

M02. Gestió de Bases de Dades

UF2. Llenguatge SQL : DDL i DML

M10. Administració de Bases de Dades

UF1. Llenguatge SQL: DCL i extensió procedimental

Autor: Robert Ventura Vall-Ilovera

Data creació: Setembre 2015

Data actualització: Març 2025



Aquest document està subjecte a una llicència de [Reconeixement-
No comercial-Compartir Igual 3.0 Espanya de Creative Commons](https://creativecommons.org/licenses/by-nc-sa/3.0/es/)

A decorative graphic consisting of a 3x2 grid of squares. The top-left square is light blue. The middle-left square is a darker blue. The middle-right square is light blue. The bottom-left square is light blue. The bottom-right square is a darker blue. The bottom-most square is light blue.

Llenguatge SQL

Estudi del llenguatge ANSI SQL. Utilització del llenguatge SQL en el SGBDR MySQL.



Contingut

Contingut ii

Capítol 1. Introducció a SQL	5
1.1. Història d'SQL	5
1.1.1. Introducció	5
1.1.2. Versions	5
1.2. Modes d'utilització	6
1.3. Elements del llenguatge SQL	7
Capítol 2. DDL SQL	9
2.1. Introducció a les sentències de definició	9
2.1.1. Introducció	9
2.1.2. Creació i esborrat d'una Base de Dades Relacional	9
2.1.3. Objectes de BD	12
2.2. Definició del nivell lògic: taules i restriccions	13
2.2.1. Creació de taules	13
Tipus de dades del MySQL	14
Tipus de dades Spatial (Extensió per dades espacial)	18
Tipus de dades JSON	19
Regles de nomenclatura en la creació de taules	19
2.2.2. Creació de restriccions	21
2.2.3. Modificació en una taula	29
2.2.4. Eliminar l'estructura de taules	32
2.2.5. Esborrar el contingut d'una taula	33
2.2.6. Afegir i treure restriccions	34
2.3. Definició del nivell extern: Vistes	37
2.3.1. Introducció	37
2.3.2. Creació de vistes	38
2.3.3. Modificació i esborrat de vistes	39
2.3.4. Utilització de vistes actualitzables	40
2.4. Índexs	41
2.4.1. Introducció	41
2.4.2. Creació d'índexs	42
Veure els índexs que tenim	45
Índexs invisibles	47
Capítol 3. DML – SQL	48
3.1. Inserció, modificació i eliminació	48
3.1.1. Introducció	48
3.1.2. Inserció de dades	48
3.1.3. Modificació de dades	52
3.1.4. Eliminació de dades	54
3.2. Consulta de dades sobre una taula	56
3.2.1. Sintaxi	56
Exemple 1	56
Exemple 2	57
Exemple 3	57
Exemple 4	58
3.2.2. Ús d'ÀLIES	58
Sintaxi	58
3.2.3. Càlculs	59
Aritmètics	59
Concatenació	61
3.2.4. Condicions de selecció	62
Operadors de comparació	63
Operador BETWEEN	63
Operador IN	65
Operador IS NULL	66

Operador LIKE	67
Funció REGEXP.....	69
Operadors lògics	71
Precedència d'operadors	72
3.2.5. Ordenació.....	73
3.2.6. Funcions.....	75
Funcions de caràcters.....	75
Funcions numèriques.....	79
Funcions per treballar amb dates.....	80
Funcions de control de flux i treballar valors NULL	85
Funcions de conversió	87
3.2.7. Agrupacions i Agregats	88
Funcions de càlcul amb grups (funcions de totals o d'agregats)	91
HAVING.....	92
3.3. Consulta de dades sobre vàries taules.....	95
3.3.1. Introducció.....	95
3.3.2. Producte cartesià de taules.....	95
3.3.3. Combinació de taules.....	97
3.3.4. Combinacions creuades.....	98
3.3.5. Combinacions internes.....	99
3.3.6. Combinacions externes.....	102
3.3.7. Combinacions especials	106
3.3.8. Operadors de conjunts.....	108
3.4. Subconsultes	113
3.4.1. Subconsulta escalar	115
3.4.2. Subconsulta de llista	119
IN	119
ANY	122
ALL	124
EXISTS	126
3.4.3. Subconsulta multi-columna	128
3.4.4. Subconsultes correlacionades	129
3.4.5. Taules derivades	130
3.4.6. Resum d'ús de subconsultes	130
3.5. DDL i DML ampliat.....	131
3.5.1. Sentència CREATE TABLE	131
3.5.2. Sentència INSERT i SELECT	131
3.5.3. Sentències UPDATE i DELETE amb subconsultes.....	132
3.5.4. Sentències UPDATE i DELETE amb relacions.	133
3.7. CTE (Common Table Expressions)	135
3.7.1. WITH QUERIES.....	135
3.7.2. WITH QUERIES - Formes recursives	136
3.8. Window Functions.....	143
3.8.1. Introducció.....	143
3.8.2. Llista de funcions de finestra RANKING	147
3.8.3. Llista de funcions de finestra VALUE.....	154
3.8.4. Named Windows	159
Capítol 4. Base de dades de tipus espacial.....	160
4.1. Introducció	160
4.2. MySQL GIS.....	163
4.2.1. Tipus de dades espacials	163
Point Class.....	164
LineString Class.....	164
Polygon Class	165
4.2.2. Funcions de tipus espacial.....	166
Funcions de relació.....	167
Funcions de Contingut / MBR (Minimum Bounding Rectangles)	169

Altres funcions spaials útils	170
Capítol 5. JSON	171
5.1. Introducció	171
5.2. MySQL JSON	171
5.2.1. Tipus de dades en format JSON	171
5.2.2. JSON Path Syntax	171
5.2.3. Funcions JSON	171
Capítol 6. Transaccions i concurrència	174
6.1. Gestió de transaccions	174
6.1.1. Introducció	174
6.1.2. Transaccions	174
6.1.3. Control de transaccions en MySQL	175
COMMIT	176
ROLLBACK	177
6.2. Gestió de la concurrència	181
6.2.1. Bloquejos	181
Bloqueig a nivell de taula	181
Bloqueig a nivell de registre	182
6.2.2. Consistència de lectura	183
Nivell d'aïllament i accés concurrent de les dades	184
REPEATABLE READ (lectures repetibles)	188

Capítol 1. Introducció a SQL

1.1. Història d'SQL

1.1.1. Introducció

L'any 1970, Codd va introduir el concepte de Base de Dades Relacional, però va portar un temps el disseny d'un sistema real de gestió de bases de dades relacional (SGBDR o RDMBS). El primer i que es recorda més és *System R* (dissenyat per IBM), el qual no era més que un prototipus, l'objectiu principal del qual era demostrar que amb aquest sistema era possible construir un Sistema Relacional, que es podia utilitzar en un entorn real, per solucionar problemes reals. No obstant, va ser Oracle qui el va introduir per primera vegada en un programa comercial (1979).

Aquest sistema incorporava un llenguatge de dades que permetia fer qualsevol accés a la base de dades, que es va anomenar *SEQUEL* (**Structured English Query Language**). Posteriorment altres sistemes van adoptar aquest subllenguatge, que es va passar a anomenar SQL., com a subllenguatge de consulta de dades, que fins avui dia ha patit diverses modificacions.

La seva utilització dins l'àmbit comercial és realment extensa. Les cases comercials l'inclouen dins els seus productes i és l'estàndard més important en aquest camp.

SQL és una abreviació de "**Structured Query Language**" (llenguatge de consulta estructurat). Amb aquest llenguatge es poden crear operacions relacionals, és a dir, operacions que permeten definir i manipular una base de dades relacional.

Encara que tots els sistemes SGBDs relacionals compleixen amb l'estàndard SQL en major o menor grau, també proveeixen d'extensions al llenguatge SQL per tal d'ampliar-hi les seves funcionalitats. Això s'ha de tenir en compte en el moment d'utilitzar-lo ja que pot produir problemes de portabilitat.

1.1.2. Versions

Nosaltres estudiarem SQL-92. Per tant, quan parlem del llenguatge SQL, ens estarem referint a la versió SQL-92. En els casos en que es parli de característiques pròpies de versions posteriors, ja s'indicarà.

ANSI SQL ha patit varies revisions al llarg del temps:

Any	Nom	Àlies	Comentaris
1986	SQL-86	SQL-87	Primera publicació feta per ANSI. Confirmada per ISO l'any 1987.
1989	SQL-89	SQL1	Revisió i ampliació menor.
1992	SQL-92	SQL2	ANSI publica una àmplia revisió. És la versió més estesa en els SGBDR comercials.
1999	SQL:1999	SQL3	ANSI publica SQL3 o ANSI/ISO SQL: 1999 Es van afegir expressions regulars, consultes recursives (per relacions jeràrquiques), triggers i algunes característiques orientades a objectes.

2003	SQL:2003		Introdueix algunes característiques de XML, canvis en les funcions, estandardització de l'objecte SEQUENCE i de les columnes auto numèriques.
2006	SQL:2006		Defineix la forma amb la que SQL es pot utilitzar conjuntament amb XML. Defineix com importar i guardar dades XML a una base de dades SQL, manipulant-les dins de la base de dades i publicant el XML i les dades SQL convencionals en forma de XML.

1.2. Modes d'utilització

El llenguatge SQL pot utilitzar-se de diferents formes:

- Execució directa (també anomenat SQL interactiu)
 - En aquest cas, les sentències SQL s'introdueixen mitjançant una eina que les tradueix immediatament a la base de dades, i per tant, s'executen a l'instant.
- Execució incrustada o submergida
 - Les sentències SQL es col·loquen com a part del codi d'un altre llenguatge (C, Java, Pascal, etc.). Aquestes comandes estan separades de la resta del codi d'una certa forma. Quan es compila el codi s'utilitza un precompilador de la pròpia base de dades que separa les instruccions del llenguatge de programació de les del llenguatge SQL.

1.3. Elements del llenguatge SQL

A part d'un llenguatge de consulta, en realitat el llenguatge SQL és molt més que això, ja que té altres funcions a més a més de les de consultar una base de dades. Entre aquestes hi ha:

- **DDL.** Llenguatge de definició de dades. Ens serveix per crear i administrar les base de dades i els seus objectes (taules, vistes, disparadors, índexs, regles, procediments,...)
- **DML.** Llenguatge de manipulació de les dades. Tal com diu el seu nom s'utilitza per la manipulació de la informació que tenim guardada. Obtenir, afegir, modificar i eliminar informació de la base de dades. Les comandes bàsiques són: SELECT, INSERT, UPDATE, DELETE
- **DCL.** Llenguatge de control de dades. Permet indicar restriccions d'accés i seguretat. S'utilitza per assignar permisos sobre els objectes de la base de dades. D'aquesta manera podem establir permisos als objectes respecte els diferents usuaris de la BD..

A continuació veurem per sobre com fer aquestes tasques, amb varis **Exemples**:

Crear una taula (comanda *DDL*) que contingui les dades dels productes de l'empresa:

```
CREATE TABLE producte
(
    codi          INTEGER,
    nom           CHAR(20),
    tipus         CHAR(20),
    descripcio    CHAR(50),
    preu          REAL,
    PRIMARY KEY (codi)
);
```

← *Nom de la taula*

} *Nom de les columnes i tipus (domini)*

← *Clau primària*

Inserir (comanda *DML*) un producte nou a la taula:

Nom de la taula

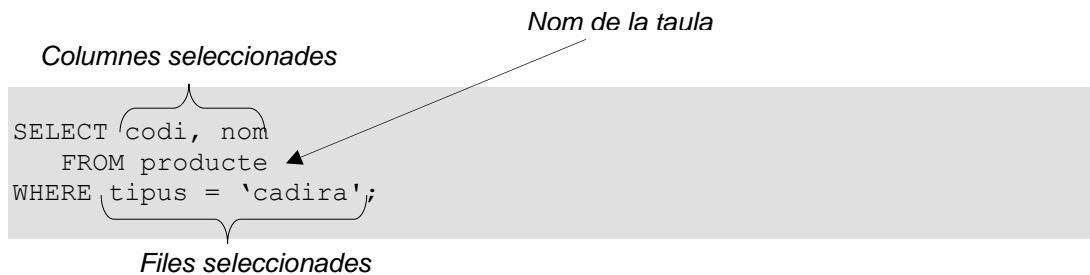
▼

```
INSERT INTO producte
VALUES (1250, 'LENA', 'taula', 'Disseny Joan Pi. Any 1920.', 2500);
```

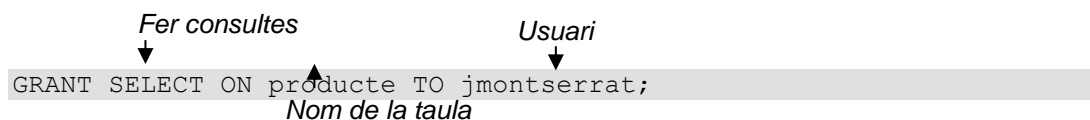
└────────────────────────────────────────────────────────────────────────────────┘

Valors de la fila

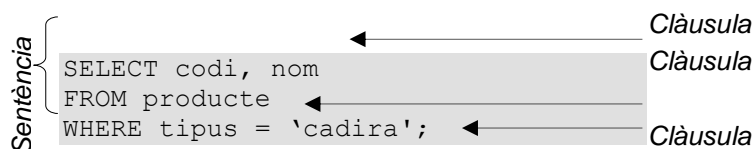
Consultar (comanda *DML*) quins productes són cadires:



Deixar accedir (comanda *DCL*) a un dels venedors de l'empresa a la informació de la taula `producte`:



Les operacions a SQL reben el nom de sentències i estan formades per diferents parts anomenades clàusules.



A més de sentències i clàusules, el llenguatge SQL té els següents elements:

- Operadors: permeten crear expressions complexes.
 - Ex: op. aritmètics → `+`, `-`, `*`, `/`, etc.
 - Ex: op. lògics → `>`, `<`, `>=`, `<=`, `<>`, `=`, `AND`, `OR`, `NOT`, etc.
- Funcions: permeten obtenir valors complexes
 - Ex: `SUM()`, `AVG()`, `COUNT()`, etc.
- Constants: valors constants per les consultes, números, textos, caràcters, etc.
- Dades: obtingudes de la pròpia base de dades.

Quan escrivim sentències del llenguatge SQL, cal tenir present que hi ha unes normes d'escriptura:

- No hi ha distinció entre majúscules i minúscules.
- Una sentència sempre acaba amb el símbol `;`.
- Les sentències SQL es poden escriure en diverses línies per facilitar-ne la lectura (molt recomanable!!)
- Per incloure comentaris en el codi SQL, aquests es posen entre `/*` i `*/`.

Capítol 2. DDL SQL

2.1. Introducció a les sentències de definició

2.1.1. Introducció

Per poder treballar amb BD relacionals, el primer que cal fer és definir-les. Veurem les instruccions de l'estàndard SQL92 per crear i esborrar una base de dades relacional i per inserir, esborrar i modificar les diferents taules que la componen.

En aquest apartat també veurem com es defineixen els dominis, les restriccions i les vistes.

La senzillesa del SQL92 fa que:

- Per crear bases de dades, taules, dominis, restriccions, índexs i vistes s'utilitza la sentència `CREATE`.
- Per modificar taules i dominis s'utilitzi la sentència `ALTER`.
- Per esborrar bases de dades, taules, dominis, restriccions, índexs i vistes s'utilitza la sentència `DROP`.

Cal tenir molta cura en utilitzar una sentència DDL, ja que aquestes no poden ser anul·lades (a diferència de les sentències d'actualització de dades).

2.1.2. Creació i esborrat d'una Base de Dades Relacional

L'estàndard SQL92 no té cap sentència de creació de BD. La idea és que una BD no és més que un conjunt de taules i, per tant, les sentències que té SQL92 es centren en la creació, modificació i esborrat d'aquestes taules.

Molts dels SGBDR han incorporat sentències de creació de BD.

Amb MySQL, segueix la següent sintaxi:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] nomBD  
[especificacio_create [,especificacio_create]...]  
[DEFAULT] CHARACTER SET joc_caràcters  
[DEFAULT] COLLATE nom_col·lecció;
```

On:

- `CHARACTER SET`: és el joc de caràcters que s'utilitzarà a la BD.
- `COLLATE`: és el nom de la col·lecció, per indicar quin és l'ordre lexicogràfic a seguir.

Més info `CHARACTER SET` i `COLLATE`:

- [Manual de referència 8.0](#)
- [Hipalinux](#)

Per comprovar quin és el CHARACTER SET i el COLLATE que té el sistema per defecte podem comprovar les variables @@character_set_database i @@collation_database;

```
SELECT @@character_set_database, @@collation_database;
```

```
+-----+-----+
| @@character_set_database | @@collation_database |
+-----+-----+
| utf8mb4                  | utf8mb4_0900_ai_ci   |
+-----+-----+
```

Per veure quins jocs de caràcters hi ha disponibles podem utilitzar **SHOW CHARACTER SET**

```
SHOW CHARACTER SET LIKE 'utf%';
```

```
+-----+-----+-----+-----+
| Charset | Description          | Default collation | Maxlen |
+-----+-----+-----+-----+
| utf16   | UTF-16 Unicode      | utf16_general_ci  | 4      |
| utf16le | UTF-16LE Unicode    | utf16le_general_ci | 4      |
| utf32   | UTF-32 Unicode      | utf32_general_ci  | 4      |
| utf8    | UTF-8 Unicode        | utf8_general_ci   | 3      |
| utf8mb4 | UTF-8 Unicode        | utf8mb4_0900_ai_ci | 4      |
+-----+-----+-----+-----+
```

Per poder veure la collation d'un joc de caràcters podem utilitzar **SHOW COLLATION WHERE = '<nom_joc_caràcters>';**

```
mysql> SHOW COLLATION WHERE Charset = 'utf8mb4';
```

En el cas de MySQL, només es poden crear BD si es tenen privilegis d'administrador (DBA).

Exemple:

```
CREATE DATABASE prova
  DEFAULT CHARACTER SET utf8
  DEFAULT COLLATE utf8_general_ci;
```

A partir de la versió 8.0 utf8 és un alias de utf8m3, però en un futur es preveu que sigui un alias de utf8mb4.

El collation 'utf8_general_ci' està obsolet per utf8mb3.

Convenció de noms per les Col·lecció dels Jocs de caràcters

<https://dev.mysql.com/doc/refman/8.0/en/charset-collation-names.html>

El nom de les col·leccions segueixen una convenció pel que fa el seu nom.

- El nom de la col·lecció comença per el nom del joc de caràcters del que està associat.
 - **utf8mb4**: **utf8mb4_bin**, **utf8mb4_general_ci**, **utf8mb4_spanish_ci**,...
- Una col·lecció d'idiom inclou un codi regional o un nom d'idioma:
 - **utf8mb4**: **utf8mb4_bin**, **utf8mb4_general_ci**, **utf8mb4_spanish_ci**,...
- L'últim sufix indica si a la col·lecció es distingeix entre majúscules i minúscules (case-sensitive), sensible als accents (accent-sensitive).

Sufix	Significat
_ai	Accent-insensitive
_as	Accent-sensitive
_ci	Case-insensitive
_cs	Case-sensitive
_ks	Kana-sensitive
_bin	Binary (comparació a nivell de byte)

<https://dev.mysql.com/doc/refman/8.0/en/create-database.html>

L'estàndard SQL, en canvi, té una sentència potser més potent que la creació de BD: la sentència de creació d'esquemes anomenada `CREATE SCHEMA`. Amb la creació d'esquemes es pot agrupar un conjunt d'elements de la BD que són d'un usuari.

La sintaxi d'aquesta sentència amb SQL92 és la següent:

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

On:

- **nom_esquema**: nom que es vol que tingui l'esquema
- **AUTHORIZATION usuari**: indicar l'usuari de l'esquema (l'esquema tindrà el nom de l'usuari).
- **llista_elements_esquema**: sentències de creació de taules, vistes, dominis, etc.

Exemple:

```
CREATE SCHEMA AUTHORIZATION usuari1;
```

```
CREATE TABLE producte (  
    color VARCHAR(10) PRIMARY KEY  
    ,quantitat NUMBER  
);  
  
CREATE VIEW producte_view AS  
    SELECT color, quantitat  
        FROM producte WHERE color = 'vermell';  
  
GRANT select ON producte_view TO usuari1;
```

2.1.3. Objectes de BD

La majoria dels SGBDR tenen varis objectes que es poden definir en una BD. Molts d'ells estan definits en l'estàndard SQL, però altres no.

Depenent del SGBD podem tenir diferents estructures de dades, cadascuna de les quals s'ha de descriure en el disseny de la base de dades de manera que es pugui crear durant la fase de creació del desenvolupament de la BD.

Els tipus d'objectes que es poden crear amb MySQL són:

Objecte	Descripció
Taula	Unitat bàsica d'emmagatzematge formada per files i columnes.
Vista	Representa lògicament subconjunts de dades d'una o més taules.
Índex	Millora el rendiment d'algunes consultes.

2.2. Definició del nivell lògic: taules i restriccions

2.2.1. Creació de taules

<https://dev.mysql.com/doc/refman/5.6/en/create-table.html>

Per crear una taula s'utilitza la sentència `CREATE TABLE`.

Sintaxi:

```
CREATE[TEMPORARY] TABLE [IF NOT EXISTS] [BD.]nomTaula  
(definicióCreate,...)  
[opcionsTaula]  
[opcionsParticio]  
;
```

On **definicióCreate** pot ser:

definicióColumna

definicióRestricció

definicióIndex

definicióCheck

On **definicióColumna** pot ser:

```
#Camp amb un valor concret  
nomColumna {tipusDades/domini} [DEFAULT valor_per_defecte]  
  
#Camp calculat  
nomColumna {tipusDads} [GENERATED ALWAYS] AS (expressió) [VIRTUAL |  
STORED]
```

nomColumna: és el nom de la columna.

tipusDades: és la mida i tipus de dades de la columna

- *domini*: és el domini dels valors admesos per la columna (Oracle no té dominis).
- *DEFAULT valor_per_defecte*: indica un valor per defecte si no s'indica a l'executar una sentència `INSERT` sobre la taula.

<https://dev.mysql.com/doc/refman/8.0/en/create-table.html>

Tipus de dades del MySQL

ANSI defineix els següents tipus de dades per definir les columnes de les taules. Això no vol dir que els diferents SGBD els implementin tots o bé que n'afegeixin de nous.

Els tipus de dades bàsics definits per ANSI són:

- CHARACTER
- CHARACTER VARYING (or VARCHAR)
- CHARACTER LARGE OBJECT
- NCHAR
- NCHAR VARYING
- BINARY
- BINARY VARYING
- BINARY LARGE OBJECT
- NUMERIC
- DECIMAL
- SMALLINT
- INTEGER
- BIGINT
- FLOAT
- REAL
- DOUBLE PRECISION
- BOOLEAN
- DATE
- TIME
- TIMESTAMP
- INTERVAL

L'experiència diu que els més utilitzats són:

- CHAR
- VARCHAR
- INTEGER
- DECIMAL
- DATE
- TIMESTAMP

MySQL utilitza diferents tiups de dades separats en 3 categories: numèriques, cadena de caràcters i data i hora

<http://dev.mysql.com/doc/refman/8.0/en/data-types.html>

NUMÈRIQUES		Tamany
INT / INTEGER [(M)]	<p>El tipus enter per defecte. Pot ser <i>signed</i> o <i>unsigned</i> Amb signe (signed): -2.147.483.648 a 2.147.483.647. Sense signe (unsigned): 0 a 4.294.967.295.</p> <p>M representa el número de dígitos a mostrar.</p>	4 bytes
TINYINT [(M)]	<p>És el tipus enter més petit. Amb signe: -128 a 127 Sense signe: 0 a 255</p>	1 byte
BOOLEAN	Tipus booleà implementat amb TINYINT (1)	1 byte
SMALLINT [(M)]	<p>Amb signe: -32.768 a 32.767 Sense signe: 0 a 65.535 Pots especificar fins a 5 dígitos</p>	2 bytes
MEDIUMINT [(M)]	<p>Amb signe: -8.388.608 a 8.388.607 Sense signe: 0 a 16.777.215</p>	3 bytes
BIGINT [(M)]	<p>El tipus enter més gran Amb signe: -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807 Sense signe: 0 a 18.446.744.073.709.551.615</p>	8 bytes
FLOAT [(M,D)]	<p>El tipus real amb punt flotant. Aquest tipus no pot ser sense signe (unsigned). Pots definir la longitud a mostrar (M) i el número de decimals (D) a mostrar. Per defecte s'utilitza (10,2). A on 2 és el número de decimals i 10 el numero total de dígitos (incloent els decimals). El valor decimal pot arribar fins a una precisió de 24 dígitos.</p> <p>MySQL arrodoneix els valors emmagatzemats. Per exemple si tenim un camp FLOAT(7,4) i volem guardar el valor 999,00009 el valor aproximat serà 999,0001.</p>	4 bytes

DOUBLE [(M,D)] REAL [(M,D)]	El tipus real de doble precisió tampoc pot ser sense signe. El seu valor de M i D són 16,4 i la precisió pot arribar a ser de 53 dígits. REAL és sinònim de DOUBLE	8 bytes
DECIMAL [(M,D)] NUMERIC [(M,D)]	Podem definir un tipus punt flotant desempaquetat que no té signe. Els valors de M i D són obligatoris. NUMERIC és sinònim de DECIMAL M pot tenir un rang de 1 a 65 (num total dígits) D pot tenir un rang de 0 a 30 (part decimal) Exemple: DECIMAL(5,2) → possibles valors que podem representar amb 5 dígits i 2 decimals. Rang -999.99 to 999.99.	Depèn Referència MySQL
BIT(M)	És un tipus per guardar valors amb bits. M: Especifiquem el número de bits que pot anar de 1-64. Exemple: BIT(6) → b'010101' http://dev.mysql.com/doc/refman/8.0/en/bit-value-literals.html	

TAMANY DECIMAL

El que ocupa un DECIMAL ve determinat pel número de dígits de la part entera i de la part decimal.

Els valors s'emmagatzemen utilitzant paquets de 9 dígits en 4 bytes. La mida utilitzada es determina de forma separada per la part entera i per la part decimal.

Cada múltiple de 9 dígits necessita 4 bytes. Si no es poden realitzar paquets de 9 dígits exactes, s'utilitzarà aquesta taula per saber-ne el número de bytes a utilitzar.

Dígits	Bytes
0	0
1–2	1
3–4	2
5–6	3

Exemple: DECIMAL (20, 6)

La columna té 14 dígits a la part entera i 6 a la part decimal. La part entera necessita de 4 bytes per 9 dígits i 3 bytes per la resta de 5 dígits. Pels 6 dígits de la part decimal necessitem 3 bytes. Aquest camp utilitzarà un total de 10 bytes.

<http://dev.mysql.com/doc/refman/8.0/en/integer-types.html>

<http://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html>

<http://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html>

AUTO_INCREMENT

Podem utilitzar aquest atribut quan volem generar identificadors únics o valors en sèrie. Normalment comencen per 1 i augmenten en aquest valor per cada fila introduïda. Quan inserim el valor NULL en una columna AUTO_INCREMENT, MySQL inserta un valor més gran que el valor màxim actual de la columna.

Atenció!, només podem tenir un a columna AUTO_INCREMENT a la taula

Qualsevol columna que volem que s'utilitzi amb AUTO_INCREMENT, s'haurà de declarar amb la restricció NOT NULL i també com a clau primària (PRIMARY KEY) o com a clau candidata (UNIQUE)

Podem especificar el número inicial de seqüència al crear la taula. Per exemple:

```
CREATE TABLE nom_taula (
  seq INT UNSIGNED AUTO_INCREMENT PRIMARY KEY
) AUTO_INCREMENT = 999
```

CADENA		Tamany
CHAR (N)	Cadena de caràcters de longitud fixe. Longitud de 1 a 255 caràcters. Per defecte la longitud és 1	1 byte per caràcter
VARCHAR (N)	Cadena de caràcters de longitud variable. Longitud de 1 a 255 caràcters.	Només es guarda la informació dels caràcters ocupats
BLOB, TEXT	Camp amb una longitud de fins a 65.535 caràcters. BLOBs són "Binary Large Objects" i s'utilitzen per guardar informació binària com imatges, fitxers,... Els TEXT són per guardar text de longitud variable que faci distinció entre majúscules i minúscules.	
TINYBLOB or TINYTEXT	A BLOB or TEXT amb un màxim de 255 caràcters	
MEDIUMBLOB or MEDIUMTEXT	A BLOB or TEXT amb un màxim de 16.777.215	

LOB or LONGTEXT	A BLOB or TEXT amb un màxim de 4.294.967.295	
ENUM	Una enumeració, el camp pren el valor d'una llista de valors possibles. Per exemple ENUM('A','B','C')	65.535 membres (1 o 2 bytes)

DATA i HORA		Tamany
DATE	Una data amb el format YYYY-MM-DD Entre 1000-01-01 i 9999-12-31	3 bytes
TIME	Valor d'una hora en format hh:mm:ss	3 bytes
DATETIME	Valor d'una data hora en format YYYY-MM-DD hh:mm:ss Valors: 1000-01-01 00:00:00 i 9999-12-31 23:59:59	8 bytes
TIMESTAMP	Valor d'instant de temps. Semplant a l'anterior, però sense utilitzar caràcters separadors. YYYYMMDDhhmmss	4 bytes
YEAR	Any en format YYYY (4 dígits)	1 byte

Tipus de dades Spatial (Extensió per dades espacial)

<https://dev.mysql.com/doc/refman/8.0/en/spatial-types.html>
<http://dev.mysql.com/doc/refman/8.0/en/spatial-extensions.html>

MySQL incorpora tipus de dades espacial corresponents a les classes definides per OpenGIS(<http://www.opengeospatial.org/standards/sfs>):

- GEOMETRY
- POINT
- LINESTRING
- POLYGON

El tipus GEOMETRY pot emmagatzemar els altres tres tipus de dades geomètriques.

Tipus de dades espacial de MySQL es basa en el model geomètric OpenGIS descrit a la documentació "The OpenGIS Geometry Model".

Per veure algun exemple de com utilitzar aquest tipus de dades podem anar a l'apartat "Using Spatial Data" de la documentació oficial.

Tipus de dades JSON

A partir de MySQL 5.7.8, MySQL suporta de forma nativa el format JSON (JavaScript Object Notation). El tipus de dades JSON té una sèrie de característiques respecte un format JSON en una columna de tipus *string*:

- Validació dels documents guardats en un camp de tipus JSON. Si intentem de guardar un document JSON invàlid donarà un missatge d'error i no el deixarà inserir.
- Optimització d'emmagatzematge. Els documents JSON es converteixen en un format intern que permet un ràpid accés en els elements d'un document. D'aquesta manera quan volem accedir a un valor no cal realitzar un *parse* del fitxer per tal d'anar a buscar els elements, sinó que ho té guardat amb un format binari que li permet obtenir els objectes de forma més ràpida i eficient.

<http://dev.mysql.com/doc/refman/8.0/en/json.html>

També hi ha disponible una sèrie de funcions GeoJSON. Veure la secció "[Spatial GeoJSON Functions](#)"

Regles de nomenclatura en la creació de taules

Els noms de les taules i columnes han de seguir les següents regles estàndard:

- Utilitzar noms descriptius per taules i altres objectes de la BD.
- Ús de tot amb majúscules o minúscules en el nom de les taules. **En MySQL utilitzarem els noms amb minúscules**
- Ús de tot en minúscules o la notació "*Camel Case*" amb alguna de les seves variants en la definició dels noms de les columnes.
- Han de començar per una lletra.
- Han de tenir entre 1 i 30 caràcters.
- Només poden tenir els caràcters A-Z, a-z, 0-9, _ (subguió)
- Els noms no han de duplicar el nom d'un altre objecte del mateix esquema.
- En el cas que el nom tingui espais en blanc o caràcters nacionals (com la Ç) s'han de posar entre cometes (no es recomana l'ús d'aquests caràcters per temes de compatibilitat i es recomana l'ús del guió baix o un caràcter semblant en substitució de l'especial.
 - Per exemple la Ç la podem substituir per C. Si volsem que tingui espais en blanc els podem substituir per el caràcter de subratllat ' _ '

- No pot coincidir amb el nom d'una paraula reservada (CREATE, SELECT, UPDATE, etc.)
- Els noms no són sensibles a majúscules/minúscules. Per Exemple, **empleat** és el mateix que **emPLEat**.

COMENTARIS

A cada columna de la taula és convenient afegir-hi un comentari que defineix el significat de la columna en la taula. Hi ha camps que poden semblar trivials, però passat un temps ja no ho són.

Sintaxi

