades.

Ja ho vorem més endavant, però seria així:

```
CREATE VIEW socis_sense_tel AS
    SELECT codi_soci, nom_soci
        FROM socis
        WHERE telefon IS NULL
```

Ni estava creada la taula *socis_sense_tel* ni l'estem creant ara, sinó que li estem posant un nom al resultat de la *select*, de forma que ho podrem utilitzar com si fóra una altra taula però no ocupa espai en disc dur, ja que les files són les mateixes que la taula original (*socis*).

EXERCICIS SOBRE EL DESTÍ DEL RESULTAT D'UNA CONSULTA (BD institut)

En els següents exercicis vorem els passos per a passar les dades d'una taula a altres, de diverses formes.

22) Crea la BD institut.

23) Crea en ella la taula alumnes amb els camps codi (enter), nom i estudis (4 caràcters).

24) Introdueix 3 alumnes amb estudis de PQPI, 3 d'ESO i 3 de DAM.

25) A partir d'eixa taula, crea la taula a_dam amb els alumnes de DAM (codi i nom)

(amb la sentència de crear taula a partir d'una select).

26) Crea la taula a_eso amb codi (enter) i nom

27) Introdueix 3 alumnes d'ESO en a_eso (amb codis que no estiguin en taula d'alumnes).

28) Copia els alumnes d'ESO de la taula alumnes a la taula a_eso (amb la sentència d'inserció de files a partir d'una select).

29) Esborra de la taula alumnes els d'ESO i DAM.

4.1.5. Eliminar files duplicades (DISTINCT)

A vegades, una consulta traurà files repetides. Per exemple:

```
SELECT  país, llengua
        FROM  autors
```

país	llengua
Espanya	Castellà
Espanya	Català
Argentina	Castellà
Espanya	Català
Argentina	Castellà

Si després de la paraula SELECT posem DISTINCT, eliminarà eixes files repetides de la consulta:

```
SELECT  DISTINCT país, llengua
        FROM autors
```

país	llengua
Espanya	Castellà
Espanya	Català
Argentina	Castellà

El contrari de DISTINCT és ALL, però és el funcionament per defecte (sí que mostra les files repetides).

Encara que la clàusula SELECT tinga més d'un camp (com en l'exemple anterior), només es posa un DISTINCT, el qual descartarà les files repetides en tots els camps de la SELECT.

EXERCICIS SOBRE LA REPETICIÓ DE DADES EN UNA CONSULTA (BD empresa)

30) Mostra els departaments que tenen algun empleat.

31) Fes el mateix però sense que apareguen repetits els departaments.

32) Mostra les parelles de comissió i departament (de la taula d'empleats) sense que apareguen parelles repetides.

33) Mostra els diferents pobles (sense repetir) on hi ha departaments.

4.1.6. La clàusula WHERE

La clàusula WHERE s'utilitza quan no volem mostrar dades de totes les files de la taula, sinó només d'algunes. Per exemple: *Mostra'm els alumnes menors d'edat*

```
SELECT  nom, edat
        FROM  alumnes
        WHERE  edat < 18      -- Condició de recerca:  edat < 18
```

nom	edat
Pep	17
Pepa	16
Toni	15
Maria	17

La clàusula WHERE consta de la paraula clau WHERE seguida d'una **condició de recerca** que especifica les files a recuperar. **Per cada fila de la taula seleccionada**, la condició de recerca pot produir un d'estos tres resultats:

- Si la condició de recerca és **true** (certa), la fila SÍ que ix en la consulta.
- Si la condició de recerca és **false** (falsa), la fila NO ix en la consulta.
- Si la condició de recerca té un valor **NULL**, la fila NO ix en la consulta. En l'exemple anterior, els alumnes que no tenien posada la seua edat (tenien un NULL), la condició de NULL<18 té com a resultat el valor NULL i, per tant, eixos alumnes NO eixirien en la consulta. Ja vorem més endavant com consultar en la WHERE si un camp té el valor NULL.

La condició de recerca és allò que podem posar dins de la clàusula WHERE (o del HAVING, ja ho vorem). Què podem posar en una condició de recerca?

- Test de comparació
- Test de rang
- Test de pertinença a un conjunt
- Test de correspondència amb patró
- Test de valor nul.

A més, la condició de recerca pot ser composta (tindre diverses condicions). Per a això usarem els operadors relacionals AND, OR i NOT.

Veiem cadascun d'aquests tests així com les condicions de recerques compostes.

- **Test de comparació (=, <>, <, <=, >, >=)**

Compara expressions (noms de columnes, constants, expressions aritmètiques o funcions).

Exemples:

... WHERE iva < 16

... WHERE nom = 'Pep'

... WHERE (entrades - sortides) <> saldo

... WHERE preu_euros > (preu_ptes / 166.386)

- **Test de rang (BETWEEN)**

```
SELECT codi_soci
        FROM prestecs
        WHERE data BETWEEN "2013/02/01" AND "2013/02/15"
```

Això mostrarà els socis amb un préstec en la primera quinzena de febrer de 2013. La condició contrària seria: WHERE data NOT BETWEEN "2013/02/01" AND "2013/02/15". O bé: WHERE NOT (data BETWEEN 2013/02/01" AND "2013/02/15").

El BETWEEN no és realment necessari, ja que podríem reemplaçar-ho amb expressions amb AND i OR:

A BETWEEN B AND C → (A >= B) AND (A <= C)

NOT A BETWEEN B AND C → (A < B) OR (A > C)

- **Test de pertinença a un conjunt (IN)**

```
SELECT  codi_llibre, codi_soci, data_pre
        FROM  prestecs
        WHERE  data_pre  IN  ('2015/09/02', '2015/04/05', '2016/01/09')
```

Mostrarà les dades dels préstecs que siguen d'alguna d'eixes dates.

codi_llibre	codi_soci	data_pre
0001	1	2015/09/02
0002	4	2016/01/09
0002	3	2015/04/05
0007	1	2015/09/02

Per a obtindre els altres resultats, caldria utilitzar el NOT IN (però recordem que no eixirien aquells préstecs que tenen un NULL en la seua data).

Igual que el test BETWEEN, el test *IN* tampoc és necessari, ja que la condició:

```
x  IN  (a, b, c)
```

...és equivalent a:

```
(x = a)  OR  (x = b)  OR  (x = c)
```

- **Test de correspondència amb patró (LIKE)**

S'utilitza per a buscar cadenes de caràcters que s'ajusten a un patró.

nom_columna **[NOT] LIKE** patró

El patró és una cadena que pot incloure un o més caràcters comodins:

_ (símbol barra baixa) → qualsevol caràcter (però 1 i només 1).

% (símbol del percentatge) → qualsevol seqüència de 0, 1 o més caràcters.

Exemple 1: Traure nom i cognoms dels alumnes que tinguen de cognom Escrivà o Escribà

```
SELECT nom, cognoms
    FROM alumnes
    WHERE cognoms LIKE '%Escri_à%'
```

nom	cognoms
Pep	Pons Escrivà
Pepa	EScribà Garcia

Exemple 2: Traure nom i cognoms dels alumnes que tenen un nom de 3 lletres

```
SELECT nom, cognoms
    FROM alumnes
    WHERE nom LIKE '___';
```

nom	cognoms
Pep	Pons Escrivà
Roc	Garcia i Garcia

Es poden localitzar cadenes que no s'ajusten a un patró utilitzant NOT LIKE.

Si el valor d'eixa columna en alguna fila és NULL, tant si posem LIKE com NOT LIKE, la fila no eixirà en el resultat.

Caràcter d'escapament

Si volguérem buscar si apareix el símbol del % o el _ en una cadena de text, caldria utilitzar un caràcter d'escapament, que en MySQL és "\".

Exemple: Troba el títol dels llibres que continguen el símbol del percentatge.

```
SELECT titol
    FROM articles
    WHERE titol LIKE "%\%%"
```

Això traurà títols com: "El 50% dels alumnes són la meitat"

- **Test de valor nul (IS NULL)**

Un camp d'una taula que no conté cap valor es diu que conté un valor **nul**. No és que el valor siga zero o que siga una cadena buida, sinó que no té valor.

Per exemple, posarem valors nuls en els següents casos:

- ✓ Quan desconeixem el valor (per exemple, el la data de naixement d'un client).
- ✓ Quan no té valor (per exemple, el correu electrònic d'algú que no en té).
- ✓ Quan no té sentit el valor (per exemple, el nom del pare d'un alumne major d'edat).

Per a consultar on apareixen valors nuls cal usar una sentència de comparació específica "IS NULL". No és correcte fer comparacions amb el comparador d'igualtat "=". Per exemple, si volem saber els clients que no ens deuen res o aquells que no sabem el que ens deuen, faríem:

```
SELECT  codi, nom, deute
        FROM  socis
        WHERE  (deute =0) OR (deute IS NULL)
```

codi	nom	deute
3	Pep	0
5	Pepa	NULL
7	Maria	NULL
9	Tomàs	0

I per a consultar on apareixen valors no nuls, usarem IS NOT NULL:

```
SELECT  codi, nom, deute
        FROM  socis
        WHERE  deute IS NOT NULL
```

Que és el mateix que ... WHERE NOT (deute IS NULL)

Lògica trivaluada

Cal tindre en compte que els valors NULL creen una lògica *trivaluada* per a les condicions de recerca en SQL. És a dir, quan SQL compara els valors de dos expressions en el test de comparació, per a una fila determinada, es poden produir tres resultats: TRUE, FALSE o NULL.

- Si la comparació és certa → el resultat és TRUE.
- Si la comparació és falsa → el resultat és FALSE.
- Si el valor d'algun camp de la comparació és NULL → el resultat és NULL.

La clàusula WHERE només seleccionarà els resultats TRUE.

Vegem-ho amb exemples. Suposem que tenim la taula d'alumnes següent:

nom	edat
Pep	20
Pepa	NULL
Pepet	15

Per a mostrar els alumnes majors d'edat farem:

```
SELECT nom FROM alumnes WHERE edat >= 18;        // Pep
```

Per a mostrar els menors d'edat, caldria indicar la condició contrària:

```
SELECT nom FROM alumnes WHERE NOT (edat >= 18);        // Pepet
```

Però veiem que Pepa no ha eixit ni com a menor d'edat ni com a major. Per a mostrar aquells que no tenen edat, hem d'utilitzar IS NULL:

```
SELECT nom FROM alumnes WHERE edat IS NULL;          // Pepa
```

I per a mostrar els que sí que tenen edat:

```
SELECT nom FROM alumnes WHERE edat IS NOT NULL;   // Pep i Pepet
```

Si volguérem mostrar els menors d'edat però també els que no tenen ficada l'edat:

```
SELECT nom FROM alumnes WHERE edat<18 OR edat IS NULL;// Pepa i Pepet
```

Cal anar en compte i NO utilitzar la condició següent:

```
... WHERE  telèfon = NULL
```

És a dir: per a comparar amb el NULL cal usar IS i no el signe =.

- **Condicions de recerca compostes: AND, OR i NOT**

Ja hem vist en algun exemple que en l'apartat del WHERE ja han aparegut alguns operadors relacionals. Els que usa MySQL són AND, OR i NOT. És a dir: les condicions de recerca poden ser compostes i així combinar diferents condicions simples. També podem fer ús de parèntesis, sobretot quan volem trencar l'ordre de prioritat dels operadors, que és el següent:

NOT (el més prioritari)

AND

OR (el menys prioritari)

La sentència SELECT calcularà eixa condició composta per cada fila de la taula i retornarà aquelles files que el resultat siga vertader.

Exemples:

```
SELECT nom, domicili
        FROM amics
        WHERE ciutat = 'Sueca' AND data_naix < '01/01/1970

SELECT títol
        FROM llibres
        WHERE autor = 'Joan Fuster' OR tipus = 'assaig'

SELECT  nom, nacionalitat
        FROM  autors
        WHERE  NOT (nom  LIKE  '%Delibes%')

SELECT títol_disc, grup
        FROM discos
        WHERE (tipus = 'rock'  OR tipus = 'pop')
           AND NOT (llengua IN ('castellà', 'català', 'anglés'))
```

EXERCICIS DE FILTRAT DE FILES (CLÀUSULA WHERE) (BD empresa)

34) Mostra nom i comissió d'aquells empleats que NO tinguen comissió (pensa que pot haver empleats amb comissió 0 i empleats amb comissió NULL).

35) Mostra nom i comissió dels empleats que siguen del departament 2 amb una comissió superior al 10%.

36) Mostra nom i comissió dels empleats que tenen una comissió entre el 20 i el 50%.

37) Mostra el nom dels empleats dels departaments 2, 5, 6 i 8. Mostra també el departament de cadascun.

38) Mostra totes les dades dels empleats que el nom comence per A, que acaben amb E i que la tercera lletra del seu nom siga una I.

39) A partir d'ara treballarem amb la base de dades lliga1213. Per a crear-la (amb totes les seues taules i registres) executa l'script lliga1213.sql que trobaràs al Moodle.

4.1.7. La clàusula GROUP BY

Suposem que tenim esta taula de llibres:

titol	*editorial*	*preu*
L'odissea	Altaya	15.00
Cinc hores amb Màrius	McGraw-Hill	10.50
La Barraca	Altaya	11.00
Canyes i fang	Planeta	12.00
El nom de la rosa	Altaya	21.00
El Quixot	McGraw-Hill	23.50
Atles d'Espanya	Plaza & Janés	17.00
Aplique SQL	Plaza & Janés	26.00

Veiem que cada llibre pertany a una editorial (i que una editorial té molts llibres). La taula *llibres* té tants llibres com files. Però imagina que no volem obtindre informació de cada llibre (de cada fila) sinó de cada editorial (de cada GRUP de llibres de la mateixa editorial). Per exemple, de cada editorial volem saber (a més del nom): el preu del llibre més car d'eixa editorial, el preu del llibre més barat d'eixa editorial, el preu mig dels llibres d'eixa editorial, la suma dels preus de tots els llibres d'eixa editorial i la quantitat de llibres que té eixa editorial.

És a dir: voldrem obtindré la següent informació:

editorial	preu màxim	preu mínim	preu mig	suma de preus	quantitat de llibres
Altaya	21.00	11.00	15.67	47.00	3
McGraw-Hill	23.50	10.50	17.00	34.00	2
Planeta	12.00	12.00	12.00	12.00	1
Plaza & Janés	26.00	17.00	21.50	43.00	2

Com obtenim això amb una sentència *select*?

Haurem de dir a la *select* que no mostre informació de cada fila (de cada llibre) sinó de cada grup de files de la mateixa editorial. I, per a mostrar la informació, usarem unes funcions de MySQL que actuen sobre grups de files (funcions d'agregat): max, min, avg, sum, count.

