SQL - MariaDB

Contingut

Llenguatge de definició de dades SQL

Introducció

Instal·lació d'aplicacions

Primer instal·larem el sistema gestor de bases de dades que utilitzarem. En el nostre cas farem servir:

- MariaDB (https://mariadb.org/) : Es la que utilitzarem al curs. La podem descarregar des de https://mariadb.org/download/
- MySQL (https://www.mysql.com/) : Aquesta aplicació no la que utilitzarem (donat que és llicència GNU, però sota la propietat d'Oracle), però es pot descarregar des de https://dev.mysql.com/downloads/mysql/

En segon lloc instal·larem l'aplicació que utilitzarem per treballar amb el sistema gestor de bases de dades:

- WorkBench (https://www.mysql.com/products/workbench/) : Eina de treball de MySQL. Es pot descarregar des de https://dev.mysql.com/downloads/workbench/
- HeidiSQL (https://www.heidisql.com/download.php) : Eina per treballar amb diferents sistemes gestors.
- PhpMyAdmin (https://www.phpmyadmin.net/downloads/) : Eina de treball online per MariaDB y MYSQL.

Accedir i sortir a mariaDB/mysql

Executar des de la línia de comandaments del Windows (executeu COMMAND o CMD) o des d'un terminal de Linux. Executeu l'ordre:

```
mysql -u root -p
```

El paràmetre -u indica l'usuari, i -p indica que sol.licitarà la contrasenya. Hem de tenir la ruta al PATH, si no buscar el directori BIN del MYSQL. Apareixerà com a prompt:

```
mysql>
```

Si sabem la base de dades amb la que volem treballar farem:

```
mysql base_de_dades
```

Si en canvi entrem només amb l'ordre mysql després haurem d'executar l'ordre USE per indicar-li amb quina base de dades volem treballar:

```
USE base_de_dades
```

Per sortir fem:

```
quit
```

Ordres a mysql

Poden ocupar diverses línies.

El programa sap que finalitza una ordre quan llegeix al final “;”.

Comentaris en SQL

Es poden utilitzar, com en qualsevol llenguatge de programació, per documentar el codi o per deshabilitar temporalment instruccions.

- **#** : es poden incorporar en la mateixa línia a executar o en una línia apart. Fa comentari tot el que troba entre els guions i el final de la línia. Si ocupa vàries línies ha d’aparèixer al principi de totes les línies.

Exemple:

```
#Selecciona el nom dels alumnes
SELECT nom #nom del alumne
   FROM alumnes; #taula d’alumnes
```

- **/* ... */** : es poden incorporar en el codi que es vol executar, en línies separades o en el codi executable. Fa comentari tot el que troba entre els dos símbols. Pot ocupar diverses línies.

Exemple:

```
/*Selecciona el nom dels alumnes
del centre*/
SELECT /*nom del alumne*/ nom
   FROM alumnes; /*taula d’alumnes*/
```

Crear bases de dades i afegir registres amb SQL en l’entorn de MySQL

Guardar l’estructura en una arxiu de text amb extensió *.sql* (no es obligatori que sigui un fitxer amb aquesta extensió, però és recomenable per poder reconèixer el tipus de fitxer). Executar l’ordre des de la línia de comandaments de Windows:

```
mysql base_de_dades<fitxer.sql
```

Creació i eliminació d'una base de dades

Per crear una base de dades utilitzarem la sentència:

```
CREATE DATABASE [IF NOT EXISTS] nom_base_dades;
```

Exemple:

```
 DROP DATABASE IF EXISTS institut;
 CREATE DATABASE institut DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
 USE institut;
```

Per eliminar una base de dades utilitzarem la sentència:

```
DROP DATABASE nom_base_dades;
```

Exemple:

```
DROP DATABASE institut;
```

Per utilitzar una base de dades haurem d'activar-la utilitzant la sentència:

```
USE nom_base_dades;
```

o bé quan entrem dintre l'aplicatiu entrarem indicant la base de dades que utilitzarem:

```
MYSQL nom_base_dades
```

Creació de taules

Per crear una taula fem servir la sentència:

```
CREATE TABLE nom_taula
   (definicio_columna
   [opcions_taula]
   );
```

on definicio_columna és:

```
nom_columna  tipus_dades  [opcions_columna]
```

Exemple:

```
CREATE TABLE alumne
   (codi            INTEGER        NOT NULL,
   nom              CHAR(20)       NOT NULL,
   cognoms          CHAR(40)       NOT NULL,
   data_naixement   DATE,
   PRIMARY KEY (codi)
   );
```

Observació:

- Per veure les taules creades utilitzarem la sentència SHOW TABLES;
- Per veure el contingut de la taula utilitzem la sentència SHOW COLUMNS FROM alumne;

El procés que farem sempre és:

1r) Decidir el nom de la taula (nom_taula)
2n) Donar nom a les columnes (nom_columna)
3r) Definir el tipus de dades/domini i les definicions per defecte i restriccions
4t) Dir les opcions de la taula

Tipus de dades

Els principals tipus de dades que podem utilitzar són els següents:

Classe	Tipus	Descripció	Rang
Númeric	SMALLINT	Nombre enters petits	-32718 a 32767
	INT o INTEGER	Nombre enters	-2147483648 a 2147483647
	FLOAT	Nombre en coma flotant i precisió simple	
	DOUBLE o REAL	Nombre en coma flotant i precisió doble	
	DECIMAL o NUMERIC (M,D)	Nombre en coma flotant emmagatzemat com a cadena,on M indica la amplada de visualització i D el nombre de decimals	
Cadena	CHAR	Cadena de longitud fixa, on D indica la longitud	1 a 255
	VARCHAR	Cadena de longitud fixa, on D indica la longitud	0 a 255
	TEXT	Text	0 a 65535
	ENUM	Enumera tipus (limita a una llista)	Exemple: ENUM('a','b','c')
Data i hora	DATE	Data en format AAAA-MM-DD	1000-01-01 a 9999-12-31
	TIME	Hora en format hh:mm:ss	
	DATETIME	Data i hora en format AAAA-MM-DD hh:mm:ss	1000-01-01 00:00:00 a 9999-12-31 23:59:59
	TIMESTAMP	Data i hora en format AAAAMMDDhhmmss	