```
CREATE TABLE nom_taula (  
  ...  
  nomColumna tipuDades COMMENT "text pel comentari"  
  ...  
);
```

2.2.2. Creació de restriccions

Les restriccions ens serveixen per forçar regles a nivell de taula o columna. D'aquesta manera el SGBD ens controla la integritat de la informació que s'introdueixen a les taules.

Les restriccions són normes que es fan complir a les dades. En el cas d'Oracle, es poden utilitzar les restriccions per:

- Forçar regles sobre les dades d'una taula quan s'insereix, actualitza o elimina una fila de la taula. La restricció s'ha de complir per tal que es faci correctament l'operació.
- Evitar l'eliminació d'una taula si hi ha dependències d'altres taules.

L'estàndard SQL92 defineix les següents restriccions d'integritat de les dades:

Restricció model relacional	Restricció SQL92	Descripció
Integritat d'entitat	NOT NULL	Indica que una columna no pot tenir un valor nul.
Clau	UNIQUE	Indica que una columna o una combinació de columnes han de tenir valors únics en totes les files de la taula.
Clau	PRIMARY KEY	Identifica de forma única cada fila de la taula.
Integritat referencial	FOREIGN KEY	Estableix i força una relació de clau forana entre la columna i una columna de la taula a la que es fa referència.
Domini	CHECK	Indica una condició que ha de ser certa.

Amb MySQL Tenim 7 tipus de restriccions:

:

- NOT NULL
- PRIMARY KEY (clau primària)
- UNIQUE
- FOREIGN KEY (clau forana)
- CHECK
- TRIGGER

Noms de restricció

En la majoria dels SGBDR les restriccions es guarden al diccionari de dades i es fàcil fer-hi referència si se'ls hi assigna un nom significatiu.

Els noms de restriccions ha de seguir les normes estàndard de nomenclatura d'objectes. Si no s'assigna nom a una restricció, SGBD genera un nom de tal manera que el nom de la restricció sigui únic per la taula.

En altres SGBD(Oracle) el nom ha de ser únic per tot l'esquema de base de dades.

Les restriccions es poden definir en el moment de la creació de la taula o un cop ja creada.

Com a conveni, pel nom de les restriccions utilitzarem la següent nomenclatura:

Restricció	Nom de la restricció	Nom aut.MySQL
PRIMARY KEY	pk_taula	PRIMARY
UNIQUE	uk_taula_columna	nom_camp
FOREIGN KEY	fk_taula1_taula2	nom_taula+ib_fk_<nº seq>
CHECK	ck_taula_columnes /ck_nom_identificat iu	No implementada

Podem consultar les constraints que tenim definides en una taula. Per fer-ho podem consultar una taula de la BD de metadades anomenada `information_schema`. Concretament hem de consultar la taula `TABLE CONSTRAINTS`.

`information_schema.TABLE_CONSTRAINTS`

La restricció NOT NULL

La restricció NOT NULL assegura que la columna no conté valors nuls. Cal recordar que no es pot definir a nivell de taula; tant sols a nivell de columna

Les columnes sense aquesta restricció poden tenir valors nuls per defecte.

Exemple:

```
CREATE TABLE empleats
(
  codi INT(6),
  nom VARCHAR(30) NOT NULL,
  cognom VARCHAR(30),
  sou NUMBER(8,2),
  comissio NUMBER(2,2),
  data_alta DATE NOT NULL,
  ...
);
```

Com es pot veure a l'**Exemple**, s'aplica la restricció NOT NULL a les columnes nom i dataAlta.

Per temes d'eficiència MySQL tracta millor les columnes NOT NULL que no les que permeten valors NULL. Per tant si podem aplicar aquesta restricció MySQL haurà de realitzar menys comprovacions de si el camp és null o no etc. Conseqüentment menys temps de còmput de càlcul.

La restricció CLAU PRIMÀRIA (PRIMARY KEY)

Podem definir la clau primària a nivell d'un atribut o una llista d'atribut que formen una clau primària.

MySQL crea un índex automàticament pel camp o conjunt de camps que formen part de la clau primària.

Sintaxi:

```
CREATE TABLE taula(
  camp tipus PRIMARY KEY,
  llista_camps
)

CREATE TABLE taula(
  llista_camps,
  [CONSTRAINT [nom_constraint]] PRIMARY KEY ( atribut1 , ... )
)
```

En el cas del MySQL si volem crear un índex compostat per més d'un camp i un d'aquests és un valor autoincremental, aquest haurà de ser el primer de la llista.

Exemples:

```
CREATE TABLE clients (  
    id          INT PRIMARY KEY,  
    nom         VARCHAR(40),  
    cod_provincia INT  
);  
  
CREATE TABLE clients (  
    id          INT,  
    nom         VARCHAR(40),  
    cod_provincia INT,  
    CONSTRAINT pk_clients PRIMARY KEY (id)  
);
```

En MySQL encara que hi posem un nom a la restricció de clau primària el nom que s'utilitzarà i es guardarà a la taula de metadades serà el nom *PRIMARY*

La restricció **UNIQUE**

Amb una restricció d'integritat de clau **UNIQUE** es garanteix que no pot haver-hi dues files d'una taula que tinguin valors duplicats en una columna o conjunt de columnes indicat.

La columna (o conjunt de columnes) inclosa a la definició de la restricció **UNIQUE** s'anomena *clau única*. Si la restricció **UNIQUE** està formada per més d'una columna, el grup de columnes s'anomena *clau única composta*.

Les restriccions **UNIQUE** permeten l'entrada de valors nuls a no ser que també defineixi restriccions **NOT NULL** per les mateixes columnes.

Vindria a ser una clau candidata per aquella taula amb la possibilitat d'introducció de valors **NULLs**.

De la mateixa manera que en el cas de la clau primària també es crea un índex de tipus **UNIQUE** a la taula pel camp o camps que formen part de la restricció

La sintaxi no varia enfront de la clau primària excepte l'ús de la paraula clau **UNIQUE**.

```
CREATE TABLE taula(  
    camp tipus UNIQUE,  
    llista_camps  
)  
  
CREATE TABLE taula(  
    llista_camps,  
    [CONSTRAINT [nom_constraint1]] UNIQUE ( atribut1 , ... )  
)
```

Exemples:

```
CREATE TABLE clients (  
    id          INT PRIMARY KEY,  
    dni         CHAR(10) UNIQUE,  
    nom        VARCHAR(40),  
    cod_provincia INT  
);  
  
CREATE TABLE clients (  
    id          INT,  
    dni         CHAR(10),  
    nom        VARCHAR(40),  
    cod_provincia INT,  
    CONSTRAINT uk_clients_dni UNIQUE (dni)  
);
```

La restricció FOREIGN KEY

Una restricció `FOREIGN KEY`, o restricció d'integritat referencial, defineix una columna o conjunt de columnes com clau forana i estableix una relació entre una clau a la mateixa taula o en una de diferent.

Quan es crea una clau forana cal tenir en compte que el domini de les claus que es relacionen ha de ser el mateix (el tipus de dades ha de ser el mateix).

<https://dev.mysql.com/doc/refman/5.6/en/create-table-foreign-keys.html>

Sintaxi:

```
CREATE TABLE taula(  
    llista_camps,  
    [CONSTRAINT [nom_fk]] FOREIGN KEY ( atribut1 , ... )  
        REFERENCES taula_ref ( atribut1 , ... )  
        [ON DELETE { RESTRICT | CASCADE | SET NULL | NOT ACTION }]  
        [ON UPDATE { RESTRICT | CASCADE | SET NULL | NOT ACTION }]  
);
```

Important: Cal tenir present l'ordre dels atributs que col·loquem en els parèntesi per definir la `FOREIGN KEY` i l'ordre dels atributs de la taula a la que fem referència.

Exemple:

```
CREATE TABLE empleats (  
    codi          INT NOT NULL PRIMARY KEY,  
    nom           VARCHAR(30) NOT NULL,  
    cognom        VARCHAR(30),  
    email         VARCHAR(40),  
    sou           INT,  
    comissio      INT,  
    data_alta     DATE,  
    ...  
    codi_dep      INT,  
    CONSTRAINT fk_empleats_departaments FOREIGN KEY (codi_dep)  
        REFERENCES departaments(codi)  
);
```

També es poden definir restriccions `FOREIGN KEY` a nivell de columna mitjançant la sintaxi següent, però és molt millor definir totes les FKs a un sol lloc, però **no l'utilitzarem en cap cas.**

```
CREATE TABLE taula(  
    llista_camps,  
    camp_tipus REFERENCES taula_ref (camp1)  
);
```

Quan definim una clau forana hem dit que estem establint una regla d'integritat referencial. Quan vam estudiar el model relacional, vam veure que la integritat referencial podia provocar problemes en certs casos. Concretament, en l'eliminació o actualització de files de la taula principal.

- Actualització i eliminació de files a la taula principal

Per defecte quan s'eliminen files de la taula principal que tenen files relacionades de la taula secundària, MySQL no ho permet.

Per solucionar-ho, MySQL ofereix dues solucions a aquest problema, que consisteix en afegir aquestes clàusules

```
[ON DELETE { RESTRICT | CASCADE | SET NULL | NOT ACTION}]
```

- **RESTRICT:** El valor per defecte. No deixarà borrar el registre si aquest té un registre relacionat en un altre taula (registre fill)
- **CASCADE:** Quan s'esborri la fila de la taula principal també s'esborraran **totes** les files que en depenen. **Molt de compte amb l'ús d'aquesta clàusula. Podem esborrar registres de taules que no ens interessava esborrar.**
- **SET NULL:** Assigna el valor NULL als valors de la clau secundària quan s'esborra un registre de la principal.

- No es podrà utilitzar aquesta opció en claus foranes que poden formar part d'una clau primària, per exemple en implementacions d'entitats febles.
- NOT ACTION: Sinònim del RESTRICT, s'ha inclòs per complir amb l'estàndard SQL.

Sense les opcions `ON DELETE CASCADE` o `ON DELETE SET NULL`, la fila de la taula principal no es pot suprimir si a la taula secundària s'hi fa referència.

Moltes vegades també utilitzarem la **implementació de disparadors (triggers)**. Per tal de realitzar certes accions en l'esborrat o actualització d'un camp.

Nota: Cal dir que la clàusula **CASCADE** no activa els disparadors (*triggers*).

Exemple 1: eliminar les files de la taula `empleats` si s'esborra la fila de qui depèn a la taula `departament`.

```
CREATE TABLE empleats
(
    ...
    codi_dep INT,
    ...
    CONSTRAINT fk_empleats_departaments
    REFERENCES departaments(codi) ON DELETE CASCADE,
    ...
);
```

Exemple 2: posar a `NULL` el valor de la columna `codi_dep` a les files de la taula `empleat` si s'esborra la fila de qui depèn a la taula `departaments`.

```
CREATE TABLE empleat
(
    ...
    codiDept INT,
    ...
    CONSTRAINT fk_empleats_departaments
    REFERENCES departaments(codi) ON DELETE SET NULL,
    ...
);
```

Nota: Si volem activar o desactivar la comprovació de les claus foranes podem utilitzar el parameter [foreign key checks](#).

```
mysql> SET foreign_key_checks = 0;
```

La restricció CHECK

La restricció `CHECK` defineix una condició que ha de complir cada fila.

Al definir una taula, a vegades ens pot interessar afegir certes restriccions en els valors que poden assolir els camps.

Per exemple, si tenim la taula llibre, ens podria interessar obligar a que el camp “numero de pàgines” fos sempre positiu, o si tenim la taula comanda ens podria interessar obligar a que el camp “data servida” fos igual o superior al camp de “data de creació” de la comanda.

Sintaxi:

```
CREATE TABLE taula(  
    llista_camps,  
    [CONSTRAINT [CK_nom]] CHECK (expr)  
)
```

Exemple:

```
CREATE TABLE empleat  
(  
    codi          INT(6),  
    nom           VARCHAR(30) NOT NULL,  
    cognom        VARCHAR(30),  
    email         VARCHAR(40),  
    sou           DECIMAL(8,2)  
    comissio      DECIMAL(2,2),  
    data_alta     DATE,  
    ...  
    codi_dep      NUMBER(3),  
    CONSTRAINT fk_empleats_departaments FOREIGN KEY (codi_dep)  
        REFERENCES departaments(codi),  
    CONSTRAINT ck_empleats_sou_min CHECK (sou>0),  
    CONSTRAINT ck_empleats_sou_max CHECK (sou<6000),  
    CONSTRAINT ck_empleats_comissio CHECK (comissio<=(sou/10))  
);
```

IMPORTANT: Actualment en MySQL(v5.7) la restricció CHECK no es comprova, però permet crear-la per futures versions i per compatibilitat amb altres SGBD.

2.2.3. Modificació en una taula

Després de crear una taula, és possible que calgui canviar la seva estructura per qualsevol motiu (ens hem oblidat d'una columna, el tipus de dades d'una columna no és correcte, volem afegir una restricció, etc.). Per poder fer aquestes tasques, s'utilitza la sentència `ALTER TABLE`.

<https://dev.mysql.com/doc/refman/8.0/en/alter-table.html>

Per modificar una taula s'utilitza la sentència `ALTER TABLE`.

Sintaxi:

```
ALTER TABLE [BD.]nomTaula  
[definicioAlter [, definicioAlter]...;
```

Canviar el nom d'una taula

Sintaxi:

```
ALTER TABLE nomTaula RENAME [TO|AS] nomNovaTaula;
```

Exemple: canviar el nom de la taula POBLACIO per POBLACIONS.

```
ALTER TABLE poblacio RENAME TO poblacions;
```

Afegir una columna

Sintaxi:

```
ALTER TABLE nomTaula  
ADD [COLUMN] definicioColumna  
[FIRST | AFTER nomColumna ];
```

Exemple: afegir una columna a la taula de DEPARTAMENT després de la columna codi.

```
ALTER TABLE departaments  
ADD COLUMN codi_treball CHAR(3)  
AFTER codi;
```

A tenir en compte:

- La nova columna si no se li diu el contrari passarà a ser l'última columna de la taula.
- Si afegim una nova columna a una taula que ja té registres de dades, les files **inicialment tindran valor NULL per aquesta nova columna**.

- Què passa si aquesta columna li afegim la restricció de NOT NULL? *Feu-ne la prova.*

Modificar una columna

Permet canviar el tipus de dades, la mida i el valor per defecte d'una columna.

Sintaxi:

```
ALTER TABLE nomTaula
  MODIFY [COLUMN] definicioColumna
    [FIRST | AFTER nomColumna ];
```

Podem també canviar el nom d'una columna mitjançant la clàusula `CHANGE`:

Sintaxi:

```
ALTER TABLE nomTaula
  RENAME COLUMN nomVell TO nomNou;
```

Podem també canviar el nom d'una columna i la definició de la columna a la vegada mitjançant la clàusula `CHANGE`:

Sintaxi:

```
ALTER TABLE nomTaula
  CHANGE [COLUMN] nomVell nomNou definicioColumna
    [FIRST | AFTER nomColumna ];
```

Exemple: modificar el tipus de dades d'una columna de la taula de departament.

```
ALTER TABLE departaments
  MODIFY COLUMN cognom CHAR(30);
```

Exemple: modificar el nom de la columna cognom anterior.

```
ALTER TABLE departaments
  RENAME COLUMN cognom TO cognoms;
```

Exemple: modificar el nom de la columna cognom i a més li canviem el tipus de dades a `CHAR(30)`.

```
ALTER TABLE departaments
  CHANGE COLUMN cognom cognoms CHAR(30);
```

A tenir en compte en MySQL:

- Es pot augmentar la precisió o l'escala d'una columna numèrica.

- Es pot augmentar la mida de columnes numèriques o de caràcters.
- Es pot disminuir la mida d'una columna, sempre cal tenir en compte que el valor serà el valor màxim del tipus de dades si tenim un valor superior en el nou tipus de dades.
- Es pot convertir una columna CHAR en VARCHAR o convertir una columna VARCHAR en CHAR. Compte amb el canvi de mida
- Es pot canviar el tipus de dades INT->VARCHAR i a la inversa
- Cal tenir present que un canvi en el valor per defecte només afecta a les insercions posteriors al canvi.

En altres SGBD aquest punt és el que hem de mirar o tenir en compte

Esborrar una columna

Sintaxi:

```
ALTER TABLE nomTaula  
DROP [COLUMN] nomColumna;
```

Exemple: esborrar una columna de la taula de departament.

```
ALTER TABLE departaments  
DROP COLUMN codi_treball;
```

A tenir en compte:

- La columna pot tenir o no dades.
- Un cop esborrada la columna, no es pot recuperar.

2.2.4. Eliminar l'estructura de taules

La sentència `DROP TABLE` elimina la definició d'una taula.

Sintaxi:

```
DROP TABLE nomTaula;
```

A tenir en compte:

- S'esborren totes les dades de la taula.
- S'esborren tots els índexs associats a la taula.
- És una sentència irreversible i no demana confirmació a l'hora d'eliminar-la.

2.2.5. Esborrar el contingut d'una taula

Esborrar el contingut d'una taula o *truncar* una taula elimina totes les files de la taula i allibera l'espai que ocupa aquesta taula.

Sintaxi:

```
TRUNCATE TABLE nomTaula;
```

Amb SQL(DML) hi ha la sentència `DELETE` que també permet eliminar totes les files d'una taula, però no allibera l'espai de disc ocupat. La sentència `TRUNCATE` és més ràpida que `DELETE` per eliminar files, ja que:

- `TRUNCATE` és una sentència del llenguatge de definició de dades i no genera informació de *rollback* ("tornar enrere").
- És més eficient en temps que la sentència DML.
- Truncar una taula no engega els disparadors (*triggers*) d'eliminació de la taula.
- Si tenim un camp numèric autoincremental el número de seqüència es reinicia.
- Si la taula és la principal en una restricció d'integritat referencial, no es pot truncar. Abans caldrà desactivar la restricció

2.2.6. Afegir i treure restriccions

La restricció NOT NULL

La restricció NOT NULL la podem tractar com una modificació d'una columna.

Sintaxi:

```
ALTER TABLE nomTaula  
MODIFY [COLUMN] nomColumna (NULL | NOT NULL);
```

Exemple: Si volem que el camp nom passi a ser obligatori i el camp cognom a opcional:

```
ALTER TABLE empleats  
MODIFY COLUMN nom VARCHAR2(30) NOT NULL,  
MODIFY COLUMN cognom VARCHAR2(30) NULL  
;
```

La restricció UNIQUE, PRIMARY KEY, FOREIGN KEY

Les restriccions es poden afegir un cop la taula està creada mitjançant la sintaxi següent:

Sintaxi:

```
ALTER TABLE nomTaula  
ADD CONSTRAINT [nom_constraint] PRIMARY KEY (camp1, ... )  
| ADD CONSTRAINT [nom_constraint] UNIQUE (camp1,...)  
| ADD [CONSTRAINT [nom_fk]] FOREIGN KEY (camp1,..)  
REFERENCES taula_ref (camp1,...)  
[ON DELETE {RESTRICT|CASCADE|SET NULL|NOT ACTION }]  
[ON UPDATE {RESTRICT|CASCADE|SET NULL|NOT ACTION }]  
| ADD CONSTRAINT [CK_nom] CHECK (expr)
```

Exemple:

```
ALTER TABLE TABLE departaments  
ADD COLUMN codi NUMBER(3),  
ADD CONSTRAINT pk_departaments PRIMARY KEY(codi)  
...;
```

NOTA: Per afegir una restricció de clau primària en una taula que ja posseeix una restricció d'aquest tipus, primer cal esborrar l'anterior.

Esborrar una restricció

En tot moment podem esborrar les restriccions d'una taula.

Sintaxi:

```
ALTER TABLE nomTaula
DROP PRIMARY KEY
| DROP {INDEX|KEY} index_name
| DROP FOREIGN KEY nom_clau_forana
| DROP CONSTRAINT nom_de_la_restricció (en el cas PK `PRIMARY`)
```

Exemple: En l'exemple volem esborra les 3 restriccions de la taula (la clau primària, una clau de tipus única i una clau forana a la taula alumnes

```
ALTER TABLE empleats
DROP PRIMARY KEY,
DROP INDEX uk_empleats_dni,
DROP FOREIGN KEY fk_empleats_departaments;
```

Exemple: En l'exemple volem esborrar el mateix que la taula anterior, però amb un altra sintaxi.

```
ALTER TABLE empleats
DROP CONSTRAINT `PRIMARY`,
DROP CONSTRAINT uk_empleats_dni,
DROP CONSTRAINT fk_empleats_departaments;
```

Atenció !!!

Quan esborreu una clau forana mitjançant ALTER TABLE DROP FOREIGN KEY O DROP CONSTRAINT no esborra l'índex associat a aquesta clau forana.

```
mysql> SHOW INDEX FROM empleats;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation
empleats	0	PRIMARY	1	empleat_id	A
empleats	0	uk_empleats_dni	1	dni	A
empleats	1	fk_empleats_departaments	1	dep_id	A

```
mysql> ALTER TABLE
        DROP FOREIGN KEY fk_empleats_departaments;
```

```
mysql> SHOW INDEX FROM empleats;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation
empleats	0	PRIMARY	1	empleat_id	A
empleats	0	uk_empleats_dni	1	dni	A
empleats	1	fk_empleats_departaments	1	dep_id	A

En canvi, si esborreu una constraint de tipus UNIQUE aquesta acció si que esborrarà l'INDEX associat a aquella CONSTRAINT.

```
mysql> SHOW INDEX FROM empleats;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation
empleats	0	PRIMARY	1	empleat_id	A
empleats	0	uk_empleats_dni	1	dni	A
empleats	1	fk_empleats_departaments	1	dep_id	A

```
mysql> ALTER TABLE
        DROP CONSTRAINT uk_empleats_dni;
```

```
mysql> SHOW INDEX FROM empleats;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation
empleats	0	PRIMARY	1	empleat_id	A
empleats	1	fk_empleats_departaments	1	dep_id	A

Exemple: Si volem esborrar la restricció de clau primària d'una taula i aquesta és un valor AUTONUMÈRIC no ens deixarà esborrar-la. Ho podem fer si abans li creem una restricció de tipus UNIQUE.

```
ALTER TABLE curs DROP PRIMARY KEY;
```

ERROR-1075 (42000) :

2.3. Definició del nivell extern: Vistes

2.3.1. Introducció

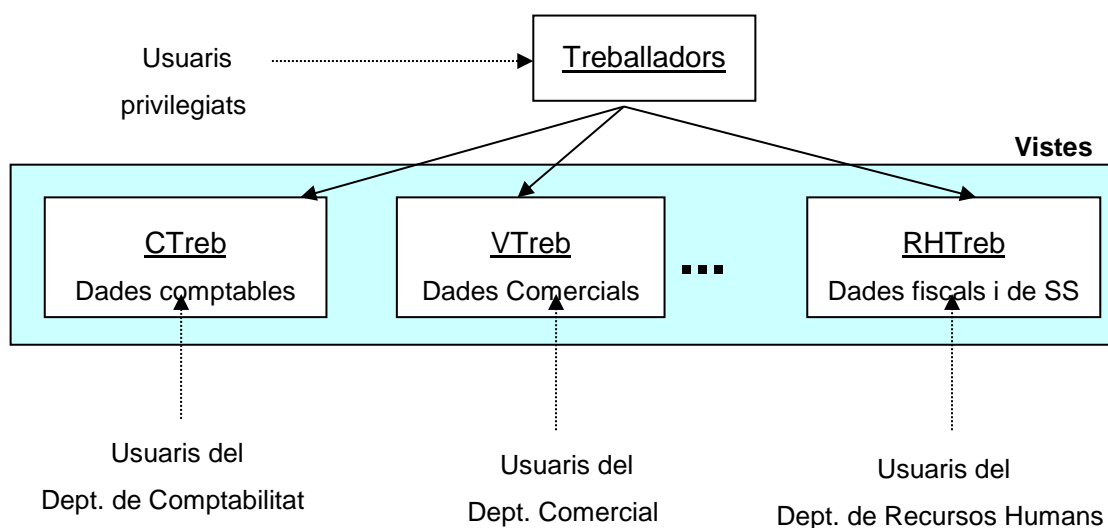
La taula ens permet representar un concepte en el món real, però:

- Gent diferent pot veure una taula conceptualment diferent.
- No tothom té necessitat d'accedir a totes les dades de la taula.
- De vegades convé amagar certa informació a determinats usuaris

Per ajudar a aconseguir aquestes característiques se solen utilitzar les vistes:

- Es poden considerar taules virtuals: no estan creades físicament.
- Permeten independència lògica de les dades.
- Es creen a través de consultes sobre les taules base o altres vistes, o bé:
- Aplicant restricció de files
- Aplicant restricció de columnes
- Els usuaris no saben que realment el que veuen no és la taula original.
- Qualsevol consulta sobre una vista està permesa.
- No sempre es permeten realitzar operacions d'inserció, d'actualització o d'esborrat.

Amb el següent esquema pot quedar més clar el concepte de vista:



Hi ha dos tipus de vistes:

Simples. Les formen una sola taula i no contenen funcions d'agrupació. El seu avantatge és que permeten realitzar operacions DML sobre elles.

Complexes. Obtenen dades de varies taules, poden utilitzar funcions d'agrupació. No sempre permeten operacions DML.

2.3.2. Creació de vistes

<https://dev.mysql.com/doc/refman/8.0/en/create-view.html>

Sintaxi:

```
CREATE [OR REPLACE] VIEW nomVista [(llistaColumnes)] AS  
  sentència_SELECT  
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

on:

`OR REPLACE`. Si la vista ja existeix, la canvia per l'actual.

nomVista. El nom de la vista.

llistaColumnes: És el llistat de noms de les columnes que volem que tingui la nostra vista. Cal tenir en compte que el número de columnes ha de coincidir amb el número de columnes retornades per la sentència `SELECT`.

sentènciaSELECT. Sentència `SELECT` que genera la consulta i que alimentarà la Vista.

`WITH CHECK OPTION`. Només permet fer operacions d'actualització (`INSERT`, `UPDATE`) si es compleix les condicions de selecció de la consulta que defineix la vista (les condicions del `WHERE`).

`CASCADED | LOCAL`. Determina l'abast de la comprovació quan la vista està definida en termes d'una altre vista. El valor per defecte és `CASCADED`.

Per obtenir més informació respecte `WITH CHECK OPTION` pots anar a la secció: <https://dev.mysql.com/doc/refman/8.0/en/view-check-option.html>

Com a criteri estès el nom de la vista es crea amb un prefix que normalment és la lletra `v` o `'v_'` per indicar que aquell objecte de la BD és una vista.

Exemple: Volem crear una vista amb els alumnes que són majors d'edat.

```
CREATE VIEW v_alumnes18 AS  
  SELECT *  
    FROM alumnes  
   WHERE edat>18;
```

Cal tenir en compte que cada vegada que utilitzem una vista la sentència `SELECT` que la genera s'executa. Per tant indirectament estem executant la consulta `SELECT` cada vegada.

Un cop tenim la vista creada la podem utilitzar per generar noves consultes sobre aquesta. De fet el SGBD la tracta com una taula amb l'excepció que no s'hi poden afegir, modificar o esborrar registres.

Podem recuperar les diferents vistes creades mitjançant la taula de metadades `information_schema.VIEWS`.

2.3.3. Modificació i esborrat de vistes

Per la modificació d'una vista l'hem de tornar a definir.

Sintaxi:

```
ALTER VIEW nomVista [(llistaColumnes)] AS  
    sentència_SELECT  
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

Aquesta sentència equivaldria a utilitzar `CREATE OR REPLACE VIEW`

Per l'esborrat d'una vista

Sintaxi:

```
DROP VIEW [IF EXISTS] nomVista1 [, nomVista2]
```

2.3.4. Utilització de vistes actualitzables

<http://dev.mysql.com/doc/refman/8.0/en/view-updatability.html>,

<https://dev.mysql.com/doc/refman/8.0/en/view-check-option.html>

Important: Podem utilitzar les vistes per tal d'afegir o modificar registres de les taules a les quals fan referència.

Cal dir però que només es poden utilitzar comandes DML contra una vista sempre i quan afectin només a camps d'una sola taula si la vista és complexa.

Exemple:

```
CREATE TABLE t1 (a INT);

CREATE VIEW v1 AS
  SELECT *
    FROM t1
   WHERE a < 2
WITH CHECK OPTION;

CREATE VIEW v2 AS
  SELECT *
    FROM v1
   WHERE a > 0
WITH LOCAL CHECK OPTION;

CREATE VIEW v3 AS
  SELECT *
    FROM v1
   WHERE a > 0
WITH CASCADED CHECK OPTION;
```

En aquest exemple s'han creat 3 vistes i v2 i v3 depenen d'una primera vista v1.