És a dir: haurem de dir a la *select* que, internament, agrupe els llibres per l'editorial...

titol	editorial	preu
L'odissea	Altaya	15.00
La Barraca	Altaya	11.00
El nom de la rosa	Altaya	21.00
Cinc hores amb Màrius	McGraw-Hill	10.50
El Quixot	McGraw-Hill	23.50
Canyes i fang	Planeta	12.00
Atles d'Espanya	Plaza & Janés	17.00
Aplique SQL	Plaza & Janés	26.00

... i, així, obtindre **una fila de resultats per cada grup**. És a dir, la sentència seria esta:

```
SELECT editorial, MAX (preu), MIN (preu), AVG (preu), SUM (preu), COUNT(*)
FROM llibres
GROUP BY editorial
```

Les funcions d'agregat també admeten expressions. Per exemple: MAX(preu_venda – preu_compra)

COUNT(*) compta les files del grup.
COUNT(preu) comptaria files del grup on el preu no és null.

La sintaxi de la sentència *select* amb la clàusula *group by* és:

```
SELECT ...
FROM ...
WHERE ...
GROUP BY...
```

Ací podrem posar o bé els noms de les columnes per les quals estem agrupant o bé **funcions d'agregat** (*max, min, avg, sum, count*) o bé expressions formades per estesa.

Ací posarem el nom de la columna (o columnes separades per comes) per la qual volem agrupar. És a dir: totes les files d'eixe grup tindran el mateix valor d'eixa columna.

Nota: Si usem funcions d'agregat però **no** posem el GROUP BY, només ens mostrarà una única fila de resultats per a tota la taula:

```
SELECT MAX (preu), MIN (preu), AVG (preu), SUM (preu), COUNT(*), editorial
FROM llibres
```

MAX(preu)	MIN(preu)	AVG(preu)	SUM(preu)	COUNT(*)
26.00	10.50	17.00	136.0	8

Com en este cas **no** estem agrupant per l'editorial, no té sentit que ens mostre l'editorial, ja que mostraria el nom d'una editorial qualsevol de tota la taula.

Agrupament per més d'una columna

També podem agrupar per més d'una columna. En eixe cas, cada grup tindrà els mateixos valors en cadascuna d'eixes columnes.

Quantes més columnes posem en el *group by*, més grups eixiran (però amb menys files cadascun, clar).

Vegem-ho amb exemples de la taula *alumnes*:

ALUMNES					
núm	nom	curs	grup	poble	edat
1	Pep	1	A	Sueca	17
2	Pepa	1	A	Sueca	17
3	Pepet	1	A	Tavernes	18
4	Pepeta	1	A	Tavernes	19
5	Pepot	1	B	Sueca	17
6	Pepota	1	B	Sueca	18
7	Pepiu	1	B	Sueca	17
8	Josep	1	B	Tavernes	17
9	Josepa	2	A	Sueca	18
10	Josepet	2	A	Sueca	18
11	Josepeta	2	A	Tavernes	21
12	Josepot	2	B	Tavernes	19

```
SELECT curs, count(*), min(edat), max(edat)
FROM ALUMNES
GROUP BY curs;
```

curs	count(*)	min(edat)	max(edat)
1	8	17	19
2	4	18	21

Del curs 2 hi ha 4 alumnes

```
SELECT curs, grup, count(*), min(edat), max(edat)
FROM ALUMNES
GROUP BY curs, grup;
```

curs	grup	count(*)	min(edat)	max(edat)
1	A	4	17	19
1	B	4	17	18
2	A	3	18	21
2	B	1	19	19

Del curs 2 i grup A hi ha 3 alumnes

```
SELECT curs, grup, poble, count(*), min(edat), max(edat)
FROM ALUMNES
GROUP BY curs, grup, poble;
```

curs	grup	poble	count(*)	min(edat)	max(edat)
1	A	Sueca	2	17	17
1	A	Tavernes	2	18	19
1	B	Sueca	3	17	18
1	B	Tavermes	1	17	17
2	A	Sueca	2	18	18
2	B	Tavernes	1	19	19

Del curs 2, grup A que siguen de Sueca hi ha 2 alumnes

Norma: si en la clàusula *select* hi ha funcions d'agregat (sum, max, min, count, avg), tots els altres camps de la clàusula *select* han d'aparéixer també a la clàusula *group by*.

4.1.8. La clàusula HAVING

SELECT ...

FROM ...

WHERE ...

GROUP BY ...

HAVING condició_de_recerca

Si la clàusula WHERE descartava files, la clàusula HAVING descarta grups. És a dir: en la condició de filtrat del HAVING indicarem què ha de complir un grup per a que no siga descartat. No pot haver clàusula HAVING si no hi ha GROUP BY.

Per exemple, modifiquem la SELECT d'abans perquè només mostre les editorials que tenen més d'un llibre i que la mitja dels seus preus siga major de 16 euros:

```
SELECT editorial, MAX (preu), MIN (preu), AVG (preu), SUM (preu), COUNT(*)
    FROM llibres
    GROUP BY editorial
    HAVING (COUNT(*) > 1) AND (AVG(preu) > 16)
```

Estes condicions NO poden anar en el *where* perquè no s'han d'aplicar apliquen a cada fila de la taula, sinó a cada grup de files.

El resultat serà este:

editorial	preu màxim	preu mínim	preu mig	suma de preus	quantitat de llibres
McGraw-Hill	23.50	10.50	17.00	34.00	2
Plaza & Janés	26.00	17.00	21.50	43.00	2

4.1.9. La clàusula ORDER BY

```
SELECT  editorial, preu, titol
        FROM  llibre
        ORDER BY  editorial DESC, titol ASC
```

Això mostrarà els llibres ordenats per l'editorial, de forma descendent. Si hi ha diverses files de la mateixa editorial, les ordenarà pel títol, de forma ascendent.

editorial	*preu*	*titol*
Plaza & Janés	26.00	Aplique SQL
Plaza & Janés	17.00	Atles d'Espanya
Planeta	12.00	Canyes i fang
McGraw-Hill	10.50	Cinc hores amb Màrius
McGraw-Hill	23.50	El Quixot
Altaya	21.00	El nom de la rosa
Altaya	15.00	L'odissea
Altaya	11.00	La Barraca

Podem ordenar respecte a qualsevol element de la clàusula SELECT o per qualsevol camp de la taula del FROM (encara que no estiga en la clàusula SELECT).

Si no s'indica DESC ni ASC, per defecte és ASC.

Per a indicar la columna en l'ORDER BY, en compte de posar el nom de la columna podem posar el número en el qual apareix eixa columna en la clàusula SELECT. El resultat serà el mateix. En el nostre exemple:

```
SELECT  editorial, preu, titol
        FROM  llibres
        ORDER BY  1 DESC, 3 ASC
```

4.1.10. Regles per a processament de consultes de taula única

Veiem els passos que segueix el SGBD per a processar una sentència SELECT:

1) Seleccionarà totes les files de la taula que hi ha a la clàusula FROM.

2) Si hi ha clàusula WHERE, aplicarà la seua condició de recerca a cada fila de la taula. Si per a una fila la condició de recerca és TRUE, la seleccionarà. Però si és FALSE o NULL, la descartarà.

3) Si hi ha clàusula GROUP BY, les files seleccionades anteriorment les classifica en grups (on cada grup té els mateixos valors en els camps del GROUP BY).

4) Si hi ha HAVING, descarta els grups anteriors que no compleixen la condició de recerca del HAVING.

5) Calcula el valor de cada element de la clàusula SELECT per a cada fila seleccionada (o bé, per a cada grup si hi havia GROUP BY).

6) Si s'especifica SELECT DISTINCT, elimina les files duplicades dels resultats que s'hagueren produït.

7) Si hi ha una clàusula ORDER BY, ordena els resultats de la consulta.

Estes regles seran ampliades posteriorment per a incloure sentències SELECT més complexes (amb més d'una taula, etc)

EXERCICIS DE TAULA ÚNICA I SENSE GROUP BY (BD lliga1213)

1. Mostra tota l'estadística d'aquells golejadors que han marcat algun penal.
2. De cada partit mostra la jornada, els gols de cada equip, el total de gols, la possessió de l'equip de casa i la de l'equip de fora. Primer eixiran els partits que s'hagen marcat més gols. En cas d'igualtat, eixiran primer els que es marcaren més gols a casa.
3. Total de gols marcats en tota la lliga.
4. Mitja de gols per partit en tota la lliga.
5. Quants partits ha guanyat el Barça (codi 'bar')
6. Quants partits li falten per jugar al València (codi 'val')?

   Nota: un partit no està jugat si no estan posats els gols (per exemple, els de casa)

7. Mostra els següents sous: el més car, el més barat i la mitjana.
8. Incrementa un 5% els pressupostos de tots els equips.
9. Mostra la quiniela de la primera jornada (equip casa, equip fora, 1x2). Hauràs d'utilitzar una select condicional. Busca en Internet.
10. Gols marcats pel pitxitxi (només els gols; el nom del jugador no).
11. Mostra les quinieles de tota la competició. Caldrà mostrar també el número de la jornada. I han d'eixir junts els resultats d'una mateixa jornada.

EXERCICIS DE TAULA ÚNICA AMB GROUP BY (BD lliga1213)

1. Mostra de cada equip: el codi, sou màxim, mínim, la suma de tots els sous, quants jugadors hi ha, de quants jugadors es coneix el sou, la mitjana de sous entre els que sabem el sou i la mitjana de sous entre tots els jugadors.
2. Mostra quants jugadors té cada equip en cada posició.
3. Gols marcats en total en cada jornada.
4. Mitja de gols per partit en cada jornada.
5. Gols marcats pel pitxitxi de cada equip. És a dir: cal mostrar el codi de l'equip i els gols marcats pel seu màxim golejador.
6. Gols marcats en total per cada equip en casa.
7. Gols que ha rebut en total cada equip com a visitant.
8. Quants partits ha guanyat cada equip jugant en casa
9. Comprova si hi ha algun nom de jugador repetit. És a dir: cal mostrar el nom del jugador i quantes voltes apareix però només per a aquells jugadors que tinguen el nom repetit.
10. Volem saber la mitja de possessió del baló de cada equip jugant a casa. Ordenat de major a menor possessió. La mitja ha d'eixir sense decimals.
11. Jornades en les quals s'han marcat més de 35 gols.

4.1.11. Combinació de resultats de consulta (UNION)

Podem "unir" dos o més consultes en una sola. És a dir, a partir del conjunt de resultats d'una SELECT i del conjunt de resultats d'una altra SELECT, podem obtindre un únic conjunt de resultats.

1a consulta: Llista tots els productes que valguen més de 200 €

```
SELECT  codi, nom
        FROM  articles
        WHERE  preu > 200
```

CODI	NOM
ACI	4100I
REI	2A44L
ACI	4100Z
REI	2A44R

2a consulta: Llista tots els productes que s'hagen venut per un import de més de 3.000 €.

```
SELECT  DISTTINCT article, descripcio
        FROM  vendes
        WHERE  import > 3000
```

ARTICLE	DESCRIPCIO
IMM	775C
REI	2A44L
REI	2A44R

*Unió: Llista tots els productes que valguen més de 200 € **o** que s'hagen venut per un import de més de 3.000 €. És a dir: llista els productes que complisquen la primera condició **i** també aquells que complisquen la segona condició.*

```
SELECT  codi AS ID_FAB, nom AS ID_PRODUCTE
        FROM  articles
        WHERE  preu > 200
UNION
SELECT  DISTINCT article, descripcio
        FROM  vendes
        WHERE  import > 3000
ORDER BY 1
```

ID_FAB	ID_PRODUCTE
ACI	4100I
ACI	4100Z
IMM	775C
REI	2A44L
REI	2A44R

Algunes consideracions sobre el UNION:

- ✓ UNION lleva els resultats duplicats. Si no volem que els lleve cal posar UNION ALL.
- ✓ Es poden posar més de dos SELECT, separades per UNION:

```
SELECT * FROM a
UNION
( SELECT * FROM b   UNION   SELECT * FROM c )
```

- ✓ Les SELECT han de tindre el mateix nombre de columnes i ser del mateix tipus.
- ✓ Els noms de columnes de les SELECT no tenen per què ser igual. Si són diferents, el nom de les columnes que es mostraran en el resultat final seran les de la primera SELECT.
- ✓ Cap de les SELECT té la clàusula ORDER BY, però se'n pot posar una al final de tot.

4.1.12. Resum de les consultes simples

- La sentència SELECT s'utilitza per a expressar una consulta SQL. Tota sentència SELECT produïx una *taula* de resultats que conté una o més columnes i zero o més files.
- La clàusula FROM especifica les taules que tenen les dades a recuperar en una consulta.
- La clàusula SELECT especifica les columnes de dades a incloure en els resultats de la consulta, que poden ser columnes de dades de la BD o columnes calculades.
- La clàusula WHERE selecciona les files a incloure en els resultats aplicant una condició de recerca a les files de la BD.
- Una condició de recerca pot seleccionar files mitjançant comparació de valors (<, >, = ...), mitjançant comparació de valor amb un rang (BETWEEN) o un grup de valors (IN), per correspondència amb un patró de columna (LIKE) o per comprovació de valors nuls (IS NULL).
- Les condicions de recerca simples poden combinar-se mitjançant AND, OR i NOT per a formar condicions de recerca més complexes.
- La clàusula ORDER BY especifica que els resultats de la consulta han de ser ordenats en sentit ascendent o descendent, basant-se en els valors d'una o més columnes.
- L'operació UNION pot ser utilitzada dins d'una sentència SELECT per a combinar dos o més conjunts de resultats i formar un únic conjunt.

EXERCICIS SOBRE LA UNIÓ DE FILES (BD lliga1213)

1. Quants partits li queda per jugar a cada equip en casa i quants fora? Mostra la informació ordenada per equip. Dins de cada equip, primer els de casa.

EQUIP	PARTITS	LLOC
ath	1	casa
ath	2	fora
atm	1	casa
atm	1	fora
bar	2	casa
bar	1	fora
...		

2. Quants partits ha guanyat/empatat/perdut cada equip jugant en casa/fora. Ordenat per equip. Així (dóna igual si apareixen primer els guanyats o empatats o perduts):

EQUIP	PARTITS	RESULTAT
...		
atm	14	guanyats a casa
atm	1	empatats a casa
atm	3	perduts a casa
atm	8	guanyats fora
atm	5	empatats fora
atm	5	perduts fora
bar	13	guanyats fora
bar	3	empatats fora
bar	2	perduts fora
bar	16	guanyats a casa
bar	1	empatats a casa
bar	**0**	**perduts a casa**
...		