Exemple:
Crear una taula anomenada *alumne* amb les següents característiques:

-**Codi**: enter, el màxim que pot tenir 2000
-**Nom**: cadena de longitud màxima 20
-**Cognoms**: cadena de longitud màxima 40
-**Inscripció**: data i hora d'inscripció
-**Data_naixement**
-**Via_acces**: conté una lletra
-**Nota d'acces**: nombre de la nota amb dos decimals

La solució és:

```
CREATE TABLE alumne
   (codi                SMALLINT(4),
   nom                  VARCHAR(20),
   cognoms              VARCHAR(40),
   inscripcio           DATETIME,
   data_naixement       DATE,
   via_acces            CHAR(1),
   nota_acces           DECIMAL(4,2)
   )
   ;
```

Opcions d'una columna

Podem tenir les següents restriccions:

Restriccions de columna

Restricció	Descripció
DEFAULT literal	Insereix un valor per defecte
NOT NULL	La columna no pot tenir valors nuls
PRIMARY KEY	La columna és una clau primària
AUTO_INCREMENT	Per columnes de tipus enter, incrementa una unitat començant a 1
UNIQUE	La columna no pot tenir valors repetits
REFERENCES taula [(columna)]	La columna és la clau forana de la columna de la taula especificada
CHECK (condició)	La columna ha de complir les condicions especificades
FOREIGN KEY	Indica la referència a una clau forana

Exemple:
A la taula anterior afegir que el codi és un valor autoincremental i la clau primària, que el nom i els cognoms no poden ser nuls, la via d'accés serà per defecte A. A més volem que els cognoms siguin valors únics i que via_acces referenciï a una altra taula anomenada VIA que conté un camp anomenat via_acces. La solució es:

```
CREATE TABLE via
   (via_acces           CHAR(1) PRIMARY KEY
   );
CREATE TABLE alumne
   (codi                SMALLINT(4)     AUTO_INCREMENT PRIMARY KEY,
   nom                  VARCHAR(20)     NOT NULL,
   cognoms              VARCHAR(40)     NOT NULL UNIQUE,
   inscripcio           DATETIME,
   data_naixement       DATE,
   via_acces            CHAR(1)         DEFAULT 'A' REFERENCES via,
   nota_acces           DECIMAL(4,2)
   )
   ;
```

Opcions d'una taula

Podem tenir les següents restriccions:

Restriccions de taula

Restricció	Descripció
CONSTRAINTS	Utilitza per establir una restricció
AUTO_INCREMENT=n	Primer valor d'AUTO_INCREMENT
CHECK (condicions)	La taula ha de complir les condicions especificades
ENGINE= (tipus_de_taula)	MySQL és capaç de treballar amb diferents tipus de taules, per defecte treballa amb MyISAM. Per poder treballar amb bases de dades relacionals la taula ha de ser del tipus InnoDB

Exemple:
A la taula anterior volem afegir que els valors del nom i cognoms no siguin iguals tots dos alhora i que comenci el valor autoincremental del codi al 1000. Recorda d'eliminar el valor UNIQUE pels cognoms. La solució és:

```
CREATE TABLE via
```

```
   (via_acces           CHAR(1)         PRIMARY KEY
   )
   ENGINE=InnoDB
   ;
CREATE TABLE alumne
   (codi                INT(4)          PRIMARY KEY AUTO_INCREMENT,
   nom                  VARCHAR(20)     NOT NULL,
   cognoms              VARCHAR(40)     NOT NULL UNIQUE,
   inscripcio           DATETIME,
   data_naixement       DATE,
   via_acces            CHAR(1)         DEFAULT 'A' REFERENCES via,
   nota_acces           DECIMAL(4,2),
   UNIQUE (nom,cognoms),
   FOREIGN KEY (via_acces) REFERENCES via(via_acces)
   )
   AUTO_INCREMENT=1000,
   ENGINE=InnoDB
   ;
```

Esborrat de taules

Per esborrar una taula utilitzem la sentència:

```
DROP TABLE nom_taula;
```

Exemple:

```
DROP TABLE via;
```

Modificació de taules

Per modificar una taula utilitzem la sentència:

```
ALTER TABLE nom_taula ...
```

on tenim les següents possibilitats:

- Afegir una columna:

```
ADD definicio_columna;
```

- Modificar una columna:

```
MODIFY nom_columna definicio_columna;
```

- Modificar una columna i canviar el nom:

```
CHANGE nom_columna nou_nom definicio_columna;
```

- Canviar el nom a una taula:

```
RENAME nou_nom_taula;
```

- Esborrar una columna

```
DROP nom_columna;
```

Exemple:

a) Crea una taula anomenada VIA que tingui un camp anomenat codi que sigui INT.
b) Afegeix un altre columna anomenada *descripcio* que sigui del tipus CHAR(4).
c) Modifica la columna anterior de manera que sigui del tipus CHAR(8).
d) Canvia el nom del columna *codi* anomenant-lo *ccc* i que sigui del tipus SMALLINT.
e) Canvia el nom de la taula *via* anomenant-lo *via_acces*.
f) Esborra la columna *descripcio*.

La solució és:

```
CREATE TABLE via
     (codi          INT
     );
ALTER TABLE via ADD descripcio CHAR(4);
ALTER TABLE via MODIFY descripcio CHAR(8);
ALTER TABLE via CHANGE codi ccc SMALLINT;
ALTER TABLE via RENAME via_acces;
ALTER TABLE via_acces DROP descripcio;
```

Creació d'índex

És similar a com funciona l'índex d'un llibre, permet trobar informació sense necessitat de llegir tot el llibre. Un índex és una estructura que proporciona accés ràpid a les files d'una taula. En SQL és una estructura associada a una taula que accelera l'obtenció de files de la taula.