```
mysql> INSERT INTO v2 VALUES (2);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO v3 VALUES (2);
ERROR 1369 (HY000): CHECK OPTION failed 'test.v3'a
```

2.4. Índexs

2.4.1. Introducció

Els índexs són objectes d'esquema que fan que el SGBD acceleri les operacions de consulta i ordenació sobre els camps als que l'índex fa referència.

Si no es té un índex a una columna per la qual es fa una consulta o ordenació, l'operació en qüestió fa una exploració completa de la taula.

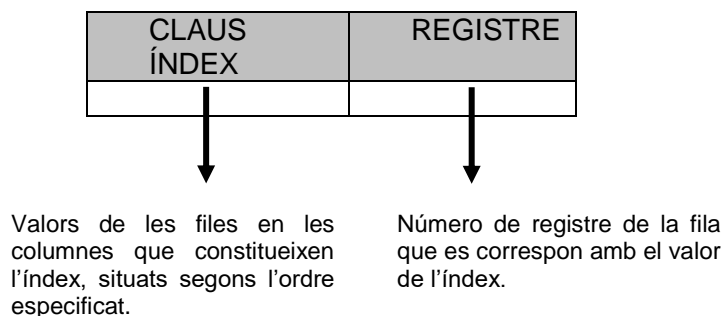
Per tant, un índex proporciona accés directe i ràpid a les files d'una taula. El seu objectiu és reduir la necessitat de fer operacions d'E/S de disc utilitzant un accés indexat per trobar les dades ràpidament. Normalment en molts casos és el propi SGBD que utilitza i manté els índexs automàticament i, un cop creat l'índex, l'usuari no ha de fer cap operació sobre l'índex.

Són objectes nous a la BD i es guarden a part de la taula a la que fa referència, el que permet crear-los i esborrar-los en qualsevol moment (són independents lògica i físicament de la taula).

Cada vegada que s'afegeix un nou registre, els índexs afectats s'actualitzen per tal que la seva informació estigui actualitzada en tot moment. D'aquí venen molts dels problemes quan tenim una sobre saturació d'índexs. El fet de crear molts índex alenteix les operacions d'afegir, modificar i esborrat.

Hi ha molts índex que es creen de forma implícita, com a conseqüència de les restriccions PRIMARY KEY (que obliga a crear un índex únic sobre els camps clau), UNIQUE (crea també un índex únic) i FOREIGN KEY (crea un índex amb possibilitat de repetir valors, índex amb duplicats). Aquests índexs són obligatoris, i gairebé sempre els crea el mateix SGBD.

Una etiqueta d'índex es pot considerar com una taula de dues columnes, tal i com es mostra a continuació:



2.4.2. Creació d'índexs

Sintaxi

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX nom_index  
    [index_type]  
    ON nom_taula (index_col_name,...)  
    [index_option]  
    [algorithm_option | lock_option] ...
```

Exemples:

```
#Crear un índex amb 1 camp  
CREATE INDEX idx_factures_data  
    USING BTREE  
    ON FACTURES (data_factura)  
  
#Crear un índex amb 3 camps  
CREATE INDEX idx_dni_cognom_codialumne  
    USING HASH  
    ON ALUMNES (dni,cognom,codi_alumne)  
  
#Crear un índex del tipus HASH  
CREATE INDEX idx_dni  
    USING HASH  
    ON ALUMNES (dni
```

Paràmetres:

UNIQUE: Un índex garanteix que no pot haver-hi dues files d'una taula que tinguin valors duplicats en una columna o conjunt de columnes indicat. Aquest índex podem afegir valors de tipus NULL.

FULLTEXT: Només aplicable a columnes de tipus CHAR, VARCHAR i TEXT. Quan volem buscar sobre gran quantitats de dades i realitzar cerques amb llenguatge natural.

SPATIAL: Aplicable a columnes de tipus geospacial (POINT, GEOMETRY). Les columnes han de tenir la restricció NOT NULL.

index_col_name:

```
col_name [(length)] [ASC | DESC]
```

Podem definir quina longitud del camp volem utilitzar en l'índex. *Això només té sentit en camps de tipus caràcter (CHAR, VARCHAR, TEXT).*

Podem indicar també l'orde: Ascendent (ASC) i Descendent (DESC). Per defecte s'utilitza l'ASC. *Aquest valor no s'aplica a la versió actual (5.7), però es manté la sintaxi per futures implementacions.*

index_type:

USING {BTREE | HASH}

Tipus d'índex BTREE(Arbre B) i HASH (funció de hash).

[Comparació de índex B-Tree i Hash](#)

Els índexs BTREE responen a consultes amb expressions que utilitzin els operadors =, >, >=, <, <=, BETWEEN, LIKE (amb un %).

Els índexs HASH responen a consultes amb expressions que utilitzin els operadors =, !=. **No són vàlids per expressions amb rangs (<,<=, >=, >, BETWEEN, etc...).**

No tots els tipus d'índex es poden crear. MySQL van en funció del tipus de Storage Engine que utilitzem. A La versió 8.0 tenim:

<https://dev.mysql.com/doc/refman/8.0/en/create-index.html>

Storage Engine	Tipus d'índex permés
InnoDB	BTREE
MyISAM	BTREE
MEMORY/HEAP	HASH, BTREE
NDB	HASH, BTREE

index_option:

```
KEY_BLOCK_SIZE [=] value
| index_type
| WITH PARSEr parser_name
| COMMENT 'string'
```

KEY_BLOCK_SIZE: es pot especificar la mida en bytes dels blocs de la clau de l'índex.

WITH PARSEr: Només es pot especificar amb índexs de FULLTEXT.

Veure [FULLTEXT parser plugins](#)

Veure [Writing Full-Text parser plugins](#)

COMMENT: Es pot afegir un comentari de fins a 1.024 caràcters.

algorithm_option:

```
ALGORITHM [=] {DEFAULT|INPLACE|COPY}
```

lock_option:

```
LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

Consideracions:

- Amb MySQL no podem crear un Índex de clau primària (PRIMARY KEY) amb la sintaxi de CREATE INDEX, s'ha de realitzar mitjançant la sentència DDL ALTER TABLE.
- Amb MySQL no podem esborrar un índex de constraint FOREIGN KEY ja que el necessita.
-

Links als manuals de referència de MySQL

- [Com MySQL utilitza els índexs](#)

Si creem un índex amb dos camps alhora no és el mateix que crear un índex per cada camp. Aquest tipus d'índex l'utilitza el SGBD quan es cerquen o ordenen valors utilitzant els dos camps a la vegada.

És aconsellable crear índexs en columnes que:

Tinguin una gran quantitat de valors.

Tinguin una gran quantitat de valors NULL.

Una o més columnes s'utilitzen juntes habitualment dins una clàusula WHERE o dins una condició de combinació.

Habitualment formen part de clàusules WHERE, GROUP BY o ORDER BY.

Formen part de llistats de consultes de grans taules sobre les que gairebé sempre es mostren com a molt el 4% de les files.

No és aconsellable en columnes que:

Pertanyin a taules petites.

No s'utilitzen habitualment en la clàusula `WHERE` de consultes.

S'espera que la majoria de les consultes recuperaran més del 2-4% de les files de la taula.

Pertanyen a taules que s'actualitzen habitualment.

S'utilitzen dins d'expressions.

Veure els índexs que tenim

Per poder mostrar els índex d'una taula tenim la sentència `SHOW INDEX`.

Exemples:

```
SHOW INDEX FROM empleats;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name
empleats	0	PRIMARY	1	empleat_id
empleats	0	uk_empleats_dni	1	dni

Atenció !!!

Quan esborreu una clau forana mitjançant `ALTER TABLE DROP FOREIGN KEY` O `DROP CONSTRAINT` no esborra l'índex associat a aquesta clau forana.

```
mysql> SHOW INDEX FROM empleats;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation
empleats	0	PRIMARY	1	empleat_id	A
empleats	0	uk_empleats_dni	1	dni	A
empleats	1	fk_empleats_departaments	1	dep_id	A

```
mysql> ALTER TABLE
        DROP FOREIGN KEY fk_empleats_departaments;
```

```
mysql> SHOW INDEX FROM empleats;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation
empleats	0	PRIMARY	1	empleat_id	A
empleats	0	uk_empleats_dni	1	dni	A
empleats	1	fk_empleats_departaments	1	dep_id	A

Índexs invisibles

A mesura que la nostra BD creix alguns dels índexs que es van crear en el seu moment no tenen sentit en l'actualitat, però no tenim la certesa que aquell índex s'estigui utilitzant o sigui d'utilitat.

A partir de la versió 8.0 MySQL ofereix la possibilitat de posar índexs en quarantena sense esborrar-los, però que l'optimitzador del SGBD els ignori per llavors fer-ne un esborrat segur.

Soft Delete (Recycle Bin)

Per posar un índex en quarantena, però no borrar-lo podem utilitzar els modificadors `INVISIBLE` i `VISIBLE` de la sentència `ALTER INDEX`.

Exemples:

```
#Si pensem que l'index nos'està utilitzant el posem com a invisible
ALTER TABLE països ALTER INDEX c INVISIBLE;

#Si necessitem revertir el canvi utilitzem VISIBLE;
ALTER TABLE països ALTER INDEX c VISIBLE;
```

Staged Rollout

Un altre cosa que ens pot passar alhora de crear un índex és que aquest canvi ens canviï els plans d'execució i aquests canvis poden comportar riscos de regressió.

Per tant una bona manera és crear l'índex de forma invisible i activar-lo en el moment oportú.

Exemple:

```
ALTER TABLE Country ADD INDEX c (Continent) INVISIBLE;

#Passat un temps el fem visible
ALTER TABLE Country ALTER INDEX c VISIBLE;
```

Capítol 3. DML – SQL

3.1. Inserció, modificació i eliminació

3.1.1. Introducció

Dins del llenguatge de manipulació de dades SQL (DML SQL) hi trobem les instruccions per poder manipular les dades que contenen les taules de la base de dades.

Ens cal poder afegir, modificar i eliminar els registres a les diferents taules que hem definit i creat mitjançant el llenguatge DDL.

3.1.2. Inserció de dades

Per poder afegir dades a una taula es fa mitjançant la instrucció `INSERT`.

<http://dev.mysql.com/doc/refman/8.0/en/insert.html>

La seva sintaxi bàsica és:

```
INSERT INTO taula_nom [(llistaDeCamps)]  
VALUES ({expr | DEFAULT},...),(...),...
```

On:

- ***Taula_nom*** indica la taula on volem inserir el nou registre de dades.
- ***llistaDeCamps*** és una dada opcional i serveix per indicar en quins camps volem inserir noves dades.

Després de la paraula clau ***VALUES*** s'indiquen els valors dels camps del nou registre que volem afegir a la taula. Aquests valors poden ser valors discrets o bé expressions.

La ***llistaDeCamps*** no és obligatori indicar-la. En aquest cas hem d'indicar la llista de valors de tots els camps de la taula i en el mateix ordre que apareixen en la definició de la taula.

Si indiquem la ***llistaDeCamps*** llavors permet indicar només els valors d'uns quants camps de la taula. Els valors que no indiquem s'omplen amb el seu valor per defecte (***DEFAULT*** en la definició dels camps de la taula) o bé amb ***NULL***. Si hi ha algun camp que té una restricció ***NOT NULL***, es produirà un error si no indiquem cap valor per aquell camp.

Es pot utilitzar la paraula ***DEFAULT*** per introduir explícitament el valor per defecte si n'hi ha. Si el camp no té valor per defecte i l'utilitzem tindrem un error indicant-nos que el camp no té valor per defecte

Exemple: Volem inserir un nou departament a la taula `DEPARTAMENT`. L'estructura de la taula és la següent:

```
mysql> desc DEPARTAMENT;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| codi           | int(11)       | NO   |     | NULL    |      |
| nom            | varchar(14)   | YES  |     | NULL    |      |
| localitzacio   | varchar(13)   | YES  |     | NULL    |      |
+-----+-----+-----+-----+-----+-----+
```

Si només volem inserir les dades del *codi* i el *nom* del departament ho podem fer així:

```
mysql> INSERT INTO DEPARTAMENT(codi,nom)
VALUES (50,'Informatica');
```

1 row created.

```
mysql> SELECT *
FROM DEPARTAMENT;
```

codi	nom	localitzacio
10	Comptabilitat	Sevilla
20	Investigacio	Madrid
30	Vendes	Barcelona
40	Produccio	Bilbao
50	Informatica	

Es pot veure com en la columna *localitzacio* del nou departament hi ha com a valor NULL, ja que no s'ha indicat aquest valor en la instrucció INSERT. Això s'ha produït perquè no hi havia cap restricció NOT NULL a la columna.

Exemple: Si volem inserir totes les dades d'un nou departament ho podem fer així sense haver d'indicar el nom de cadascuna de les columnes:

```
mysql> INSERT INTO DEPARTAMENT
VALUES (50,'Informatica', 'Blanes');
```

Nota: Hem d'introduir els valors amb el mateix ordre de la definició de les columnes.

Exemple: Podem inserir més d'una fila dins la mateixa sentència INSERT.

```
mysql> INSERT INTO DEPARTAMENT
VALUES (50, 'Informatica', 'Blanes'),
(10, 'Comptabilitat', 'Blanes'),
(20, 'Investigacio', 'Madrid');
```

Query OK, 3 rows affected (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 0

Exemple: Si no indiquem les dades d'alguna de les columnes que tenen alguna restricció d'integritat, donaria un error semblant a aquest:

```
mysql> INSERT INTO DEPARTAMENT(nom, localitzacio)
VALUES ('Informatica', 'Blanes');
ERROR 1048 (23000): Column 'codi' cannot be null
```

Important: Compte amb el mode d'execució de MySQL. No produirà error si estem executant MySQL en mode NO_STRICT.

<http://dev.mysql.com/doc/refman/8.0/en/sql-mode.html#sql-mode-important>

Per veure quin mode de SQL estem podem fer:

```
mysql> SHOW VARIABLES LIKE 'sql_mode';
```

Per activar un mode de SQL només cal executar:

```
mysql> SET sql_mode = 'TRADITIONAL';
```

Amb aquest mode es tindran en compte les restriccions de NOT NULL,

Errors d'inserció

Els errors més habituals que es poden produir quan s'insereixen o es modifiquen dades són els següents:

- Falta un valor obligatori per una columna NOT NULL.
- Un valor duplicat viola la restricció d'unicitat.
- Es viola una restricció de clau forana (FK).
- Es viola una restricció CHECK (**no implementat a MySQL**).
- El tipus de dades no coincideix (compte amb els canvis implícits).
- El valor és massa gran per la columna.
 - Quan volem inserir 600 en un camp TINYINT
 - Quan volem inserir 'Pere' en un camp VARCHAR(2)

Inserció de valors especials

Quan es fa la inserció d'una nova fila a una taula es poden utilitzar funcions.

Per **Exemple**:

```
CREATE TABLE taula1 (
  col_data DATE,
  col_datatemp DATETIME)
);
```

<https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html>

```
INSERT INTO taula1 (col_data, col_datatemp)
VALUES (CURRENT_DATE(), CURDATE())
, (NOW(), NOW());
```

```
mysql> select * from taula1;
+-----+-----+
| col_data | col_datatemp |
+-----+-----+
| 2016-01-15 | 2016-01-15 00:00:00 |
| 2016-01-15 | 2016-01-15 11:07:32 |
+-----+-----+
2 rows in set (0.00 sec)
```

Inserció de registres a partir de les files d'una taula

Hi ha un tipus de consulta, anomenada d'addició de dades, que permet omplir dades d'una taula copiant el resultat d'una consulta.

Aquest tipus d'addició de dades es basa amb la consulta SELECT que serà la que retornarà les dades a afegir. Com és lògic, l'ordre d'aquests camps ha de coincidir amb la llista de camps indicada en la instrucció INSERT.

Sintaxi:

```
INSERT INTO taula (camp1, camp2, ...)
SELECT campCompatibleCamp1, campCompatibleCamp2, ...
FROM taula(es)
[altres clàusules del SELECT...];
```

Exemple: Suposem que hem creat una taula nova que contindrà les dades dels empleats que són Analistes i que la volem omplir a partir d'una consulta feta a la taula EMPL.

En primer lloc és necessari que la nova taula estigui creada. Suposem que la taula que creem és la següent:

```
CREATE TABLE EMPLEATS_ANALISTES
(
    num    INTEGER PRIMARY KEY,
    nom    VARCHAR(16),
    dept   INTEGER,
    CONSTRAINT FOREIGN KEY(dept) REFERENCES DEPARTAMENTS(dept_num)
)
```

La consulta que ompliria la taula amb tots els analistes de la taula EMPL seria:

```
INSERT INTO EMPLEATS_ANALISTES(num, nom, dept)
SELECT empl_num, empl_nom, empl_dept_num
FROM empl
WHERE ofici='ANALISTA';
```

3.1.3. Modificació de dades

Podem modificar / actualitzar les dades dels registres d'una taula amb la instrucció UPDATE.

Sintaxi:

```
UPDATE taula
SET columna1 = valor1
[,columna2 = valor2...]
[WHERE condició];
```

On:

taula indica la taula on volem fer alguna modificació de les dades dels seus registres.

A partir de la paraula **SET** s'indiquen les assignacions de les noves dades per cada una de les columnes on hem de canviar valors.

Si es vol no cal actualitzar totes les columnes.

La condició WHERE és opcional i permet indicar quines seran les files que es veuran afectades per l'actualització de dades.

Exemple: Actualitzar a Barna el camp dept_loc d'aquells departaments que estan ubicats a Barcelona.

```
mysql> UPDATE dept
      SET dept_loc='Barna'
      WHERE UPPER(dept_loc)='BARCELONA';
```

Exemple: Actualitzar el salari i la comissió dels empleats que guanyen menys de 200.000 amb un 10% més.

```
mysql> UPDATE empleats
      SET empl_salari = empl_salari*1.10
        , empl_comissio = empl_comissio*1.10
      WHERE empl_salari<200000;
```

En el primer **Exemple** s'assigna un nou valor utilitzant una constant de tipus cadena. En el segon **Exemple**, el nou valor que s'assigna s'obté a partir d'una operació de producte.

En general, el nou valor que assignem es pot obtenir de qualsevol forma per complexa que sigui, fins i tot a partir de subconsultes.

Exemple: Assignar als empleats del departament 10 el mateix departament que l'empleat Fernandez.

```
mysql> UPDATE empl
      SET empl_dept_num=(
                        SELECT empl_dept_num
                        FROM empl
                        WHERE UPPER(empl_nom)='FERNANDEZ'
                        )
      WHERE empl_dept_num=10;
```

Aquest tipus d'actualitzacions només són vàlides si la subconsulta retorna un únic valor i a més a més que sigui compatible amb el tipus de dades de la columna a actualitzar.

S'ha de tenir molt en compte que les actualitzacions no es poden saltar les regles d'integritat que tinguin les taules.

Exemple d'error d'integritat. Volem actualitzar el valor del camp emp_dep_num a un valor que no existeix a la taula relacionada:

```
mysql> UPDATE empleats
      SET empl_dept_num=55
      WHERE empl_dept_num=10;
ERROR at line 1: integrity constraint (FK_EMPL_DEPT_NUM) violated-
parent key not found
```

3.1.4. Eliminació de dades

Es realitza mitjançant la instrucció `DELETE`.

<http://dev.mysql.com/doc/refman/8.0/en/delete.html>

Sintaxi:

```
DELETE FROM taula
[WHERE condició]
[ORDER BY nom_col, [nom_col,...]]
[LIMIT num_files]
```

On:

FROM Indiquem el nom de la taula d'on volem esborrar registres. En alguns SGBD aquesta paraula clau es pot obviar.

WHERE Identifiquem quines files s'han d'esborrar. Si no indiquem clàusula `WHERE` s'intentaran esborrar **TOTS** els registres de la taula. Com a condició també podem afegir-hi consultes.

ORDER BY El podem utilitzar per especificar l'ordre en què s'esborren els registres. Normalment l'utilitzarem juntament amb el següent paràmetre (`LIMIT`)

LIMIT El podem utilitzar per limitar el número de registres a esborrar.

Cal tenir en compte que l'eliminació d'un registre no pot provocar error d'integritat si la opció d'integritat `ON DELETE CASCADE` (veure els apunts de DDL, on es parla de la creació de claus foranes). Això pot provocar l'esborrat de no només dels registres de la taula indicada, sinó tots els registres relacionats amb els que s'eliminen.

Exemple: Esborrar els empleats del departament amb identificador 50

```
mysql> DELETE FROM EMPLEATS
      WHERE departament_id =50 AND cog1='Pi';
```

Exemple: Esborrar els empleats del departament 20 que guanyen menys que l'empleat amb el millor sou del departament 30.

```
mysql> DELETE FROM EMPLEATS
      WHERE departament_id = 20 and salari < (
                                                    SELECT MAX(salari)
                                                    FROM EMPLEATS
                                                    WHERE departament_id=30
                                                    );

ERROR 1093 (HY000): You can't specify target table 'EMPLEATS' for
update in FROM clause
```


En el SGBD MySQL no podem esborrar registres utilitzant una subconsulta fent referència a la mateixa taula, però hi ha altres SGBD que sí que ho permeten.

El que sí que podem utilitzar són subconsultes a altres taules. Imagineu que teniu dues taules EMPLEATS I DIRECTIUS que tenen la mateixa estructura.

```
mysql> DELETE FROM EMPLEATS
      WHERE departament_id = 20 AND salari < (
                                           SELECT MAX(salari)
                                           FROM DIRECTIUS
                                           );

Query OK, 1 row affected (0.03 sec)
```

En aquest exemple es pot veure com s'han esborrat els empleats de la taula empleats utilitzant la taula directius. O sigui volem esborrar els empleats del departament 20 que guanyen menys que el sou del millor directiu.

3.2. Consulta de dades sobre una taula

Sens dubte la sentència més utilitzat de llenguatge SQL és la sentència SELECT. Aquesta permet:

- ⊙ Obtenir dades de certes columnes d'una taula (**projecció**).
- ⊙ Obtenir registres (files) d'una taula que compleixen certs criteris (**selecció**).
- ⊙ Combinar dades de taules diferents (**combinació**, join).
- ⊙ Amb aquesta instrucció l'usuari especifica **QUÈ** és el que vol obtenir, **no ON ni COM**.
- ⊙ L'usuari pot realitzar sentències SELECT de diferents nivells de complexitat.
- ⊙ Les dades poden ser representades en un ordre determinat.

3.2.1. Sintaxi

```
SELECT * | {[DISTINCT] columna | expressió, ...}  
FROM taula  
[WHERE condició];
```

On:

- ⊙ SELECT s'utilitza per determinar les columnes que es volen obtenir.
- ⊙ FROM especifica una llista de taules que s'utilitzen a la consulta.
- ⊙ * significa que se seleccionen totes les columnes.
- ⊙ DISTINCT fa que no es mostrin els valors duplicats.
- ⊙ columna. És el nom d'una columna de la taula que es vol mostrar.
- ⊙ expressió. És una expressió SQL vàlida (una operació, una funció de totals, etc.)
- ⊙ WHERE especifica una condició per restringir els registres seleccionats. La clàusula WHERE sempre es col·loca després de la llista de taules de la clàusula FROM.

La condició serà una expressió booleana que s'avaluarà i que permetrà cercar totes les files que la compleixen. Aquesta expressió pot ser molt complexa i es poden utilitzar múltiples operadors en la seva construcció.

Exemple 1

Seleccionar tots els números de departament de la taula DEPT:

```
mysql> SELECT departament_id  
        FROM departaments;  
  
+-----+  
| departament_id |  
+-----+  
|          60 |  
|          50 |  
|          10 |
```

```

|          90 |
|         110 |
|         190 |
|          20 |
|          80 |
+-----+
8 rows in set (0.01 sec)

```

Exemple 2

Seleccionar totes les columnes de la taula DEPT:

```

mysql> SELECT *
        FROM departaments;

+-----+-----+-----+
| departament_id | nom          | localitzacio_id |
+-----+-----+-----+
|          10 | Administració |          1700 |
|          20 | Marketing     |          1800 |
|          50 | Vendes       |          1500 |
|          60 | IT           |          1400 |
|          80 | Compres      |          2500 |
|          90 | Directiu     |          1700 |
|         110 | Contable     |          1700 |
|         190 | Recursos Humans |          1700 |
+-----+-----+-----+
8 rows in set (0.01 sec)

```

Exemple 3

Seleccionar la columna departament_id de la taula empleats.

```

mysql> SELECT departament_id
        FROM departaments;

+-----+
| departament_id |
+-----+
|          NULL |
|          10 |
|          20 |
|          20 |
|          50 |
|          50 |
|          50 |
|          50 |
|          50 |
|          50 |
|          60 |
|          60 |
|          60 |
|          80 |
|          80 |
|          80 |

```

```

|          90 |
|          90 |
|          90 |
|         110 |
|         110 |
|         110 |
+-----+
21 rows in set (0.01 sec)

```

Exemple 4

Seleccionar els diferents departaments on hi ha empleats:

```

mysql> SELECT DISTINCT departament_id
        FROM departaments;

+-----+
| departament_id |
+-----+
|          NULL |
|           10 |
|           20 |
|           50 |
|           60 |
|           80 |
|           90 |
|          110 |
+-----+
8 rows in set (0.00 sec)

```

3.2.2. Ús d'ÀLIES

Quan es consulta una taula, els noms de columnes s'utilitzen com a capçaleres de presentació. Si aquests noms són massa llargs o críptics es poden canviar amb la mateixa sentència SQL de consulta, creant un *ÀLIES*.

Sintaxi

```

SELECT columna alies | columna AS alies
FROM taula

```

Com es pot veure tenim dues maneres de definir un àlies, però es preferible utilitzar la segona.

La utilització de l'àlies només ens serveix per canviar el nom de la columna en el resultat de la consulta, però **no es poden utilitzar en les condicions del WHERE**.

Exemple:

```

mysql> SELECT departament_id AS num_dep
        FROM empleats
        WHERE num_dep = 10;

```

```
ERROR 1054 (42S22): Unknown column 'num_dep' in 'where clause'
```

3.2.3. Càlculs

Aritmètics

Els operadors + (suma), - (resta), * (multiplicació) i / (divisió), es poden utilitzar per fer càlculs a les consultes. Quan s'utilitzen com expressió en una consulta SELECT, no modifiquen les dades originals sinó que com a resultat de l'execució del SELECT, apareix una columna amb el resultat de l'expressió.

La prioritat d'aquests operadors és:

1. tenen més prioritat la multiplicació i divisió,
2. després la suma i la resta.

En cas d'igualtat de prioritat es realitza primer la operació que estigui més a l'esquerra (associativitat per l'esquerra).

Com és lògic, es pot evitar complir aquesta prioritat utilitzant parèntesis; l'interior dels parèntesis és el que s'executa primer.

Quan una expressió aritmètica es calcula sobre valors NULL, el resultat de l'expressió és sempre NULL.

Exemple:

```
mysql> SELECT empleat_id
        ,nom
        ,salari
        ,salari*2
        FROM empleats;
```

empleat_id	nom	salari	salari * 2
100	Steven	24000.00	48000.00
101	Neena	17000.00	34000.00
102	Lex	17000.00	34000.00
103	Alexander	9000.00	18000.00
104	Bruce	6000.00	12000.00
...			
200	Jennifer	4400.00	8800.00
201	Michael	13000.00	26000.00
202	Pat	6000.00	12000.00
205	Shelley	12000.00	24000.00
206	William	8300.00	16600.00
207	Sarah	9000.00	18000.00

```
21 rows in set (0.01 sec)
```

Aquesta consulta obté tres columnes. La tercera té com a títol, l'expressió utilitzada. És en aquests casos quan és més freqüent la utilització d'àlies.

Exemple:

```
mysql> SELECT empleat_id
        ,nom
        ,salari
        ,salari*2 AS doble_salari
        FROM empleats;
```

empleat_id	nom	salari	doble_salari
100	Steven	24000.00	48000.00
101	Neena	17000.00	34000.00
102	Lex	17000.00	34000.00
103	Alexander	9000.00	18000.00
104	Bruce	6000.00	12000.00
107	Diana	4200.00	8400.00
..			
178	Kimberely	7000.00	14000.00
200	Jennifer	4400.00	8800.00
201	Michael	13000.00	26000.00
202	Pat	6000.00	12000.00
205	Shelley	12000.00	24000.00
206	William	8300.00	16600.00
207	Sarah	9000.00	18000.00

21 rows in set (0.01 sec)

Els noms d'àlies poden portar espais si es posen entre cometes dobles:

```
mysql> SELECT empleat_id
        ,nom
        ,salari
        ,salari*2 AS "doble salari"
        FROM empleats;
```

empleat_id	nom	salari	doble salari
100	Steven	24000.00	48000.00
101	Neena	17000.00	34000.00
102	Lex	17000.00	34000.00
103	Alexander	9000.00	18000.00
104	Bruce	6000.00	12000.00
107	Diana	4200.00	8400.00
..			
178	Kimberely	7000.00	14000.00
200	Jennifer	4400.00	8800.00
201	Michael	13000.00	26000.00
202	Pat	6000.00	12000.00
205	Shelley	12000.00	24000.00
206	William	8300.00	16600.00
207	Sarah	9000.00	18000.00

```
21 rows in set (0.01 sec)
```

Concatenació

A diferència d'altres SGBD MySQL no disposa de cap operador per realitzar l'operació de concatenació de dues cadenes de caràcters.

Per tal de realitzar aquesta acció MySQL disposa d'una funció anomenada CONCAT

CONCAT(str1,str2,...)

La funció retorna NULL si algun dels arguments és NULL.

<http://dev.mysql.com/doc/refman/5.6/en/string-functions.html>

Exemple:

```
mysql> SELECT CONCAT(empleat_id, ' - ', nom) AS "Numero i Nom"
        FROM empleats;
```

```
+-----+
| Numero i Nom |
+-----+
| 100 - Steven |
| 101 - Neena  |
| 102 - Lex    |
| 103 - Alexander |
| 104 - Bruce  |
| 107 - Diana  |
| 124 - Kevin  |
| 141 - Tenna  |
| 142 - Curtis |
| 143 - Randall |
| 144 - Peter  |
| 149 - Eleni  |
| 174 - Ellen  |
| 176 - Jonathan |
| 178 - Kimberly |
| 200 - Jennifer |
| 201 - Michael |
| 202 - Pat     |
| 205 - Shelley |
| 206 - William |
| 207 - Sarah   |
+-----+
21 rows in set (0.01 sec)
```

3.2.4. Condicions de selecció

Es poden realitzar consultes que filtrin les dades de sortida de les taules. Per això s'utilitza la clàusula `WHERE`. Aquesta clàusula permet col·locar una condició que han de complir tots els registres, els que no la compleixin no apareixeran en el resultat.