Nota: Voràs que no ixen les línies on la quantitat de partits és 0. No et preocupes. Per a que isquen, caldria usar subconsultes.

3. Quants uns, quantes x i quants 2 en la primera jornada.

SIGNE	QUANTITAT
1	5
X	2
2	3

4. Quants uns, quantes x i quants 2 en cada jornada (ordenat per la jornada). Així:

JORNADA	SIGNE	QUANTITAT
1	1	5
1	X	2
1	2	3
2	1	4
2	X	2
2	2	3
...		

4.2. Consultes compostes

4.2.1. Introducció

Moltes consultes sol·liciten **dades procedents de dos o més taules** en la BD. SQL permet recuperar dades que responen a estes peticions mitjançant **consultes compostes** o multitaula.

Per exemple, tenim estes dos taules:

POBLES = codi_postal + nom

ALUMNES = num + nom + grup + cp

C. Ali: cp → POBLES (codi_postal)

Volem fer una consulta on aparega el nom de l'alumne i el nom del poble on viu. Com que la informació està en 2 taules, haurem de:

- ✓ Posar les 2 taules en el FROM
- ✓ Posar en el WHERE com estan relacionades les 2 taules: sol ser una condició sobre la **clau aliena**.

```
SELECT  alumnes.nom, pobles.nom
FROM  alumnes, pobles
WHERE  alumnes.cp = pobles.codi_postal
```

Si hi ha una columna que està en les 2 taules (conflicte de noms) cal usar noms qualificats (nom_taula.nom_columna).

Taules necessàries separades per ",". Dóna igual l'ordre.

Enllaç entre les taules del FROM.

NOM	NOM
Pep Garcia	Sueca
Pepa Sales	Sueca
Andreu Albors	Sollana
Martí Manyes	Cullera

4.2.2. Explicació de per què posar l'enllaç entre les taules

Imaginem que en la taula d'alumnes tenim 3 registres i en la taula de pobles, 2:

ALUMNES		
num	nom	cp
1	Pep	46410
2	Pepa	46410
3	Pepet	46760

POBLES		
cpostal	nom	comarca
46410	Sueca	Ribera Baixa
46760	Tavernes	La Valldigna

C. Ali: ALUMNES(cp) → POBLES(cpostal)

Si volem obtindré en una consulta totes les dades dels alumnes i les corresponents dades dels pobles de cada alumne, si **no** posem l'enllaç, obtindrem el que s'anomena "producte cartesià" de les files. És a dir: cada fila d'alumnes la relacionarà amb cada fila de pobles. Per tant, tindrem 3 x 2 = 6 files:

```
SELECT alumnes.*, pobles.*
    FROM alumnes, pobles
```

num	nom	cp	cpostal	nom	comarca
1	Pep	46410	46410	Sueca	Ribera Baixa
1	Pep	46410	46760	Tavernes	La Safor
2	Pepa	46410	46410	Sueca	Ribera Baixa
2	Pepa	46410	46760	Tavernes	La Safor
3	Pepet	46760	46410	Sueca	Ribera Baixa
3	Pepet	46760	46760	Tavernes	La Safor

Però està clar que no és això el que volíem, sinó que d'eixes files només ens interessa aquelles que el cp és igual al cpostal. Per tant:

```
SELECT alumnes.*, pobles.*
    FROM alumnes, pobles
    WHERE alumnes.cp = pobles.cpostal
```

num	nom	cp	cpostal	nom	comarca
1	Pep	46410	46410	Sueca	Ribera Baixa
2	Pepa	46410	46410	Sueca	Ribera Baixa
3	Pepet	46760	46760	Tavernes	La Safor

I ara, si volem, com que el codi postal apareix repetit en dos columnes, en llevarem una qualsevol de les dos (o les dos, si no volguérem mostrar-lo):

```
SELECT alumnes.*, pobles.nom, pobles.comarca
    FROM alumnes, pobles
    WHERE alumnes.cp = pobles.cpostal
```

num	nom	cp	nom	comarca
1	Pep	46410	Sueca	Ribera Baixa
2	Pepa	46410	Sueca	Ribera Baixa
3	Pepet	46760	Tavernes	La Safor

4.2.3. Exemple per a més de dos taules

Suposem que tenim estes 3 taules:

ASSIGNATURES = codi + nom

ALUMNES = num + nom + grup + cp

MATRÍCULES = alu + assig + nota

C. Ali: alu → ALUMNES(num)

assig → ASSIGNATURES(codi)

Volem obtindré de cada alumne: el seu nom i el nom de les assignatures on està matriculat.

Solució:

- El nom de l'alumne està en la taula ALUMNES i el nom de les assignatures en la taula ASSIGNATURES. Però també necessitem la taula MATRÍCULES, ja que fa d'enllaç entre elles dos (és en eixa taula on està la informació de quines assignatures està matriculat cada alumne.
- Dit d'una altra manera: la informació de les matrícules està en la taula MATRIÍCULES però els camps a obtindré estan en altres dos taules. Per tant, caldrà fer l'enllaç de la taula MATRÍCULES a les altres dos.

Per tant, tindrem:

```
SELECT alumnes.nom, assignatures.nom
        FROM alumnes, assignatures, matricules
        WHERE alumnes.num = matricules.alu
            AND matricules.assig = assignatures.codi
```

Si tenim 3 taules, per a enllaçar-les caldrà posar els 2 enllaços entre elles (claus alienes).

NOM	NOM
Pep Garcia	Matemàtiques
Pep Garcia	Valencià
Pepa Sales	Valencià
Pepa Sales	Anglés
Martí Manyes	Valencià

4.2.4. Columnes d'emparellament

El procés de formar parelles de files fent coincidir els continguts de les columnes relacionades es denomina compondre les taules (o bé: "fer un *join*" de les taules)..

Les columnes usades en la composició de les taules es diuen columnes d'emparellament. En la majoria dels casos, les columnes d'emparellament fan referència a les claus alienes d'una taula respecte a altra. Caldrà anar en compte si entre les taules hi ha alguna clau aliena composta o bé hi ha distintes claus alienes que relacionen diverses taules:

a) Relacionar taules amb clau composta

Caldrà relacionar cadascuna de les parelles de camps. Per exemple:

FACTURES = any + numero + client + data

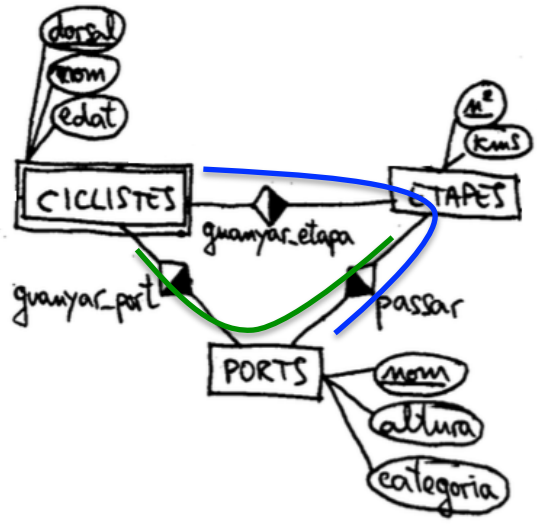
LINIES_FAC = any + numero + linia + article + quantitat + preu

C. Ali: (any + numero) → FACTURES(any + numero)

```
SELECT factures.*, línies_fac.*
FROM factures, linies_factura
WHERE factures.any = linies_fac.any
   AND  factures.numero = linies_fac.numero
```

b) Relacionar taules amb més d'una clau aliena entre elles

Caldrà "seguir el camí" de claus alienes segons la consulta que volem fer. Per exemple:



b.1) Mostra de cada ciclista, els ports guanyats i l'etapa per on passa cadascun d'eixos ports:

→ Rrelacionarem:

- ciclistes amb ports
- ports amb etapes

b.2) Mostra de cada ciclista, les etapes guanyades i els ports que té cadascuna d'eixes etapes:

→ Rrelacionarem:

- ciclistes amb etapes
- etapes amb ports

És a dir, si tenim el següent esquema relacional:

CICLISTES = dorsal + nom + edat

ETAPES = numero + kms + ciclista
C. Ali: ciclista → CICLISTES (dorsal)

PORTS = nom + altura + categoria + etapa + ciclista
C. Ali: etapa → ETAPES(numero)
C. Ali: ciclista → CICLISTES(dorsal)

Les consultes anteriors serien:

b.1) SELECT ciclistes.*, ports.*, etapes.*
FROM ciclistes, ports, etapes
WHERE ciclistes.dorsal = ports.ciclista
AND ports.etapa = etapes.numero

b.2) SELECT ciclistes.*, ports.*, etapes.*
FROM ciclistes, ports, etapes
WHERE ciclistes.dorsal = etapes.ciclista
AND etapes.numero = ports.etapa

4.2.5. Condicions de recerca en consultes multitaula

En la clàusula WHERE podem combinar emparellament de columnes i altres condicions de recerca. Per exemple:

```
SELECT  nom_edit, títol_llibre
        FROM  llibre, editorial
        WHERE  llibre.codi_edit = editorial.codi_edit
          AND  preu_llibre > 30
```

4.2.6. Ús d'àlies en les taules

En la clàusula FROM podem fer ús d'àlies de taules, per diferents motius:

a) Per a simplificar noms qualificats de les taules:

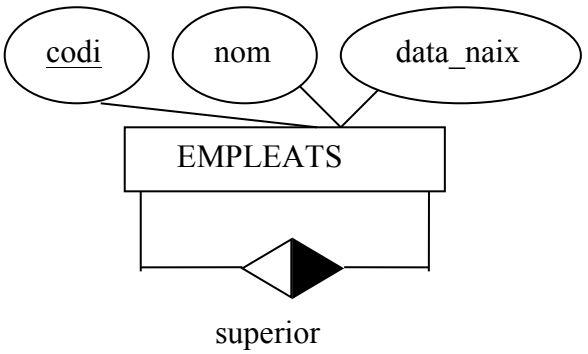
```
SELECT ven.*, ofi.ciutat, ofi.comarca
    FROM vendes AS ven, oficines AS ofi
    WHERE ven.oficina = ofi.codi
```

Usem la paraula reservada "AS" per a indicar l'àlies, però és opcional.
També podem usar àlies en consultes amb una única taula.

b) Per a crear una consulta multitaula que relaciona una taula amb ella mateixa.

Per exemple, en la taula empleats tenim tots els empleats d'una empresa: tant els treballadors subordinats com els seus superiors.

En l'esquema E/R tindríem:



En el disseny relacional tindríem:

EMPLEATS = codi + nom + data_naix + superior

C. Ali: superior → EMPLEATS (codi)

En eixa taula, cada empleat té el codi del seu superior, però non el nom.

Volem obtindre una consulta on aparega el nom de l'empleat junt al nom (no el codi) del seu superior. Implementant la consulta que volem en SQL, tindríem:

```
SELECT sub.nom, cap.nom
    FROM empleats AS sub, empleats AS cap
    WHERE sub.superior = cap.codi
```

El truc està en fer com si tinguérem dos taules d'empleats: una on estan els empleats com a subordinats i altra on estan els empleats com a superiors (caps).

4.2.7. JOIN: una altra forma de fer composicions de taules

Hi ha una altra forma de fer les composicions de taules: l'ús de JOIN. Per exemple, estes dos sentències trauen el mateix resultat:

```
SELECT alumnes.*, pobles.nom, pobles.comarca
        FROM alumnes, pobles
        WHERE alumnes.cp = pobles.cpostal
           AND alumnes.edat > 17;

SELECT alumnes.*, pobles.nom, pobles.comarca
        FROM alumnes JOIN pobles
             ON alumnes.cp = pobles.cpostal
        WHERE alumnes.edat > 17;
```

Esta segona forma és més eficient quan alguna de les dos taules té un índex en alguna de les columnes d'emparellament (per exemple, si una és clau principal).

A més, amb el JOIN, podrem dir més coses sobre com volem enllaçar dos taules. Què passaria si hi ha alumnes sense ordinadors? Que no eixirien en el llistat (ni amb l'operador coma ni amb el JOIN). Tampoc apareixeran els ordinadors que no tenen alumnes. Per a solucionar això, ho indicarem d'alguna forma en el JOIN. Ara veurem els distints tipus de JOIN que podem fer entre dos taules.

- **join** = inner join = cross join: no agafa files d'una taula si no troba correspondència en l'altra (és el mateix funcionament que faríem en l'operador coma (,) del FROM.
  A join B → Només parelles de A i B relacionades

- **left join** = left outer join: agafa files de la taula de l'esquerra (alumnes) encara que no tinga correspondència amb la taula de la dreta (ordinadors).
  A left join B → Tots els A (amb els B relacionats)

- **right join** = right outer join: agafa files de la taula de la dreta (ordinadors) encara que no tinga correspondència amb la taula de l'esquerra (alumnes).
  A right join B → Tots els B (amb els A relacionats)

- **outer join** = full outer join: agafa files de les dos taules encara que no tinguen correspondència amb l'altra taula.
  A outer join B → Tots els A i tots els B (amb els corresponents relacionats).

  Problema: el outer join (o full outer join) no funciona en MySQL (almenys en algunes versions). Veiem la solució.

Solució: fer un UNION de la left i de la right:

```
SELECT alumnes.*, ordinadors.*
        FROM alumnes LEFT JOIN ordinadors
                ON alumnes.ordinador = ordinadors.codi

UNION

SELECT alumnes.*, ordinadors.*
        FROM alumnes RIGHT JOIN ordinadors
                ON alumnes.ordinador = ordinadors.codi
```

Nota: esta forma de fer les composicions (amb JOIN) és millor que l'altra, ja que l'altra primer fa el producte cartesià i després elimina files el WHERE, mentre que esta és més ràpida ja que simplement relaciona les files amb el mateix valor en la clau aliena.

EXERCICIS MULTITAULA (BD lliga1213)

1. De cada partit volem mostrar els codis dels equips i el nom de la ciutat on juguen.
2. De cada partit que falta per jugar volem mostrar en quina data es disputarà, els noms curts dels equips, els noms de les ciutats respectives i el total d'habitants de les dos ciutats.
3. De cada equip: el pressupost, el que es gasta amb els jugadors i el percentatge que representa.
4. Llista de jugadors on conste: nom del jugador i nom de la ciutat on juga.
5. Quantitat total de gols de penal marcats per equips de ciutats de menys de 200000 habitants.
6. En quines dates s'han enfrontat el València i el Barça (sabent que els codis són "val" i "bar"). Mostra quin jugaba a casa i qui fora i el resultat de gols.
7. En quines dates s'han enfrontat el València i el Barça (sabent que els noms curts són "València" i "Barça"). Mostra quin jugaba a casa i qui fora (nom llargs) i el resultat de gols.
8. Mostra parelles de jugadors on un d'ells cobra més de 10 voltes que l'altre. Mostra també els seus sous.