Es crearan índex per les columnes que s'utilitzin per consultes (que s'utilitzen per cercar dades). Les taules amb índex necessiten més espai, per tant la inserció, actualització o eliminació de dades necessitarà més temps.

SQL crea automàticament índex pels valors amb restriccions del tipus PRIMARY KEY i UNIQUE. Per veure els índex creats utilitzem la sentència:

```
SHOW INDEX FROM taula;
```

Per crear índex utilitzem la sentència:

```
CREATE INDEX nom_index ON nom_taula (columnes_index);
```

Exemple:

a) Crea una taula anomenada MATERIA que tingui les següents columnes: CODI_MAT que serà un camp enter i autonumèric, i a més serà la clau principal, MATERIA, camp de text de com a màxim 20 caràcters,

DEPARTAMENT, camp de text de com a màxim 20 caràcters.
b) Comprova els índexs que hi ha creats.
c) Fes ara que el camp MATERIA sigui únic.
d) Torna a comprova els índexs existents.
e) Crea un índex anomenat xdepartament amb el camp DEPARTAMENT.
f) Torna a comprova els índexs existents.

La solució és:

```
CREATE TABLE materia
       (codi_materia          INT             AUTO_INCREMENT PRIMARY KEY,
       materia                VARCHAR(20),
       departament            VARCHAR(20)
       );
SHOW INDEX FROM materia;
ALTER TABLE materia MODIFY materia VARCHAR(20) UNIQUE;
SHOW INDEX FROM materia;
CREATE INDEX xdepartament ON materia (departament);
SHOW INDEX FROM materia;
```

Eliminació d'índex

Per eliminar un índex utilitzarem la sentència:

```
DROP INDEX nom_index ON nom_taula;
```

Exemple:

a) Esborra l'índex que has creat en l'apartat anterior (apartat e).
b) Intenta esborrar els índex que s'han creat automàticament pel programa

La solució és:

```
a) DROP INDEX xdepartament ON materia;
b) DROP INDEX materia ON materia;
DROP INDEX primary ON materia;
```

(no es pot esborrar aquest índex perquè és una clau principal)

Llenguatge de manipulació de dades SQL

Introduir dades en taules

Per introduir dades en taules fem servir la següent sentència:

```
INSERT INTO nom_taula [llistat_columnes] VALUES valors_de_dades;
```

Per introduir valors en format text i en format data els escriurem entre ' ' o " ". Per introduir valors en blanc escriurem ' ', " " o NULL.

Exemple: Tenim una taula amb tres camps (nom, cognoms, data):

- codi INT AUTO_INCREMENT
- nom VARCHAR NOT NULL
- cognoms VARCHAR NOT NULL
- data DATE

Exemple: Per introduir dades a la taula utilitzem:

```
INSERT INTO alumne VALUES (3,'Josep', 'Pla', '1953/12/07');
INSERT INTO alumne (nom, cognoms) VALUES ('Miquel','Garcia');
INSERT INTO alumne (nom, cognoms, data) VALUES ('Miquel','Garcia', NULL);
```

Si volem introduir dades massivament des d'un fitxer extern utilitzarem la sentència:

```
LOAD DATA LOCAL INFILE "fitxer" INTO TABLE nom_taula;
```

on el fitxer estarà situat a C:\mysql\data\(base_de_dades), i els camps estaran separats per tabuladors i els registres pel canvi de línia.

Exemple:
a) Creem una base de dades anomenada "empresa". Dins aquesta base de dades crear una taula anomenada "personal" amb els següents camps:

- **codi**: valor enter autonumèric, és la clau primària
- **nom**: cadena de caràcters amb longitud 20, no pot ser nul
- **cognoms**: cadena de caràcters amb longitud 40, no pot ser nul
- **data_naixement**
- **salari**: valor de quatre xifres enteres i dos decimals

S'ha de tenir en compte que els valors de nom-cognoms ha de ser únic i que el sou ha de ser un valor positiu.

b) Per introduir els valor fem servir:

```
INSERT INTO personal VALUES (null,'Josep','Font',null,1867.56);
INSERT INTO personal VALUES (null,'Jordi','Vila','1979/2/20',1243.06);
INSERT INTO personal VALUES (null,'Anna','Torner',null,1243.06);
INSERT INTO personal (nom,cognoms) VALUES ('Miquel','Ferrando');
```

c) Introduïu els següents valors i apunteu-vos els problemes amb els que us trobeu:

codi	nom	cognoms	data_naixement	salari
	Josep	Font		1867,56
	Jordi	Vila	20/2/79	1243,06
	Anna	Torner		1243,06
	Miquel	Ferrando		
	Carla		13/4/68	1765,00
	Gerard	Codina	27/5/74	1402,89

6	Mercè	Vila	27/6/78	1765,00
	Anna	Torner	1/2/73	
15	Marta	Casas		
	Joel	Codina	14/2/81	1402,89
	Marta	Pérez	20/2/92	0,00

Solució de de l'exemple:
a)

```
CREATE TABLE personal
   (codi             INTEGER(4)       PRIMARY KEY AUTO_INCREMENT,
   nom               VARCHAR(20)         NOT NULL,
   cognoms           VARCHAR(40)         NOT NULL,
   data_naixement    DATE,
   salari            NUMERIC(6,2),
   CONSTRAINT unic_nom_cog UNIQUE(nom,cognoms),
   CONSTRAINT sal_pos CHECK(salari>0)
   ) ENGINE=InnoDB
   ;
```

c)

```
INSERT INTO personal VALUES (null,'Josep','Font',null,1867.56);
INSERT INTO personal VALUES (null,'Jordi','Vila','1970/2/20',1243.06);
INSERT INTO personal VALUES (null,'Anna','Torner',null,1243.06);
INSERT INTO personal VALUES (null,'Miquel','Ferrando',null,null);
#Els cognoms no poden ser nuls
#INSERT INTO personal VALUES (null,'Carla',null,'1968/4/13',1765.00);
INSERT INTO personal VALUES (null,'Gerard','Codina','1974/5/27',1402.89);
INSERT INTO personal VALUES (6,'Mercè','Vila','1978/6/27',1765.00);
#Està duplicat el nom-cognom amb un altre registre
#INSERT INTO personal VALUES (null,'Anna','Torner','1973/2/1',null);
INSERT INTO personal VALUES (15,'Marta','Casas',null,null);
INSERT INTO personal VALUES (null,'Joel','Codina','1981/2/14',1402.89);
#El sou ha de ser positiu
#INSERT INTO personal VALUES (null,'Marta','Pérez','1992/2/20',0.00);
```