Exemple: Seleccionar el número, nom i ofici dels empleats.

```
mysql> SELECT empleat_id
        ,nom
        ,feina_codi
        FROM empleats;
```

empleat_id	nom	feina_codi
100	Steven	AD_PRES
101	Neena	AD_VP
102	Lex	AD_VP
103	Alexander	IT_PROG
104	Bruce	IT_PROG
107	Diana	IT_PROG
...		
200	Jennifer	AD_ASST
201	Michael	MK_MAN
202	Pat	MK_REP
205	Shelley	AC_MGR
206	William	AC_ACCOUNT
207	Sarah	AC_ACCOUNT

```
21 rows in set (0.00 sec)
```

Exemple: Seleccionar el número, nom i ofici dels empleats que són `IT_PROG`.

```
mysql> SELECT empleat_id
        ,nom
        ,feina_codi
        FROM empleats
        WHERE feina_codi = 'IT_PROG';
```

empleat_id	nom	feina_codi
103	Alexander	IT_PROG
104	Bruce	IT_PROG
107	Diana	IT_PROG

```
3 rows in set (0.00 sec)
```

O també

```
mysql> SELECT empleat_id
```



```

        , nom
        , feina_codi
    FROM empleats
    WHERE feina_codi = "IT_PROG";

```

empleat_id	nom	feina_codi
103	Alexander	IT_PROG
104	Bruce	IT_PROG
107	Diana	IT_PROG

3 rows in set (0.00 sec)

Operadors de comparació

Funcions i operadors de comparació:

<http://dev.mysql.com/doc/refman/8.0/en/comparison-operators.html>

Es poden utilitzar en la clàusula `WHERE`. Són:

Operador	Significat
>	Major que
<	Menor que
>=	Major o igual que
<=	Menor o igual que
=	Igual
<> , !=	Diferent

Es poden utilitzar tant per comparar números com per comparar textos i dates. En el cas dels textos, les comparacions es fan en ordre alfabètic.

Operador *BETWEEN*

Permet obtenir dades que es trobin (o no es trobin) en un rang de dades (valor inferior, valor superior).

Sintaxi

```
expressió1 [NOT] BETWEEN expMin AND expMax
```

on:

○ *expressió1* és una expressió avaluable de la que es vol comprovar si està dins els rang definit entre *expMin* i *expMax*.

⊙ [NOT] és opcional. S'utilitza per indicar que es vol seleccionar tots els registres que no tenen valors entre *expMin* i *expMax*.

⊙ *expMin* és una expressió avaluable que indica el límit inferior del rang de valors.

⊙ *expMax* és una expressió avaluable que indica el límit superior del rang de valors.

Exemple: Obtenir tots els empleats que tenen un salari entre 10.000 i 20.000.

```
mysql> SELECT empleat_id
        ,nom
        ,salari
        FROM empleats
        WHERE salari BETWEEN 10000 AND 20000;
```

empleat_id	nom	salari
101	Neena	17000.00
102	Lex	17000.00
149	Eleni	10500.00
174	Ellen	11000.00
201	Michael	13000.00
205	Shelley	12000.00

6 rows in set (0.00 sec)

Exemple: Obtenir tots els empleats que no tenen un salari entre 10.000 i 20.000.

```
mysql> SELECT empleat_id
        ,nom
        ,salari
        FROM empleats
        WHERE salari NOT BETWEEN 10000 AND 20000;
```

empleat_id	nom	salari
100	Steven	24000.00
103	Alexander	9000.00
104	Bruce	6000.00
...		
202	Pat	6000.00
206	William	8300.00
207	Sarah	9000.00

35 rows selected.

Pregunta: Com podries realitzar els dos exemples anteriors sense utilitzar la sentència BETWEEN?

Operador IN

Permet obtenir registres que tenen (o no tenen) valors dins una llista.

Sintaxi

`expressió1 [NOT] IN (llistaValors)`

on:

- ⊙ *expressió1* és una expressió avaluable de la que es vol comprovar si està dins la *llistaValors*.
- ⊙ [NOT] és opcional. S'utilitza per indicar que es vol seleccionar tots els registres que no tenen valors dins la *llistaValors*.
- ⊙ *llistaValors* és una llista de valors.

Exemple: Obtenir els empleats que són president o programador.

```
mysql> SELECT empleat_id
        ,nom
        ,feina_codi
        FROM empleats
        WHERE feina_codi IN ('AD_PRES','IT_PROG');
```

empleat_id	nom	feina_codi
100	Steven	AD_PRES
103	Alexander	IT_PROG
104	Bruce	IT_PROG
107	Diana	IT_PROG

Exemple: Obtenir els empleats que no són president ni programador.

```
mysql> SELECT empleat_id
        ,nom
        ,feina_codi
        FROM empleats
        WHERE feina_codi NOT IN ('AD_PRES','IT_PROG');
```

empleat_id	nom	feina_codi
------------	-----	------------

	101	Neena	AD_VP	
	102	Lex	AD_VP	
	124	Kevin	ST_MAN	
	141	Trena	ST_CLERK	
	142	Curtis	ST_CLERK	
	143	Randall	ST_CLERK	
	144	Peter	ST_CLERK	
	149	Eleni	SA_MAN	
	174	Ellen	SA_REP	
	176	Jonathan	SA_REP	
	178	Kimberely	SA_REP	
	200	Jennifer	AD_ASST	
	201	Michael	MK_MAN	
	202	Pat	MK_REP	
	205	Shelley	AC_MGR	
	206	William	AC_ACCOUNT	
	207	Sarah	AC_ACCOUNT	
+-----+-----+-----+-----+				

Com rescriuries IN o el NOT IN amb operadors lògics? Mirar Àlgebra de Bool (Llei de Morgan)

Operador IS NULL

Retorna CERT si una expressió conté un valor NULL.

Sintaxi:

<i>expressió1</i> IS [NOT] NULL

on:

- ⊙ *expressió1* és una expressió avaluable de la que es vol comprovar si el seu valor resultant és NULL.
- ⊙ [NOT] és opcional. S'utilitza per indicar que es vol comprovar si el valor de l'expressió no és NULL.

També es pot utilitzar la funció *ISNULL(exp)*

Exemple: Seleccionar el número, el nom i la comissió dels empleats que tenen el valor NULL al camp `pct_comissio`.

```
mysql> SELECT empleat_id, nom, pct_comissio
        FROM empleats
        WHERE pct_comissio IS NULL;
```

+-----+-----+-----+-----+			
empleat_id	nom		pct_comissio
+-----+-----+-----+-----+			
	100	Steven	NULL
	101	Neena	NULL
	102	Lex	NULL
	103	Alexander	NULL
	104	Bruce	NULL

	107	Diana		NULL	
	124	Kevin		NULL	
	141	Trena		NULL	
	142	Curtis		NULL	
	143	Randall		NULL	
	144	Peter		NULL	
	200	Jennifer		NULL	
	201	Michael		NULL	
	202	Pat		NULL	
	205	Shelley		NULL	
	206	William		NULL	
	207	Sarah		NULL	
+-----+-----+-----+					

Exemple: Seleccionar el número, el nom i la comissió dels empleats en tenen. És a dir no tenen el valor NULL al camp pct_comissio.

```
mysql> SELECT empleat_id, nom, pct_comissio
        FROM empleats
        WHERE pct_comissio IS NULL;
```

+-----+-----+-----+					
	empleat_id		nom		pct_comissio
	+-----+				
	149		Eleni		0.20
	174		Ellen		0.30
	176		Jonathan		0.20
	178		Kimberely		0.15
+-----+-----+-----+					

Operador LIKE

S'utilitza sobretot amb textos. Permet obtenir registres que tenen alguna columna que es vol que compleixi una condició textual.

Sintaxi

expressió1 [NOT] LIKE cadena

on:

- ⊙ *expressió1* és una expressió avaluable de la que es vol comprovar si és com cadena.
- ⊙ [NOT] és opcional. S'utilitza per indicar que es vol seleccionar tots els registres que no tenen valors que compleixen cadena.
- ⊙ *cadena* és un valor textual. Aquest valor textual pot incloure dos símbols especials:

Operador	Significat
----------	------------

%	Indica que on hi hagi aquest símbol s'interpretarà com si hi hagués 0 o més caràcters qualsevols en aquella posició.
—	Indica que on hi hagi aquest símbol s'interpretarà com un sol caràcter qualsevol en aquella posició

NOTA: Per defecte i en funció com tenim definim el COLLATE quan es fan comparacions amb cadenes textuais la diferència entre minúscules i majúscules **no** és significativa.

Tot i que es podem tornar significativa mitjançant l'ordre COLLATE :
<http://dev.mysql.com/doc/refman/8.0/en/case-sensitivity.html>)

Pots mirar els diferents COLLATE disponibles amb *CASE SENSITIVE* realitzant la consulta següent:

```
SHOW COLLATION WHERE COLLATION LIKE "%_cs"
```

Exemple: Obtenir tots els empleats que el seu nom acabi amb a.

```
mysql> SELECT empleat_id, nom
      FROM empleats
      WHERE nom like '%A';
+-----+-----+
| empleat_id | nom    |
+-----+-----+
|         101 | Neena  |
|         107 | Diana  |
|         141 | Trena  |
+-----+-----+
3 rows in set (0.01 sec)

mysql> SELECT empleat_id, nom
      FROM empleats
      WHERE nom like '%A' COLLATE utf8_bin;

Empty set (0.00 sec)
```

Exemple: Obtenir tots els empleats que el seu nom tingui només 3 caràcters i comenci amb LE.

```
mysql> SELECT empleat_id, nom
      FROM empleats
      WHERE nom like 'le_';
+-----+-----+
| empleat_id | nom    |
+-----+-----+
|         102 | Lex    |
+-----+-----+
```

```
+-----+-----+
```

Funció REGEXP

Una de les eines més potents per tractar textos són les expressions regulars.

MySQL utilitza la implementació Henry Spencer d'expressions regulars i compleix amb POSIX 1003.2.

<http://dev.mysql.com/doc/refman/8.0/en/regexp.html>

<http://dev.mysql.com/doc/refman/8.0/en/pattern-matching.html>

Sintaxi

expressió **REGEXP** patró, expressió **RLIKE** patró

expressió **NOT REGEXP** patró, expressió **NOT RLIKE** patró

on:

- *expressió1* és una expressió avaluable de la que es vol comprovar si és com patró.

- *Patró* és una cadena que conté la cadena de caràcters que forma el patró de cerca. El patró ha de ser un **string literal**, **no pot ser una funció/expressió**

Retorna 1 (True) si l'expressió coincideix (*fa match*) amb el patró, altrament retorna 0 (False)

Si l'expressió o el patró són NULL, el resultat serà NULL.

Exemples:

```
mysql> SELECT 'Monty!' REGEXP '.*' as resultat;
-> 1

mysql> SELECT 'a' REGEXP '^[a-d]' as resultat;
-> 1

mysql> SELECT empleat_id, nom
      FROM empleats
      where nom REGEXP '^[a-c]';
+-----+-----+
| empleat_id | nom      |
+-----+-----+
|          103 | Alexander |
|          104 | Bruce    |
|          142 | Curtis   |
+-----+-----+
```

Compte amb els rangs del patró. Encara que REGEXP no sigui case sensitive per defecte

```
mysql> SELECT empleat_id, nom
        FROM empleats
        where nom REGEXP '^[a-C]';

ERROR 1139 (42000): Got error 'invalid character range' from regexp
```

Compte també amb el COLLATE definit a la base de dades ja que no funcionaran `[:upper:]` i `[:lower:]`
Per exemple en un `utf8_collate_ci`

```
mysql> SELECT 'WORD' REGEXP '[:lower:]{4}' as resultat;
-> 1 (true)

mysql> SELECT 'WORD' REGEXP '[:lower:]{4}' as resultat;
-> 1 (true)

mysql> SELECT 'WORD' REGEXP '[:lower:]{4}' COLLATION utf8_bin as
resultat;
-> 0 (false)
```

Veure la pàgina de la documentació oficial per veure els diferents operadors que podem col·locar com a patró de cerca. <http://dev.mysql.com/doc/refman/8.0/en/regexp.html>

Operadors lògics

<http://dev.mysql.com/doc/refman/8.0/en/logical-operators.html>

Operador	Significat
AND, &&	Retorna CERT si les expressions de la seva esquerra i dreta són les dues certes.
OR,	Retorna CERT si qualsevol de les expressions de la seva esquerra i dreta són certes.
NOT, !	Inverteix la lògica de l'expressió que està a la seva dreta. Si era cert, mitjançant NOT passa a ser fals.
XOR	Retorna CERT quan les dues expressions no tenen el mateix valor. Per exemple: CERT XOR CERT = FALS CERT XOR FALS = CERT FALS XOR CERT = CERT FALS XOR FALS = FALS

En SQL tots els operadors lògic avaluats poden donar CERT(true), FALS(false), o NULL (null).

MySQL, implementa els valors lògics com 1 (CERT), 0 (FALS), i NULL, encara que podem utilitzar TRUE,FALSE i NULL en les nostres expressions.

```
mysql> SELECT false or false AS resultat;
+-----+
| resultat |
+-----+
|          0 |
+-----+

mysql> SELECT false or true AS resultat;
+-----+
| resultat |
+-----+
|          1 |
+-----+
```

Exemple: Obtenir els empleats que són president o programador.

```
mysql> SELECT empleat_id
        , nom
        , feina_codi
      FROM empleats
     WHERE feina_codi='AD_PRES' OR feina_codi ='IT_PROG';

+-----+-----+-----+
| empleat_id | nom       | feina_codi |
+-----+-----+-----+
|          100 | Steven    | AD_PRES    |
|          103 | Alexander | IT_PROG    |
|          104 | Bruce     | IT_PROG    |
|          107 | Diana     | IT_PROG    |
+-----+-----+-----+
```

Exemple: Obtenir els empleats que són director i programador.

```
mysql> SELECT empleat_id
        , nom
        , feina_codi
      FROM empleats
     WHERE feina_codi='AD_PRES' AND feina_codi ='IT_PROG';

Empty set (0.00 sec)
```

Mira't Àlgebra de Bool i Llei de Morgan

Precedència d'operadors

De vegades les expressions que podem arribar a crear en instruccions de selecció són molt extenses i és difícil saber quina part de l'expressió és la que s'avalua primer.

<http://dev.mysql.com/doc/refman/8.0/en/operator-precedence.html>

En la següent taula s'indica la precedència dels operadors:

Operador
^
*, /, DIV, %, MOD
+ -
=, >=, <, <=, <, <>, !=, IS, LIKE, IN
BETWEEN, CASE, WHEN, THEN, ELSE
NOT
AND, &&
XOR

OR, ||

3.2.5. Ordenació

L'ordre dels registres obtinguts per una instrucció **SELECT** és imprevisible, però normalment en tots els SGBD té relació amb l'ordre en que es van introduir les dades.

Per tal d'ordenar en base a algun criteri, s'utilitza la clàusula **ORDER BY**.

En aquesta clàusula es col·loca una llista de camps que indica la forma d'ordenar. S'ordena primer pel primer camp de la llista, si hi ha coincidències pel segon, si també hi ha coincidències pel tercer, i així successivament.

Es poden col·locar les paraules **ASC** o **DESC** (per defecte sempre és **ASC**). Aquestes paraules indiquen l'ordenació que es vol fer (**ASC**endent o **DESC**endent, respectivament).

Sintaxi completa de la instrucció **SELECT**

```
SELECT expressions
FROM taula
[WHERE condició]
[ORDER BY expressionsAOrdenar1 {ASC|DESC}];
```

Exemple: Obtenir l'ofici, nom i salari dels empleats que tenen un salari superior a 10.000, ordenat ascendentment per ofici i nom.

```
mysql> SELECT feina_codi
        ,nom
        ,departament_id
        ,salari
        FROM empleats
        WHERE salari > 10000
        ORDER BY feina_codi, nom ASC;
```

feina_codi	nom	departament_id	salari
AC_MGR	Shelley	110	12000.00
AD_PRES	Steven	90	24000.00
AD_VP	Lex	90	17000.00
AD_VP	Neena	90	17000.00
MK_MAN	Michael	20	13000.00
SA_MAN	Eleni	80	10500.00
SA_REP	Ellen	80	11000.00

¹ Per indicar quines expressions es volen ordenar es pot fer indicant l'expressió o bé el número d'expressió tenint en compte l'ordre en què han estat escrites. Exemple: ...**ORDER BY** empl_nom; o bé ... **ORDER BY** 1;

Exemple: Obtenir l'ofici, nom i salari dels empleats que tenen un salari superior a 10.000, ordenat ascendentment per ofici i descendentment per nom.

```
mysql> SELECT  empl_ofici, empl_nom, empl_dept_num
              FROM empl
              WHERE empl_salari > 200000
              ORDER BY feina_codi, empl_nom DESC;
```

feina_codi	nom	departament_id	salari
AC_MGR	Shelley	110	12000.00
AD_PRES	Steven	90	24000.00
AD_VP	Neena	90	17000.00
AD_VP	Lex	90	17000.00
MK_MAN	Michael	20	13000.00
SA_MAN	Eleni	80	10500.00
SA_REP	Ellen	80	11000.00

7 rows selected.

Important: Fixa't en els empleats Neena i Lex. S'han intercanviat.

3.2.6. Funcions

SQL no incorpora funcions específiques pel tractament dels diferents tipus de dades. Aquestes funcions doncs, són pròpies de cada SGBD.

Aquí veurem algunes de les funcions més significatives del SGBD MySQL.

Totes les funcions reben dades per poder operar (paràmetres) i retornen un resultat (que depèn dels paràmetres passats a la funció).

Els paràmetres es passen entre parèntesi:

nomFunció [(paràmetre1 [, paràmetre2,...])]

Funcions de caràcters

<https://dev.mysql.com/doc/refman/8.0/en/string-functions.html>

Conversió de text a majúscules i minúscules

Funció	Descripció
CONCAT(text,text,...)	Retorna un string concatenant els valors passats per paràmetre
LENGTH(text)	Retorna la longitud del string text
LOWER(text) / LCASE(text)	Retorna el string text a minúscules.
UPPER(text) / UCASE(text)	Retorna el string text a majúscules.
LEFT(text,len)	Retorna els <i>len</i> primers caràcters del string text començant per l'esquerra
RIGHT(text,len)	Retorna els <i>len</i> primers caràcters del string text començant per la dreta
MID(text, pos, len) / SUBSTRING(text, pos) SUBSTRING(text, pos, len) SUBSTR == SUBSTRING	S'utilitza per obtenir porcions d'una cadena de text. Si el paràmetre pos és negatiu indicarà la posició en sentit dreta-esquerra.
INSTR(text, sub_text)	Retorna la posició de la primera aparició de sub_text dins a text
LOCATE(sub_text, text), LOCATE(sub_text, text, pos)	Igual que la funció anterior, però amb l'ordre dels paràmetres al revés. A més podem indicar la posició a iniciar la cerca.
LPAD(text, len, padtext)	Retorna el string <i>text</i> omplint-lo amb el text <i>padtext</i> per l'esquerra fins obtenir la longitud <i>len</i> . Si <i>len</i> és inferior a la longitud del text original el modifica.
RPAD(text,len,padtext)	Retorna el string <i>text</i> omplint-lo amb el text <i>padtext</i> per la dreta fins obtenir la longitud <i>len</i> . Si <i>len</i> és inferior a la longitud del text original el modifica.
TRIM(text) LTRIM(text)	Retorna el text traient els possibles espais en blanc d'esquerra i dreta.

RTRIM(text)	
REPEAT(text, count)	Retorna un string de longitud count repetint el string text.
SPACE(len)	Retorna un string de longitud <i>len</i> només format per espais en blanc
REPLACE(text, from, to)	Retorna el string <i>text</i> reemplaçant les ocurrences del string <i>from</i> per el string <i>to</i> .
REVERSE(text)	Retorna string text amb l'orde dels caràcters en format invers
FORMAT(numero, D, [locale]) Locale info	Formata un decimal arrodonint a D decimals i el retorna com un string. Amb el tercer paràmetre podem definir la localització Com es mostren els milers, el caràcter de decimal,...
ELT(índex, sstring, string,...)	Donat un índex i una sèrie de strings retorna l'string situat a la posició indicat pel paràmetre índex.

Podem combinar funcions per crear noves funcionalitats que no ens donen. Per exemple en MySQL no existeix la funció INITCAP (text): retorna el text amb la primera lletra en majúscules, però podem combinar-ne per obtenir el mateix resultat:

```
CONCAT(UPPER(SUBSTRING(x,1,1)),LOWER(SUBSTRING(x,2)) );
```

Exemples:

```
mysql> SELECT CONCAT ("Hola", ' ', "món") AS resultat;
+-----+
| resultat |
+-----+
| Hola món |
+-----+

mysql> SELECT LENGTH('hola món') as LONGITUD;
+-----+
| LONGITUD |
+-----+
|          9 |
+-----+

mysql> SELECT UPPER('Hola Món') AS resultat;
+-----+
| resultat |
+-----+
| HOLA MÓN |
+-----+

mysql> SELECT RTRIM('hola món      ') AS resultat;
+-----+
| resultat |
+-----+
| hola món |
+-----+
```

```
mysql> SELECT LTRIM('    hola món') as missatge;
+-----+
| missatge |
+-----+
| hola món |
+-----+

mysql> SELECT TRIM('    hola món    ') as missatge;
+-----+
| missatge |
+-----+
| hola món |
+-----+

mysql> SELECT SUBSTR('hola món',6,3) as missatge;
+-----+
| missatge |
+-----+
| món      |
+-----+

mysql> SELECT SUBSTR('hola món',6) as missatge;
+-----+
| missatge |
+-----+
| món      |
+-----+

mysql> SELECT INSTR('hola món','món') as posicio;
+-----+
| posicio |
+-----+
|        6 |
+-----+

mysql> SELECT REPLACE('hola món','món','a tots') as canvi;
+-----+
| canvi    |
+-----+
| hola a tots |
+-----+

mysql> SELECT LPAD('HOLA MON',10,'*') AS NOM;
+-----+
| NOM      |
+-----+
| **HOLA MON |
+-----+

mysql> SELECT RPAD('HOLA MON',10,'*') AS NOM;
+-----+
| NOM      |
+-----+
| HOLA MON** |
+-----+

mysql> SELECT RPAD(nom,10,'*') AS nom
        FROM empleats;
```

```

+-----+
| nom    |
+-----+
| Steven**** |
| Neena***** |
| Lex***** |
| Alexander* |
| Bruce***** |
| . . . . |
| Jonathan** |
| Kimberely* |
| Jennifer** |
| Michael*** |
| Pat***** |
| Shelley*** |
| William*** |
| Sarah***** |
+-----+

```

Podem utilitzar combinacions de funcions per tal de crear-ne de noves. Per exemple MySQL no incorpora una funció INITCAP (retorna string en minúscules excepte la primera lletra amb majúscules). Combinant les funcions anteriors:

```
CONCAT( UPPER(SUBSTR(x,1,1)), LOWER(SUBSTR(x,2)) )
```

```

mysql> SELECT CONCAT( UPPER(SUBSTR('hola món',1,1)),
LOWER(SUBSTR('hola món',2)) ) AS initcap;
+-----+
| initcap |
+-----+
| Hola món |
+-----+

mysql> SELECT CONCAT(UPPER(SUBSTR(nom,1,1)),LOWER(SUBSTR(nom,2)) ) AS nom
FROM empleats;
+-----+
| nom    |
+-----+
| Steven |
| Neena  |
| Lex    |
| ...    |
| Kimberely |
| Jennifer |
| Michael |
| Pat     |
| Shelley |
| William |
| Sarah   |
+-----+

```


Funcions numèriques

<https://dev.mysql.com/doc/refman/8.0/en/numeric-functions.html>

Arrodoniments

Funció	Descripció
ROUND(n,[decimals])	Arrodona el número n al següent número amb el nombre de decimals indicat més proper. Ex: ROUND(8.239, 2) → 8.24
TRUNCATE(n,decimals)	Retorna el numero n només amb el nombre de decimals indicat. Ex: TRUNCATE(8.239, 2) → 8.23
FLOOR(n)	Obté l'enter immediatament més petit o igual que n. Ex: FLOOR(8.6) → 8
CEIL(n)	Obté l'enter immediatament més gran o igual que n. Ex: CEIL(8.3) → 9

Matemàtiques

Funció	Descripció
MOD(m,n) / N % M / N MOD M	Retorna el residu de dividir m entre n. Ex: MOD(7,5) → 2 7 % 5 → 2 7 MOD 5 → 2
POWER(m,n)	Retorna el numero m elevat a la potència n. Ex: POWER(2,3) → 8
SQRT(n)	Obté l'arrel quadrada de n. Ex: SQRT(2) → 1.41421356
SIGN(n)	Retorna -1 si n és negatiu; 0 si n és 0 i 1 si n és positiu. Ex: SIGN(-2.3) → -1 SIGN(0) → 0 SIGN(2.3) → 1
ABS(n)	Calcula el valor absolut de n. Ex: ABS(-2.3) → 2.3
EXP(n)	Calcula e ⁿ , és a dir, l'exponent amb base e del número n. Ex: EXP(2) → 7.3890561
LOG(m,n)	Calcula el logaritme amb base m de n. Ex: LOG(10,100) → 2
LOG2(n)	Calcula el logaritme amb base 2 de n. EX: LOG2(65536) → 16
LOG10(n)	Calcula el logaritme amb base 10 de n EX: LOG10(100) → 2
PI()	Retorna el valor de PI
SIN(n)	Calcula el sinus de n (n ha d'estar amb radiants) Ex: SIN(0) → 0
COS(n)	Calcula el cosinus de n (n ha d'estar amb radiants) Ex: COS(0) → 1
TAN(n)	Calcula la tangent de n (n ha d'estar amb radiants)

	Ex: TAN(0) → 0
--	----------------

Altres

Funció	Descripció
RAND([n])	Retorna un número en punt flotant entre 0 i 1.0 El paràmetre n és un enter que el podem utilitzar com a llavor per obtenir sempre els mateixos valors. Ex: RAND() → 0.61914388706828

Funcions per treballar amb dates

<https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html>

<http://dev.mysql.com/doc/refman/8.0/en/date-calculations.html>

Les dates s'utilitzen molt en totes les bases de dades.

MySQL proporciona cinc tipus de dades per gestionar dates, els tipus DATE, TIME, DATETIME, TIMESTAMP i YEAR.

- DATE: Una data concreta. Dia mes i any. Amb format 'YYYY-MM-DD'
- TIME: Valor d'una hora en format hh:mm:ss
- DATETIME: Una data i hora concrets. Dia mes i any i l'hora. Amb format 'YYYY-MM-DD hh:mm:ss'
- TIMESTAMP: Valor d'un instant de temps que pot contenir fraccions de segon.
- YEAR: Valor d'un any amb format de 'YYYY';

Cal tenir en compte que als valors de tipus data se'ls hi pot sumar números i s'entendria que aquesta suma és de dies. En general es pot operar amb números (sumes, restes, multiplicacions i divisions). Si té decimals llavors se sumen dies, hores, minuts i segons.

Funció	Descripció
CURDATE() CURRENT_DATE() CURRENT_DATE	Retorna la data actual en format 'YYYY-MM-DD' o bé 'YYYYMMDD' depenent del context. Ex: SELECT CURDATE() → '2016-08-25' SELECT CURDATE()+1 → 20160826
CURTIME() CURRENT_TIME() CURRENT_TIME	Retorna l'hora actual en format 'HH:MM:SS' o bé 'HHMMSS' depenent del context. Ex: SELECT CURTIME() → '18:41:30' SELECT CURTIME()+1 → 184131
NOW() CURRENT_TIMESTAMP() CURRENT_TIMESTAMP	Obté la data i hora actuals de manera constant a quan la sentència s'ha començat a executar. EX: SELECT NOW(), SLEEP(2), NOW();
SYSDATE()	Retorna la data i hora actuals per cada vegada que s'executa la funció. Mira la diferència amb l'exemple entre NOW();

UNIX_TIMESTAMP()	Retorna el Unix timestamp. El número de segons que han passat des del '1970-01-01 00:00:00' UTC
------------------	-------------------------------------------------------------------------------------------------

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+-----+-----+-----+
| NOW()          | SLEEP(2) | NOW()          |
+-----+-----+-----+
| 2006-04-12 13:47:36 |          0 | 2006-04-12 13:47:36 |
+-----+-----+-----+
mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+-----+-----+-----+
| SYSDATE()      | SLEEP(2) | SYSDATE()      |
+-----+-----+-----+
| 2006-04-12 13:47:44 |          0 | 2006-04-12 13:47:46 |
+-----+-----+-----+
```

Calcular dates

Funció	Descripció
DATE_ADD(data, INTERVAL expr unit)	Afegeix un interval de temps expressat amb el paràmetre <i>expr</i> i amb el tipus d'unitats expressat amb el paràmetre <i>unit</i> . Mirar documentació Ex: SELECT NOW(), DATE_ADD(NOW(), interval 1 MONTH) → 2016-08-01, 2016-09-01
DATE_SUB(data, INTERVAL expr unit)	Mateix funcionament que DATE_ADD, però en aquest cas sostreu l'interval indicat.
ADDDATE(data, INTERVAL expr unit) ADDDATE(data,dies)	Mateix funcionament que DATE_ADD o bé si només volem afegir-hi dies podem utilitzar la sobrecàrrega de la funció.
DATEDIFF(data1,data2) <i>Compte amb l'ordre dels paràmetres: resta data1-data2</i>	Realitza la resta entra dues dates i retorna el resultat en número de dies.
TIMESTAMPDIFF(unit,data1,data2) <i>Compte amb l'ordre dels paràmetres: resta data2-data1</i>	Realitza la diferència entre dos DATETIMES. El número retornat és un enter amb les unitats expressades en el paràmetre UNITAT (YEAR, MONTH,MINUTE,...)
EXTRACT(unit FROM data)	Extreu un valor d'una data. El valor pot ser DAY (dia), MONTH (mes), YEAR (any), etc... La data s'ha d'indicar com a 'yyyy-mm-dd'.
DAY(data) DAYOFMONTH(data)	Retorna el dia del mes donada una data.

	Ex: DAY('2016-08-06') → 6
DAYOFWEEK (data)	Retorna el dia de la setmana donada una data. (1 = Diumenge, 2 = Dilluns, ..., 7 = Dissabte) Ex: DAYOFWEEK('2016-08-06') → 7
WEEKDAY (data)	Retorna el dia de la setmana donada una data. (0 = Dilluns, 1 = Dimarts, ..., 6 = Diumenge) Ex: DAYOFWEEK('2016-08-06') → 6
DAYOFYEAR(data)	Retorna el dia de L'any entre 1 i 366.

Exemples:

```
mysql> SELECT DATEDIFF('2016-08-01','2016-08-01') resta;
+-----+
| resta |
+-----+
|      0 |
+-----+

mysql> SELECT DATEDIFF('2016-08-31','2016-08-01') resta;
+-----+
| resta |
+-----+
|     30 |
+-----+

mysql> SELECT TIMESTAMPDIFF(DAY,'2016-08-31','2016-08-01') AS resta;
+-----+
| resta |
+-----+
|    -30 |
+-----+

mysql> SELECT TIMESTAMPDIFF(MINUTE,'2016-08-31','2016-08-01') AS
resta;
+-----+
| resta |
+-----+
| -43200 |
+-----+
```

Locale Support

<https://dev.mysql.com/doc/refman/8.0/en/locale-support.html>

Mitjançant la variable de sessió `lc_time_names` podem modificar la configuració regional de la nostra connexió i mostrar els formats decimals, noms de dies de la setmana, etc.. amb la configuració regional més propera a la nostra.

En aquest exemple veiem que la configuració local és **en_US**:

```
mysql> SELECT @@lc_time_names;
+-----+
| @@lc_time_names |
+-----+
| en_US           |
+-----+

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----+-----+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----+-----+
| Friday                | January                  |
+-----+-----+

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+-----+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+-----+
| Friday Fri January Jan                  |
+-----+
```

Si canviem aquest configuració local a **ca_ES**:

```
mysql> SET lc_time_names = 'ca_ES';

mysql> SELECT @@lc_time_names;
+-----+
| @@lc_time_names |
+-----+
| ca_ES           |
+-----+

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----+-----+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----+-----+
| divendres              | gener                    |
+-----+-----+

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+-----+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+-----+
| divendres dv gener gen                  |
+-----+
```


Funcions de control de flux i treballar valors NULL

Permeten definir els valors a utilitzar en el cas que les expressions agafin el valor NULL.

Funció	Descripció
IFNULL(expr1,expr2)	Si expr1 és un valor NULL, retorna l'expressió 2, altrament retorna expr1 Ex: IFNULL(234, 5) → 234 IFNULL(NULL, 6) → 6
NULLIF(expr1,expr2)	Retorna NULL si expr1 = expr2; altrament retorna expr1 Ex: NULLIF(2,2) → NULL

CASE

Instrucció que permet establir condicions de sortida a l'estil CASE, SWITCH d'altres llenguatges.

Sintaxi (I)

```
CASE expressió WHEN valor1 THEN resultat1
[ WHEN valor2 THEN resultat2 ....
...
ELSE resultatElse
]
END
```

Aquesta versió retorna el resultat quan expressió == valor.

El funcionament és el següent:

S'avalua l'expressió indicada.

Es comprova si aquesta expressió és igual al valor del primer WHEN. Si és així, retorna el primer resultat (qualsevol valor menys NULL).

Si l'expressió no és igual al valor1, llavors es comprova si és igual que el segon i així successivament fins a trobar ELSE que només s'escriu si l'expressió no val cap valor dels indicats.