9. Modifica l'exercici anterior perquè també apareguen els respectius noms (llargs) dels equips.

10. Migcampistes que cobren més que algun davanter del seu equip. Cal mostrar els 2 noms i els 2 sous.

11. Parelles de porter i golejador d'un mateix equip on el golejador haja marcat més gols que els gols que ha encaixat el porter. Cal mostrar l'equip i els noms del porter i golejador amb els gols respectius. Ordenat per l'equip i el nom del porter.

12. Volem comparar els gols de Messi i Ronaldo (no sabem el nom complet d'ells). Mostra el nom del jugador i tota l'estadística dels gols com a jugadors però només d'ells dos.

13. Mitjana de gols marcats en cada jornada i la data de la jornada (un decimal).

14. Quants partits ha guanyat/empatat/perdut cada equip, però sense diferenciar si és a casa o fora (només els totals).

EQUIP	PARTITS	RESULTAT
ath	11	guanyats
ath	8	empatats
ath	16	perduts
atm	22	guanyats
atm	6	empatats
atm	8	perduts
bar	29	guanyats

4.2.8. Regles per a processament de consultes multitaula

Veiem què fa SQL per a generar els resultats d'una consulta amb una sentència SELECT:

1) Si la sentència és una UNION de sentències SELECT, aplica els passos 2 fins al 7 a cada una de les sentències per a generar els resultats individuals de cada SELECT.

2) Forma el producte cartesià de les taules indicades en la clàusula FROM. Si la clàusula FROM designa una sola taula, el producte és eixa taula.

3) Si hi ha una clàusula WHERE, aplica la seua condició de recerca a cada fila de la taula producte, retenint aquelles files per a les quals la condició de recerca és TRUE (i descartant aquelles per a les quals és FALSE o NULL).

4) Si hi ha clàusula GROUP BY, les files retingudes anteriorment són agrupades pels camps d'esta clàusula.

5) Si hi ha HAVING, es retenen aquells grups que compleixen la condició d'esta clàusula.

6) Es calcula el valor de cada element de la clàusula SELECT per a cada fila retinguda (o bé, per a cada grup si hi ha el GROUP BY).

7) Si s'especifica SELECT DISTINCT, elimina les files duplicades dels resultats que s'hagueren produït.

8) Si la sentència és una UNION de sentències SELECT, mescla les files retingudes en cada SELECT, en una única taula de resultats. Elimina les files duplicades a no ser que s'haja especificat UNION ALL.

9) Si hi ha una clàusula ORDER BY, ordenar els resultats de la consulta segons s'haja especificat.

4.2.9. Resum de les consultes multitaula

- En una consulta multitaula, les taules que contenen les dades són designades en la clàusula FROM.

- Si no posem el join de les taules del FROM, es generen tantes files com combinacions diferents de les files de les taules (producte cartesià).

- Les consultes multitaula més habituals utilitzen les relacions creades per les claus primàries i claus alienes.

- Una taula pot compondre's amb ella mateixa; per a això es requereix l'ús d'àlies.

- Amb l'operador JOIN podem fer més variants de les composicions.

- Per a processar una consulta multitaula, els SGBD fan els següents passos:
    - Per a tota sentència SELECT de la UNION:
        - Si el FROM té més d'una taula, forma el producte cartesià
        - Si WHERE, selecciona les files
        - Calcula el valor de cada columna de la SELECT per a cada fila
        - Si DISTINCT, elimina files repetides
    - Si UNION, junta files obteses. Si no té ALL, elimina files repetides.
    - Si ORDER BY, ordena files.

4.3. Subconsultes

4.3.1. Introducció

Les subconsultes són sentències SELECT que s'utilitzen dins d'una altra sentència SELECT, a la qual anomenarem consulta principal.

Exemple introductori:

Si volem saber l'edat màxima de la taula *alumnes* faríem:

```
SELECT max(edat) FROM alumnes
```

Però... i si volem saber el nom d'eixe alumne (el que té la màxima edat)?

Cal anar en compte perquè açò estaria mal:

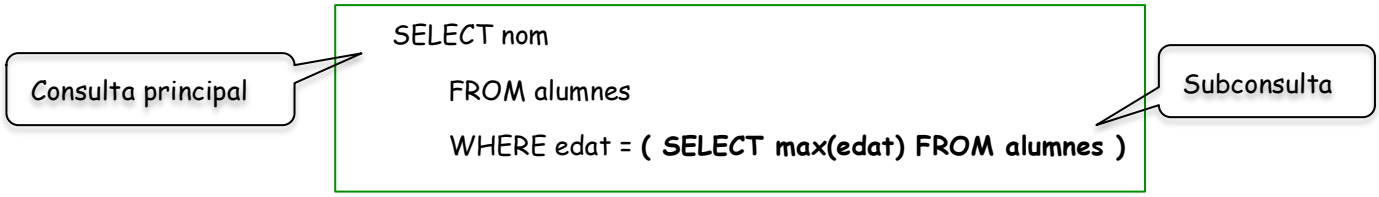
```
SELECT nom, max(edat) FROM alumnes
```

Està mal ja que mostraria l'edat màxima... i un nom qualsevol de la taula *alumnes*. Per què qualsevol? MySQL no és prou intel·ligent per a saber-ho? La resposta és que no té per què saber-ho, ja que si férem:

```
SELECT nom, max(edat), min(edat), avg(edat), min(nota) FROM alumnes
```

... quin nom d'alumne hauria d'agafar? El de la màxima edat? El de la mínima nota? ...

Per tant, el que volem és mostrar el nom d'aquell alumne **que té la condició** que la seua edat és... la màxima edat de tots els alumnes:



La select anterior mostrarà el nom de l'alumne (o alumnes) que tenen la màxima edat. Si, a més, volguérem saber quina és eixa edat, la posarem també a la clàusula select:

```
SELECT nom, edat
    FROM alumnes
    WHERE edat = ( SELECT max(edat) FROM alumnes )
```

SQL permet formar sentències SELECT combinant qualsevol de les formes que hem vist fins ara. És a dir: tant la consulta principal com la subconsulta poden ser multitaula, amb union, group by, etc. Una consulta pot tindre diferents subconsultes. Fins i tot, una subconsulta pot tindre altres subconsultes. Les sentències UPDATE i DELETE també poden tindre subconsultes en el WHERE. També poden anar subconsultes en la clàusula FROM i en la SELECT, però no són tan freqüents i no ho vorem. A més, en la condició on posem la subselect, en compte de l'operador relacional =, també en podem posar altres: <, >, !=, ...

Exemple on la WHERE té altres condicions, a més de la subconsulta:

Nom d'alumnes de 1r majors que tots els alumnes de 2n:

```
SELECT nom
        FROM alumnes
        WHERE curs =1
           AND edat > ( SELECT max(edat) FROM alumnes WHERE curs=2 );
```

Exemple on el FROM de la subconsulta no és el de la consulta principal:

Nom d'alumnes majors que tots els professors:

```
SELECT nom
        FROM alumnes
        WHERE edat > ( SELECT max(edat) FROM professors )
```

Exemple amb moltes subconsultes:

Noms de l'alumne més major i el del més menor:

```
SELECT nom
        FROM alumnes
        WHERE edat = ( SELECT max(edat) FROM alumnes )
             OR edat = ( SELECT min(edat) FROM alumnes )
```

Nota: si hi ha 2 o més persones de major edat, eixiran tots ells. Igual amb els menors.

Exemple amb subconsulta que enllaça amb la consulta principal:

Nom, curs i edat de l'alumne més major de cada curs.

Amb la solució de l'exercici anterior, per a obtindre l'alumne de major edat, teníem:

```
SELECT nom
    FROM alumnes
    WHERE edat = ( SELECT max(edat) FROM alumnes )
```

Però ara no volem comparar l'edat de cada alumne amb la màxima edat de tots els alumnes, sinó que, de cada alumne voldrem comparar la seua edat amb tots els alumnes DEL MATEIX CURS DE L'ALUMNE QUE ESTEM INTENTANT MOSTRAR (en la consulta principal):

"Nom, curs i edat dels alumnes...

... tals que la seua edat és...

... la màxima edat dels alumnes...

... del curs de l'alumne que estem intentant mostrar"

```
SELECT nom, curs, edat
FROM alumnes
WHERE edat =
        ( SELECT max(edat)
          FROM alumnes AS alumnes2
          WHERE alumnes2.curs = alumnes.curs);
```

És a dir: en la subconsulta hem d'enllaçar la taula de la subconsulta amb la taula de la consulta principal. I, com en este cas és la mateixa taula, necessitem fer ús de l'àlies de les taules (encara que no hi haguera conflicte sempre és recomanable).

Les subconsultes que havíem vist fins ara treien el mateix resultat per a cada fila seleccionada en la consulta principal. Però en este cas, la subconsulta s'executa per a cada fila de la consulta principal, de forma que;

- ✓ L'alumne A serà mostrat si la seua edat és la màxima dels alumnes del curs de l'alumne A.
- ✓ L'alumne B serà mostrat si la seua edat és la màxima dels alumnes del curs de l'alumne B.
- ✓ ...

És a dir: ara hem vist que la subconsulta pot agafar el valor de la consulta principal. Per tant, la subconsulta pot traure un resultat diferent per a cada fila de la consulta principal.

Nota: si volguérem enllaçar amb una subconsulta amb una condició composta, caldria posar una condició en l'enllaç i l'altra (o altres) en la where de la subconsulta.

EXERCICIS INTRODUCTORIS DE SUBCONSULTES (BD lliga1213)

Nota: s'ha intentat classificar els exercicis de subconsultes pels tipus de solucions de cadascun d'ells (conforme avancem en els apunts, hi haurà més tipus de subconsultes). Ara bé, com que un exercici pot tindre moltes solucions, potser no estiguen ben classificats segons la solució aportada per l'alumne.
En este apartat són tipus de subconsultes que retornaran només 1 únic valor.

1. Dorsal, equip i gols del pitxitxi (el que ha marcat més gols).

2. Nom del pitxitxi

3. Mostra el nom i sou del jugador millor pagat de tota la lliga.

4. Mostra el nom i sou del jugador millor pagat de cada equip.

5. Jugador que més cobra en cada equip dins la seua categoria (lloc). Cal mostrar el nom de l'equip, el nom del jugador, el lloc i el sou (expressat en milions d'euros, amb 1 decimal). Ordenat per equip i lloc.

6. Mostra totes les dades dels partits on més gols es marcaren de tot el campionat.

7. Mostra totes les dades dels partits on més gols es marcaren de cada jornada. Ordenat per la jornada.

8. Noms dels jugadors dels equips del partit on més gols es marcaren. Mostra també el codi dels seus equips. Ordenat per equip i nom de jugador.

9. Jornades en les quals s'han marcat més gols que la jornada anterior.

10. Nom llarg d'equips que tenen més de 2 porters, més de 2 defenses, més de 2 mitjos i més de 2 davanters.

4.3.2. Conjunt de resultats d'una subconsulta. Ús de ALL, ANY i IN.

En les subselects que hem vist només retornaven un únic valor. Però sabem que una select i, per tant, una subselect, poden donar com a resultat un conjunt de files i de columnes.

Condicions que han de complir les **columnes** de la subconsulta:

- Només podrem posar una única columna. És a dir: només pot aparèixer un camp en la clàusula SELECT. **No** podem fer: ... WHERE a = (SELECT ~~b, c, d~~ FROM...)

  (Com a excepció està l'operador EXISTS, que ho vorem més avant).

- Hem d'anar en compte que el que posem en la condició que enllaça la consulta principal amb la subconsulta, ha de poder-se comparar:
  - han de tindre un tipus "semblant" (els dos *varchar* encara que siguen de distinta capacitat; o bé un enter amb un *float*, etc)
  - han de tindre el mateix contingut semàntic: no podem comparar un codi de client amb l'edat d'una persona, per exemple.

Condicions que han de complir les **files** de la subconsulta:

- Si, com en l'exemple anterior, la subselect només va a retornar una fila, no hi ha problema. Podem usar d'enllaç (entre la consulta principal i la subselect) qualsevol operador relacional: =, !=, <, > ...

- Però si la subselect pot retornar més d'una fila, **estarem obligats a usar**, junt amb els operadors relacionals (=, >=, >, <=, <, <>), els operadors **ANY** (algun) o **ALL** (tots), ja que voldrem comparar un valor de la consulta principal amb ALGÚN valor retornat per la subselect o amb TOTS els valors retornats per la subselect. Com vorem després, en compte de "= ANY" també podem usar "IN"; i, en compte de "!=ALL" també podem usar "NOT IN".

Exemple ANY. Volem obtindre els noms dels alumnes que són majors que algun professor:

```
SELECT nom
FROM alumnes
WHERE edat > ANY (SELECT edat FROM professors)
```

Exemple ALL. Volem obtindre noms d'alumnes que són majors que tots els professors:

```
SELECT nom
FROM alumnes
WHERE edat > ALL (SELECT edat FROM professors)
```

Equivalències a expressions amb ANY i ALL:

Comparació	Subconsultes amb: ANY o ALL	Subconsultes equivalents amb: max, min ; IN, NOT IN
Major que tots	... WHERE edat > ALL (SELECT edat ...)	... WHERE edat > (SELECT max(edat) ...)
El major	... WHERE edat >= ALL (SELECT edat ...)	... WHERE edat = (SELECT max(edat) ...)
No major	... WHERE edat < ANY (SELECT edat ...)	... WHERE edat < (SELECT max(edat) ...)
No menor	... WHERE edat > ANY (SELECT edat ...)	... WHERE edat > (SELECT min(edat) ...)
El menor	... WHERE edat <= ALL (SELECT edat ...)	... WHERE edat = (SELECT min(edat) ...)
Menor que tots	... WHERE edat < ALL (SELECT edat ...)	... WHERE edat < (SELECT min(edat) ...)
Igual a algun	... WHERE edat = ANY (SELECT edat ...)	... WHERE edat IN (SELECT edat...)
Distint de tots	... WHERE edat != ALL (SELECT edat ...)	... WHERE edat NOT IN (SELECT edat ...)