Esborrar dades de taules

Per eliminar dades en taules fem servir la següent sentència:

```
DELETE FROM nom_taula [WHERE condició] [LIMIT n];
```

on WHERE estableix la condició de recerca i LIMIT n estableix en n el nombre màxim de files per esborrar.

Exemple:

```
a) DELETE FROM empleat WHERE codi=1;
b) DELETE FROM empleat WHERE cognoms='Codina' LIMIT 2;
d) DELETE FROM empleat; (esborra tota la taula)
```

Actualitzar dades de taules

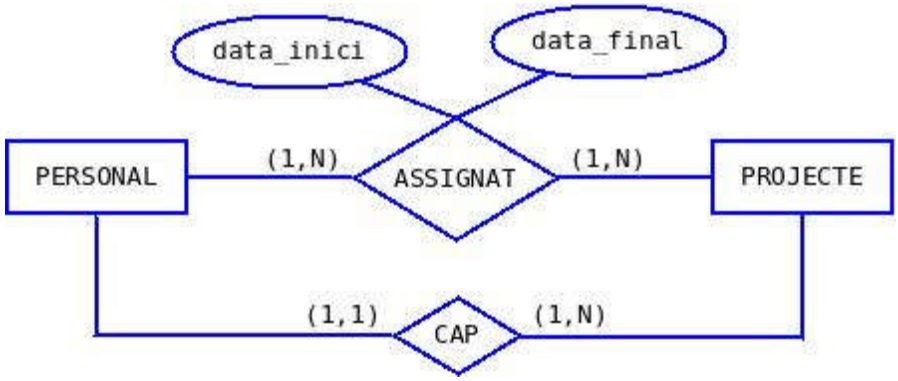
Per actualitzar dades en taules fem servir la següent sentència:

```
UPDATE nom_taula SET columna=expressió [,columna=expressió] [WHERE condició] [LIMIT n];
```

Exemple:
Actualitzar el sou als usuaris pujant el IPC de l'any passat que va estar del 3,2%.

```
UPDATE empleat SET salari=salari*1.032
```

Exemple:
a) Crear un nou camp anomenat sexe com un sol caràcter (H,D).
b) Actualitzar les dades dels usuaris.
c) Afegir un nou camp que posi el càrrec que ocupen, en Josep Font i Gerard Codina són "Caps de projecte" i la resta són "Programadors".
d) Tenim el disseny següent:



Afegir una nova taula anomenada projecte que contingui els projectes en els que treballen de manera que es pugui introduir el codi del projecte (integer), la descripció (fins a 40 caràcters) i el cap d'obra, que serà un dels treballadors, i una altra taula que contingui les dades de la relació.
e) Introduïu les següents dades:

Treballador	Cap de projecte	Descripció projecte	Data inici	Data final
Jordi Vila	Josep Font	Disseny de BD agència de viatges	12/3/03	23/12/04
Anna Torner	Josep Font	Disseny de BD agència de viatges	12/3/03	17/7/04
Miquel Ferrando	Josep Font	Disseny de BD agència de viatges	18/7/04	23/12/04
Mercè Codina	Gerard Codina	Disseny programa gestió hotel	7/1/04	13/9/04
Marta Casas	Gerard Codina	Disseny programa gestió hotel	7/1/04	
Joel Codina	Gerard Codina	Disseny programa gestió hotel	7/1/04	13/9/04
Anna Torner	Gerard Codina	Disseny programa gestió hotel	18/7/04	
Jordi Vila	Josep Font	Programa gestió botiga informàtica	7/1/05	
Miquel Ferrando	Josep Font	Programa gestió botiga informàtica	7/1/05	
Mercè Codina	Josep Font	Programa gestió botiga informàtica	14/9/04	
Joel Codina	Josep Font	Programa gestió botiga informàtica	14/9/04	

Solució de de l'exemple:

a)

```
ALTER TABLE personal ADD sexe ENUM('H','D');
```

b)

```
UPDATE personal SET sexe='H';
UPDATE personal SET sexe='D' WHERE codi IN (3,6,16);
```

c)

```
ALTER TABLE personal ADD carrec VARCHAR(20);
UPDATE personal SET carrec='Programador';
UPDATE personal SET carrec='Cap de projecte' WHERE codi IN (1,5);
```

d)

```
CREATE TABLE projecte
   (codi             SMALLINT         AUTO_INCREMENT PRIMARY KEY,
   descripcio        VARCHAR(30)      NOT NULL,
   cap               SMALLINT,
   CONSTRAINT fk_cap FOREIGN KEY (cap) REFERENCES personal(codi)
   );


CREATE TABLE assignat
   (codi_empleat        SMALLINT        NOT NULL,
   codi_projecte        SMALLINT        NOT NULL,
   data_inici           DATE            NOT NULL,
   data_final           DATE,
   PRIMARY KEY (codi_empleat,codi_projecte),
   CONSTRAINT fk_empleat FOREIGN KEY (codi_empleat) REFERENCES                personal(codi),
   CONSTRAINT fk_projecte FOREIGN KEY (codi_projecte) REFERENCES       projecte(codi)
   );


INSERT INTO projecte VALUES (null,'Disseny de BD agència de viatges',1);
INSERT INTO projecte VALUES (null,'Disseny programa gestió hotel',5);
INSERT INTO projecte VALUES (null,'Programa gestió botiga informàtica',1);


INSERT INTO assignat VALUES (2,1,'2003/3/12','2004/12/23');
INSERT INTO assignat VALUES (3,1,'2003/3/12','2004/7/17');
INSERT INTO assignat VALUES (4,1,'2004/7/18','2004/12/23');
INSERT INTO assignat VALUES (6,2,'2004/1/7','2004/9/13');
INSERT INTO assignat VALUES (15,2,'2004/1/7',null);
INSERT INTO assignat VALUES (16,2,'2004/1/7','2004/9/13');
INSERT INTO assignat VALUES (3,2,'2004/1/7',null);
INSERT INTO assignat VALUES (2,3,'2005/1/7',null);
INSERT INTO assignat VALUES (4,3,'2005/1/7',null);
INSERT INTO assignat VALUES (6,3,'2004/9/14',null);
INSERT INTO assignat VALUES (16,3,'2004/9/14',null);
```