```
mysql> SELECT CASE 1+1
      WHEN 1 THEN 'one'
      WHEN 2 THEN 'two'
      ELSE 'more'
      END AS resultat;

+-----+
| resultat |
+-----+
| two      |
+-----+
```

Sintaxi (II)

```

CASE WHEN condicio1 THEN resultat1
[ WHEN condicio2 THEN resultat2 ....
...
ELSE resultatElse
]
END

```

Aquesta versió retorna el resultat de la primer condició que es compleix començant per la primera. Si no troba cap condició que es compleix retorna resultat de l'apartat ELSE, si no hi ha apartat de ELSE retorn NULL.

```

mysql> SELECT CASE WHEN 1>0 THEN 'true-1'
                  WHEN 2>0 THEN 'true-2'
                  ELSE 'impossible'
                  END AS resultat;

+-----+
| resultat |
+-----+
| true-1   |
+-----+

```

IF

Funció per permet establir condicions de sortida a l'estil d'un IF d'un llenguatge de programació.

Sintaxi (II)

```
IF(expr1,expr2,expr3)
```

Comprova la expr1 si aquesta es certa retorna expr2, altrament retorna expr3. Equivalència amb un llenguatge de de programació:

```

//JAVA
if(expr1){
    expr2
}
else {
    expr3
}

```

```

mysql> SELECT IF(1>0,'cert','fals') AS resultat;

+-----+
| resultat |
+-----+
| cert     |
+-----+

```


Funcions de conversió

MySQL, com molts llenguatges de programació, permet realitzar una sèrie de conversions implícites entre els diferents tipus de dades:

Per assignacions, MySQL automàticament pot convertir:

- VARCHAR o CHAR → NUMBER
- VARCHAR o CHAR → DATE
- NUMBER → VARCHAR o CHAR
- DATE → VARCHAR

Per avaluar una expressió MySQL pot convertir automàticament:

- VARCHAR o CHAR → NUMBER
- VARCHAR o CHAR → DATE

De totes maneres, MySQL disposa de dues funcions per realitzar conversions de tipus de dades de manera explícita.

CAST i CONVERT

Sintaxi:

CAST(expressió **as** tipus)

CONVERT(expressió, tipus)

Tipus només pot ser: BINARY, CHAR, DATE, DATETIME, DECIMAL, SIGNED|UNSIGNED [INTEGER], TIME

Exemples:

```
SELECT CAST (123 AS CHAR);

SELECT CONVERT (123, CHAR);

SELECT CONVERT(salari, CHAR)
  FROM empleats;
```

3.2.7. Agrupacions i Agregats

És molt habitual crear consultes en les que es vol agrupar les dades per tal de realitzar càlculs en vertical, és a dir, calculats a partir de dades de diferents registres.

Per poder fer aquestes agrupacions s'utilitza la clàusula `GROUP BY` que permet indicar en base a quins registres es fa l'agrupació.

Amb `GROUP BY` la sintaxi de la instrucció `SELECT` queda d'aquesta forma:

```
SELECT llistaExpressions
FROM llistaTaules
[WHERE] condicionsSeleccio
[GROUP BY] llistaCampsPerAgrupar
[HAVING] condicionsSeleccioDeGrups
[ORDER BY] llistaCampsOrdenació;
```

A la clàusula `GROUP BY`, s'indiquen les columnes per les que s'agrupa. La funció d'aquesta clàusula és crear un únic registre per cada valor diferent en les columnes del grup.

Si per **Exemple** agrupem en base a les columnes *hospital_codi* i *sala_codi* de la taula *plantilla*, es crearà un únic registre per cada hospital i sala diferents:

```
SQL> SELECT hospital_codi, sala_codi, COUNT(*)
      FROM      plantilla
      GROUP BY hospital_codi, sala_codi;
```

Si la taula `PLANTILLA` sense agrupar és:

HOSPITAL_CODI	SALA_CODI	NOM
13	6	Diaz B.
13	6	Hernandez J.
18	4	Karplus W.
22	6	Higuera D.
22	6	Bocina G.
22	2	Nunez C.
22	1	Rivera G.
22	1	Carlos R.
45	4	Amigo R.
45	1	Frank H.

HOSPITAL_CODI	SALA_CODI	NOM
13	6	Diaz B.
13	6	Hernandez J.
18	4	Karplus W.
22	6	Higuera D.
22	6	Bocina G.
22	2	Nunez C.
22	1	Rivera G.
22	1	Carlos R.
45	4	Amigo R.
45	1	Frank H.

La consulta anterior produeix el següent resultat:

```
+-----+-----+
```

hospital_codi	sala_codi	
13	6	2
18	4	1
22	1	2
22	2	
22	6	
45	1	
45	4	

És a dir, **és un resum de les dades anteriors**. La columna *nom* no està disponible directament ja que és diferent en els registres del mateix grup. Només es podria utilitzar des de funcions (com es veurà a més endavant). És a dir, aquesta consulta és errònia:

En molts SGBD dona un error semblant a aquest i en MySQL també

```
mysql> SELECT hospital_codi, sala_codi, nom
        FROM plantilla
        GROUP BY hospital_codi, sala_codi;
```

*

ERROR 1055 (42000): Expression #2 of SELECT list is not in GROUP BY clause and contains nonaggregated column 'hospital.plantilla.nom' which is not functionally dependent on columns in GROUP BY clause; this is incompatible with sql_mode=only_full_group_by

En MySQL, podem canviar el mode SQL mitjançant la variable de sistema `sql_mode` per tal de que no doni error, però el resultat segurament no seria l'esperat. La sentència no dona error i en el camp `nom` si col·loca el primer element de la coincidència.

hospital_codi	sala_codi	nom
13	6	Diaz B.
18	4	Karplus W.
22	1	Rivera G.
22	2	Nunez C.
22	6	Higuera D.
45	1	Frank H.
45	4	Amigo R.

Web a on explica els diferents `sql_modes`:

- <https://dev.mysql.com/doc/refman/8.0/en/sql-mode.html>
- <http://www.mysqlfaq.net/mysql-client-server-commands/what-is-sql-mode-in-mysql-and-how-can-we-set-it>

Mitjançant les funcions `list_add` i `list_drop` de `SYS SCHEMA` (afegides a partir de la versió MySQL 5.7.9.) podem afegir o treure modes de la nostra variable d'entorn `sql_mode`:

```
mysql> SELECT @@sql_mode;
```

```
| @@sql_mode |
+-----+
| ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES |
+-----+

mysql> SET @@sql_mode = list_add(@@sql_mode,
'NO_ENGINE_SUBSTITUTION');

mysql> SELECT @@sql_mode;
+-----+
| @@sql_mode |
+-----+
| ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION |
+-----+

mysql> SET @@sql_mode = list_drop(@@sql_mode,
'NO_ENGINE_SUBSTITUTION');
```

Funcions de càlcul amb grups (funcions de totals o d'agregats)

El que és realment interessant de la creació de grups és les possibilitats de càlcul que ofereix. Per poder fer càlculs de grups s'utilitzen funcions que permeten treballar amb els registres d'un grup.

Són:

Funció	Descripció
COUNT(*)	Compta el nombre d'elements d'un grup.
COUNT(expressió)	Expressió pot ser una columna o expressió on hi intervé alguna columna. Compta el número de vegades que expressió no és NULL .
SUM(expressió)	Suma els valors de expressió.
AVG(expressió)	Calcula la mitjana aritmètica sobre l'expressió indicada.
MIN(expressió)	Mínim valor que agafa l'expressió indicada.
MAX(expressió)	Màxim valor que agafa l'expressió indicada.
STD(expressió)	Desviació estàndard sobre l'expressió indicada

Exemple: Obtenir la suma de salaris per cada ofici dels empleats.

```
mysql> SELECT  feina_codi, SUM(salari)
            FROM empleats
            GROUP BY feina_codi;
```

```
+-----+-----+
| feina_codi | SUM(salari) |
+-----+-----+
| AC_ACCOUNT | 17300.00 |
| AC_MGR     | 12000.00 |
| AD_ASST    | 4400.00  |
| AD_PRES    | 34000.00 |
| AD_VP      | 24000.00 |
| IT_PROG    | 79200.00 |
| MK_MAN     | 13000.00 |
| MK_REP     | 6000.00  |
| SA_MAN     | 10500.00 |
| SA_REP     | 26600.00 |
| ST_CLERK   | 11700.00 |
| ST_MAN     | 5800.00  |
+-----+-----+
```

Exemple: Obtenir quantes comissions diferents tenim a la taula empleats.

```
mysql> SELECT COUNT(DISTINCT pct_comissio) AS quants
        FROM empleats;
+-----+
| quants |
+-----+
| 3      |
+-----+
```

En aquest exemple algú es podria pensar que hauria de ser 4, ja que hi ha valors NULLS, però el COUNT no comptabilitza valors NULLS.

Exemple: Volem obtenir el salari total de tots els empleats juntament amb la seva mitjana.

```
mysql> SELECT SUM(salari) AS total
        ,AVG(salari) AS mitjana
        FROM empleats;
+-----+-----+
| total   | mitjana   |
+-----+-----+
| 244500.00 | 10630.434783 |
+-----+-----+
```

Exemple: Volem obtenir la mitjana i la desviació estàndard dels empleats.

HAVING

De vegades es vol restringir el resultat d'una expressió agrupada, com per **Exemple:** Volem obtenir aquelles oficines que la suma del salari de tots els seus empleats sigui superior a 700.000

```
mysql> SELECT feina_codi, SUM(salari)
        FROM empleats
        WHERE SUM(salari)>700000
        GROUP BY feina_codi;
```

Però molts SGBD retornen un error:
Oracle

```
WHERE      SUM(empl_salari)>700000
          *
ERROR at line 3:
ORA-00934: aquí no es permet la funció d'agrupament
```

MySQL

ERROR 1111 (HY000): Invalid use of group function

La raó d'aquest error és perquè el SGBD sempre executa primer el `WHERE` i llavors els grups; i per tant aquesta condició del `WHERE` no la pot fer pel fet de que encara no hi ha els grups fets.

Per això s'utilitza la clàusula `HAVING`, que s'executa després de fer els grups. S'utilitzaria d'aquesta forma:

```
mysql> SELECT  feina_codi, SUM(salari)
            FROM empleats
            GROUP BY feina_codi;
            HAVING SUM(salari)>20000;
```

feina_codi	SUM(salari)
AD_PRES	34000.00
AD_VP	24000.00
IT_PROG	79200.00
SA_REP	26600.00

Això no significa que no puguem utilitzar la clàusula `WHERE` quan fem agrupacions.

Per **Exemple**, aquesta consulta si que seria vàlida:

```
mysql> SELECT  feina_codi, SUM(salari)
            FROM empleats
            WHERE feina_codi != 'AD_PRES'
            GROUP BY feina_codi;
            HAVING SUM(salari)>20000;
```

feina_codi	SUM(salari)
AD_VP	24000.00
IT_PROG	79200.00
SA_REP	26600.00

En definitiva, l'ordre d'execució de la consulta marca el que es pot utilitzar amb el `WHERE` i el que es pot utilitzar amb `HAVING`.

Passes en l'execució d'una instrucció d'agrupació per part del gestor de bases de dades:

- Selecció de les files desitjades utilitzant `WHERE`. Aquesta clàusula eliminarà files en base a la condició indicada.
- S'estableixen els grups indicats en la clàusula `GROUP BY`.
- Es calculen els valors de les funcions de totals (`COUNT`, `SUM`, `AVG`,...)
- Es filtren els grups que compleixen la clàusula `HAVING`.
- El resultat s'ordena en base a l'apartat `ORDER BY`.

3.3. Consulta de dades sobre v ries taules

3.3.1. Introducci 

 s molt normal que en una consulta es necessitin dades que es troben en diferents taules.

Les bases de dades relacionals es basen, com hem vist, en que les dades es distribueixen en relacions que es poden relacionar mitjan ant un atribut. Aquest atribut  s el que permet combinar les dades de les relacions.

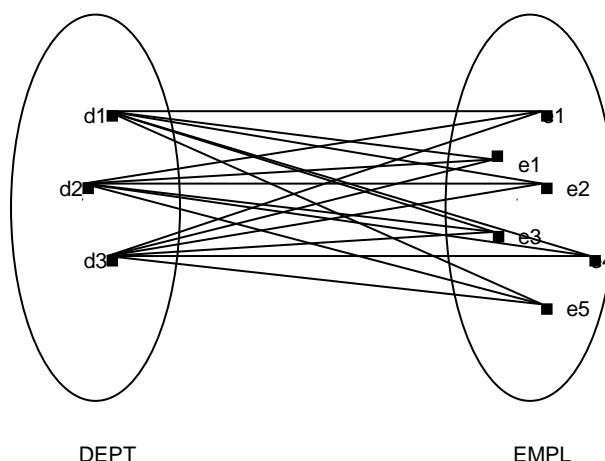
Per **Exemple**, si tenim una relaci  de departaments amb clau prim ria *dept_codi* i una altra relaci  d'empleats que es refereix als empleats que treballen a cada departament, segur que (si el disseny est  ben fet) a la relaci  d'empleats hi haur  el codi de departament el qual pertany l'empleat.

3.3.2. Producte cartesi  de taules

Si per **Exemple**, es vol obtenir una llista de les dades dels departaments i dels empleats, es podria fer de la seg ent forma:

```
SELECT *  
FROM departaments, empleats ;
```

La sintaxi  s correcta ja que es pot indicar m s d'una taula a l'apartat de la cl usula FROM. Aix  produeix el que es coneix com a producte cartesi : El producte cartesi  de dos conjunts consisteix en construir totes les possibles parelles entre cada un dels elements d'un conjunt amb els dels altres.



Example:

```
mysql> SELECT dept_num,dept_nom,empl_num,empl_nom
        FROM   dept, empl;
```

DEPT_NUM	DEPT_NOM	EMPL_NUM	EMPL_NOM
-----	-----	-----	-----
10	Comptabilitat	7369	SANCHEZ
20	Investigacio	7369	SANCHEZ
30	Vendes	7369	SANCHEZ
40	Produccio	7369	SANCHEZ
10	Comptabilitat	7499	ARROYO
20	Investigacio	7499	ARROYO
30	Vendes	7499	ARROYO
40	Produccio	7499	ARROYO
10	Comptabilitat	7521	SALA
20	Investigacio	7521	SALA
30	Vendes	7521	SALA
40	Produccio	7521	SALA
10	Comptabilitat	7566	JIMENEZ
20	Investigacio	7566	JIMENEZ
30	Vendes	7566	JIMENEZ
40	Produccio	7566	JIMENEZ
10	Comptabilitat	7654	MARTIN
20	Investigacio	7654	MARTIN
30	Vendes	7654	MARTIN
40	Produccio	7654	MARTIN
10	Comptabilitat	7698	NEGRO
20	Investigacio	7698	NEGRO
30	Vendes	7698	NEGRO
40	Produccio	7698	NEGRO
10	Comptabilitat	7782	CEREZO
20	Investigacio	7782	CEREZO
30	Vendes	7782	CEREZO
40	Produccio	7782	CEREZO
10	Comptabilitat	7788	GIL
20	Investigacio	7788	GIL
30	Vendes	7788	GIL
40	Produccio	7788	GIL
10	Comptabilitat	7839	REY
20	Investigacio	7839	REY
30	Vendes	7839	REY
40	Produccio	7839	REY
10	Comptabilitat	7844	TOVAR
20	Investigacio	7844	TOVAR
30	Vendes	7844	TOVAR
40	Produccio	7844	TOVAR
10	Comptabilitat	7876	ALONSO
20	Investigacio	7876	ALONSO
30	Vendes	7876	ALONSO
40	Produccio	7876	ALONSO
10	Comptabilitat	7900	JIMENO
20	Investigacio	7900	JIMENO
30	Vendes	7900	JIMENO
40	Produccio	7900	JIMENO
10	Comptabilitat	7902	FERNANDEZ
20	Investigacio	7902	FERNANDEZ
30	Vendes	7902	FERNANDEZ
40	Produccio	7902	FERNANDEZ
10	Comptabilitat	7934	MUNOZ
20	Investigacio	7934	MUNOZ
30	Vendes	7934	MUNOZ
40	Produccio	7934	MUNOZ

56 rows selected.

Com es pot observar a l'**Exemple** anterior, el producte cartesià entre dues taules no té molta utilitat.

Si suposem que la taula departaments està composta per **4 registres** i la taula empleats per **14 registres**, el producte cartesià serà una consulta amb un resultat de **56 registres**

Cal discriminar el producte per tal que només ens doni com a resultat els departaments relacionats amb els seus empleats. **A aquesta discriminació se l'anomena combinació (*join*) de taules.**

En general nosaltres tindrem diferents opcions per treballar amb múltiples taules:

- Combinacions de taules (*joins*).
- Combinacions especials
- Subconsultes (*subqueries*).

3.3.3. Combinació de taules

<http://dev.mysql.com/doc/refman/8.0/en/join.html>

Sintaxi:

```
SELECT <columnes de les taules de la clàusula FROM>
  FROM taula1 [[AS] alias_t1]
    ,taula2 [[AS] alias_t2]
    ,taulaN [[AS] alias_tN]
 WHERE [{taula1|alias_t1}.]columna = [{taula2|alias_t2}.]columna
 [ AND [{taulaX|aliasX}.]columna = {taulaY|aliasY}.]columna ...]
```

Exemple:

```
mysql> SELECT dept_num, dept_nom, empl_num, empl_nom
       FROM   empl, dept
       WHERE  dept_num = empl_dept_num;
```

Sintaxi ANSI SQL-92

En les versions anteriors (ANSI SQL-89) la operació de JOIN s'indicava a la clàusula FROM i a la clàusula WHERE com s'ha vist en l'exemple anterior, però a partir del **ANSI SQL-92** es va crear una nova sintaxi per consultar varies taules. La raó va ser la de separar **les condicions de combinació** de les **condicions de selecció** de registres.

Sintaxi:

```
SELECT taula1.columna1, taula1.columna2,...
       taula2.columna1, taula2.columna2,...
FROM taula1
[CROSS JOIN taula2] |
[NATURAL JOIN taula2] |
[JOIN taula2 USING(columna1,columna2,...)] |
[[INNER]JOIN taula2 ON (taula2.columa=taula1.columna)] |
[LEFT|RIGHT|FULL OUTER JOIN taula2 ON (taula1.columa=taula2.columna)] ;
```

3.3.4. Combinacions creuades

CROSS JOIN

S'utilitza per fer un producte cartesià entre les taules.

Exemple :

```
SELECT empl_nom, dept_num
FROM empl
CROSS JOIN dept;
```

És equivalent a:

```
SELECT empl_nom, dept_num
FROM dept
, empl;
```

3.3.5. Combinacions internes

NATURAL JOIN

Estableix una relació d'igualtat entre les dades de les taules a partir de les columnes que **tinguin el mateix nom en les dues taules**.

Si no hi ha columnes que tinguin el mateix nom en les dues taules, es produirà un producte cartesà.

Si les columnes que tenen el mateix nom tenen tipus de dades diferents, dóna un error.

És una drecera per no haver d'especificar els camps que intervenen en la relació. Veureu que és un cas particular del INNER JOIN.

Exemple 1: Suposem que tenim el següent esquema relacional de taules:

EMPL(empl_num, empl_nom, dept_num)

DEPT(dept_num, dept_nom)

```
mysql> SELECT empl_nom, dept_nom
        FROM   empl
              NATURAL JOIN dept;
```

EMPL_NOM	DEPT_NOM
SANCHEZ	Investigacio
ARROYO	Vendes
SALA	Vendes
JIMENEZ	Investigacio
MARTIN	Vendes
NEGRO	Vendes
CEREZO	Comptabilitat
GIL	Investigacio
REY	Comptabilitat
TOVAR	Vendes
ALONSO	Investigacio
JIMENO	Vendes
FERNANDEZ	Investigacio
MUNOZ	Comptabilitat

14 rows selected.

És equivalent a:

```
mysql> SELECT empl_nom, dept_nom
        FROM empl e, dept d
        WHERE e.dept_num = d.dept_num;
```

JOIN USING

Permet establir relacions indicant quina columna (o columnes) comuns a les dues taules cal utilitzar per fer la combinació.

Exemple: Suposem que tenim les taules:

EMPL(empl_num, empl_nom, dept_num) i DEPT(dept_num, dept_nom)

```
mysql> SELECT empl_nom, dept_nom
        FROM empl
        JOIN dept USING (dept_num);
```

EMPL_NOM	DEPT_NOM
-----	-----
SANCHEZ	Investigacio
ARROYO	Vendes
SALA	Vendes
JIMENEZ	Investigacio
MARTIN	Vendes
NEGRO	Vendes
CEREZO	Comptabilitat
GIL	Investigacio
REY	Comptabilitat
TOVAR	Vendes
ALONSO	Investigacio
JIMENO	Vendes
FERNANDEZ	Investigacio
MUNOZ	Comptabilitat

14 rows selected.

És equivalent a:

```
mysql> SELECT empl_nom, dept_nom
        FROM empl e, dept d
        WHERE e.dept_num = d.dept_num;
```

Nota: Si no s'utilitza la clàusula USING es produirà un producte cartesià.

Quan utilitzarem la clàusula USING?

- Quan es vol fer coincidir només una columna quan en coincideix més d'una. És a dir a les dues taules tenim varies columnes que coincideixen i només volem realitzar la combinació amb una d'aquestes columnes.

Nota: Cal tenir en compte també de que no podrem utilitzar els àlies dins de la clàusula USING.

[INNER] JOIN ON

En general, una operació JOIN combina dos o més taules i genera un conjunt de resultats a partir de les informacions contingudes en aquestes taules.

Per establir aquestes combinacions han de tenir columnes similars, normalment claus externes (FKs), que són les que s'utilitzaran en les operacions de JOIN per vincular les taules entre si.

Cal destacar que el nom de les columnes no cal que siguin iguals.

L'operació INNER JOIN entre dues taules retorna totes les files coincidents que hi ha en ambdues taules. INNER JOIN avalua la condició de combinació per cada fila de cada taula, i si la condició és compleix s'afegeix al conjunt de resultats.

El fet d'especificar la paraula clau INNER no afecta amb el resultat, només és de cares a diferenciar-lo dels JOINS que veurem a continuació (LEFT, RIGHT, OUTER). Recomano l'ús de la paraula clau INNER per tal de facilitar la lectura de la sentència i entendre més ràpidament que es vol aconseguir.

Exemple:

```
mysql> SELECT empl_nom, dept_nom  
        FROM empl  
        INNER JOIN dept ON empl_dept_num = dept_num;
```

EMPL_NOM	DEPT_NOM
SANCHEZ	Investigacio
ARROYO	Vendes
SALA	Vendes
JIMENEZ	Investigacio
MARTIN	Vendes
NEGRO	Vendes
CEREZO	Comptabilitat
GIL	Investigacio
REY	Comptabilitat
TOVAR	Vendes
ALONSO	Investigacio
JIMENO	Vendes
FERNANDEZ	Investigacio
MUNOZ	Comptabilitat

14 rows selected.

És equivalent a:

```
mysql> SELECT empl_nom, dept_nom  
        FROM empl e, dept d  
        WHERE empl_dept_num = dept_num;
```

Nota: Cal tenir en compte que les columnes que contenen valors NULL no coincideixen amb cap valor, (NULL no significa res). I per tant no apareixeran en el resultat de la JOIN.

I si es combinen 3 taules, com s'utilitza la clàusula JOIN ON?

Exemple : Seleccionar el nom dels empleats, i el número de telèfon de l'hospital dels empleats que treballen a la sala de Recuperacions de l'hospital La Paz.

Nota: mostrar petit E/R.

```
mysql> SELECT p.nom, h.telefon
        FROM plantilla p
        INNER JOIN sala s ON s.codi = p.sala_codi
        INNER JOIN hospital h ON h.codi = s.hospital_codi
        WHERE s.sala_nom = 'Recuperacio'
        AND h.hospital_nom = 'La Paz';
```

PLANTILLA_NOM	HOSPITAL
Rivera G.	923-5411
Carlos R.	923-5411

No és imprescindible que les columnes utilitzades en la condició de la combinació siguin del mateix tipus de dades, però com a mínim, han de ser compatibles:

- Les columnes tenen el mateix tipus de dades, però hi ha més precisió en una que l'altre. Per exemple SMALLINT amb INT.
- Si les columnes són de tipus de dades diferents, però algun del dos tipus de dades es pot convertir implícitament en l'altre tipus de dades. Per exemple una columna que conté valors numèrics de tipus CHAR/VARCHAR amb una columna de tipus INT.

MySQL té la funció [CAST/CONVERTER](http://dev.mysql.com/doc/refman/8.0/en/type-conversion.html) per tal de convertir tipus de dades

<http://dev.mysql.com/doc/refman/8.0/en/type-conversion.html>

La llista de columnes d'una consulta que combina taules pot fer referència a qualsevol de les columnes de qualsevol taula. Si existeixen noms de columnes iguals en taules diferents caldrà especificar de quina taula és aquella columna.

Exemple: EMPLEATS (id,nom,cognoms,...), DEPARTAMENTS(id,nom,num_empleats).

Si volem fer referència a la columna nom de la taula empleats haurem d'utilitzar la sintaxi "EMPLEATS.nom"

També podem fer referència a tots els camps de la taula empleats mitjançant *.

Exemple: EMPLEATS.*

Nota: El rendiment de la JOIN millorarà si les columnes estan indexades

3.3.6. Combinacions externes

Les combinacions externes retornen totes les files que compleixin la condició de combinació, **però a més a més** poden incloure registres d'una o ambdues taules de consulta que no tinguin registres corresponents a l'altre taula.

Per exemple, de quina manera podria obtenir els empleats que no pertanyen a cap departament si INNER JOIN precisament només selecciona aquells que hi hagi coincidència de valors entre la taula empleats i departaments?.

Això s'aconsegueix mitjançant la clàusula **OUTER JOIN**, la qual comença a on els resultats de INNER JOIN acaben.

LEFT [OUTER] JOIN ON

La paraula clau OUTER es pot utilitzar o no.

Exemple:

```
mysql> SELECT e.nom AS emp_nom,d.nom AS dep_nom
->      FROM empleats e
->      LEFT JOIN departaments d ON d.departament_id=e.departament_id;
```

emp_nom	dep_nom
Steven	Directiu
Neena	Directiu
Lex	Directiu
Alexander	IT
Bruce	IT
Diana	IT
Kevin	Vendes
Trenna	Vendes
Curtis	Vendes
Randall	Vendes
Peter	Vendes
Eleni	Compres
Ellen	Compres
Jonathan	Compres
Kimberely	NULL
Jennifer	Administració
Michael	Marketing
Pat	Marketing
Shelley	Contable
William	Contable
Sarah	Contable
Pere	NULL
Pau	NULL

Els empleats Pere i Pau no apareixerien utilitzant un INNER JOIN.

RIGHT [OUTER] JOIN ON

Seria el cas contrari que el LEFT. Per exemple, volem que apareguin els registres de la taula departaments que no estan relacionats amb empleats. O sigui volem veure tots els empleats amb el seu departament i tots aquells departament a que no tenen cap empleat assignat.

Exemple:

```
mysql> SELECT e.nom AS emp_nom,d.nom AS dep_nom
-> FROM empleats e
-> RIGHT JOIN departaments d ON d.departament_id=e.departament_id;
```