Cal recordar que açò estaria **mal**:

```
SELECT nom
FROM alumnes
WHERE ~~edat > (SELECT edat FROM professors)~~
```

Mal perquè estem comparant l'edat d'un alumne amb MOLTES edats. Caldria posar ALL o ANY (o una expressió equivalent).

Equivalències de subconsultes amb multitaula.

Vegem un exemple de consulta equivalent a partir de les següents taules:

ALUMNES = codi + nom

MATRÍCULES = alumne + assig + nota

C. Ali: alumne → ALUMNES(codi)

	Amb subconsulta (tipus IN)	**Amb multitaula**
Alumnes matriculats de BD	SELECT nom FROM alumnes WHERE codi **IN** (SELECT alumne FROM matricules WHERE assig = 'BD');	SELECT nom FROM alumnes, matricules WHERE alumnes.codi = matricules.alumne AND assig = 'BD';
Alumnes NO matriculats de BD	SELECT nom FROM alumnes WHERE codi **NOT IN** (SELECT alumne FROM matricules); WHERE assig = 'BD');	Este cas no té una consulta multitaula equivalent.

La següent consulta NO seria correcta:

```
SELECT nom
FROM alumnes, matricules
WHERE alumnes.codi = matricules.alumne
   AND assig != 'BD');
```

No donarà error però no és el que volem mostrar. Esta consulta mostrarà els alumnes matriculats d'alguna assignatura que no siga BD.

És a dir: les consultes amb subconsultes tipus IN (o bé "= ANY") podrien resoldre's també com una consulta multitaula. Però no les de NOT IN.

EXERCICIS DE SUBCONSULTES QUE RETORNEN MÉS D'UNA FILA: ÚS D'*ANY, ALL, IN, NOT IN* (BD lliga1213)

11. Mostra els noms dels jugadors que cobren més que tot un altre equip sencer.

12. Nom dels jugadors que han marcat més gols que tot un altre equip sencer.

13. Nom dels jugadors que han marcat més gols que tot un altre equip sencer. També ha d'aparéixer el codi de l'equip del golejador i el codi de l'equip a qui supera. Ordenat per l'equip del golejador, nom del golejador i equip a qui supera.

14. Igual que l'exercici anterior però també ha d'aparéixer els gols del golejador i els gols de l'equip amb qui es compara.

15. Jugadors (equip i nom) que encara no han marcat cap gol. Ordenat per equip i nom.

16. Equip amb més jugadors

17. Equip amb més jugadors i la quantitat de jugadors.

18. Equip amb més jugadors i equip amb menys jugadors. També ha d'aparéixer les quantitats i una parauleta al costat que diga "MAX" o "MIN"

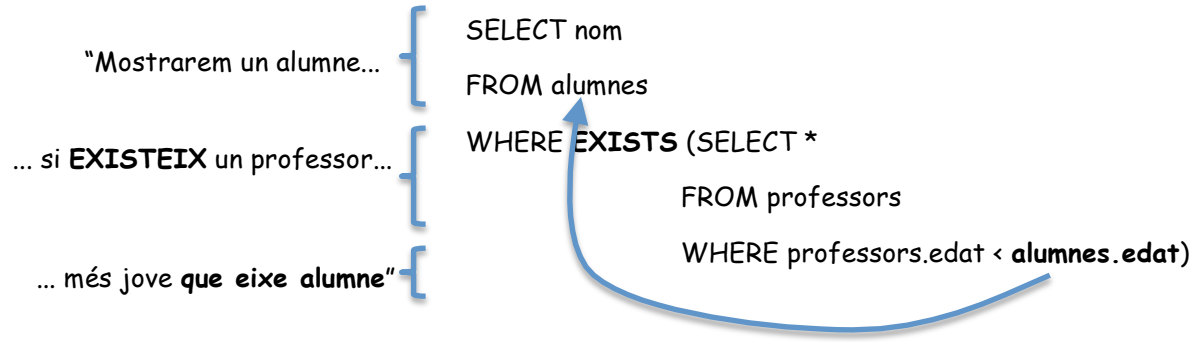
4.3.3. L'operador EXISTS

Vegem primer un exemple. Volem mostrar els alumnes majors que algun professor. Diverses formes de fer-ho:

- ✓ SELECT nom FROM alumnes WHERE edat > ANY (SELECT edat FROM professors);
- ✓ SELECT nom FROM alumnes WHERE edat > (SELECT min(edat) FROM professors);
- ✓ SELECT DISTINCT alumnes.nom
  FROM alumnes, professors
  WHERE alumnes.edat > professors.edat;

Cal posar DISTINCT per a no repetir noms d'alumnes (ja que un alumne el traurà tantes voltes com la quantitat de professors majors que ell hi hagen).

Però a vegades és més senzill expressar el requeriment així: "**Mostrarem un alumne si EXISTEIX un professor més jove que eixe alumne**". La consulta seria:



Veiem que l'enllaç entre la consulta principal i la subconsulta no es fa en el WHERE de la consulta principal sinó en el de la subconsulta. La subconsulta s'executa per a cada fila de la consulta principal, de forma que;

- ✓ L'alumne A serà mostrat si existeix un professor de menor edat que eixe alumne A.
- ✓ L'alumne B serà mostrat si existeix un professor de menor edat que eixe alumne B
- ✓ ...

L'expressió " WHERE EXISTS (subconsulta) " produïx un resultat vertader sempre que en la subconsulta s'obtinga alguna fila, sense importar els camps seleccionats en la subconsulta (per això he posat "*" però podria posar qualsevol camp de la taula professors). I produirà un resultat fals en cas contrari: si la subconsulta no obté cap resultat.

Al principi del present apartat hem vist que allò que fem amb l'EXISTS ho podem fer amb altre tipus de sentències. Per tant, podríem viure sense l'EXISTS però ja hem dit que és una altra forma de plantejar la resolució de les consultes. A més, en els apartats següents vorem el NOT EXISTS, que sí que ajuda a resoldre diferents tipus de consultes.

EXERCICIS DE SUBCONSULTES QUE RETORNEN MÉS D'UNA FILA: ÚS D'*EXISTS*, ETC (BD lliga1213)

19. Mostra el nom de les ciutats que tinguen algun equip de futbol. Fes-ho almenys amb 3 solucions possibles:
    a. exists
    b. in
    c. multitaula

20. Mostra el nom de les ciutats que no tinguen cap equip de futbol. Fes-ho almenys amb 2 solucions possibles:
    a. not exists
    b. not in

Ara vorem dos tipus de consultes per vore de quines formes se solen resoldre. Una forma serà amb el NOT EXISTS. Estes consultes són:

- ✓ Seleccionar una fila si totes les files que estan relacionades amb ella compleixen una mateixa condició. ("Totes les que estan, són").

  Per exemple: "*Mostra els alumnes tals que totes les assignatures que s'ha matriculat són de 1r curs*".

- ✓ Seleccionar una fila si totes les files que compleixen una mateixa condició estan relacionades amb ella. ("Totes les que són, estan").

  Per exemple: "*Mostra els alumnes que s'han matriculat de totes les assignatures de 1r curs*".

4.3.4. Seleccionar una fila si totes les files relacionades amb ella complixen una mateixa condició

Per exemple: "Mostra els alumnes que han aprovat TOTES les assignatures que s'han matriculat".

ALUMNES = codi + nom + nota_mitja

MATRÍCULES = alumne + assig + nota

C. Ali: alumne → ALUMNES(codi)

C. Ali: assig → ASSIGNATURES(codi)

ASSIGNATURES = codi + nom + curs

Quan volem mostrar una fila (alumne) si un conjunt de files relacionades (assignatures que s'ha matriculat) complix una mateixa condició (estan aprovades), hi ha diverses possibles solucions, **depenent de cada cas**. Vegem-les:

a) Amb l'operador ALL
b) Comparant quantitats
c) Doble negació:
    1. usant la quantitat 0
    2. usant l'operador NOT EXISTS
    3. usant l'operador NOT IN

Anem a vore diferents solucions per a una mateixa consulta: "Mostra els alumnes que ho han aprovat tot".

a) Amb l'operador ALL

"Mostra els alumnes on el 5 és menor o igual que **totes** les seues notes".

```
SELECT nom
FROM alumnes
WHERE 5 <= ALL (SELECT nota
                FROM matricules
                WHERE matricules.alumne = alumnes.codi );
```

Nota: hem de posar el ALL a la dreta de l'operador relacional. La següent consulta és sintàcticament incorrecta: ... WHERE ~~ALL (SELECT ...) >= 5~~;

b) Comparant quantitats

"Mostra els alumnes on la **quantitat** de matrícules seues és igual a la **quantitat** de matrícules seues que estan aprovades".

Esta solució consisteix en comparar els *COUNT* de 2 subconsultes:

```
SELECT nom
FROM alumnes
WHERE (SELECT count(*)
                FROM matricules
                WHERE matricules.alumne = alumnes.codi )
      =
   (SELECT count(*)
                FROM matricules
                WHERE matricules.alumne = alumnes.codi
                   AND matricules.nota >= 5 )
```

c) Doble negació

Si recordem una de les lleis de De Morgan, deia que: A = NOT (NOT A)

Per tant, trobarem una consulta equivalent amb la "doble negació". En el nostre exemple seleccionarem aquells alumnes que **NO** compleixen la condició **contrària**. És a dir: "Alumnes que no en tenen cap suspesa".

c.1) Doble negació: usant la quantitat 0

"Mostra els alumnes on **0 és la quantitat** de **matrícules seues suspeses**".

```
SELECT nom
FROM alumnes
WHERE 0 = (SELECT count(*)
                        FROM matricules
                        WHERE matricules.alumne = alumnes.codi
                           AND matricules.nota < 5 );
```

c.2) Doble negació: usant l'operador NOT EXISTS

"Mostrarem un alumne si **NO EXISTEIX** una matrícula d'eixe alumne que **no estiga aprovada**":

"Mostrarem un alumne...
... si NO EXISTEIX una matrícula...
... d'eixe alumne...
... que no estiga aprovada"

```
SELECT nom
FROM alumnes
WHERE NOT EXISTS (SELECT *
                  FROM matricules
                  WHERE matricules.alumne = alumnes.codi
                      AND matricules.nota < 5) );
```

Per a cada fila de la consulta principal el NOT EXISTS produïx un resultat vertader quan no s'obté cap fila resultant en la subconsulta. Fals en cas contrari.

Veiem que les dos últimes solucions són pràcticament iguals. És a dir: són equivalents les següents expressions:

EXISTS / NOT EXISTS	Comparar quantitat amb 0
... WHERE EXISTS (SELECT * FROM...)	... WHERE 0 < (SELECT count(*) FROM...)
... WHERE NOT EXISTS (SELECT * FROM...)	... WHERE 0 = (SELECT count(*) FROM...)

c.3) Doble negació: usant l'operador NOT IN

"Mostra els alumnes que **no estan** entre els alumnes que tenen alguna matrícula **suspesa**".

```
SELECT nom
FROM alumnes
WHERE codi NOT IN (SELECT alumne
                   FROM matricules
                   WHERE nota <5 );
```

Un altre exemple: "Mostra els alumnes tals que totes les assignatures que s'ha matriculat són de 1r curs".

Solució a) ALL:

```
SELECT nom
FROM alumnes
WHERE 1 = ALL (SELECT assignatures.curs
                    FROM matricules, assignatures
                    WHERE matricules.assig = assignatures.codi
                       AND matricules.alumne = alumnes.codi);
```

Solució b) Comparar quantitats:

```
SELECT nom
FROM alumnes
WHERE (SELECT count(*)
            FROM matricules
            WHERE matricules.alumne = alumnes.codi)
      = (SELECT count(*)
            FROM matricules, assignatures
            WHERE matricules.assig = assignatures.codi
               AND assignatures.curs = 1);
```

Solució c.1) Doble negació. Quantitat 0:

```
SELECT nom
FROM alumnes
WHERE 0 = (SELECT count(*)
                    FROM matricules, assignatures
                    WHERE matricules.assig = assignatures.codi
                       AND matricules.alumne = alumnes.codi
                       AND assignatures.curs != 1);
```

Solució c.2) Doble negació. NOT EXISTS:

```
SELECT nom
FROM alumnes
WHERE NOT EXISTS (SELECT *
                  FROM matricules, assignatures
                  WHERE matricules.assig = assignatures.codi
                    AND matricules.alumne = alumnes.codi
                    AND assignatures.curs != 1);
```

Solució c.3) Doble negació. NOT IN:

```
SELECT nom
FROM alumnes
WHERE codi NOT IN (SELECT matricules.alumne
                   FROM matricules, assignatures
                   WHERE matricules.assig = assignatures.codi
                     AND assignatures.curs != 1);
```

EXERCICIS DE SUBCONSULTES: ELS RELACIONATS COMPLIXEN UNA CONDICIÓ (BD lliga1213)

21. Equip (nom llarg) que en tots els partits (jugats, clar) ha superat el 65% de possessió del baló jugant en casa.

22. Equip (nom llarg) que en tots els partits (jugats, clar) ha superat el 55% de possessió del baló jugant fora.

23. Nom curt dels equips que mai han perdut a casa.

4.3.5. Seleccionar una fila si totes les files que compleixen una mateixa condició estan relacionades amb ella.

Per exemple:

"Mostra els alumnes que s'han matriculat de totes les assignatures de 1r curs".

Quan volem mostrar una fila (alumne) si complix la condició d'estar relacionada amb MOLTES altres files (matriculat de totes les assignatures de 1r), hi ha diverses possibles solucions, **depenent de cada cas**. Vegem-les:

~~a) Amb l'operador ALL~~

b) Comparant quantitats

c) Doble negació:
   1. usant la quantitat 0
   2. usant l'operador NOT EXISTS
   3. ~~usant l'operador NOT IN~~

a) Amb l'operador ALL

En els casos de l'apartat 4.3.4 sí que era possible solucionar-ho amb el ALL però no en els casos del 4.3.5.

b) Comparant quantitats

"Mostra els alumnes on la **quantitat** de matrícules d'assignatures de 1r seues és igual a la **quantitat** d'assignatures de 1r".

```
SELECT nom
FROM alumnes
WHERE (SELECT count(*)
                FROM matricules, assignatures
                WHERE matricules.assig = assignatures.codi
                  AND assignatures.curs = 1
                  AND matricules.alumne = alumnes.codi )
      =
   (SELECT count(*)
                FROM assignatures
                WHERE assignatures.curs = 1 );
```

c) Doble negació

"Alumnes tals que no hi ha una assignatura de 1r de la qual no estiguen matriculats".

c.1) Doble negació: usant la quantitat 0

"Mostra els alumnes on **0 és la quantitat** d'assignatures de 1r que **no està matriculat** eixe alumne".

```
SELECT nom
FROM alumnes
WHERE 0 = (SELECT count(*)
           FROM assignatures
           WHERE curs = 1
              AND codi NOT IN (SELECT ass
                               FROM matricules
                               WHERE matricules.alumne=alumnes.codi)
           );
```

c.2) Doble negació: usant l'operador NOT EXISTS

"Mostrarem un alumne si **NO EXISTEIX** una assignatura de 1r de la qual **no estiga matriculat** eixe alumne"

"Mostrarem un alumne...
... si NO EXISTEIX una assignatura...
... de 1r...
... de la qual no estiga matriculat..."
... eixe alumne (el que intentem mostrar)"

```
SELECT nom
FROM alumnes
WHERE NOT EXISTS (SELECT *
                  FROM assignatures
                  WHERE curs = 1
                  AND codi NOT IN (SELECT ass
                                   FROM matricules
                                   WHERE alumne =
                                         alumnes.codi)
```

c.3) Doble negació: usant l'operador NOT IN

"Mostra els alumnes que **no estan** entre els alumnes on hi ha alguna assignatura de 1r que no estan matriculats".