Consultes

Per fer consultes utilitzem la sentència:

```
SELECT nom_columna_a_seleccionar [[AS]columna_reanomenada]
```

```
   [,nom_columna_a_seleccionar [[AS]columna_reanomenada]]
   FROM taula_a_consultar [[AS]taula_reanomenada]]
   WHERE condicions;
```

Es pot veure que la paraula AS és opcional.

Exemple:
Utilitzarem la base de dades PLANTES:
a) Per veure tot el contingut d'una taula farem:

```
SELECT *
   FROM planta;
```

b) Si volem veure només els camps nom_cientific i floracio escriurem:

```
SELECT nom_cientific, floracio
   FROM planta;
```

c) Si volem canviar el nom als camps i que es mostrin com a nom i estacio_de_floracio escriurem:

```
SELECT nom_cientific nom, floracio estacio_de_floracio
   FROM planta;
```

d) Si volem escriure una condició posarem, per exemple les plantes que la seva estació de floració és Hivern:

```
SELECT nom_cientific nom, floracio estacio_de_floracio
   FROM planta
   WHERE floracio='Hivern';
```

Exemple:
a) Mostra les firmes comercials que existeixen.
b) Mostra el nom científic de les plantes que necessiten llum directa.
c) Mostra el nom científic de les plantes i les quantitats d'adobs de les que el reben major o igual a 50.
d) Mostra el nom científic de les plantes que el seu mètode de reproducció és Esqueix amb un grau d'èxit alt.
e) Mostra el nom científic de les plantes que tenen 4 o més exemplars.

Solució a l'exemple:
a)

```
SELECT nom_firma
   FROM firma_comercial;
```

b)

```
SELECT nom_planta
   FROM planta_interior
   WHERE ubicacio='Llum directa';
```

c)

```
SELECT nom_planta, quantitat_adob
   FROM dosi_adob
   WHERE quantitat_adob>=50;
```

d)

```
SELECT nom_planta
   FROM reproduccio
   WHERE metode_reproduccio='Esqueix' AND grau_exit='Alt';
```

e)

```
SELECT nom_planta
   FROM exemplar_planta
   WHERE num_exemplar=4;
```

Operadors

Per definir el WHERE podem utilitzar diferents operadors:

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

	Operadors lògics
AND	Per a la conjunció de condicions (A i B)
OR	Per a la disjunció de condicions (A o B)
NOT	Per a la negació de condició (no A)

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

Exemple:
a) Mostra les marques comercials que tenen algun producte.
b) Mostra el nom de les plantes que durant la primavera se les subministra adob.
c) Mostra el nom de les plantes que el seu mètode de reproducció és Esqueix amb un grau d'èxit alt.

Solució a l'exemple:
a)

```
SELECT DISTINCT nom_firma FROM adob;
```

b)

```
SELECT DISTINCT nom_planta
   FROM dosi_adob
   WHERE nom_estacio='Primavera';
```

c)

```
SELECT nom_planta
   FROM reproduccio
   WHERE metode_reproduccio='Esqueix' AND grau_exit='Alt';
```

Altres predicats

- BETWEEN

Per a expressar una condició que vol trobar un valor entre uns límits concrets podem fer servir BETWEEN. El format és:

```
SELECT nom_columnes_a_seleccionar
   FROM taula_a_consultar
   WHERE columna BETWEEN limit1 AND limit2;
```

Exemple:
Mostra les plantes que a la primavera necessiten entre 40 i 50 unitats d'adob (inclosos).

```
SELECT nom_adob
   FROM dosi_adob
   WHERE quantitat_adob BETWEEN 40 AND 50;
```

- [NOT] IN

Per a veure si un valor coincideix amb els elements d'una llista utilitzarem IN, i per a veure si no coincideix, NOT IN. El format és:

```
SELECT nom_columnes_a_seleccionar
   FROM taula_a_consultar
   WHERE columna [NOT] IN (valor1, ..., valorN);
```

Exemple:
Mostra el nom dels adobs de la firma TIRSADOB i PRISADOB.

```
SELECT nom_adob
   FROM adob
   WHERE nom_firma IN ('TIRSADOB','PRISADOB');
```

- LIKE

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

a) Posarem un caràcter _ per a cada caràcter individual que vulguem considerar.
b) Posarem un caràcter % per a expressar una seqüència de caràcters, que pot ser cap.

El format és:

```
SELECT nom_columnes_a_seleccionar
   FROM taula_a_consultar
   WHERE columna LIKE caracteristica;
```

Exemple:
a) Mostra el nom popular de les plantes que el seu nom comença amb C.

```
SELECT nom_popular
   FROM planta
   WHERE nom_popular LIKE 'C%';
```

b) Mostra el nom popular de les plantes que el seu nom comença amb C i tenen sis caràcters

```
SELECT nom_popular
   FROM planta
   WHERE nom_popular LIKE 'C_____';
```

- IS [NOT] NULL

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

```
SELECT nom_columnes_a_seleccionar
   FROM taula_a_consultar
   WHERE columna IS [NOT] NULL;
```

Exemple:
Mostra el nom popular i científic de les plantes que sabem l'estació de floració.

```
SELECT nom_popular, nom_cientific
   FROM planta
   WHERE floracio IS NOT NULL;
```

Funcions d'agregació

SQL ens ofereix les següents funcions d'agregació:

- COUNT: Ens dóna el número total de files seleccionades.