```
mysql> SELECT e.nom AS emp_nom,d.nom AS dep_nom
-> FROM departaments d
-> RIGHT JOIN empleats e ON d.departament_id=e.departament_id;
```

emp_nom	dep_nom
Jennifer	Administració
Michael	Marketing
Pat	Marketing
Kevin	Vendes
Trenna	Vendes
Curtis	Vendes
Randall	Vendes
Peter	Vendes
Alexander	IT
Bruce	IT
Diana	IT
Eleni	Compres
Ellen	Compres
Jonathan	Compres
Steven	Directiu
Neena	Directiu
Lex	Directiu
Shelley	Contable
William	Contable
Sarah	Contable
NULL	Recursos Humans
NULL	Producció

Apareixen els departaments 'Recursos Humans' i 'Producció' que no tenen cap empleat assignat, més tots els empleats que tenen algun departament assignat.

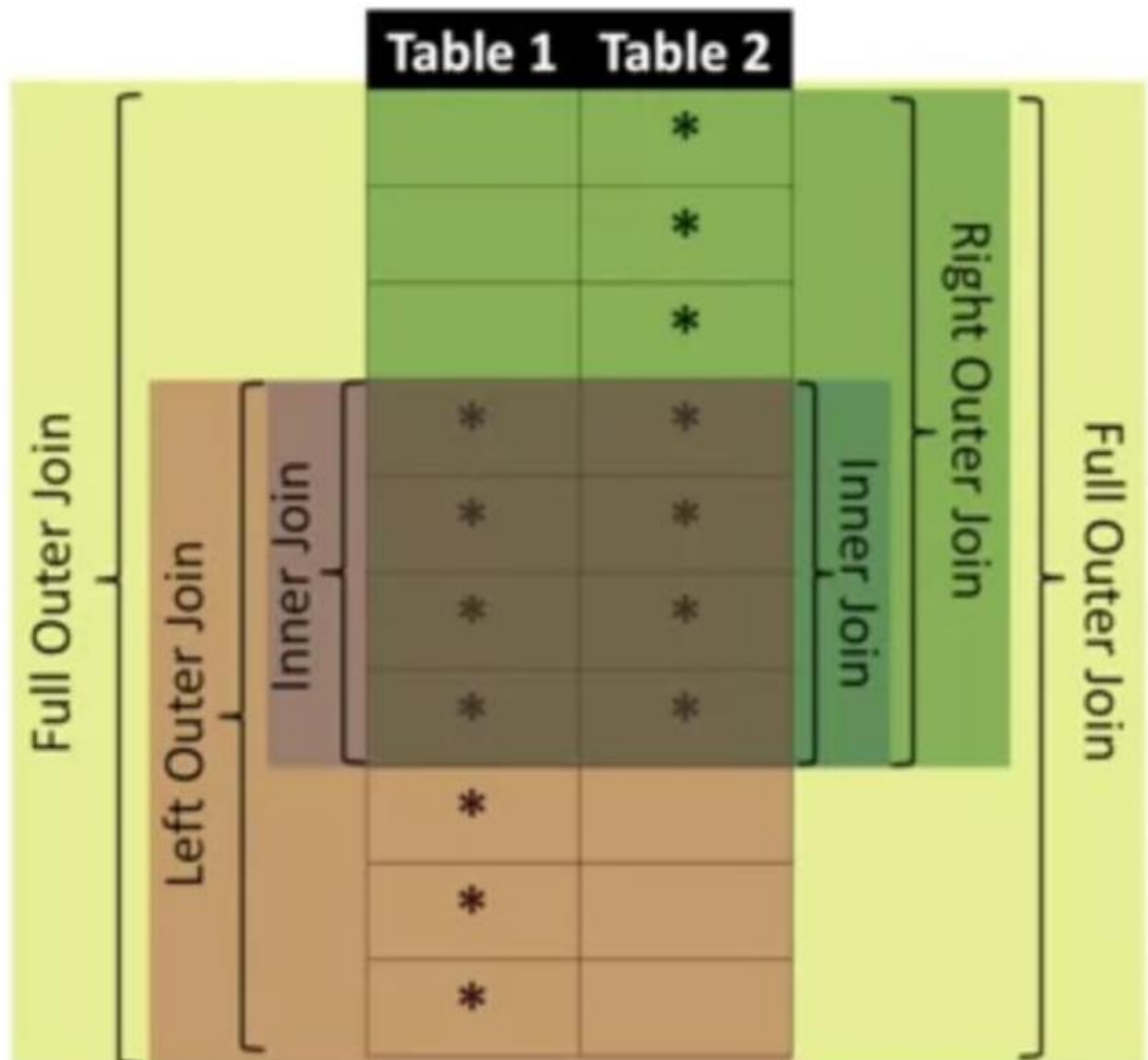
FULL OUTER JOIN ON

Per últim podem voler que apareguin tots aquells registres que es mostraven mitjançant LEFT OUTER JOIN i tots els que es mostraven mitjançant RIGHT OUTER JOIN.

Nota: MySQL **no suporta** aquest tipus de combinació de taules, però es pot simular

utilitzant la combinació del LEFT I RIGHT JOIN mitjançant la clàusula UNION que explicarem més endavant.

Resum



3.3.7. Combinacions especials

Autocombinacions

Les autocombinacions no són res més que combinacions internes o externes aplicades a la mateixa taula. Quan a la mateixa taula hi tenim una relació de llaç (reflexiva)

Per exemple, a la taula EMPLEATS tenim una columna anomenada id_cap que indica quin és l'identificador de l'empleat el qual aquell empleat n'és subordinat.

Exemple:

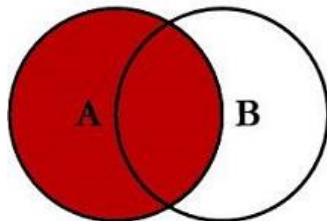
```
mysql> SELECT e.nom emp_nom, e.cognoms emp_cognoms
        ,c.nom cap_nom, c.cognoms cap_cognoms
        FROM empleats e
        INNER JOIN empleats c ON c.empleat_id = e.id_cap;
```

emp_nom	emp_cognoms	cap_nom	cap_cognoms
Neena	Kochhar	Steven	King
Lex	De haan	Steven	King
Alexander	Hunold	Lex	De haan
Bruce	Ernst	Alexander	Hunold
Diana	Lorentz	Alexander	Hunold
Kevin	Mourgos	Steven	King
Trenna	Rajs	Kevin	Mourgos
Curtis	Davies	Kevin	Mourgos
Randall	Matos	Kevin	Mourgos
Peter	Vargas	Kevin	Mourgos
Eleni	Zlotkey	Steven	King
Ellen	Abel	Eleni	Zlotkey
Jonathan	Taylor	Eleni	Zlotkey
Kimberely	Grant	Eleni	Zlotkey
Jennifer	Whalen	Neena	Kochhar
Michael	Hartstein	Steven	King
Pat	Fay	Michael	Hartstein
Shelley	Higgins	Neena	Kochhar
William	Gietz	Shelley	Higgins
Sarah	Connor	Shelley	Higgins

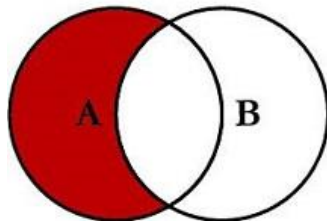
20 rows in set (0.00 sec)

Resum de les JOINS

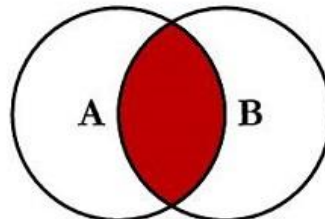
SQL JOINS



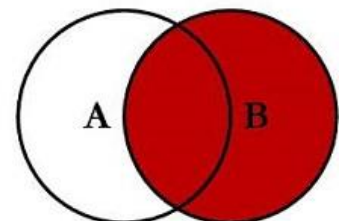
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



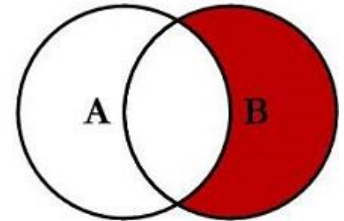
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



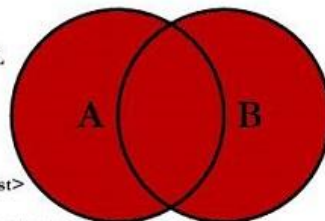
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



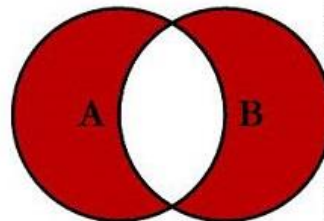
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



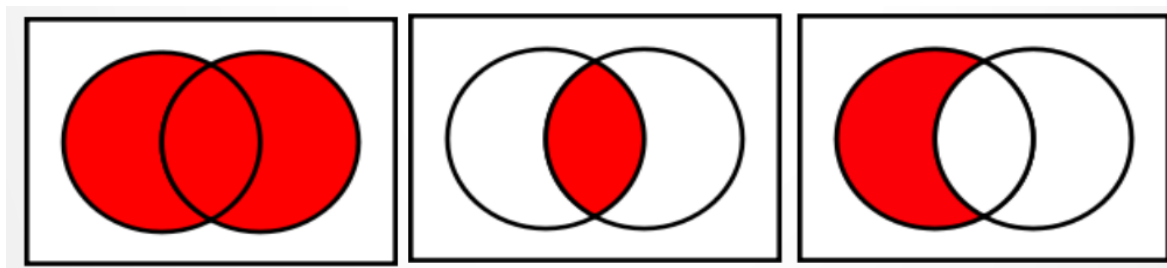
```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

3.3.8. Operadors de conjunts

Teoria de conjunts

El resultat d'una consulta es pot veure com un conjunt de valors (o instàncies) i, de vegades, ens és útil fer operacions entre els **resultats** de dos o més consultes.

L'estàndard SQL preveu fer les següents operacions entre conjunts: UNIÓ , INTERSECCIÓ , DIFERÈNCIA



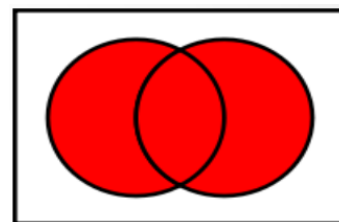
Unió (UNION)

Aquest operador equival a l'operació UNIÓ de l'àlgebra relacional.

L'operador UNION s'utilitza per combinar dos o més resultats d'instruccions SELECT i generar un únic conjunt de resultats.

Permetent afegir en el resultat d'un SELECT el resultat d'una altra SELECT. Per poder-ho fer, les instruccions SELECT han de satisfer certes condicions:

- Han de tenir el mateix nombre de columnes/camps.
- Els tipus de dades de les columnes han de ser compatibles. En altres paraules, els tipus de dades han de ser implícitament convertibles o explícitament mitjançant CAST o CONVERT.



Els noms de les columnes que es mostrarà en el conjunt de resultats s'utilitzaran els noms indicats en la primera SELECT.

Sintaxi

```
SELECT ....  
  FROM ...  
 WHERE ...  
 UNION [DISTINCT | ALL]  
 SELECT ....  
  FROM ...  
 WHERE ...
```

Exemple:

```
mysql> SELECT prov_nom AS Nom
        FROM proveidors
        UNION
        SELECT cli_nom
        FROM clients;

NOM
-----
ALONSO
ARROYO
Amigo R.
Bocina G.
CEREZO
Carlos R.
Diaz B.
FERNANDEZ
Frank H.
GIL
Hernandez J.
Higuera D.
JIMENEZ
JIMENO
Karplus W.
MARTIN
MUNOZ
NEGRO
Nunez C.
REY
Rivera G.
SALA
SANCHEZ
TOVAR

24 rows selected.
```

El resultat és una taula que tindrà els noms de la taula proveïdors i els noms de la taula clients.

Si hi ha elements o files **repetides s'eliminen**.

Si es volen mostrar tots els registres i incloure també els duplicats cal utilitzar la paraula clau **ALL** al costat de l'`UNION`.

Seria l'equivalent a “afegir” les dades d'una consulta a l'altre.

Exemple:

```
mysql> SELECT prov_nom AS Nom
        FROM proveidors
        UNION ALL
        SELECT cli_nom
        FROM clients;

NOM
-----
```

```
ALONSO  
ARROYO  
Amigo R.  
Nunez C.
```

```
Bocina G.  
CEREZO  
Carlos R.  
Diaz B.  
FERNANDEZ  
Frank H.  
GIL  
Hernandez J.  
Higuera D.  
JIMENEZ  
JIMENO  
Karplus W.  
MARTIN  
MUNOZ  
NEGRO  
Nunez C.  
REY  
Rivera G.  
SALA  
Higuera D.  
SANCHEZ  
TOVAR  
  
24 rows selected.
```

Per defecte, s'ordena el resultat per la primera columna seleccionada.

Nota: Dins una SELECT UNION o UNION ALL no es pot utilitzar la clàusula ORDER BY!

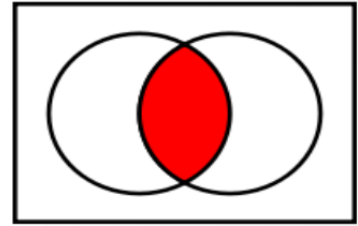
Com ho fem si volem ordenar el resultat de l'UNION? I si volem ordenar per codi empleat?

Per realitzar una ordenació sobre el resultat d'una UNION caldrà utilitzar una subconsulta

Intersecció (INTERSECT)

Aquest operador equival a l'operació INTERSECCIÓ de l'àlgebra relacional.

Amb l'operació INTERSECT s'obtenen els registres COMUNS A LES DUES consultes



Sintaxi:

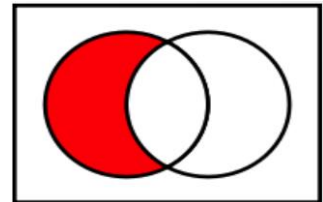
```
SELECT ....  
  FROM ...  
WHERE ...  
INTERSECT  
SELECT ....  
  FROM ...  
WHERE ...
```

Nota: MySQL no implementa aquesta operació, però es pot resoldre mitjançant subconsultes mitjançant els operadors **IN** i **EXISTS**.

Diferència (MINUS/EXCEPT)

Aquest operador equival a l'operació DIFERÈNCIA de l'àlgebra relacional. Fa la diferència d'un conjunt envers l'altre.

Amb l'operació MINUS/EXCEPT s'obtenen TOTS els registres de la consulta A menys els registres que també estan a la consulta B



Nota: MySQL no implementa aquesta operació, però es pot resoldre mitjançant l'ús de subconsultes i els operadors **NOT IN** i **NOT EXISTS**

MINUS és l'operador de diferència de conjunts per **Oracle**

EXCEPT és l'operador de diferència de conjunts per **SQL Server**.

Exemple:

```
SQL> SELECT nom  
      FROM professors  
MINUS  
      SELECT nom  
      FROM empleats;
```

NOM

ALONSO
ARROYO
CEREZO
FERNANDEZ
GIL
JIMENEZ

```
JIMENO  
MARTIN  
MUNOZ  
NEGRO  
REY  
SALA  
SANCHEZ  
TOVAR  
  
14 rows selected.
```

Es poden utilitzar aquests operadors de forma niuada (una unió on el resultat es utilitzat amb la diferència amb un altre SELECT, per **Exemple**). En aquest cas cal posar parèntesis per indicar quina operació es fa primer:

```
(SELECT....  
  ....  
  UNION  
  SELECT....  
    ...  
)  
MINUS  
SELECT....  
/* Primer es fa la unió i després la diferència */
```

3.4. Subconsultes

Fins ara hem utilitzat la clàusula WHERE per definir/filtrar els subconjunts de dades que desitgem. Utilitzen un valor d'una columna amb una constant o un grup de constants, etc..

Les subconsultes s'utilitzen per realitzar filtres amb dades d'una altra consulta o bé per utilitzar-les com a “noves taules” o taules derivades.

Aquests filtres poden ser aplicats a la clàusula WHERE (filtrar registres) o a la clàusula HAVING (filtrar grups)

A la clàusula FROM / JOIN per utilitzar-les com a “noves taules / taules derivades” noves fonts d'informació.

Exemple 1 : Quins empleats tenen un salari igual o superior a en Kevin Mourgos?

Calen dues consultes:

1. Una per saber quin és el salari d'en Kevin Mourgos

```
mysql> SELECT salari AS salari_kevin_mourgos
        FROM empleats
        WHERE nom = 'Kevin' AND cognoms = 'Mourgos';

salari_kevin_mourgos
-----
24300
```

2. Buscar els empleats que guanyen més que aquesta quantitat.

```
mysql> SELECT *
        FROM empleats
        WHERE salari >= 24300;
```

Solució (JOIN) *(Compte que hem de treure en Kevin de la segona taula.)*

```
SELECT e2.*
       FROM empleats e
       JOIN empleats e2 ON e.salari < e2.salari
                        AND e2.empleat_id != e.empleat_id
      WHERE e.nom = 'Kevin' AND e.cognoms = 'Mourgos';
```

Exemple 2: Quins empleats tenen un salari superior a la mitjana?

1. Una per saber quina és la mitja dels salaris dels empleats

```
mysql> SELECT AVG(salari) mitja
        FROM empleats;

mitja
-----
230210
```

2. Buscar els empleats que guanyen més que aquesta quantitat.

```
mysql> SELECT *
        FROM empleats
        WHERE salari>=230210;
```

Tots dos exemples es poden resoldre combinant les dues consultes, col·locant una dins l'altra.

La subconsulta s'executa abans de la consulta principal i el resultat de la subconsulta s'utilitza en la consulta principal.

D'aquesta manera, la consulta interna o *subconsulta* retorna un valor que s'utilitzarà en la consulta externa o principal.

Exemple de sintaxi:

```
SELECT llistaExpressions
  FROM taula
 WHERE expressió operador
      (SELECT llistaExpressions
        FROM taula);
```

La subconsulta sempre haurà d'anar tancada amb parèntesi per identificar-la.

3.4.1. Subconsulta escalar

Les subconsultes escalars són aquelles que retornen un únic valor . **Retornen una sola columna i una sola fila.**

Les subconsultes escalars es poden utilitzar com subconsultes en qualsevol lloc de la instrucció SQL a on volem utilitzar un valor escalar:

- A la clàusula WHERE o HAVING com a valor per comparar amb el valor d'un camp, una constant, una variable o el resultat d'una altra subconsulta escalar.

```
SELECT llistaExpressions1
FROM taula_1
WHERE expressió operador (SELECT expressió(1 sol valor)
                           FROM taula_2)
...
```

Exemple:

```
## Consulta per saber quins empleats tenen un salari superior a
la mitjana

SELECT nom
FROM plantilla
WHERE salari > ( SELECT AVG(salari)
                 FROM plantilla);
```

Exemple: Retornar la informació sobre els empleats que guanyen més que l'empleat que guanya més del departament 60.

```
SELECT *
FROM empl
WHERE empl_salari > (SELECT MAX(empl_salari)
                    FROM empl
                    WHERE empl_dept_num = 60);
```

Nota: Compte de que si la subconsulta **retorna NULL!** (per exemple si el departament no existeix)

- Com a part de la clàusula SELECT(part de la llista de camps de sortida) d'una instrucció SELECT.

```
SELECT (SELECT expressió(1 sol valor)
        FROM taula_2)
, llistaExpressions1
FROM taula_1
...
```

Exemples:

```
## Obtenim els salaris de tots els empleats sumant a cadascun  
d'ells el salari mínim de tots els empleats.
```

```
SELECT salari + (SELECT MIN(salari)  
                  FROM empleats)  
FROM empleats;
```

```
SELECT 1,2,(SELECT 3);
```

- A la clàusula SET d'una instrucció UPDATE, per indicar el valors amb que es vol assignar el camp.

```
UPDATE taula_1  
  SET camp_taula1 = (SELECT expressió(1 sol valor)  
                    FROM taula_2)  
...
```

Exemple:

```
## Modifiquem tots els salari de tots els empleats sumant-li el  
salari mínim de tots els empleats.  
  
UPDATE empleats  
  SET salari = salari + (SELECT MIN(salari)  
                        FROM empleats);
```

- En la clàusula FROM d'una instrucció SELECT, com a taula derivada amb una sola columna i una sola fila.

```
SELECT ,llistaExpressions (pot contenir algun camp de la subconsulta)  
FROM taula_1  
  <clàusula JOIN>(SELECT llistaExpressions  
                  FROM taula_2) AS taula_3 [ON]  
...
```

Exemple:

```
## Volem comparar el preu unitari amb els preus mínim i màxim.
```

```

SELECT p.producte_id
      ,p.preu
      ,pminmax.preu_min
      ,pminmax.preu_max
FROM productes p
INNER JOIN (SELECT producte_id
              ,MIN(preu) AS preu_min
              ,MAX(preu) AS preu_max
            FROM productes) AS pminmax
ON p.producte_id = pminmax.producte_id;

```

Més exemples

Exemple: Obtenir els empleats que tenen el mateix ofici que l'empleat Kevin i guanyen més que l'empleat MARTIN.

```

mysql> SELECT e1.nom, e1.feina_codi, e1.salari
      FROM empleats e1
      WHERE e1.feina_codi = (SELECT feina_codi
                            FROM empleats e2
                            WHERE e2.nom = 'Kevin'
                            AND e2.cognoms = 'Mourgos')
      AND e1.salari > (SELECT e3.salari
                      FROM empleats e3
                      WHERE e3.nom = 'Martin'
                      AND e3.cognoms = 'Martin');

Empty set (0.02 sec)

```

Exemple: Quina error hi ha en aquesta consulta?

```

mysql> SELECT e.empleat_id, e.nom
      FROM empleats e
      WHERE e.salari = (SELECT MIN(e2.salari)
                      FROM empleats e2
                      GROUP BY e2.departament_id);

```

El motiu de l'error és el d'utilitzar un operador d'una sola fila amb una subconsulta de varies files.

Depenent de la versió de **MySQL** no donarà cap error, però no retornarà cap fila. O bé donarà el següent error:

```

ERROR 1242 (21000): Subquery returns more than 1 row

```

En **Oracle** donarà el missatge d'error:

```

(SELECT MIN(empl_salari)
*)

```

```
ERROR at line 4:  
ORA-01427: single-row subquery returns more than one row
```

Exemple: Quin motiu pot ser que aquesta consulta no retorni cap fila?

```
mysql> SELECT e1.empleat_id, e1.nom  
        FROM empleats e1  
        WHERE e1.feina_codi = (SELECT e2.feina_codi  
                               FROM empleats e2  
                               WHERE e2.nom = 'PERE');  
  
no rows selected
```

No hi ha cap empleat que es digui PERE, per tant, la subconsulta no torna cap fila. La consulta externa agafa els resultants de la subconsulta (valor NULL) i els utilitza dins la seva clàusula WHERE. Un altre motiu pot ser que l'empleat PERE no tingui assignada cap feina_codi.

Recorda: La comparació de dos valors NULLs sempre dona NULL.

3.4.2. Subconsulta de llista

Les subconsultes de llista són aquelles que retornen **més d'una fila** per una **sola columna**. S'utilitzen amb operadors de comparació de varietats de files:

TAULA RESUM

Operador	Significat
[NOT] IN	Serveix per comprovar si un valor es troba en el resultat de la subconsulta.
[NOT] ANY	Compara els valors de la taula amb cada valor retornat per la subconsulta. Si ALGUN dels valors de la comparació compleix (és CERT), la consulta serà CERT.
[NOT] ALL	Compara els valors de la taula amb cada valor retornat per la subconsulta. Si TOTS els valors compleixen la condició de la comparació, llavors la consulta serà CERT.
[NOT] EXISTS	Ens diu si la subconsulta retorna o no <u>algun</u> resultat. Cert si té resultats respecte a la consulta principal.

Sintaxi:

```
SELECT llistaExpressions
  FROM taula
 WHERE expressió ([NOT]IN | [NOT]ANY | [NOT]ALL | [NOT]EXISTS)
        (SELECT llistaExpressions
          FROM taula);
```

IN

Exemple: Retornar els noms de tots els empleats del departament 20 que tenen un ofici idèntic als empleats que tenen un salari igual o superior a 2.000

```
SELECT e1.nom, e1.departament_id, e1.feina_codi
  FROM empleats e1
 WHERE e1.departament_id = 20
        AND e1.feina_codi IN (SELECT DISTINCT e2.feina_codi
                              FROM empleats e2
                              WHERE e2.salari >= 2000);
```

```
+-----+-----+-----+
| nom    | departament_id | feina_codi |
+-----+-----+-----+
| Michael |                20 | MK_MAN     |
| Pat     |                20 | MK_REP     |
+-----+-----+-----+
```

Exemple: Retornar tots els empleats que treballen en els departaments situats a les ciutats d'Oxford i Seattle.

```
SELECT e.nom,e.departament_id
  FROM empleats e
 WHERE e.departament_id IN
        (SELECT d.departament_id
          FROM departaments d
          INNER JOIN localitzacions l
            ON l.localitzacio_id = d.localitzacio_id
          WHERE ciutat IN ('Oxford', 'Seattle')
        );
```

nom	departament_id
Jennifer	10
Steven	90
Neena	90
Lex	90
Shelley	110
William	110
Sarah	110
Eleni	80
Ellen	80
Jonathan	80

Nota: Aquesta també es podria utilitzar una JOIN amb les 3 taules, però és per veure el funcionament de les subconsultes amb l'operador IN.

Exemple: Retornar les feines dels empleats del departament 20 que tenen un treball idèntic al del departament 30.

```
--Obtenir les feines que hi ha al departament 20
SELECT e.feina_codi
  FROM empleats e
 WHERE e.departament_id = 20;

--Obtenir les feines que hi ha al departament 30
SELECT e.feina_codi
  FROM empleats e1
 WHERE e1.departament_id = 30;

SELECT e.feina_codi
  FROM empleats e
 WHERE e.departament_id = 20
 AND feina_codi IN (SELECT e.feina_codi
                    FROM empleats e1
                    WHERE e1.departament_id = 30);
```

En l'exemple anterior es pot veure l'equivalència de l'operació **INTERSECCIÓ** de l'àlgebra relacional.

Sintaxi:

Exemple: Retornar els noms de tots els empleats del departament 20 que tenen un treball idèntic al de qualsevol dels empleats del departament de MARKETING

```
SELECT e1.nom
  FROM empleats e1
 WHERE e1.departament_id = 20
       AND e1.feina_codi IN (SELECT e2.feina_codi
                             FROM empleats e2
                             WHERE  e2.departament_id IN
                                     (SELECT d.departament_id
                                      FROM departaments d
                                      WHERE d.nom = 'Marketing'))
       );
```

Observant l'últim **Exemple**:

Una subconsulta pot contenir una altra subconsulta. Es poden tenir un número infinit de nivells.

L'operador de comparació utilitzat és IN, ja que la consulta interior retorna més d'una fila.

L'ordre en que la consulta interior retorna les files no és important i per tant, no pot especificar-se.


```
AND e.feina_codi != 'IT_PROG';
```

nom	feina_codi	salari
Steven	AD_VP	24000.00
Neena	AD_PRES	17000.00
Lex	AD_PRES	17000.00
Kevin	ST_MAN	5800.00
Trenna	ST_CLERK	3500.00
Curtis	ST_CLERK	3100.00
Randall	ST_CLERK	2600.00
Peter	ST_CLERK	2500.00
Eleni	SA_MAN	10500.00
Ellen	SA_REP	11000.00
Jonathan	SA_REP	8600.00
Kimberely	SA_REP	7000.00
Jennifer	AD_ASST	4400.00
Michael	MK_MAN	13000.00
Pat	MK_REP	6000.00
Shelley	AC_MGR	12000.00
William	AC_ACCOUNT	8300.00
Sarah	AC_ACCOUNT	9000.00

Exemple: Obtenir el nom i el salari dels empleats que tenen el salari igual a algun dels empleats del departament 20.

```
mysql> SELECT e1.nom, e1.salari
        FROM empleats e1
        WHERE e1.salari = ANY (SELECT e2.salari
                               FROM empleats e2
                               WHERE e2.departament_id = 20);
        e1.departament_id != 20;
```

nom	salari
Bruce	6000.00

salari
13000.00
6000.00

ALL

Exemple: Retornar tots els treballadors de la plantilla que no són Programadors i que el seu salari sigui inferior al de tots els Programadors.

```
mysql> SELECT e.nom, e.feina_codi, e.salari
        FROM empleats e
        WHERE e.salari < ALL
              (SELECT it_prog.salari
               FROM empleats it_prog
               WHERE it_prog.feina_codi = 'IT_PROG')
              AND e.feina_codi != 'IT_PROG';
```

nom	feina_codi	salari
Trenna	ST_CLERK	3500.00
Curtis	ST_CLERK	3100.00
Randall	ST_CLERK	2600.00
Peter	ST_CLERK	2500.00

salari
9000.00
6000.00
4200.00
30000.00

Exemple: Obtenir l'empleat que cobra més.

```
mysql> SELECT e1.nom, e1.salari
        FROM empleats e1
        WHERE e1.salari >= ALL (SELECT e2.salari FROM empleats e2);
```

nom	salari
Pere	40000.00

< ALL significa menys que el valor mínim de la subconsulta. Equivalent a la subconsulta amb una funció d'agrupament MIN.

> ALL significa més que el valor màxim de la subconsulta. Equivalent a la subconsulta amb una funció d'agrupament MAX.

<> ALL és equivalent a NOT IN

Resum

Subconsulta amb un MIN	>ANY (<u>més gran</u> que el valor mínim)	<ALL (més petit que el valor mínim)
Subconsulta amb un MAX	<ANY (<u>més petit</u> que el valor màxim)	>ALL (<u>més gran</u> que el valor màxim)
IN	=ANY	-
NOT IN	-	<>ALL

Quins creus que són els resultats d'aquestes consultes i perquè?

Consulta 1

```
SELECT e1.nom, e1.salari
      FROM empleats e1
     WHERE e1.salari <> ALL (SELECT e2.salari FROM empleats e2);
```

Consulta 2

```
SELECT e1.nom, e1.salari
      FROM empleats e1
     WHERE e1.salari <> ANY (SELECT e2.salari FROM empleats e2);
```

Nota: Els operadors ANY, ALL només es poden utilitzar mitjançant subconsultes.

Per exemple la següent sentència donarà un error:

```
SELECT *
      FROM empleats
     WHERE salari < ALL (9000,6000);
```

Lookup Error - MySQL Database Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '9000,6000)' at line 3

EXISTS

EXISTS utilitza una subconsulta com a condició a on la condició serà certa si la subconsulta conté alguna fila (la subconsulta retorna alguna fila), altrament serà falsa si la subconsulta no retorna cap fila.

Exemple: Obtenir els departaments que tenen empleats assignats.

```
SELECT d.departament_id, d.nom
FROM departaments d
WHERE EXISTS (SELECT e.empleat_id
              FROM empleats e
              WHERE e.departament_id = d.departament_id);
```

Aquest exemple també es podria solucionar mitjançant una consulta amb JOIN, però **no** significa que totes les consultes amb EXISTS es poden solucionar amb un JOIN

```
SELECT DISTINCT d.codi, d.nom
FROM departaments d
INNER JOIN empleats e ON e.id_departament = d.id
```

Exemple: Obtenir els departaments que **NO** tenen empleats assignats.

```
SELECT d.departament_id, d.nom
FROM departaments d
WHERE NOT EXISTS (SELECT e.empleat_id
                  FROM empleats e
                  WHERE e.departament_id = d.departament_id);
```

departament_id	nom
190	Recursos Humans
200	Producció

```
#Comprovació:
SELECT *
FROM empleats
WHERE departament_id in (190,200);
```

Creus que aquest exemple anterior també es podria solucionar sense utilitzar l'EXISTS?
La resposta és Sí, com?