Es podria fer però és massa complicat: ixen 3 subconsultes niuades.

Nota: la solució de comparar quantitats no sempre podrà servir. Per exemple:

"Nom dels ciclistes que han portat tots els mallots".

La següent solució NO serviria:

"Ciclistes tals que la quantitat de mallots existents és igual a la quantitat de mallots portats pel ciclista".

```
SELECT nom
FROM ciclistes
WHERE (SELECT count(*)
         FROM mallots)
       =
       (SELECT count(*)
        FROM portar
        WHERE portar.ciclista = ciclistes.dorsal)
```

Quantitat de mallots existents

Quantitat de mallots portats pel ciclista. Però potser un mallot l'ha dut diverses vegades... i altre mallot cap!

Caldria fer-la amb el NOT EXISTS (o el que és el mateix: 0 = (SELECT count(*)...)):

"Ciclistes tals que **no existeix** un mallot que **no haja sigut portat** per eixe ciclista".

```
SELECT nom
FROM ciclistes
WHERE NOT EXISTS (SELECT *
                  FROM mallots
                  WHERE codi NOT IN (SELECT mallot
                                     FROM portar
                                     WHERE portar.ciclista = ciclistes.dorsal);
```

"Ciclistes tals que no la quantitat de mallots **no portats** per ell **és 0**":

```
SELECT nom
FROM ciclistes
WHERE  0 = (SELECT count(*)
                  FROM mallots
                  WHERE codi NOT IN (SELECT mallot
                                     FROM portar
                                     WHERE portar.ciclista = ciclistes.dorsal);
```

Conclusió: la solució amb el NOT EXISTS (o la de comparar la quantitat 0) sempre ens servirà per als 2 tipus d'exercicis vistos:

- ✓ Seleccionar una fila si totes les files que estan relacionades amb ella compleixen una mateixa condició. ("Totes les que estan, són").
- ✓ Seleccionar una fila si totes les files que compleixen una mateixa condició estan relacionades amb ella. ("Totes les que són, estan").

Però per a això, cal tindre clar el que es demana i saber construir un enunciat tipus "doble negació" a partir d'un enunciat que continga un "tots" o una cosa semblant.

EXERCICIS DE SUBCONSULTES: LES QUE COMPLIXEN UNA CONDICIÓ ESTAN RELACIONADES AMB ELLA (BD lliga1213)

24. Dades dels equips que hagen jugat en casa contra tots els equips.
25. Equips que hagen jugat fora de casa en totes les ciutats.
26. Equips que haja jugat tots els partits de la ciutat de Madrid fora de casa.
27. Equips on els han marcat més de 20 gols a cadascun dels seus porters (a tots).

4.3.6. *DELETE* i *UPDATE* amb subconsultes

Exemple de *DELETE* amb subconsulta

"Esborra aquells alumnes que no estan matriculats de res":

```
DELETE FROM alumnes
    WHERE codi NOT IN (SELECT alumne FROM matricules);
```

Exemple d'*UPDATE* amb subconsulta

"Puja mig punt en el camp *nota_mitja* dels alumnes que ho tenen tot aprovat".

```
UPDATE alumnes
SET nota_mitja = nota_mitja + 0.5
WHERE codi in (SELECT alumne
                    FROM matricules
                    GROUP BY alumne
                    HAVING min(nota>=5));
```

Nota: no podem posar en un UPDATE una taula que estarà també en el FROM de la subselect. Per exemple: "augmenta les notes en mig punt per a les assignatures que amb nota mitja inferior a 5". La següent sentència donaria error:

```
UPDATE matricules
SET nota = nota + 0.5
WHERE assig in (SELECT assig
                    FROM matricules
                    GROUP BY assig
                    HAVING avg(nota) < 5);
```

Exemple d'UPDATE amb subconsulta que enllaça amb taula que s'actualitza

"Posa la nota mitja dels alumnes amb la mitja de les matrícules de cadascú":

```
UPDATE alumnes
SET nota_mitja = (SELECT avg(nota)
                    FROM matricules
                    WHERE matricules.alumne = alumnes.codi);
```

Veiem que en els UPDATE poden haver subconsultes en el SET i/o en el WHERE.

EXERCICIS DE DELETE I UPDATE AMB SUBCONSULTES (BD lliga1213)

28. Esborra les ciutats que no tinguen equip de futbol.

29. Incrementa 1000 euros a cada jugador que haja guanyat algun partit el seu equip;

30. Incrementa 1000 euros a cada jugador per cada partit que haja guanyat el seu equip.

31. Incrementa 1000 euros a cada porter que ha rebut menys de 20 gols;

32. Incrementa 1000 euros a cada jugador per cada gol marcat.

5. DDL: Vistes

5.1 Introducció

Suposem que tenim la taula: ALUMNES = codi + nom + edat + grup + domicili

Ja veiérem que a partir del resultat d'una SELECT podíem crear una taula:

```
CREATE TABLE alumnes_menors AS
    SELECT codi, nom, grup, edat
    FROM alumnes
    WHERE edat <18 ;
```

Eixa instrucció crearà una nova taula amb les columnes especificades en la SELECT (codi, nom, grup i edat) i amb les files que complisquen la WHERE (menors d'edat). Ara bé: els canvis que es facen en la taula alumnes no implicaran canvis en la taula alumnes_menors ni viceversa:

- ✓ Si inserim un nou alumne de 15 anys en la taula alumnes, eixe alumne no estarà en la taula alumnes_menors (ni viceversa).
- ✓ Si esborrem un alumne de alumnes_menors, no s'esborrarà d'alumnes (ni viceversa).
- ✓ Si en la taula d'alumnes modifiquem l'edat alumne que abans era menor d'edat i fem que ara tinga 18 anys, l'alumne no desapareixerà de la taula alumnes_menors.
- ✓ ...

Volem solucionar això. És a dir: volem crear una taula especial a partir d'una altra, de forma que els canvis d'una es reflectisquen en l'altra. Realment, el que farem no serà copiar les dades d'una taula a una altra, sinó que crearem una "finestra" a partir de la qual vore les dades que volem de la taula. Això s'aconsegueix amb les vistes:

```
CREATE VIEW alu_men AS
    SELECT codi, nom, grup, edat
    FROM alumnes
    WHERE edat < 18;
```

Així, amb la "finestra" (vista) anomenada **alu_men**, podrem vore el codi, nom, grup i edat dels alumnes **de la taula alumnes** que siguen menors de 18 anys. Per tant, si inserim un alumne nou de 1DAM en la taula alumnes, també el vorem des de la vista alu_men.

És a dir: si fem

```
SELECT *
FROM alu_men
WHERE grup = "1DAM";
```

... és com si férem:

```
SELECT nom, edat
FROM alumnes
WHERE edat < 18 AND grup = "1DAM";
```

5.2 Què és una vista

Una vista és una taula *virtual*, ja que no té existència pròpia encara que exteriorment ho parega. És a dir, no existeix físicament (no ocupa espai en disc dur), sinó que és com una finestra a partir de la qual només podem veure una part (files i columnes) de les taules sobre les quals està definida.

5.3. Creació d'una vista

```
CREATE VIEW nom_vista  [ (comallista_nom_col) ]
        AS sentència_select
        [ WITH CHECK OPTION ]
```

Noms de les columnes que tindrà la vista. És opcional.

Tipus de la select de la vista:

La select sobre la que es defineix la vista pot ser de qualsevol tipus: pot tindre columnes calculades (expressions), pot ser multitaula, pot tindre unions, group by, subconsultes....

També podem crear vistes a partir de *selects* sobre altres vistes.

Operacions que es poden fer sobre la vista:

En principi, una vista és com una taula. Per tant, podrem fer sobre ella: select, insert, update, delete. Ara bé: cal tindre en compte que les sentències sobre la vista "es tradueixen" a sentències sobre la taula (o taules) sobre les quals està definida. Per tant, sempre podrem fer

select sobre elles però no sempre un insert/delete/update sobre la vista es podrà traduir a un insert/delete/update sobre la taula (o taules) corresponent. Per exemple, si una vista està definida amb un group by, si intentem fer un update sobre la vista, no hi haurà traducció per a fer update sobre la taula corresponent. Per exemple, suposem la següent vista:

```
CREATE VIEW quantitats AS
SELECT grup, count(*) AS quant
FROM alumnes
GROUP BY grup;
```

Podrem fer la següent sentència sobre la vista?

```
UPDATE quantitats
SET quant = 10
WHERE grup = "1ASIX"
```

La resposta és no. En eixe cas es diu que la vista no és *actualitzable*.

Una vista és ***actualitzable*** si la select sobre la qual està definida no té "coses rares": no està creada amb UNION ni DISTINCT, ni GROUP BY, ni subconsultes, ni multitaula...

Clàusula *WITH CHECK OPTION*

Si en la definició d'una vista actualitzable s'inclou la clàusula *WITH CHECK OPTION*, esta no permetrà *inserts* o *updates* que no complisquen les condicions de la vista. Per exemple, si la vista *alu_men* l'haguérem creada amb esta clàusula, les següents instruccions donarien error:

```
INSERT INTO alu_men VALUES (100, "Pep", "1DAM", 20);
UPDATE alu_men SET edat = 18 WHERE edat = 17;
```

5.4. Destrucció d'una vista

```
DROP VIEW nom_vista [ RESTRICT | CASCADE ];
```

- RESTRICT: no deixa esborrar la vista si hi ha altres vistes definides sobre ella.
- CASCADE: si hi ha altres vistes definides sobre ella, també les esborra.

MySQL accepta el RESTRICT i CASCADE, però passa d'ells. És a dir: deixa esborrar una vista V1 encara que hi haja alguna altra vista V2 definida a partir d'ella. Si férem una select sobre V2 donaria error.

5.5. Utilitats de les vistes

- Permetre tindre en una taula informació derivada d'altres taules, de forma que les modificacions en eixes taules primitives també es reflectisquen en la "taula derivada" (vista).

- Facilitar la construcció de *selects* complexes. Així podrem fer *selects* sobre altres "*selects*" ja fetes.

- Permetre donar permisos sobre parts d'una taula (ja vorem el tema de permisos).

EXERCICIS SOBRE VISTES (BD lliga1213)

33. Crear una vista:
    a) Crea la vista *jug_sue* amb el dorsal, nom i lloc dels jugadors de l'equip de codi 'sue' (no existeix en la taula de jugadors però dóna igual). No li poses la clàusula del *check option*.
    b) Comprova el contingut de la vista *jug_sue*. Caldrà fer un *select* sobre la vista. Voràs que no té res, ja que no hi ha jugadors d'eixe equip.

34. Comprovació del funcionament de la vista:
    a) Inserix l'equip tav (Tavernes C.F.) i el sun (Sueca United) en la taula d'equips.
    b) Inserix en la taula de jugadors dos nous davanters: Pep, del Tavernes C.F. i Pau, Sueca United.
    c) Comprova el contingut de la vista jug_sue. Voràs que ara la vista "sí que té" un jugador (Pau). Realment "no el té" però a través de la vista estem mirant els jugadors de la taula de jugadors que són del Sueca United.
    d) Fes que el jugador del Tavernes C.F. ara el fitxe el Sueca United (si cal, canviar-li també el dorsal). Caldrà fer un update.
    e) Torna a comprovar el contingut de la vista jug_sue. Ara estarem veient dos jugadors (Pau i Pep).
    f) Esborra a Pep de la taula jugadors. Cal fer delete sobre la taula.
    g) Esborra a Pau a través de la vista jug_sue. Cal fer delete sobre la vista.
    h) Comprova el contingut de la vista jug_sue. Ja no deu estar ningú dels dos. Comprova que tampoc estan en la taula jugadors.
    i) Intenta inserir a Pau a través de la vista jug_sue. Et donarà error perquè en l'insert no li posem el codi de l'equip i, per tant, quan s'intente inserir en la taula de jugadors no admetrà un null en el camp del equip. Però si no fóra per això, sí que es permet inserir a través d'una vista.

35. Eliminar una vista:
    a) Elimina la vista anterior (jug_sue). No que esborres els seus registres, sinó que la destruïsques. Caldrà fer un drop de la vista. Això no afecta a la taula sobre la qual estava definida.
36. Crear una vista amb el *check option*:
    a) Crea la vista equipets amb el codi, nomcurt i pressupost de tots els equips que tinguen un pressupost inferior a 30 milions d'euros. Fes-ho amb el check option.
    b) Insereix a través de la vista equipets estos equips, en 2 inserts:
        - ✓ Equip "gan", "C.F. Gandia", amb 0 milions d'euros de pressupost.
        - ✓ Equip "and", "Andorra C.F", amb 31 milions d'euros de pressupost. Ha de donar error per no complir la condició del check option.
    c) Esborra a través de la vista equipets els equips de més de 40 m. de pressupost. No donarà error però no esborrarà cap equip, degut al check option.
    d) A través de la vista equipets fes que el nou pressupost del C.F. Gandia ara siga de 31 m. Donarà error per no complir el check option.

37. Crear una vista no actualitzable:
    a) Crea la vista *equips_nombrosos* amb el codi de l'equip, el nom curt, el nom de la ciutat i la quantitat de jugadors de cadascun. Però només amb aquells equips que tinguen més de 30 jugadors en plantilla. Els noms de les columnes seran: codi, nom, ciutat, plantilla.
    b) En la vista *equips_nombrosos* modifica el nom de l'equip del Betis (codi "bet"): ara es dirà "Betis". Donarà un error dient que la vista no és actualitzable. No ho és perquè com té *group by* (i multitaula), no es pot "traduir" eixe *update* sobre la vista a un *update* sobre taules. El mateix passaria si intentem fer *inserts* o *deletes* sobre eixa vista.