Exemple:
Mostra el nombre de plantes que tenen la seva floració a primavera.

```
SELECT COUNT(*) AS floracio_primavera
   FROM planta
   WHERE floracio='Primavera';
```

- SUM: Suma els valors d'una columna.

Exemple:
Mostra la quantitat total d'adob 'Casadob' que necessiten a la primavera les plantes .

```
SELECT SUM(quantitat_adob) AS total_adob
   FROM dosi_adob
   WHERE nom_estacio='Primavera' AND nom_adob='Casadob';
```

- MIN: Ens dóna el valor mínim d'una columna. MAX:
- Ens dóna el valor màxim d'una columna.

Exemple:
Mostra la temperatura mínima i màxima que necessiten les plantes d'interior.

```
SELECT MIN(temperatura) AS temperatura_minima,
   MAX(temperatura) AS temperatura_maxima
   FROM planta_interior;
```

- AVG: Calcula el valor mitjà d'una columna.

Exemple:
Mostra la mitjana d'adob que necessiten les plantes al hivern.

```
SELECT AVG(quantitat_adob) AS mitjana_adob
   FROM dosi_adob
   WHERE nom_estacio='Hivern';
```

Funcions i operadors

- **Funcions de caràcters**:

CONCAT(str1,str2,...): Retorna la cadena resultat de concatenar els arguments.

```
SELECT CONCAT('a',' ','b');
 -> a b
```

LEFT(str,len): Retorna los *len* caràcters començant per l'esquerra de la cadena *str*.

```
SELECT LEFT('autopista',5);
 -> autop
```

LENGTH(str): Retorna la longitud de la cadena *str*, mesurada en bytes.

```
SELECT LENGTH('autopista');
 -> 9
```

LOWER(str): Retorna la cadena *str* amb tots els caràcters canviats a minúscules.

```
SELECT LOWER('Casual');
 -> casual
```

RIGHT(str,len): Retorna los *len* caràcters començant per la dreta de la cadena *str*.

```
SELECT RIGHT('autopista',5);
 -> pista
```

UPPER(str): Retorna la cadena *str* amb tots els caràcters canviats a majúscules.

```
SELECT UPPER('Casual');
 -> CASUAL
```

- **Funcions numèriques**:

Operadors aritmètics:
+: Suma.

```
SELECT 2+5;
 -> 7
```

-: Resta.

```
SELECT 2-5;
 -> -3
```

*: Producte.

```
SELECT 2*5;
 -> 10
```

/: Divisió.

```
SELECT 10/5;
 -> 2
```

DIV: Divisió entera.

```
SELECT 12 DIV 5;
 -> 2
```

Funcions matemàtiques:
ABS(X): Retorna el valor absolut de *X*.

```
SELECT ABS(-2.5);
 -> 2.5
```

CEIL(X): Retorna l'enter més gran a *X*.

```
SELECT CEIL(-2.5);
 -> -2
```

FLOOR(X): Retorna l'enter més petit a *X*.

```
SELECT FLOOR(3.8);
 -> 4
```

MOD(N,M): Operació de mòdul. Retorna el residu de *N* dividit per *M*.

```
SELECT MOD(8,3);
 -> 2
```

POW(X,Y): Retorna el valor de *X* a la potència de *Y*.

```
SELECT POW(4,3);
 -> 64
```

ROUND(X,D): Retorna el argumento *X*, arrodonit *D* decimals.

```
SELECT ROUND(5/3,3);
 -> 1.667
```

SQRT(X): Retorna la arrel quadrada d'un nombre no negativo *X*.

```
SELECT SQRT(64);
 -> 8
```

- **Funcions de data i hora**:

CURDATE(): Retorna la data actual amb valor en format 'YYYY-MM-DD'.

```
SELECT CURDATE();
 -> 2012-01-31
```

CURTIME(): Retorna la hora actual amb valor en format 'HH:MM:SS'.

```
SELECT CURTIME();
 -> 20:01:38
```

DATE(expr): Extreu la part de data de l'expressió de data o data i hora *expr*.

```
SELECT DATE('2012-01-31 20:01:38');
 -> 2012-01-31
```

DAYOFMONTH(date): Retorna el dia del mes per *date*, en el rang 1 a 31.

```
SELECT DAYOFMONTH('2012-01-31 20:01:38');
 -> 31
```

DAYOFWEEK(date): Retorna l'índex del dia de la setmana per *date* (1 = diumenge, 2 = dilluns, ..., 7 = dissabte).

```
SELECT DAYOFWEEK('2012-01-31 20:01:38');
 -> 3
```

DAYOFYEAR(date): Retorna el dia de l'any per *date*, en el rang 1 a 366.

```
SELECT DAYOFYEAR('2012-01-31 20:01:38');
 -> 31
```

LAST_DAY(date): Pren una data o data/hora i retorna el valor corresponent per l'últim dia del mes.

```
SELECT LAST_DAY('2012-01-25');
 -> 2012-01-31
```

WEEKDAY(date): Retorna l'índex de dies de la setmana per *date* (0 = dilluns, 1 = dimarts, ... 6 = diumenge).

```
SELECT WEEKDAY('2012-01-31 20:01:38');
 -> 1
```