Exemple: Retorna les feines_codi de tots els empleats del departament 20 que no tinguin un treball idèntic dels empleats del departament 30.

```
--Obtenir les feines que hi ha al departament 20
SELECT e.feina_codi
      FROM empleats e
 WHERE e.departament_id = 20;

--Obtenir les fines que hi ha al departament 30
SELECT e.feina_codi
      FROM empleats e1
 WHERE e1.departament_id = 30;

SELECT e.feina_codi
      FROM empleats e
 WHERE e.departament_id = 20
 AND feina_codi NOT IN (SELECT e.feina_codi
                        FROM empleats e1
                        WHERE e1.departament_id = 30);
```

En l'exemple anterior es pot veure l'equivalència de l'operació **DIFERÈNCIA** de l'àlgebra relacional.

3.4.3. Subconsulta multi-columna

Les subconsultes multi-columna o matricials són aquelles que retornen una o més d'una fila i més d'una columna.

La consulta principal compara amb els valors d'una subconsulta multi-registre i multi-columna.

Les podem utilitzar en:

- En la clàusula FROM d'una instrucció SELECT, com una taula derivada amb varies files i columnes. En definitiva tractar la subconsulta com una vista o una taula.
- En la clàusula WHERE, quan s'utilitza juntament amb la paraula clau IN, EXISTS, per verificar l'existència de valors en la llista. O bé quan volem filtrar per més d'un camp en la nostra clàusula.

Exemple: Obtenir els empleats amb el mateix ofici i salari que l'empleat Neena Kochhar

```
SELECT e1.nom, e1.cognoms
FROM   empleats e1
WHERE  (e1.feina_codi, e1.salari) =
      (SELECT e2.feina_codi, e2.salari
       FROM   empleats e2
       WHERE  e2.nom = 'Neena'
              AND e2.cognoms='Kochhar'
              AND e1.empleat_id != e2.empleat_id);
```

nom	cognoms
Lex	De haan

Nota: Cal dir que aquesta subconsulta ha de retornar un sol registre.

Exemple Obtenir els empleats que tenen un salari superior a la mitjana del seu departament:

```
SELECT a.nom, a.salari, a.departament_id, b.salavg
FROM   empleats a
      INNER JOIN (SELECT em.departament_id
                  , avg(em.salari) salavg
                  FROM   empleats em
                  GROUP BY em.departament_id) b
ON a.departament_id = b.departament_id
WHERE a.salari > b.salavg;
```

departament_id	salavg
NULL	25666.666667
10	4400.000000
20	9500.000000
50	3500.000000
60	6400.000000
80	10033.333333
90	19333.333333
110	9766.666667

nom	salari	departament_id	salavg
Michael	13000.00	20	9500.000000
Kevin	5800.00	50	3500.000000
Alexander	9000.00	60	6400.000000
Eleni	10500.00	80	10033.333333
Ellen	11000.00	80	10033.333333
Steven	24000.00	90	19333.333333
Shelley	12000.00	110	9766.666667

Exemple: Obtenir els empleats a on el seu salari i la seva comissió corresponguin amb el salari i el % de comissió de qualsevol empleat del departament amb id 60

```
SELECT e1.nom,e1.salari,e1.pct_comissio
  FROM empleats e1
 WHERE (e1.salari,IFNULL(e1.pct_comissio,0)) IN
      (SELECT e2.salari
        ,IFNULL(e2.pct_comissio,0)
      FROM empleats e2
      WHERE e2.departament_id = 60);
```

nom	salari	pct_comissio
Alexander	9000.00	NULL
Bruce	6000.00	NULL
Diana	4200.00	NULL
Pat	6000.00	NULL
Sarah	9000.00	NULL

Nota: Recorda que aquesta consulta es podria haver utilitzat = ANY

3.4.4. Subconsultes correlacionades

S'anomenen subconsultes correlacionades si s'estableix alguna relació (operació de JOIN) entre la consulta principal i la subconsulta.

Quan les dues consultes (principal i subconsulta) es poden executar una sense l'altre es pot dir que **no** estan correlacionades. Per exemple el cas més clar d'una consulta correlacionada normalment és una subconsulta que intervé l'EXIST.

En canvi, quan hem d'escriure una subconsulta que depengui de valors de la consulta principal direm que la subconsulta és una subconsulta correlacionada. En general , aquesta situació motiva que la subconsulta s'executi una vegada per cada fila retornada de la consulta principal.

3.4.5. Taules derivades

En alguns exemples de la secció subconsultes hem vist que la subconsulta es comporta com si fos una taula o una vista col·locant-la a la clàusula FROM. Aquestes subconsultes reben el nom de taules derivades.

Nota: Les subconsultes correlacionades **no** es poden utilitzar com a taules derivades.

3.4.6. Resum d'ús de subconsultes

Escriurem subconsultes quan una consulta estigui basada o necessiti valors que no es poden trobar en taules o vistes, però si que mitjançant la manipulació de taules i vistes es poden generar aquestes dades.

També podem dir que escriurem subconsultes quan una consulta estigui basada o necessiti de valors desconeguts i s'han de calcular.

Instruccions a tenir en compte alhora d'utilitzar subconsultes:

- Escriure la subconsulta entre parèntesi.
- Una subconsulta en la clàusula WHERE o HAVING ha d'aparèixer a la dreta de l'operador.
- No afegixis mai ORDER BY a una subconsulta (alenteix la consulta innecessàriament tot i que molts Parser ho tenen en compte).
- Utilitza operadors a nivell de fila per consultes que retornin una sola fila.
- Utilitza operadors que actuen sobre diferents registres per subconsultes que retornin més d'una fila.
- Per evitar ambigüitats és important definir el nom de la subconsulta i les columnes de la mateixa afegint-hi àlies.

Models per dades amb herència:

<https://www.slideshare.net/billkarwin/models-for-hierarchical-data>

3.5. DDL i DML ampliat

3.5.1. Sentència CREATE TABLE

Moltes vegades ens trobem que hem de crear una taula amb la mateixa estructura que una altra.

Sintaxi:

```
CREATE TABLE <nom_taula_nova> LIKE <nom_taula_copiar>
```

La sentència anterior només crearà una nova taula amb l'estructura i no les dades.

Si necessitem crear una taula i omplir-la amb un conjunt de dades que els podem obtenir mitjançant una consulta SELECT.

Sintaxi:

```
CREATE TABLE <nom_taula> [(definició_camps)]  
  SELECT .....  
  FROM ....  
  WHERE...
```

En aquest cas definició_camps pot ser opcional. Si no s'indiquen s'agafaran els noms i els tipus de dades dels camps definits en la SELECT.

Cal dir que en aquest cas també es poden afegir restriccions (claus primàries, foranes, etc...), però cal tenir en compte que aquesta informació no s'agafarà d'una SELECT.

3.5.2. Sentència INSERT i SELECT

Una variant de la sentència INSERT consisteix en combinar-la amb la sentència SELECT per obtenir un conjunt de dades i posteriorment inserir-los a un taula corresponent.

Sintaxi:

```
INSERT INTO taula_nom [(llistaDeCamps)]  
  SELECT ...  
  FROM ...  
  WHERE ....
```

Exemple. Volem guardar en una nova taula anomenada vehicles_stock tots els vehicles que tenim en stock. Seran tots aquells que la seva data de venda sigui nul·la.

```
INSERT INTO vehicles_stock(codi_vehicle, marca, model, preu)
  SELECT id, marca, model, preu
    FROM vehicles
   WHERE data_venda is NULL;
```

La sentència SELECT ha de retornar tantes columnes i tipus de dades compatibles com columnes tingui la taula o el llistat de camps que s'indiquen a la sentència INSERT.

La sentència SELECT pot ser tant complexa com es vulgui incloent operacions de conjunts, JOINS, subconsultes, etc....

3.5.3. Sentències UPDATE i DELETE amb subconsultes.

És possible actualitzar o esborrar registres d'una taula filtrant-los a través d'una subconsulta.

La única limitació és que hi ha gestors que no permeten realitzar canvis en la taula que s'està llegint (SELECT) a través de la subconsulta. És a dir no podem intentar executar una sentència de modificació i esborrat d'una taula la qual també n'estem llegint els registres a través de la subconsulta

Sintaxi:

```
DELETE FROM taula_nom
WHERE (
  SELECT ...
  FROM ...
  WHERE ....)
```

Exemple: Aquest exemple ens donaria un error perquè estem esborrant clients amb límit de crèdit igual a 0 i a més estem llegint dades de la mateixa taula que s'esborren.

```
DELETE FROM clients
  WHERE codi_client IN (
    SELECT codi_client
      FROM clients
     WHERE limit_credit = 0);

ERROR 1093 (HY000): You can't specify target table 'clients' for
update
```

3.5.4. Sentències UPDATE i DELETE amb relacions.

També ens podem ajudar de les relacions entre les taules per tal de seleccionar els registres els quals volem realitzar-hi l'operació d'esborrat o modificació.

La sintaxi d'aquestes sentències UPDATE i DELETE amb relacions pot variar depenent del SGBD que s'utilitzi.

UPDATE

Sintaxi:

```
UPDATE <nom_taula_> [AS nom_alies]
<sentències INNER JOIN>
  SET <llistat de camps a modificar de la taula>
<sentències de filtre (WHERE)>
```

Exemple: Volem modificar el sou de tots els empleats del departament de Direcció.

```
UPDATE EMPLEATS e
  INNER JOIN departaments d ON e.departament_id=d.departament_id
  SET e.salari = 1000
WHERE d.nom = 'Direccio';
```

Important: Especificar els camps a modificar mitjançant els àlies de la taula

DELETE

Sintaxi:

```
DELETE <noms_taulas/alies que volem borrar-ne registres>
FROM <nom_taula>
<sentències INNER JOIN>
<sentències de filtre (WHERE)>
```

Exemple: Volem esborrar tots els empleats que siguin del departament de direcció.

```
DELETE e
FROM empleats e
  INNER JOIN departaments d ON e.departament_id = d.departament_id
WHERE d.nom = 'Direccio';
```

Nota: Compte amb les restriccions de claus foranes (FKs).

Exemple: Volem esborrar tots els empleats i el departament de direcció

```
DELETE e,d
  FROM empleats e
  INNER JOIN departaments d ON e.departament_id = d.departament_id
 WHERE d.nom = 'Direccio';
```


3.7. CTE (Common Table Expressions)

<https://dev.mysql.com/doc/refman/8.0/en/with.html>

Les CTE s'utilitzen en diferents àmbits:

- “With queries”
- Formes recursives i no recursives
- Simplificació de SQL complexes.

3.7.1. WITH QUERIES

El primer cas (WITH QUERIES) s'utilitza per donar nom a una consulta per llavors fer-hi referència més endavant.

Amb la sentència WITH li donem un nom a una consulta per llavors utilitzar-la.

Exemple 1:

```
WITH empleatsIT AS (SELECT *
                     FROM empleats
                     WHERE feina_codi = 'IT_PROG')
SELECT *
FROM empleatsIT;
```

Exemple ús de subconsultes: Buscar aquells empleats que no siguin Programadors que tinguin el mateix salari que els programadors

```
WITH emplIT AS (SELECT empleat_id, salari
                  FROM empleats
                  WHERE feina_codi = 'IT_PROG')
SELECT e1.empleat_id, e1.nom, e1.feina_codi, e1.salari, e2.empleat_id
FROM empleats e1
INNER JOIN emplIT e2 ON e.empleat_id != e2.empleat_id
                   AND e.salari = e2.salari
WHERE e.feina_codi != 'IT_PROG'
```

Podem especificar tantes *common table expressions* com vulguem:

```
WITH
  cte1 AS (SELECT a, b FROM table1),
  cte2 AS (SELECT c, d FROM table2)
SELECT b, d
FROM cte1
INNER JOIN cte2 ON cte1.a = cte2.c
WHERE cte1.b < 100;
```

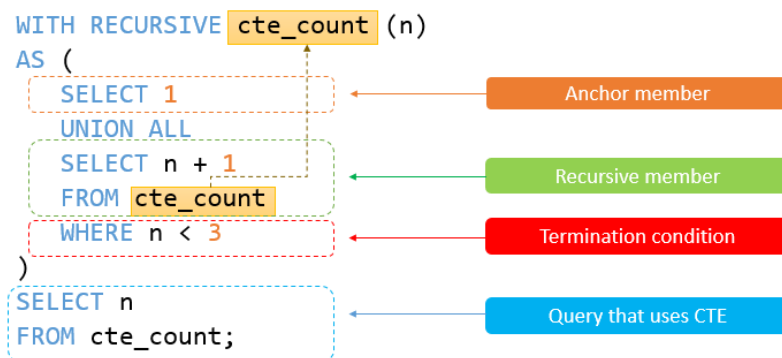
3.7.2. WITH QUERIES - Formes recursives

Podem executar recursivament una sentència SELECT sobre nosaltres mateixos per recórrer estructures jeràrquiques tal i com ho fa ORACLE mitjançant CONNECT BY.

```
with_clause:
WITH [RECURSIVE]
  cte_name [(col_name [, col_name] ...)] AS (subquery)
  [, cte_name [(col_name [, col_name] ...)] AS (subquery)] ...

WITH RECURSIVE cte[(col_name,...)] AS (
  SELECT ... FROM table_name /* "llavor" membre inicial */
  UNION ALL
  SELECT ... FROM cte      /* "recursiva" membre recursiu */
  [WHERE condició]        /* condició de fi */
)
SELECT ... FROM cte;/
```

Mira l'esquema d'aquest altre exemple



Articles: <https://dev.mysql.com/blog-archive/mysql-8-0-labs-recursive-common-table-expressions-in-mysql-ctes/>

Generació de sèries

Exemple: Volem imprimir una columna amb valors del 1 al 3.

```
WITH RECURSIVE qn(contador) AS (
  SELECT 1 AS contador
  UNION ALL
  SELECT 1 + contador
    FROM qn
   WHERE contador < 3
)
SELECT contador
  FROM qn;
```

```
+-----+
| contador |
+-----+
|         1 |
|         2 |
|         3 |
+-----+
```

Com es va construir ?

1a iteració

```
SELECT 1 AS contador
```

```
+-----+
| contador |
+-----+
|         1 |
+-----+
```

2a iteració

+-----+		+-----+
contador	SELECT 1 + 1	contador
+-----+	FROM qn	+-----+
1	WHERE 1 < 3	2
+-----+	----	+-----+

Fa la UNION entre l'anterior i el següent

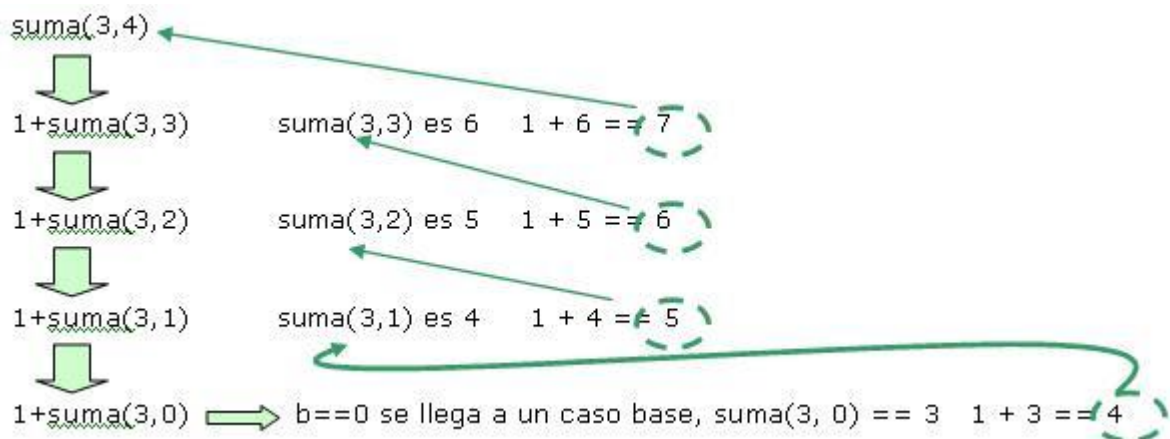
3a iteració

+-----+		+-----+
contador	SELECT 1 + 2	contador
+-----+	FROM qn	+-----+
1	WHERE 2 < 3	3
2	----	+-----+
+-----+		

4a iteració

contador	SELECT 1 + 3 FROM qn WHERE 3 < 3	---	Ja no es comleix i finalitza
1			
2			
3			

Exemple de descomposició d'un programa amb execució recursiva



Com es pot veure amb aquest exemple d'execució d'un programa que té una funció anomenada `suma(a,b)` el que fa el programa és sumar 1 a l'anterior i així successivament fins que el paràmetre `b` valgui 0(zero).

```
/* Exemple d'un codi amb Java de la funció recursiva SUMA */
int suma(a,b){
    // cas trivial
    if(b==0){
        return a;
    }
    // cas recursiu
    else {
        return 1 + suma(a,b-1);
    }
}
```

Exemple: Sèrie de Fibonacci

La sèrie de Fibonacci és una sèrie molt coneguda descrita pel matemàtic italià Leonardo de Pisa. Es tracta de construir una sèrie amb la qual cada element és la suma dels anteriors prenent com a valors inicials el 0 i l'1.

0,1,1,2,3,5,8,13,21

Aquest exemple ens mostra els 10 primers números de la sèrie de Fibonacci

```
WITH RECURSIVE fibonnacci(n,f,sequent) AS (  
    SELECT 0 AS n,  
           0 AS f,  
           1 AS sequent  
    UNION ALL  
    SELECT n+1 AS n,  
           sequent AS f,  
           f+sequent AS sequent  
    FROM fibonnacci  
    WHERE n<10  
)  
SELECT n,f,sequent  
FROM fibonnacci;
```

```
-- n: conté la posició del número  
-- f: és el número de la seqüència de fibonnaci  
-- sequent: és el valor sequent que cal sumar a l'anterior (f).
```

n	f	sequent
0	0	1
1	1	1
2	1	2
3	2	3
4	3	5
5	5	8
6	8	13
7	13	21
8	21	34
9	34	55
10	55	89

Exemple: Imaginem que volem generar una sèrie de dades successives, per exemple dates entre una data mínima i una màxima.

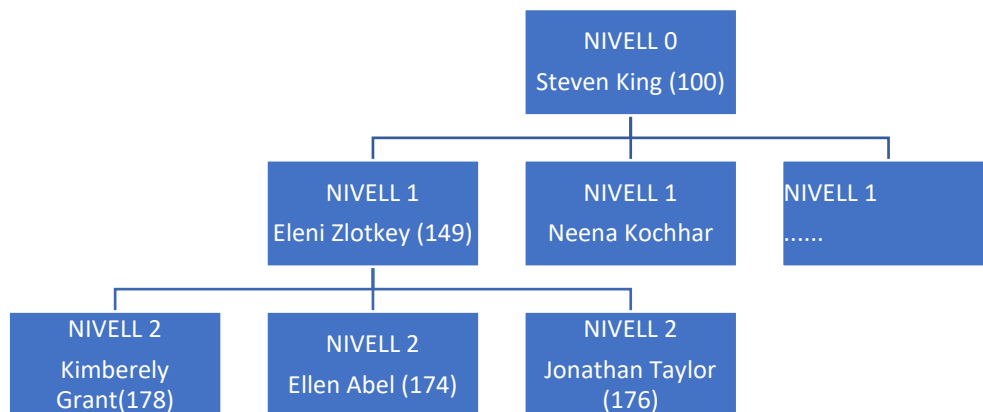
```
WITH RECURSIVE dates (data) AS
(
    SELECT MIN(data) AS data /* data mínima (la busquem taula vendes) */
    FROM vendes
    UNION ALL
    SELECT ADDDATE(data,1) AS data
    FROM dates
    WHERE ADDDATE(data,1) <= (SELECT MAX(data) FROM vendes)
)
SELECT * FROM dates;
```

Recorreguts amb herència

Tal i com s'ha dit anteriorment podem utilitzar aquesta fórmula per recórrer dades jeràrquiques.

Exemple: Imaginem que tenim una taula d'empleats amb un camp reflexiu indicant quin és el cap de cada empleat i volem saber quin és el nivell a on es troba el seu cap

```
WITH RECURSIVE emp_path AS (
  SELECT e0.empleat_id, e0.nom, e0.cognoms, e0.id_cap, 0 nivell
    FROM empleats e0          ---> PRIMER NIVELL 0
  WHERE e0.id_cap IS NULL
  UNION ALL
  SELECT en.empleat_id, en.nom, en.cognoms, en.id_cap, nivell+1
    FROM empleats en          ---> RESTA NIVELLS
    INNER JOIN emp_path ep ON ep.empleat_id = en.id_cap
)
SELECT *
  FROM emp_path
ORDER BY nivell;
```



```
WITH RECURSIVE empleats_paths (empleat_id, nom, cognoms, path) AS
(
  SELECT empleat_id, nom, cognoms
    , CAST(CONCAT(nom, ' ', cognoms) AS CHAR(500))
    FROM empleats
    WHERE id_cap IS NULL
  UNION ALL
  SELECT e.empleat_id
    , e.nom, e.cognoms
    , CONCAT(ep.path, '->', CONCAT(e.nom, ' ', e.cognoms))
    FROM empleats_paths AS ep JOIN empleats AS e
      ON ep.empleat_id = e.id_cap
)
SELECT * FROM empleats_paths ORDER BY path;
```

De la mateixa manera que el cas anterior tenim una taula de categories i aquestes depenen jeràrquicament d'unes amb les altres. Volem mostrar el *path* dels nivells.

```
#Creació de la taula
CREATE TABLE category (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  title varchar(255) NOT NULL,
  parent_id int(10) unsigned DEFAULT NULL,
  PRIMARY KEY (id),
  FOREIGN KEY (parent_id) REFERENCES category (id)
  ON DELETE CASCADE ON UPDATE CASCADE
);

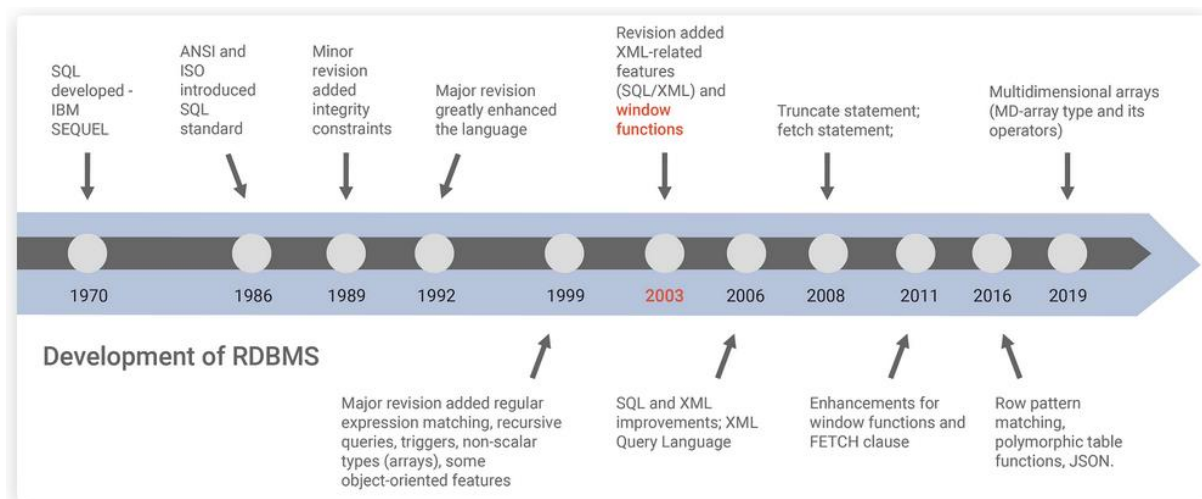
#Consulta per obtenir les categories
WITH RECURSIVE category_path (id, title, path) AS
(
  SELECT id, title, title as path
    FROM category
   WHERE parent_id IS NULL
  UNION ALL
  SELECT c.id, c.title, CONCAT(cp.path, ' > ', c.title)
    FROM category_path AS cp JOIN category AS c
      ON cp.id = c.parent_id
)
SELECT * FROM category_path
ORDER BY path;
```

id	title	path
1	Electronics	Electronics
5	Cameras & photo	Electronics > Cameras & photo
6	Camera	Electronics > Cameras & photo > ...
...		

3.8. Window Functions

3.8.1. Introducció

Aquest tipus de funcions s'ha introduït a partir de la versió MySQL 8.0. Però es va definir a l'estàndard SQL al 2003.



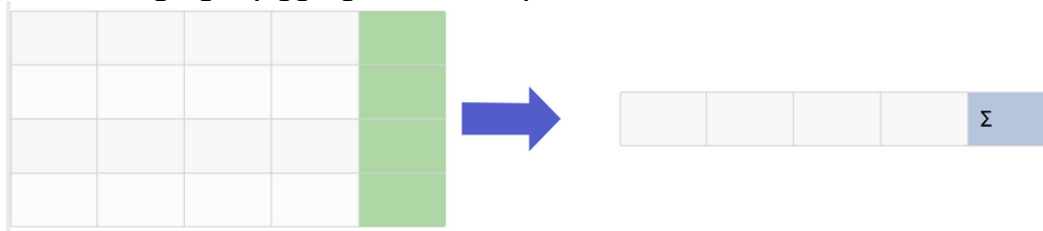
<https://learnsql.com/blog/why-should-i-learn-sql-window-functions/>

Les funcions de finestra són una eina SQL poderosa que permet realitzar càlculs en un conjunt de files que estan relacionades d'alguna manera amb la fila actual

Són similars a les funcions d'agregat mitjançant la clàusula de `GROUP BY`, ja que calculen valors agregats per a un grup de files. Però la principal diferència amb el `GROUP BY` és que no s'agrupen les files i es conserva el detall de les files.

En una funció d'agregat perdem l'accés a les files individuals ja que les agrupem en un sol resultat (fent paquets) dependent de la clàusula `GROUP BY`. En canvi a la funció de finestra tenim accés a cada fila.

Funció d'agregat (*aggregate function*)



Funció de finestra (*window function*)



Algunes de les avantatges són:

- **Simplificació de consultes complexes:** Permeten realitzar càlculs complexos en una sola consulta que, en SQL anterior, requerien de múltiples consultes o subconsultes.
- **Millora del rendiment de consultes:** Poden millorar el rendiment de les consultes reduint la quantitat de dades a processar. El fet d'utilitzar-les pot evitar la necessitat d'unir taules o crear subconsultes.
- **Millora de la lectura i el manteniment de les consultes:** Moltes vegades les consultes utilitzant funcions de finestra són més llegibles i fàcils de mantenir ja que permeten expressar lògiques complexes d'una manera senzilla i declarativa.
- **Milloren les capacitats analítiques:** Permeten realitzar càlculs analítics com totals acumulats, mitjanes mòbils i sumes acumulatives.

Sintaxi:

<https://dev.mysql.com/doc/refman/8.0/en/window-functions-usage.html>

```
<funció de finestra>(expressió) OVER (
  PARTITION BY <expressió>
  ORDER BY <expressió> [ASC | DESC]
  [RANGE | ROWS BETWEEN ( VALOR | UNBOUNDED PRECEDING )
    AND ( VALOR | UNBOUNDED FOLLOWING ) ]
)
```

<window function name> ()

OVER (

PARTITION BY <expression>

ORDER BY <expression> [ASC | DESC]

)

Per exemple si volem saber la mitjana de salari per cada departament tenim:

```
SELECT departament_id, AVG(salari) mitjana
  FROM empleats
 GROUP BY departament_id;
```

Amb aquesta sentència SQL obtindríem de cada empleat la mitjana de salari del seu propi departament.

```
SELECT empleat_id, nom, cognoms, salari, departament_id
      ,AVG(salari) OVER (PARTITION BY departament_id) AS mitjana
  FROM empleats;
```

id	nom	cognoms	salari	dep_id	mitjana
178	Kimberely	Grant	7000.00	NULL	7000.000000
200	Jennifer	Whalen	4400.00	10	4400.000000
201	Michael	Hartstein	13000.00	20	9500.000000
202	Pat	Fay	6000.00	20	9500.000000
124	Kevin	Mourgos	5800.00	50	3500.000000
141	Trenna	Rajs	3500.00	50	3500.000000
...					

Les funcions de finestra (**WINDOW FUNCTION NAME**) les dividim en diferents tipus: **Aggregat**, de **Ranking** i de **Valor**.



Amb la clàusula **OVER** determinem com s'organitzen les files i després les processa la funció de finestra.

Dins de la clàusula OVER hi ha la clàusula **PARTITION BY** i **ORDER BY**.

- **PARTITION BY**, és **opcional** i serveix per dividir les columnes de la finestra en grups/paquets (particions), com es veu en l'exemple anterior. Fem paquets per `departament_id`.
- **ORDER BY** és **opcional** i serveix per ordenar la taula resultant per els valors d'una columna seleccionada per l'usuari això té sentit quan utilitzem funcions de finestra de tipus Ranking o de Valor perquè l'ordre dins de la partició és important.

3.8.2. Llista de funcions de finestra AGGREGATE

No es descriuen perquè són les mateixes que tenim en el SQL 92 – GROUP BY: COUNT(), SUM(), MIN(), MAX(), AVG()

3.8.3. Llista de funcions de finestra RANKING

Les funcions de classificació (ranking). Una funció de classificació és un tipus de funció que assigna un rang o un número de fila a cada fila dins d'una partició del conjunt de resultats, segons un ordre especificat. Per tant en aquestes funcions L'ORDER BY de dins la partició té molt de sentit.

RANK	Retorna el rang de la fila actual dins d'una partició definida. Si hi ha una o més files que comparteixen el mateix valor de classificació, alguns números de classificació s'ometran de la seqüència. Té molt sentit en aquesta funció afegir l'ORDER BY. Per exemple si hi ha dues files empatades per a la segona classificació, la seqüència de classificació serà 1,2,2,4,...
DENSE_RANK	Assigna un rang a totes les files de la partició segons la clàusula ORDER BY. Assigna el mateix rang a les files amb valors iguals. Si dues o més files tenen el mateix rang, no hi haurà buits en la seqüència de valors classificats. Podríem dir que és el mateix que RANK, però que no contindrà buits. Per exemple: 1,2,2,3,4....
PERCENT_RANK	Calcula el rang percentil d'una fila d'una partició o conjunt de resultats. Els valors van de 0 a 1.
ROW_NUMBER	Assigna un enter seqüencial a cada fila de la partició.
NTILE	Divideix una partició en N grups (cubs), assigna a cada fila de la partició el seu número de cub i retorna el número de cub de la fila actual dins de la seva partició. Per exemple, si N és 4, NTILE() divideix les files en quatre cubs. Si N és 100, NTILE() divideix les files en 100 cubs. Aquesta funció es podria assimilar els quartils estadístics d'un box plot quan N=4. Estarem dividint les dades a on cada grup inclourà aprox. El 25% de les dades.