38. Crear una vista a partir d'altra (exercici solucionat):
    a) Crea la vista resultats_equips amb estos camps:
        - ✓ equip: codi de l'equip
        - ✓ pgc: quantitat de Partits Guanyats a Casa
        - ✓ pec: quantitat de Partits Empatats a Casa
        - ✓ ppc: quantitat de Partits Perduts a Casa
        - ✓ pgf: quantitat de Partits Guanyats Fora
        - ✓ pef: quantitat de Partits Empatats Fora
        - ✓ ppf: quantitat de Partits Perduts Fora

Este tipus de consulta es fa amb subconsultes dins de la pròpia clàusula *select*. No et preocupes si no t'ix. Estes consultes només s'han vist en algun exercici. La solució seria esta:

```
create view resultats_equips as
    select codi as equip,
            (select count(*) from partits where equipc=e.codi and golsc>golsf) as pgc,
            (select count(*) from partits where equipc=e.codi and golsc=golsf) as pec,
            (select count(*) from partits where equipc=e.codi and golsc<golsf) as ppc,
            (select count(*) from partits where equipf=e.codi and golsf>golsc) as pgf,
            (select count(*) from partits where equipf=e.codi and golsf=golsc) as pef,
            (select count(*) from partits where equipf=e.codi and golsf<golsc) as ppf
    from equips e;
```

b) Crea la vista *classif* amb els camps següents. Crea-la a partir de la vista *resultats_equips*:

equip = Codi de l'equip

pjc = Partits Jugats a Casa
pgc = Partits Guanyats a Casa
pec = Partits Empatats a Casa
ppc = Partits Perduts a Casa
puntsc = Punts a Casa

pjf = Partits Jugats Fora
pgf = Partits Guanyats Fora
pef = Partits Empatats Fora
ppf = Partits Perduts Fora
puntsf = Punts Fora

pjt = Partits Jugats en Total
pgt = Partits Guanyats en Total
pet = Partits Empatats en Total
ppt = Partits Perduts en Total
puntst = Punts en Total

La solució seria:

```
create view classif as
    select equip, (pgc + pec + ppc) as pjc,
                        pgc, pec, ppc, (3*pgc + pec) as puntsc,

                (pgf + pef + ppf) as pjf,
                        pgf, pef, ppf, (3*pgf + pef)  as puntsf,

                (pgc + pec + ppc + pgf + pef + ppf) as pjt,
                        (pgc+ pgf) as pgt,
                        (pec + pef) as pet,
                        (ppc + ppf) as ppt,
                        (3*pgc + pec + 3*pgf + pef) as puntst
    from resultats_equips;
```

c) Crea la vista classif2 amb els camps de *classif* més altres camps calculats: gols marcats (a casa, fora i en total), rebuts (a casa, fora i en total) i els que vulgues.

```
create view classif2 as
select *,   (select sum(golsc)      -- Gols Marcats a Casa
                    from partits
                    where partits.equipc = classif.equip) as gmc,

            (select sum(golsf)      -- Gols Marcats Fora
                    from partits
                    where partits.equipf = classif.equip) as gmf,

            (select sum(golsc)      -- Gols Marcats en Total
                    from partits
                    where partits.equipc = classif.equip)
           + (select sum(golsf)
                    from partits
                    where partits.equipf = classif.equip) as gmt,

           (select sum(golsf)       -- Gols Rebuts a Casa
                    from partits
                    where partits.equipc = classif.equip) as grc,

            (select sum(golsc)      -- Gols Rebuts Fora
                    from partits
                    where partits.equipf = classif.equip) as grf,

           (select sum(golsf)       -- Gols Rebuts en Total
                    from partits
                    where partits.equipc = classif.equip)
           + (select sum(golsc)
                    from partits
                    where partits.equipf = classif.equip) as grt

from classif;
```

d) Estes vistes ens serviran per a fer més fàcils consultes com les següents:

d.1) Mostra la taula classificatòria (vista classif) ordenada pels punts totals descendents. També ha d'eixir (en la primera columna) el nom llarg de l'equip.

```
select equips.nomllarg, classif.*
from classif, equips
where classif.equip = equips.codi
order by puntst desc;
```

d.2) Equips que han aconseguit més del doble de punts a casa que fora de casa. Mostra també els punts totals a casa, els punts totals fora i els punts totals. Decreixent pels punts a casa.

```
select equip, puntsc, puntsf, puntst
from classif
where puntsc > 2*puntsf
order by puntsc desc;
```

d.3) Mostra els punts que tenen els equips de la ciutat amb més habitants i de la ciutat amb menys habitants. Mostra també el nom de la ciutat, el nom curt de l'equip i els habitants de la ciutat.

```
select ciutats.nom as ciutat, equips.nomcurt as equip,
       ciutats.habitants, classif.puntst
from ciutats, equips, classif
where classif.equip = equips.codi
   and equips.ciutat = ciutats.codi
   and habitants in (select max(habitants)
                        from ciutats, equips
                        where ciutats.codi = equips.ciutat
                     union
                     select min(habitants)
                        from ciutats, equips
                        where ciutats.codi = equips.ciutat)
order by 1, 2
```

6. DCL (Permisos)

Fins ara hem fet servir només l'usuari 'root', que és l'administrador, i que disposa de tots els privilegis en MySQL. Però no convé que tots els usuaris tinguen tots els permisos.

MySQL permet definir usuaris i assignar-los permisos. Vorem només algunes opcions de les sentències sobre usuaris i permisos però en el manual hi ha moltes més.

6.1. Nivells de permisos

En MySQL existeixen 5 nivells de permisos o privilegis:

- Globals: s'apliquen a totes les BD d'un servidor.
- De base de dades: per a una BD individual (i per a tots els seus objectes).
- De taula: per a una taula individual (i per a totes les seues columnes).
- De columna: per a una columna d'una taula concreta.
- De rutina: per als procediments emmagatzemats. No ho hem vist, però en MySQL es poden guardar procediments consistents en diverses consultes SQL.

6.2. Crear usuaris

Es pot fer de 2 formes:

- Amb la mateixa sentència que serveix per a donar permisos:

```
GRANT USAGE ON *.* TO alumne IDENTIFIED BY 'clau';
```

- Amb una sentència exclusiva per a crear usuaris (a partir de la versió 5.0.2):

```
CREATE USER alumne IDENTIFIED BY 'clau';
```

A partir d'eixe moment, podrem obrir una sessió MySQL amb:

mysql -u alumne -pclau

Nota: en una mateixa clàusula podem crear diversos usuaris (separats per coma):

CREATE USER pep IDENTIFIED BY '1234', pepa IDENTIFIED BY '5555', ...

6.3. Esborrar usuaris

DROP USER alumne;

6.4. Concedir permisos

Per a donar permís per a seleccionar dades d'una taula concreta a un usuari, farem:

```
GRANT SELECT
    ON institut.assignatures
    TO alumne;
```

Esta sentència dóna permís a l'usuari 'alumne' per a executar sentències SELECT sobre la taula 'assignatures' de la BD 'institut'. Si ara obrim una sessió amb l'usuari 'alumne', podrem executar sentències com:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| institut |
+----------+

mysql> USE institut;
Database changed

mysql> SHOW TABLES;
+--------------------+
| Tables_in_institut |
+--------------------+
| assignatures       |
+--------------------+

mysql> SELECT * FROM assignatures;
+---------+-----+
| nom     | dept|
+---------+-----+
| PRG     | inf |
| BD      | inf |
| Mate-1  | mat |
| Val-1   | val |
| HTML    | inf |
|         |     |
+---------+-----+
```

Este usuari només veu la BD institut i, dins d'ella, només veu la taula assignatures. I només podrà fer consultes sobre esta taula, però no podrà ni afegir, ni modificar, ni esborrar dades en ella. I tampoc podrà crear ni destruir taules ni bases de dades, clar.

Altre exemple:

```
GRANT SELECT(assig, ava, nota), UPDATE(domicili, tel)
    ON institut.notes
    TO alumne, paremare;
```

La sintaxis (simplificada) de GRANT és:

```
GRANT tipus_de_permís [(llista_de_camps)]
    ON {nom_taula | * | *.* | nom_bd.*}
    TO usuari [IDENTIFIED BY 'contrasenya']
    [WITH GRANT OPTION]
```

Nota: les clàusules GRANT i TO admeten llistes (separat per comes).

La clàusula WITH GRANT OPTION és donar permís per a donar eixos permisos.

En resum és:

GRANT llista_de_permisos

ON taula_o_bd

TO llista_usuaris

WITH GRANT OPTION (part opcional)

Exemples d'ús de clàusula ON:

ON nom_bd.nom_taula Dóna permisos a una taula d'una BD.

ON nom_bd.* Dóna permisos a tota una base de dades.

ON *.* Dóna permisos a tots els objectes (bases de dades, taules, etc).

Els tipus de permís que es poden donar són:

- **USAGE:** equival a cap permís (serveix per a crear usuaris)
- **SELECT:** per a fer consultes de taules (permet especificar columnes)
- **INSERT:** per a inserir dades en una taula (permet especificar columnes)
- **UPDATE:** per a modificar dades d'una taula (permet especificar columnes)
- **DELETE:** per a esborrar dades d'una taula
- **CREATE:** per a crear taules
- **DROP:** per a eliminar taules
- **ALTER:** per a modificar l'estructura de les taules
- **ALL:** tots els permisos

6.5. Revocar permisos

Per a revocar (llevar) permisos s'usa la sentència REVOKE. Per exemple:

REVOKE SELECT(nota)

ON institut.assignatures

FROM alumne;

En esta sentència estem llevant a l'usuari *alumne* el permís de vore la *nota* en la taula *assignatures*. Com veiem, la sintaxi és similar a la de GRANT.

La sintaxi és esta:

```
REVOKE tipus_de_permís [(llista_de_camps)]
    ON {nom_taula | * | *.* | nom_bd.*}
    FROM usuari
```

Nota: les clàusules REVOKE i FROM admeten llistes (separat per comes).

Veiem ara uns casos especials:

Revocar tots els permisos, llevat de GRANT OPTION:

REVOKE ALL **ON** *.* **FROM** usuari

Revocar el GRANT OPTION:

REVOKE GRANT OPTION **ON** *.* **FROM** usuari

Revocar tots els permisos, inclòs el GRANT OPTION:

REVOKE ALL, GRANT OPTION **FROM** usuari

(Nota: en este cas no es posa el ON *.*)

6.6. Mostrar els permisos d'un usuari

SHOW GRANTS FOR alumne;

Això ens mostra els permisos en forma de llistes de sentències de GRANT:

```
+-----------------------------------------------------------------+
| Grants for alumne@%                                             |
+-----------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'alumne'@'%' IDENTIFIED BY PASSWORD '*5...'  |
| GRANT SELECT ON `institut`.`assignatures` TO 'alumne'@'%'       |
|                                                                 |
+-----------------------------------------------------------------+
```

6.7. Noms d'usuaris i contrasenyes

Noms d'usuaris

Quan donem permisos a un usuari, estem indicant un nom d'usuari. Però eixe nom pot estar format per dos parts, separats pel signe @:

nom_usuari@nom_maquina

El *nom_maquina* (host) pot ser 'localhost', altre nom o una IP. Ens indica des de quina màquina tindrà eixe usuari els permisos que se li han donat.

Exemples:

- ✓ GRANT USAGE ON * TO **alumne@10.28.56.15** IDENTIFIED BY 'clau';
  L'usuari 'alumne' només pot connectar-se des d'un ordinador amb eixa IP.

- ✓ GRANT USAGE ON * TO **alumne@localhost** IDENTIFIED BY 'clau';
  L'usuari 'alumne' només pot connectar-se des del mateix ordinador on està executant-se el servidor.

- ✓ GRANT USAGE ON * TO **alumne** IDENTIFIED BY 'clau';
  L'usuari 'alumne' podrà connectar-se des de qualsevol màquina. En este cas, el SHOW GRANTS FOR ho indica usant el comodí '%' per al nom de la màquina: alumne@%

Contrasenyes:

No és obligat assignar una contrasenya, però és recomanable.

Si quan donem privilegis a un usuari existent indiquem una altra contrasenya en l'IDENTIFIED BY, li canvia la contrasenya.

EXERCICIS SOBRE USUARIS I PERMISOS (BD lliga1213)

39. Crea els usuaris entrenador, jugador, aficionat, prova i prova2, que tinguen com a clau el mateix que el nom d'usuari

40. Canvia el password de l'entrenador i posa-li: mister. No està als apunts. Consulta a Internet com fer-ho.

41. Esborra l'usuari prova.

42. Dóna permís a l'usuari entrenador per a vore totes les dades de la bd, de forma que l'entrenador també puga donar eixos permisos a qui vullga.

43. Mostra els permisos de l'usuari entrenador.

44. Dóna permís a l'usuari aficionat per a vore les dades de la vista classif2.

45. Dóna permís de lectura a prova sobre la taula equips per a vore totes les dades llevat del pressupost. Fes-ho així:
    a. Dóna-li permís per a vore totes les dades d'equips
    b. Lleva-li permís per a vore el pressupost

46. Dóna permís a l'entrenador per a modificar el lloc i el nom dels jugadors

47. Dóna permís a l'entrenador per a esborrar jugadors, golejadors i porters.

48. Lleva tots els permisos a l'usuari prova.

49. Dóna permís per a modificar totes les dades de la taula equips a l'usuari prova però només per a fer-ho des del mateix ordinador on està guardada la base de dades.

50. Dóna permís a prova per a vore només les dades dels partits del barça. Esta té truc.

7. TCL (Transaccions)

7.1. Definició

Les transaccions són un concepte fonamental de tots els sistemes de base de dades. Una transacció és un conjunt de sentències INSERT, UPDATE i/o DELETE que constitueixen una operació única. És a dir: podrem fer que s'executen totes les sentències de la transacció o cap.

Les sentències entre el BEGIN i COMMIT es diuen **bloc de transacció** o **transacció**.

Nota: les transaccions no funcionen en el motor MyISAM de MySQL. Cal usar InnoDB.

7.2. Sintaxi

Les sentències que volem que formen part d'una transacció han de començar amb el comandament BEGIN i han d'acabar amb el comandament COMMIT (per a que es facen efectives les sentències) o ROLLBACK (per a anul·lar les sentències de la transacció):

BEGIN; (sentència 1); (sentència 2); ... (sentència N); **COMMIT**;	**BEGIN**; (sentència 1); (sentència 2); ... (sentència N); **ROLLBACK**;
Es CONFIRMA la transacció.	S'ANUL·LA la transacció.

7.3. Utilitats

1) Si ocorre algun error durant la transacció, podem anul·lar la transacció i així cap dels passos no afectarà la base de dades (es desfan les accions realitzades).

2) Els estats entremitjos en una transacció sí que és visible en eixa transacció però no en unes altres transaccions concurrents. És a dir, altres sessions no veuen cap canvi d'eixa transacció fins que no s'ha fet el COMMIT.

3) Problema de l'ou i la gallina (inserció en dos taules interdependents).

4) Podem desfer les accions fetes mentre no confirmem la transacció.

5) Podem desfer part de les accions mentre no confirmem la transacció (SAVEPOINT)

Exemple utilitat 1: prevenció d'errors inesperats

Pensem en una base de dades d'un banc que conté saldos dels comptes dels clients. Volem enregistrar un pagament de 100 € des del compte d'Alícia fins al compte de Pep:

```
UPDATE comptes
        SET saldo = saldo - 100
        WHERE nom = 'Alícia';

UPDATE comptes
        SET saldo = saldo + 100
        WHERE nom = 'Pep';
```

Este tipus d'operacions sol ser més complicat i implicar més sentències (introduir la transferència en un històric de moviments, etc). Però el que importa és que hi ha més d'una sentència d'actualització per a realitzar esta transferència bancària.

Els clients del nostre banc voldran que se'ls assegure que: o bé estes 2 actualitzacions es fan, o que cap d'elles es fa. No voldrien que el saldo es decrementara d'Alícia i no s'incrementara en Pep.

Necessitem una garantia que si hi ha algun error en alguna d'eixes operacions anteriors, que cap dels passos executats tinga efecte. Agrupar les actualitzacions en una transacció ens dóna esta garantia:

```
BEGIN;
  UPDATE comptes SET saldo = saldo – 100 WHERE nom = 'Alícia';
  UPDATE comptes SET saldo = saldo + 100 WHERE nom = 'Pep';
COMMIT;
```

Si després del 1r UPDATE es perdera la connexió amb la BD, no es faria el 2n UPDATE però tampoc el COMMIT. Això fa que els efectes del 1r UPDATE no es facen efectius realment ne la base de dades.

Exemple utilitat 2: canvis entremitjos transparents a altre sessions

Una altra propietat important de bases de dades transaccionals està relacionat amb la idea d'actualitzacions atòmiques: quan les transaccions múltiples estan corrent concurrentment (és a dir: simultàniament, al mateix temps), cada una no deu veure els canvis incomplets fets per altres.

Per exemple, si una transacció està ocupada sumant tots els saldos dels comptes, no sumarà els saldos fins que no haja acabat la transacció que està fent el traspàs d'un compte a altre.

SESSIÓ 1	SESSIÓ 2
BEGIN; UPDATE comptes SET saldo = saldo - 100 WHERE nom = 'Alícia'; UPDATE comptes SET saldo = saldo + 100 WHERE nom = 'Pep'; COMMIT;	SELECT SUM(saldo) FROM comptes

Si la sessió 2 s'executa en el moment en que acaba el primer UPDATE, no veurà els efectes d'eixe UPDATE. Si no haguérem posat la sessió 1 dins d'una transacció, la sessió 2 s'hauria deixat 100 € per sumar, ja que haguera sumat el saldo d'Alícia sense els 100 i el saldo de Pep sense haver-li-ho sumat encara.

Exemple utilitat 3: problema de l'ou i la gallina

Suposem una base de dades d'una acadèmia d'informàtica, on hi ha tants alumnes com ordinadors. Cada alumne ha de tindre assignat un ordinador. I cada ordinador ha de tindre assignat un alumne:

ORDINADORS = codi + carac + alumne

ALUMNES = exp + nom + ordinador

Nota: Este disseny té redundància d'informació, però ens serveix per a l'exemple.

Per a introduir un ordinador deurà tindre assignat un alumne, però per a introduir un alumne deurà tindre assignat un ordinador.

Problema: el SGBD no ens deixarà inserir res (problema de l'ou i la gallina).

Solució: en la definició de les claus alienes, hem de posar la propietat DEFERRABLE, que farà que les comprovacions d'existència es realitzen en el moment en que acabe la transacció. Però! MySQL no admet el DEFERRABLE. Sí que és admès per SGBD com Oracle o PostgreSQL.

```
Create table ordinadors (
        Codi integer primary key,
        Carac char(50),
        Alumne integer not null references alumnes DEFERRABLE);
Create table alumnes (
        Exp integer primary key,
        Nom char(40),
        Ordinador integer not null references ordinadors DEFERRABLE);
BEGIN;
        Insert into ordinadors(1, 'Pentium III', 1001);
        Insert into alumnes(1001, 'Pep', 1);
COMMIT;
```

Quan inserim l'ordinador, no es comprova que existisca l'alumne 1001, ja que al ser la seua clau aliena de tipus DEFERRABLE, ho comprovarà en el moment del COMMIT. Igual passa en la inserció de l'alumne. Quan fem el COMMIT, es comprovaran les 2 coses i sí que es compliran.

Exemple utilitat 4: desfer canvis voluntàriament

Si durant l'execució de la transacció decidim que no volem acabar-la (potser ens adonem que el saldo d'Alícia es feia negatiu), podem executar l'ordre ROLLBACK en compte de COMMIT. Això farà que s'anul·le l'efecte de les actualitzacions de la transacció que estaven fetes:

```
BEGIN;
    UPDATE comptes
        SET saldo = saldo - 100
        WHERE nom = 'Alícia';
    -- Ací ens adonem que el compte d'Alícia és negatiu.
    -- Per tant, volem anul·lar l'update anterior, i farem el ROLLBACK
ROLLBACK;
```

Exemple utilitat 5: desfer parcialment la transacció en curs

És possible controlar les accions en una transacció d'una manera més detallada. És a dir, en compte de descartar tota la transacció, podem descartar només part de la transacció.

Això es fa amb els SAVEPOINTS.

SAVEPOINTS

Els savepoints són "punts de salvament", que permeten descartar selectivament parts de la transacció (admetent la resta de la transacció).

Exemple:

Suposem que traspassem 100 € del compte d'Alícia al compte de Pep, però que, només haver-ho fet, ens adonem que volíem traspassar-ho al compte de Maria i no al de Pep. Si ens haguérem curat en salut utilitzant transaccions i savepoints, podríem desfer només unes sentències de la transacció i no tota:

```
BEGIN;
        UPDATE comptes SET saldo = saldo - 100.00
           WHERE nom = 'Alícia';
        SAVEPOINT seguretat1;
        UPDATE comptes SET saldo = saldo + 100.00
           WHERE nom = 'Pep';
        -- Ai, no! No li ho havia d'haver sumat a Pep, sinó a Maria.
        ROLLBACK TO seguretat1;
        UPDATE comptes SET saldo = saldo + 100.00
           WHERE nom = 'Maria';
COMMIT;
```

Després d'haver definit un savepoint amb *SAVEPOINT nom_del_savepoint*, podré retrocedir, si cal, a eixe punt mitjançant l'ordre *ROLLBACK TO nom_del_savepoint*.

Les accions fetes en la transacció entre la definició del SAVEPOINT i el ROLLBACK TO corresponent es descarten, però es queden els canvis anteriors a la definició del SAVEPOINT.

Podem retrocedir a un savepoint sempre que volguem:

```
BEGIN;
        Acció 1;
        Acció 2;
        SAVEPOINT segur1;
        Acció 3;
        Acció 4;
        ROLLBACK TO segur1;
        Acció 5;
        ROLLBACK TO segur1;
        ...
COMMIT;
```

Ara bé, si retrocedint a un savepoint en concret, s'esborraran automàticament tots els savepoints definits després d'eixe savepoint. Açò ho fa el sistema per a alliberar recursos.

```
BEGIN;
        Acció 1;
        Acció 2;
        SAVEPOINT segur1;
        Acció 3;
        Acció 4;
        SAVEPOINT segur2;
        Acció 5;
        ROLLBACK TO segur2
        Acció 6;
        ROLLBACK TO segur1; -- Ací s'esborrarà el SAVEPOINT segur2
        ...
COMMIT;
```

Savepoints: exemple 1

La següent transacció inserirà els clients 1000 i 1004

```
begin;
        insert into clients values(1000);
        savepoint s1;
        insert into clients values (1001);
        savepoint s2;
        insert into clients values (1002);
        rollback to s2;
        insert into clients values (1003);
        rollback to s1;
        insert into clients values (1004);
commit;
```

Savepoints: exemple 2

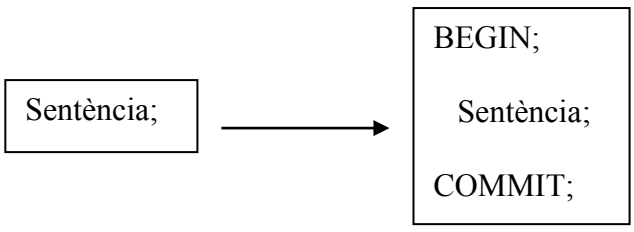
Què fa la següent transacció?

```
begin;
        insert into clients values(1000);
        savepoint s1;
        insert into clients values (1001);
        savepoint s2;
        insert into clients values (1002);
        rollback to s1;
        insert into clients values (1003);
        rollback to s2;
        insert into clients values (1004);
commit;
```

Donarà error en el *rollback to s2* pq el *rollback to s1* esborra el *savepoint s2*:

Nota:

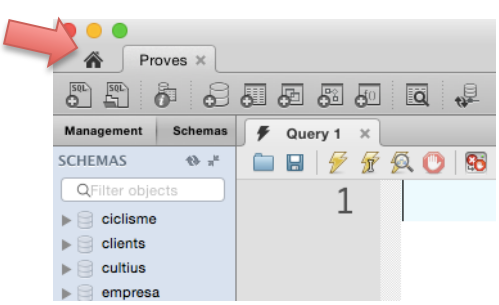
Si no treballem amb transaccions, PostgreSQL tracta cada sentències SQL com si fóra una transacció d'una sola sentència:

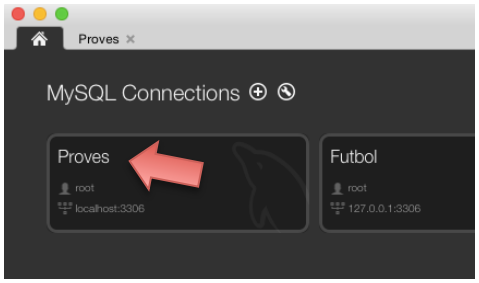


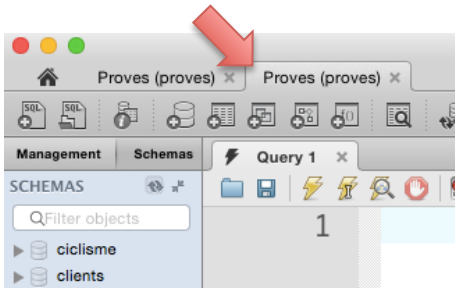
És a dir, si per exemple s'està fent un UPDATE molt costós, mentre dura l'actualització, qualsevol altre intent d'actualització de la taula serà paralitzat fins que no acabe el primer UPDATE.

EXERCICIS SOBRE TRANSACCIONS

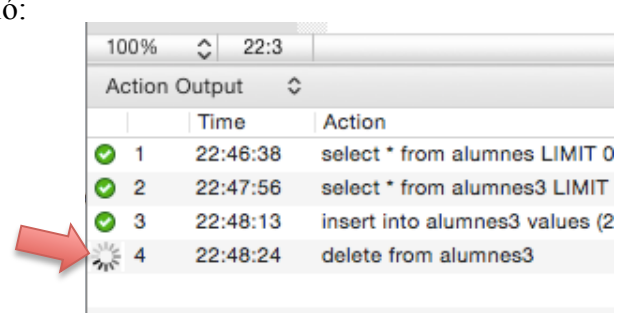
51. Comprova que es poden desfer els canvis d'una transacció no acabada:
    a. Inicia una transacció
    b. Fes diversos update/delete en algunes taules
    c. Anul·la la transacció
    d. Comprova que els canvis que has fet no han tingut efecte.
52. Comprova que mentre una transacció no acaba, una altra no pot vore els canvis fets:
    e. Inicia una transacció i fes algun canvi (update/delete) en una taula i comprova que, de moment, els canvis es poden veure.
    f. Obri en una altra sessió altra connexió a la mateixa bd:







    g. Comprova en esta 2a sessió que els canvis fets en la 1a sessió no es poden vore (ja que no està acabada la transacció).
    h. Ves a la 1a sessió i confirma la transacció.
    i. Ves a la 2a sessió i comprova que ara sí que estan els canvis fets.
53. Comprova que mentre una transacció no acaba, si havia fet una modificació en alguna taula, una altra taula no pot fer cap modificació en eixa taula:
    j. Inicia una transacció i fes algun canvi (update/delete) en una taula i comprova que, de moment, els canvis es poden veure.
    k. Obri en una altra sessió altra connexió a la mateixa bd.
    l. Intenta fer un canvi (update/delete) en la mateixa taula. Voràs com el servidor deixa en espera eixa petició esperant que acabe la transacció de la primera sessió:



    m. Comprova que, quan acabe la transacció de la 1a sessió (tant si la confirmes com si l'acabes), l'update/delete de la 2a sessió acabarà d'immediat.
54. Comprova el funcionament dels *savepoints*:
    n. Inicia una transacció i fes algun canvi en una taula
    o. Posa un savepoint
    p. Fes algun altre canvi
    q. Cancela la transacció només fins al savepoint
    r. Confirma la transacció i comprova quin dels dos canvis ha quedat.

ANNEX: CÒPIES DE SEGURETAT

Fer una còpia de seguretat d'una Base de Dades és una feina d'administració obligatòria per mantenir la informació protegida. MySQL et permet realitzar esta senzilla tasca amb ordres des de la consola, una volta hem entrat en l'entorn textual de MySQL:

Exportar una base de dades:

Sintax:

```
mysqldump -h servidor -u usuari -p nomBaseDades > nomFitxer
```

Exemple :

```
mysqldump -h localhost -u root -p lliga1213 > lliga1213.sql
```

Importar una base de dades:

Sintaxi:

```
mysql -h servidor -u usuari -p nomBaseDades < nomFitxer
```

Exemple:

```
mysql -h localhost -u root -p lliga1213 < lliga1213.sql
```