Subconsulta

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

Exemple:
a) Mostra el nom científic de les plantes d'interior que han d'estar a una temperatura superior a la mitjana de temperatura.

```
SELECT nom_planta,temperatura
 FROM planta_interior
 WHERE temperatura>(SELECT AVG(temperatura) FROM planta_interior);
```

b) Mostra el nom científic, el nom de l'adob, l'estació i la quantitat d'adob de les plantes que necessiten una dosi d'adob mínima.

```
SELECT nom_planta,nom_adob,nom_estacio,quantitat_adob
 FROM dosi_adob
 WHERE quantitat_adob = (SELECT MIN(quantitat_adob) FROM dosi_adob);
```

c) Mostra el nom popular, el nom de l'adob, l'estació i la quantitat d'adob de les plantes que necessiten una dosi d'adob entre la mitjana menys 10 i la mitjana més 10.

```
SELECT nom_popular,nom_adob,nom_estacio,quantitat_adob
 FROM planta,dosi_adob
 WHERE nom_cientific=nom_planta
     AND quantitat_adob > (SELECT AVG(quantitat_adob) FROM dosi_adob)-10
     AND quantitat_adob < (SELECT AVG(quantitat_adob) FROM dosi_adob)+10;
```

Ordenació de les dades

Per ordenar dades cal utilitzar la clàusula ORDER BY en la sentència SELECT, que té el format següent: Si no s'especifica res més, l'ordre que se seguirà és ascendent (si no es pot utilitzar ASC), però si es desitja seguir un ordre descendent cal afegir DESC.

```
SELECT nom_columnes_a_seleccionar
   FROM taula_a_consultar
   [WHERE condicions]
   ORDER BY columna_ordenacio [DESC] [, col_ordenacio [DESC]...];
```

Exemple:
a) Mostra el nom popular de les plantes que floreixen a primavera ordenades en ordre alfabètic.

```
SELECT nom_popular
   FROM planta
   WHERE floracio ='Primavera'
   ORDER BY nom_popular;
```

b) Mostra el nom popular de les plantes que floreixen a primavera ordenades en ordre alfabètic invers.

```
SELECT nom_popular
   FROM planta
   WHERE floracio <>'Primavera'
   ORDER BY nom_popular DESC;
```

Consultes amb agrupació de files d'una taula

Les clàusules següents permeten organitzar les files per grups:

- La clàusula GROUP BY ens serveix per a agrupar files segons les columnes que indiqui aquesta clàusula.
- La clàusula HAVING especifica condicions de cerca per a grups de files; porta a terme la mateixa funció que abans feia la clàusula WHERE per a les files de tota la taula, però ara les condicions s'apliquen als grups obtinguts.

Presenta el format següent:

```
SELECT nom_columnes_a_seleccionar
   FROM taula_a_consultar
   [WHERE condicions]
   GROUP BY columnes_per_agrupar
   [HAVING condicions_per_grups]
   [ORDER BY columna_ordenacio [DESC][, columna [DESC]...]];
```

Exemple:
Volem saber les plantes que utilitzen adob Casadob i Plantavit, i que la quantitat d'aquest que reben és superior a 90.
Seguim el següent procediment:
a) Busquem les plantes que utilitzen adobs Casadob i Plantavit:

```
SELECT nom_planta, quantitat_adob
   FROM dosi_adob
   WHERE nom_adob='Casadob' OR nom_adob='Plantavit';
```

b) Agrupem aquestes plantes mostrant la quantitat total utilitzada:

```
SELECT nom_planta, SUM(quantitat_adob)
   FROM dosi_adob
   WHERE nom_adob='Casadob' OR nom_adob='Plantavit'
   GROUP BY nom_planta;
```

c) Fem que la quantitat total d'adob sigui més gran que 90.

```
SELECT nom_planta, SUM(quantitat_adob)
   FROM dosi_adob
   WHERE nom_adob='Casadob' OR nom_adob='Plantavit'
   GROUP BY nom_planta
   HAVING SUM(quantitat_adob)>90;
```

La unió

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

```
SELECT columnes
   FROM taula
   [WHERE condicions]
UNION [ALL]
   SELECT columnes
   FROM taula
   [WHERE condicions];
```

Si posem l'opció ALL sortiran totes les files obtingudes de fer la unió. No la posarem si volem eliminar les files repetides.

La intersecció

Per a fer la intersecció entre dues o més sentències SELECT FROM podem utilitzar la clàusula INTERSECT, el format de la qual és: SELECT columnes

```
   FROM taula
   [WHERE condicions]
INTERSECT [ALL]
SELECT columnes
   FROM taula
   [WHERE condicions]
```

Si posem l'opció ALL sortiran totes les files obtingudes de fer la intersecció. No la posarem si volem eliminar les files repetides.

La diferència

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

```
SELECT columnes
   FROM taula
   [WHERE condicions]
```

```
EXCEPT [ALL]
SELECT columnes
   FROM taula
   [WHERE condicions];
```

Si posem l'opció ALL sortiran totes les files que dóna la diferència. No la posarem si volem eliminar les files repetides.

Consultes a més d'una taula

Fins ara s'havia fet que posàvem com una condició de WHERE la igualtat entre dues claus.

Exemple:
Creem una base de dades anomenada EMPRESA que tindrà dos taules:
a) La primera tindrà dos camps: el codi de treballador i el seu nom. En ella posarem els treballadors que estan contractats per l'empresa.

TREBALL	
1	Josep Font
2	Joan Cambra
3	Marta Serra

b) La segona tindrà dos camps: el nom del treballador i el seu cost. En ell posarem els comercials (siguin treballadors de l'empresa o autònoms) que cobraran alguna comissió.

COMISSIO	
Josep Font	200
Joan Cambra	300
Montse Rubió	500
Pere Forner	400

Solució de l'exemple:

```
#Creem la base de dades
CREATE DATABASE empresa;
#Utilitzem la base de dades empresa
USE empresa;
#Creem la taula treballador
CREATE TABLE treballador
   (codi     INT         PRIMARY KEY,
   nom        VARCHAR(20)
   );
#Creem la taula comissio
CREATE TABLE comissio
   (nom      VARCHAR(20)    PRIMARY KEY,
   cost      INT(4)
   );
#Introduim els valors a la taula treballador
INSERT INTO treballador VALUES (1,'Josep Font');
INSERT INTO treballador VALUES (2,'Joan Cambra');
INSERT INTO treballador VALUES (3,'Marta Serra');
#Introduim els valors a la taula comissio
INSERT INTO comissio VALUES ('Josep Font',200);
INSERT INTO comissio VALUES ('Joan Cambra',300);
INSERT INTO comissio VALUES ('Montse Rubió',500);
INSERT INTO comissio VALUES ('Pere Forner',400);
```

Per trobar els treballadors de l'empresa que cobren comissió fem:

```
SELECT codi, tr.nom, cost
   FROM treballador tr, comissio co
   WHERE tr.nom=co.nom;
```

Però com es pot fer quan es volen mostrar els treballador i no treballadors amb la seva comissió?

Ara utilitzarem la ordre JOIN que té la següent estructura:

```
SELECT nom_columnes_a_seleccionar
   FROM taula1 JOIN taula2 [ON condicions]
   [WHERE condicions];
```

Exemple:
Volem seleccionar els treballador de l'empresa amb la comissió que cobren i el seu codi d'empresa:

```
SELECT codi, tr.nom, cost
   FROM treballador tr JOIN comissio co ON tr.nom=co.nom;
```

La combinació natural el que fa és combinar columnes amb el mateix nom:

```
SELECT nom_columnes_a_seleccionar
   FROM taula1 NATURAL JOIN taula2
   [WHERE condicions];
```

Exemple:
Volem seleccionar tots els treballadors que cobren comissió:

```
SELECT codi, tr.nom, cost
   FROM treballador tr NATURAL JOIN comissio;
```

Qualsevol combinació pot ser interna o externa:
a) Combinació interna:
Es queda amb les files que tenen valors idèntics a les dues, això fa que es pugui perdre alguna que tingui valors NULL. Es la que s'executa per defecte. Millor no utilitzar-lo.
Seria el cas vist anteriorment: *Exemple*:
Com l'exemple anterior:

```
SELECT codi, tr.nom, cost
   FROM treballador tr INNER JOIN comissio co ON tr.nom=co.nom;
```

b) Combinació externa:
Es queda amb les files de les dues taules, això fa que agafi tots els valors. La seva estructura seria:

```
SELECT nom_columnes_a_seleccionar
   FROM taula1 [NATURAL] [LEFT|RIGHT] JOIN taula2 [ON condicions]
   [WHERE condicions];
```

Exemple:

Volem veure tots els treballadors i les comissions que aquests han cobrat.

```
SELECT codi, tr.nom, cost
   FROM treballador tr LEFT JOIN comissio co ON tr.nom=co.nom;
```

També podríem haver fet:

```
SELECT codi, tr.nom, cost
    FROM treballador tr NATURAL LEFT JOIN comissio co;
```

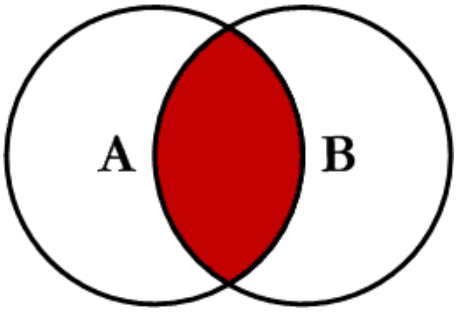
Si volem mostrar el codi de treballador de les persones que han cobrat alguna comissió, aleshores tenim:

```
SELECT codi, co.nom, cost
   FROM treballador tr NATURAL RIGHT JOIN comissio co;
```

INNER JOIN

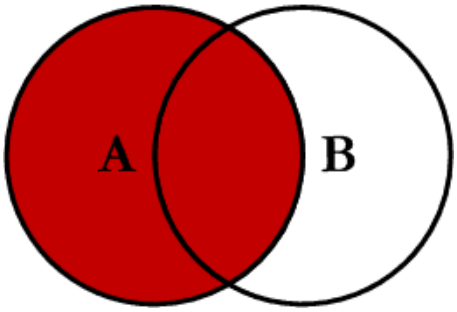
El més usual, el primer que se sol aprendre, és l'ús de INNER JOIN, o generalment abreujat com JOIN.

Aquesta clàusula cerca coincidències entre 2 taules, en funció d'una columna que tenen en comú. De manera que només la intersecció es mostrarà en els resultats .



LEFT JOIN

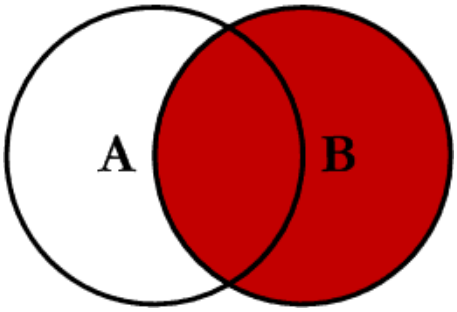
A diferència d'un INNER JOIN, on es busca una intersecció respectada per ambdues taules, amb LEFT JOIN, prioritzem la taula de l'esquerra, i busquem a la taula dreta.



Si no hi ha cap coincidència per a alguna de les files de la taula de l'esquerra, es mostren tots els resultats de la primera taula . Per tant, si es troben coincidències, es mostraran els valors corresponents, però si no, apareixerà NULL als resultats.

RIGHT JOIN

En el cas de RIGHT JOIN la situació és molt semblant, però aquí es dóna prioritat a la taula de la dreta.



De manera que si usem aquesta consulta, estarem mostrant totes les files de la taula de la dreta i si es troben coincidències, es mostraran els valors corresponents, però si no, apareixerà NULL als resultats.

FULL JOIN

Mentre que LEFT JOIN mostra totes les files de la taula esquerra, i RIGHT JOIN mostra totes les corresponents a la taula dreta, FULL OUTER JOIN (o simplement FULL JOIN) s'encarrega de mostrar totes les files de les dues taules, sense importar que no hi hagi coincidències (usarà NULL com un valor per defecte per casos de no coincidència).