Exemple de RANK

Volem obtenir el rànquing dels salaris per cada departament.

```
SELECT empleat_id, nom, cognoms, salari, departament_id,
       RANK() OVER (
         PARTITION BY departament_id ORDER BY salari
       ) AS salari_rank
FROM empleats
WHERE departament_id IS NOT NULL
ORDER BY departament_id;
```

empleat_id	nom	salari	departament_id	salari_rank
200	Jennifer	4400.00	10	1
202	Pat	6000.00	20	1
201	Michael	13000.00	20	2
144	Peter	2500.00	50	1
143	Randall	2600.00	50	2
142	Curtis	3100.00	50	3
141	Trenna	3500.00	50	4
124	Kevin	5800.00	50	5
101	Neena	17000.00	90	1
102	Lex	17000.00	90	1
100	Steven	24000.00	90	3
...				

Atenció! En aquest exemple hi ha el cas de dos empleats del departament_id 90 (Neena, Lex) que estan empatats amb un salari de 17000 i el rang del salari és 1. El següent empleat d'aquest departament amb un altre salari és Steven que és 3.

Exemple amb DENSE_RANK

Mirem el mateix exemple, però utilitzant DENSE_RANK

```
SELECT empleat_id, nom, cognoms, salari, departament_id,
       DENSE_RANK() OVER (
         PARTITION BY departament_id ORDER BY salari
       ) AS salari_rank
FROM empleats
WHERE departament_id IS NOT NULL
ORDER BY departament_id;
```

empleat_id	nom	salari	departament_id	salari_rank
200	Jennifer	4400.00	10	1
202	Pat	6000.00	20	1
201	Michael	13000.00	20	2
144	Peter	2500.00	50	1
143	Randall	2600.00	50	2
142	Curtis	3100.00	50	3
141	Trenna	3500.00	50	4
101	Neena	17000.00	90	1
102	Lex	17000.00	90	1
100	Steven	24000.00	90	2
...				

Amb DENSE_RANK veiem que es similar a RANK, però sense salt en cas d'empat. Veiem que el departament_id 90 tenim un rang de salari 1,1,2 perquè els empleats 101 i 102 comparteixen el mateix salari. En canvi amb RANK el rang era 1,1,3

Exemple amb ROW_NUMBER

Mirem el mateix exemple, però utilitzant ROW_NUMBER

```
SELECT empleat_id, nom, cognoms, salari, departament_id,
       row_number() OVER (
         PARTITION BY departament_id ORDER BY salari
       ) AS salari_row
FROM empleats
WHERE departament_id IS NOT NULL
ORDER BY departament_id;
```

empleat_id	nom	salari	departament_id	salari_row
200	Jennifer	4400.00	10	1
202	Pat	6000.00	20	1
201	Michael	13000.00	20	2
144	Peter	2500.00	50	1
143	Randall	2600.00	50	2
142	Curtis	3100.00	50	3
141	Trenna	3500.00	50	4
124	Kevin	5800.00	50	5
101	Neena	17000.00	90	1
102	Lex	17000.00	90	2
100	Steven	24000.00	90	3
...				

Amb ROW_NUMBER assigna un número de fila únic dins de cada departament sense tenir en compte els empats. Per exemple en el cas del departament_id 90 tenim un row number seqüencial 1,2,3 encara que hi hagin els empleats 101 i 102 amb el mateix salari. Amb aquest exemple no podem controlar el número assignat. Si volguéssim controlar l'ordre s'ha de fer a través de l'ORDER BY de dins de la funció de finestra.

```
SELECT empleat_id, nom, cognoms, salari, departament_id,
       ROW_NUMBER() OVER (
         PARTITION BY departament_id ORDER BY salari, cognom
       ) AS salari_row
FROM empleats
WHERE departament_id IS NOT NULL
ORDER BY departament_id;
```

empleat_id	nom	salari	departament_id	salari_row
200	Jennifer	4400.00	10	1
202	Pat	6000.00	20	1
201	Michael	13000.00	20	2
144	Peter	2500.00	50	1
143	Randall	2600.00	50	2
142	Curtis	3100.00	50	3
141	Trenna	3500.00	50	4
124	Kevin	5800.00	50	5
102	Lex	17000.00	90	2
101	Neena	17000.00	90	1
100	Steven	24000.00	90	3
...				

Exemple amb PERCENT_RANK

```

SELECT empleat_id, nom, cognoms, salari, departament_id,
       PERCENT_RANK() OVER (
         PARTITION BY departament_id ORDER BY salari
       ) AS salari_perc
FROM empleats
WHERE departament_id IS NOT NULL
ORDER BY departament_id;

```

empleat_id	nom	salari	departament_id	salari_perc
200	Jennifer	4400.00	10	0
202	Pat	6000.00	20	0
201	Michael	13000.00	20	1
144	Peter	2500.00	50	0
143	Randall	2600.00	50	0.25
142	Curtis	3100.00	50	0.5
141	Trenna	3500.00	50	0.75
124	Kevin	5800.00	50	1
101	Neena	17000.00	90	0
102	Lex	17000.00	90	0
100	Steven	24000.00	90	1
...				

En aquí podem veure que la funció de finestra va distribuint les dades del salari per cada departament.

Exemple amb NTILE

Utilitzant el mateix exemple, però sense fer les particions per `departament_id` amb `NTILE (4)` veiem en quin quartil està cada empleat segons el seu salari.

```
SELECT empleat_id, nom, cognoms, salari, departament_id,
       NTILE(4) OVER (
         ORDER BY salari
       ) AS quartil
FROM empleats
WHERE departament_id IS NOT NULL
ORDER BY salari;
```

empleat_id	nom	cognoms	salari	quartil
144	Peter	Vargas	2500.00	1
143	Randall	Matos	2600.00	1
142	Curtis	Davies	3100.00	1
141	Trenna	Rajs	3500.00	1
107	Diana	Lorentz	4200.00	1
200	Jennifer	Whalen	4400.00	2
124	Kevin	Mourgos	5800.00	2
202	Pat	Fay	6000.00	2
104	Bruce	Ernst	6000.00	2
206	William	Gietz	8300.00	2
176	Jonathan	Taylor	8600.00	3
207	Sarah	Connor	9000.00	3
103	Alexander	Hunold	9000.00	3
149	Eleni	Zlotkey	10500.00	3
174	Ellen	Abel	11000.00	3
205	Shelley	Higgins	12000.00	4
201	Michael	Hartstein	13000.00	4
102	Lex	De Haan	17000.00	4
101	Neena	Kochhar	17000.00	4
100	Steven	King	24000.00	4

En aquest exemple es veu que tenim els empleats dividits en 4 grups i `NTILE` ens ha dividit els empleats en funció del seu salari dins d'aquests 4 grups.

Aquí els grups tenen la mateixa quantitat exacte d'empleats perquè tenim 20 empleats que no tenen departament així que els grups que fa són de 5 empleats en funció del salari.

Atenció

Quan s'utilitza la funció `NTILE (n)` per dividir un conjunt de dades en `n` grups iguals, si el nombre total de files no és divisible per `n`, el SGBD fa la distribució següent:

- Els primers grups reben **una fila addicional** per compensar la diferència.
- Això garanteix que **la mida dels grups difereixi com a màxim en una fila**

Per exemple en un total de 20 files si apliquem `NTILE(3)` obtindrem 2 grups de 7 i 1 grup de 6. Si fem l'operació matemàtica $20/3 = 6,6666..$ cada grup li tocaria 6 elements, però el que fem és col·locar 7 elements als dos primers grups i el tercer es quedarà en 6.

Taules d'exemple

Quantitat de files	Nº Grups	Divisió	Resultat de grups
20	3	66.6666	2 grups 7 1 grup 6
100	3	33.333	1 grup 34 2 grups 33
33	4	8.25	1 grups 9 3 grup 8
33	6	5.5	3 grups 6 3 grups 5

3.8.4. Llista de funcions de finestra VALUE

Amb aquestes funcions de finestra també és important la clàusula ORDER BY de dins la funció de finestra perquè determinarà l'ordre dels elements.

LAG	Busca el valor de fila anterior per a una columna especificada dins de la partició actual.
LEAD	Busca un valor de fila posterior per una columna especificada dins de la partició actual. El mateix que el LAG, però aquest cop obtenint el valor de la fila següent.
FIRST_VALUE	Retorna el valor d'una columna especificada per a la primera fila de la partició actual.
LAST_VALUE	Retorna el valor d'una columna especificada per a l' última fila de la partició actual.
NTH_VALUE	Retorna el valor d'una columna especificada per a la fila n de la partició actual, on n és definida per l'usuari.

Exemple de LAG

```
SELECT empleat_id, nom, cognoms, salari,
       LAG(salari) OVER (
         ORDER BY empleat_id
       ) AS salari_ant
FROM empleats;
```

empleat_id	nom	cognoms	salari	salari_ant
100	Steven	King	24000.00	NULL
101	Neena	Kochhar	17000.00	24000.00
102	Lex	De Haan	17000.00	17000.00
103	Alexander	Hunold	9000.00	17000.00
...				

Podem veure que a cada empleat hi ha una columna anomenada salari_ant (anterior) a on es mostra el salari de la fila de l'empleat anterior. En el cas de la primera fila aquest camp valdrà NULL.

Si haguéssim utilitzat PARTITION BY el salari anterior seria l'anterior dins de cada subgrup.

Exemple de LEAD

```
SELECT empleat_id, nom, cognoms, salari,
       LEAD(salari) OVER (
         ORDER BY empleat_id
       ) AS salari_seg
FROM empleats;
```

empleat_id	nom	cognoms	salari	salari_seg
100	Steven	King	24000.00	17000.00
101	Neena	Kochhar	17000.00	17000.00
102	Lex	De Haan	17000.00	9000.00
103	Alexander	Hunold	9000.00	6000.00
...				
205	Shelley	Higgins	12000.00	8300.00
206	William	Gietz	8300.00	9000.00
207	Sarah	Connor	9000.00	NULL

Veiem que ara el que ens mostra la columna salari_seg (següent) és el valor del salari de la fila següent. Aquí és la última fila la que conté aquest valor a NULL.

Exemple de FIRST_VALUE

Per exemple volem obtenir tots els empleats de tal manera que mostrem el seu, nom cognoms, departament, salari juntament amb el salari més petit del seu propi departament.

```
SELECT empleat_id, nom, cognoms, departament_id AS dep, salari,
       FIRST_VALUE(salari) OVER (
                               PARTITION BY departament_id
                               ORDER BY salari
                               ) AS salari_min
FROM   empleats
WHERE  departament_id IS NOT NULL
ORDER BY dep;
```

empleat_id	nom	cognoms	dep	salari	salari_min
200	Jennifer	Whalen	10	4400.00	4400.00
202	Pat	Fay	20	6000.00	6000.00
201	Michael	Hartstein	20	13000.00	6000.00
144	Peter	Vargas	50	2500.00	2500.00
143	Randall	Matos	50	2600.00	2500.00
142	Curtis	Davies	50	3100.00	2500.00
141	Trenna	Rajs	50	3500.00	2500.00
124	Kevin	Mourgos	50	5800.00	2500.00
103	Alexander	Hunold	60	9000.00	4200.00
...					

En aquest exemple veiem que el departament 50 que té 5 empleats a la columna salari_min es mostra el salari mínim del departament que en aquest cas és de 2.500.

En aquest segon exemple podem veure que podem utilitzar una funció de finestra operant-la amb una columna que tinguem.

```
SELECT empleat_id, nom, cognoms, departament_id AS dep, salari,
       FIRST_VALUE(salari) OVER (
                               PARTITION BY departament_id
                               ORDER BY salari
                               ) AS salari_min,
       salari - FIRST_VALUE(salari) OVER (
                               PARTITION BY departament_id
                               ORDER BY salari
                               ) AS dif
FROM   empleats
WHERE  departament_id IS NOT NULL
ORDER BY dep;
```

empleat_id	nom	cognoms	dep	salari	salari_min	dif
200	Jennifer	Whalen	10	4400.00	4400.00	0.00
202	Pat	Fay	20	6000.00	6000.00	0.00
201	Michael	Hartstein	20	13000.00	6000.00	7000.00
144	Peter	Vargas	50	2500.00	2500.00	0.00
143	Randall	Matos	50	2600.00	2500.00	100.00
142	Curtis	Davies	50	3100.00	2500.00	600.00
141	Trenna	Rajs	50	3500.00	2500.00	1000.00
124	Kevin	Mourgos	50	5800.00	2500.00	3300.00

```
|      103 | Alexander | Hunold      | 60 | 9000.00 | 4200.00 | 4800.00 |
...

```

Exemple de LAST_VALUE

Igual que en l'exemple anterior del FIRST_VALUE volem obtenir tots els empleats juntament amb el salari més alt. En aquest cas només canviarem LAST_VALUE.

```
SELECT empleat_id, nom, cognoms, departament_id dep, salari,
       LAST_VALUE(salari) OVER (
                                PARTITION BY departament_id
                                ORDER BY salari ASC
                                ) AS salari_max
FROM empleats
WHERE departament_id IS NOT NULL
ORDER BY dep;
```

empleat_id	nom	cognoms	dep	salari	salari_max
200	Jennifer	Whalen	10	4400.00	4400.00
202	Pat	Fay	20	6000.00	6000.00
201	Michael	Hartstein	20	13000.00	13000.00
144	Peter	Vargas	50	2500.00	2500.00
143	Randall	Matos	50	2600.00	2600.00
142	Curtis	Davies	50	3100.00	3100.00
141	Trenna	Rajs	50	3500.00	3500.00
124	Kevin	Mourgos	50	5800.00	5800.00
103	Alexander	Hunold	60	9000.00	9000.00
...					

Atenció

En aquest cas veiem que `salari_max` no retorna el valor que volem sinó que ens retorna el valor de la fila actual.

Amb `LAST_VALUE` Cal que utilitzem la clàusula del marc de finestra (*Window Frame*) perquè agafi els valors de la partició correctament.

Per exemple en aquest cas hauríem d'incloure `RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING` (sense límit precedent i sense límit posterior).

Tal i com hem avisat hem d'incloure el marc de finestra.

```
SELECT empleat_id, nom, cognoms, departament_id dep, salari,
       LAST_VALUE(salari) OVER (
                                PARTITION BY departament_id
                                ORDER BY salari ASC
                                RANGE BETWEEN
                                    UNBOUNDED PRECEDING AND
                                    UNBOUNDED FOLLOWING
                                ) AS salari_max
FROM empleats
WHERE departament_id IS NOT NULL
ORDER BY dep;
```

empleat_id	nom	cognoms	dep	salari	salari_max
200	Jennifer	Whalen	10	4400.00	4400.00
202	Pat	Fay	20	6000.00	13000.00
201	Michael	Hartstein	20	13000.00	13000.00
144	Peter	Vargas	50	2500.00	5800.00
143	Randall	Matos	50	2600.00	5800.00
142	Curtis	Davies	50	3100.00	5800.00
141	Trenna	Rajs	50	3500.00	5800.00
124	Kevin	Mourgos	50	5800.00	5800.00
103	Alexander	Hunold	60	9000.00	9000.00
...					

Window Frames

En l'exemple anterior ja hem vist que tenim un altra clàusula dins de les funcions de finestra que és la definició de finestra (Window definition):

- RANGE: Opera sobre valors (p. ex., valors de la columna ordenada).
- ROWS: Opera sobre files físiques (posicions relatives).

Les clàusules específiques utilitzades seran:

- UNBOUNDED PRECEDING: Fa referència a la primera fila de la partició
- N PRECEDING: Fa referència a la fila que està a **N** files abans de la fila actual.
- CURRENT ROW: Es refereix a la fila actual
- N FOLLOWING: Es refereix a la fila que està **N** files després de la fila actual.
- UNBOUNDED FOLLOWING: Es refereix a l'última fila de la partició.

3.8.5. Named Windows

Podem definir les funcions de finestra i donar-li un nom.

```
WINDOW window_name AS (window_spec)
    [, window_name AS (window_spec)] ...
```

A on window_spec és:

```
[window_name] [partition_clause] [order_clause] [frame_clause]
```

Exemple: Imaginem que volem obtenir per cada empleat que tingui assignat un departament i una feina el salari mig del departament el qual pertany i la mitjana salarial de la feina assignada per aquell empleat.

En aquest cas volem crear dues finestres. Una per departament_id i l'altra per feina_codi.

```
SELECT empleat_id, nom, cognoms, departament_id dep,
       feina_codi,
       salari,
       AVG(salari) OVER w_dep AS avg_dep,
       AVG(salari) OVER w_feina AS avg_feina
FROM empleats
WHERE departament_id IS NOT NULL
WINDOW w_dep AS ( PARTITION BY departament_id ),
       w_feina AS ( PARTITION BY feina_codi )
ORDER BY dep;
```

id	nom	cognoms	dep	feina_codi	salari	avg_dep	avg_feina
200	Jennifer	Whalen	10	AD_ASST	4400.00	4400.00	4400.00
202	Pat	Fay	20	MK_REP	6000.00	9500.00	6000.00
201	Michael	Hartstein	20	MK_MAN	13000.00	9500.00	13000.00
124	Kevin	Mourgos	50	ST_MAN	5800.00	3500.00	5800.00
141	Trenna	Rajs	50	ST_CLERK	3500.00	3500.00	2925.00
142	Curtis	Davies	50	ST_CLERK	3100.00	3500.00	2925.00
143	Randall	Matos	50	ST_CLERK	2600.00	3500.00	2925.00
144	Peter	Vargas	50	ST_CLERK	2500.00	3500.00	2925.00
104	Bruce	Ernst	60	IT_PROG	6000.00	6400.00	6400.00
103	Alexander	Hunold	60	IT_PROG	9000.00	6400.00	6400.00
107	Diana	Lorentz	60	IT_PROG	4200.00	6400.00	6400.00
149	Eleni	Zlotkey	80	SA_MAN	10500.00	10033.33	10500.00
174	Ellen	Abel	80	SA_REP	11000.00	10033.33	9800.00
176	Jonathan	Taylor	80	SA_REP	8600.00	10033.33	9800.00
101	Neena	Kochhar	90	AD_VP	17000.00	19333.33	17000.00
102	Lex	De Haan	90	AD_VP	17000.00	19333.33	17000.00
100	Steven	King	90	AD_PRES	24000.00	19333.33	24000.00
205	Shelley	Higgins	110	AC_MGR	12000.00	9766.67	12000.00
206	William	Gietz	110	AC_ACCOUNT	8300.00	9766.67	8650.00
207	Sarah	Connor	110	AC_ACCOUNT	9000.00	9766.67	8650.00

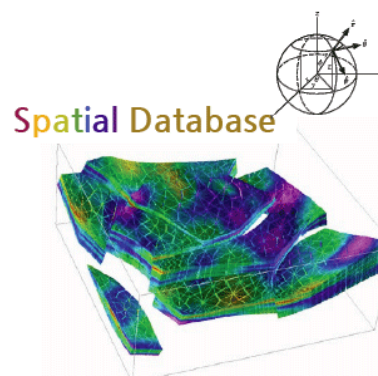
Fixem-nos amb les tres últimes files que corresponen en els treballadors del departament 110. Aquí la mitjana salarial és de 9.766,67, però com que en aquest departament hi ha empleats amb diferents feines hi ha diferents mitjanes per cada feina. La mitjana salarial dels AC_ACCOUNT és de 8.650 i la dels AC_MGR de 12.000.

Capítol 4. Base de dades de tipus espacial

4.1. Introducció

Els SGBDR Espacial son sistemes que permeten processar dades de tipus espacial. Cada vegada més i alguns gestors en més o menys mesura incorporen característiques o afegitons per tractar aquest tipus d'informació.

Per donar aquest tipus de servei el SGBD incorporen nous tipus de dades geomètrics (PUNTS, RECTES, POLÍGONS), certes funcions pel tractament d'aquests tipus nous de dades per buscar inclusions, interseccions, longituds, etc.... Un altre cosa a incorporar és nous tipus d'índexs per millorar l'eficiència de certes consultes sobre aquestes dades.



Explicacions i xifres que fan referència a temes espacials el podem trobar en aquest enllaç: <http://www.sharpgis.net/post/2007/05/Spatial-references2c-coordinate-systems2c-projections2c-datums2c-ellipsoids-e28093-confusing.aspx>.

Aquest tipus d'objectes i funcions són s'han utilitzat des dels seus inicis en l'àmbit del transport aeri per tal de calcular rutes aèries, però l'aparició de Google Maps va transformar l'ús d'aquesta tecnologia donant-li un ús civil per buscar els restaurant a 10 kms a la rodona, calcular rutes amb el cotxe d'un punt a un altre mitjançant GPS. Cal que els desenvolupadors i tècnics de sistemes entenguin la manera d'incorporar informació geogràfica dins dels seus conjunts de dades per donar més informació útil.

SGBR Espacials

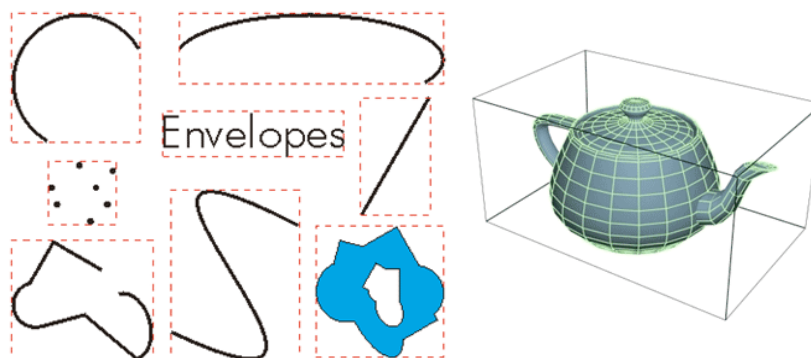
Els SGBDR més representatius són PostgreSQL per la part d'open source i MS SQL Server i Oracle per la part de sistemes comercials. MySQL encara està enrere respecte els altres, però podem dir que a partir de la versió 5.7 MySQL incorpora certes millores en aquest aspecte.

Article de comparació d'aquests SGBD: [article 1](#), [article 2](#)

Índexs Espacials

Un dels conceptes més utilitzats en els índexs de tipus spaial és el MBR (Minum Boun Rectangle), també conegut com a caixa/quadrat delimitador.

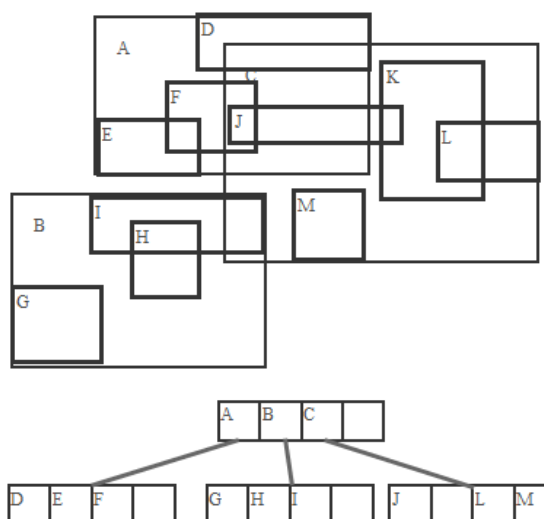
Les imatges següents mostren aquest MBR per les diferents figures 2D i 3D.



Aquest MBR ens permet visualitzar/comprovar relacions d'equivalència/grandària i poder expressar en l'espai com es creuen, dins o fora. S'utilitza el MBR per la computació lineal i facilitar els càlculs.

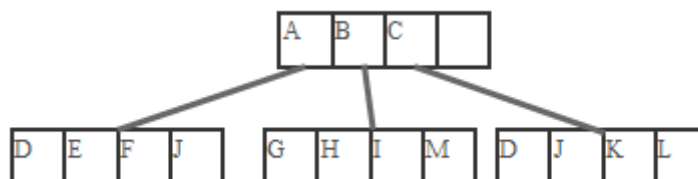
El MBR també serveix per crear estructures d'índex de dades per millorar temps de resposta. Les estructures d'arbres B i B+ no es poden utilitzar.

A la figura següent hi ha l'exemple d'un arbre R i per exemple si busquem un punt aleatori dins d'aquests MBR, imaginem un punt dins de D, G o M l'arrel del node fulla es pot trobar immediatament.



En canvi si el punt que volem buscar està a la intersecció entre F i J, llavors visitarem el node arrel A per primera vegada, després F i a continuació passarem pel node arrel altre cop per seleccionar C i J.

Per això hi ha una estructura anomenada arbre R+ que soluciona aquest tipus de problema. Com la següent:



La configuració de dades espacials de D a M és el mateix que en la configuració de l'arbre R. No obstant, l'àrea A i C és diferent a que les àrees de solapament es troben al mateix

nilli podem veure que J i D son redundants. Per tant si cerquem un punt dins la zona F i J no cal anar a buscar C, ja ho tenim en la mateix branca de l'arbre.

4.2. MySQL GIS

Referències respecte MySQL GIS:

- <http://dev.mysql.com/doc/refman/8.0/en/spatial-extensions.html>
- <http://dev.mysql.com/doc/refman/8.0/en/spatial-convenience-functions.htm>
- <http://mysqlserverteam.com/why-boost-geometry-in-mysql/>
- <http://mysqlserverteam.com/making-use-of-boost-geometry-in-mysql-gis/>
- <http://mysqlserverteam.com/innodb-spatial-indexes-in-5-7-4-lab-release/>

4.2.1. Tipus de dades espacials

<https://dev.mysql.com/doc/refman/8.0/en/spatial-datatypes.html>

<https://dev.mysql.com/doc/refman/8.0/en/populating-spatial-columns.html>

MySQL té diferents tipus de dades que corresponent a classes OpenGIS que podem utilitzar com a tipus de dades per columnes de les nostres taules

- GEOMETRY: geomètric
- POINT: punt (dos valors)
- LINESTRING: línia
- POLYGON

El tipus GEOMETRY pot guardar valors geomètrics de qualsevol tipus

Exemple de creació d'una taula:

```
CREATE TABLE espai (  
    punt POINT,  
    segment LINESTRING,  
    poligon POLYGON  
);
```

Altres tipus de dades que permet MySQL són les col·leccions de valors:

- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION

Point Class

Un punt geomètric és una localització dins d'un espai de coordenades.

Per exemple un punt en el mapamundi seria una ciutat. O dins d'una ciutat una parada de bus.

Propietats:

- Coordenada X
- Coordenada Y
- Té dimensió zero !

Per poder introduir punts en un camp de la taula ens podem ajudar de les funcions:

`ST_GeomFromText` i `ST_PointFromText`. La primera ens serveix per introduir qualsevol element geogràfic i per tant interpreta la cadena de caràcters que li passem per paràmetre. En canvi la segona es més específica i només serveix per introduir punts en un camp de tipus `POINT`.

Exemple d'introducció de punts a la taula `espai`

```
// Inserció del punt 1,1
INSERT INTO espai (punt) VALUES (ST_GeomFromText('POINT(1 1)'));

// Inserció del punt 2,1
SET @punt = ST_GeomFromText('POINT(2 1)');
INSERT INTO ESPAI (punt) VALUES (@punt);

// Inserció del punt 3,1
SET @g = 'POINT(3 1)';
INSERT INTO espai (punt) VALUES (ST_PointFromText(@g));
```

LineString Class

A `LineString` és una forma geomètrica de dues dimensions. Bàsicament és la interpolació de punts (~segment). Aquesta classe ens permet representar: rius, carrers, etc...

Propietats:

- Té les coordenades dels segments definits per cada consecució de cada parell de punts.
- És un línia si només consisteix en exactament dos punts

Podem utilitzar les funcions `ST_GeomFromText` i `ST_LineStringFromText`.

Exemple d'introducció de segments a la taula `espai`

```
// Inserció de la línia formada pels punts (0,0), (1,1), (2,2)
INSERT INTO espai (segment)
VALUES (ST_GeomFromText('LINESTRING(0 0,1 1,2 2)'));

// Inserció de la línia formada pels punts (0,0), (-1,-1), (-2,-2)
SET @g = 'LINESTRING(-0 -0,-1 -1,-2 -2)';
INSERT INTO espai (segment) VALUES (ST_LineStringFromText(@g));
```

Polygon Class

Un polígon és una superfície plana que representa una geometria amb diversos costats. Es defineix per un únic límit exterior i zero o més límits interiors, on cada límit interior defineix un forat al polígon.

Permet representar regions: boscos, districtes, etc..

Propietats:

- El límit d'un Polígon consisteix en un conjunt d'objectes LinearRing (és a dir, objectes LineString que són tant simples com tancats) que componen els seus límits exterior i interior.
- Un Polígon no té anells que es creuen. Els anells en el límit d'un Polígon poden tallar en un punt, però només com una tangent.
- Un Polígon no té línies, pics o punxades.
- Un polígon té un interior que és un conjunt de punts connectats.
- Un polígon pot tenir forats. Exterior d'un polígon amb forats no està connectat. Cada orifici defineix un component connectat l'exterior.

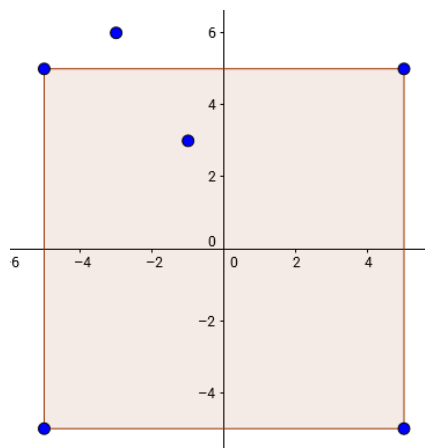
Exemple d'introducció de punts a la espai

Compte amb els espais i les comes. Cal tancar el polígon (l'últim punt a de ser l'inicial)

```
// Inserció del polígon format pels punts:
(5 5), (-5 5), (-5 -5), (5 -5)
SET @poligon = 'POLYGON((5 5,-5 5,-5 -5,5 -5,5 5))';
INSERT INTO espai (poligon)
VALUES(ST_GeomFromText(@poligon));

// Inserció del punt -1,3
SET @punt1 = ST_GeomFromText('POINT(-1 3)');
INSERT INTO ESPAI (punt) VALUES(@punt1);

// Inserció del punt -3,6
SET @punt2 = 'POINT(-3 6)';
INSERT INTO espai (punt) VALUES (ST_PointFromText(@g));
```



Mitjançant la funció *MBContains* podem preguntar si un punt és a dins del polígon.

```
//Comprovem  
punt1 està  
del polígon
```

Creat mitjançant

<https://www.math10.com/en/geometry/geogebra/geogebra.html>

```
que el  
dins  
SELECT
```

```
IF(MBContains(@poligon,@punt1),'dins','fora');  
  
//Comprovem que el punt2 no està dins del polígon  
SELECT IF(MBContains(@poligon,@punt2),'dins','fora');
```

4.2.2. Funcions de tipus espacial

MySQL proporciona un seguit de funcions per tal de poder treballar amb elements espacials (punts, línies, polígons, etc...).

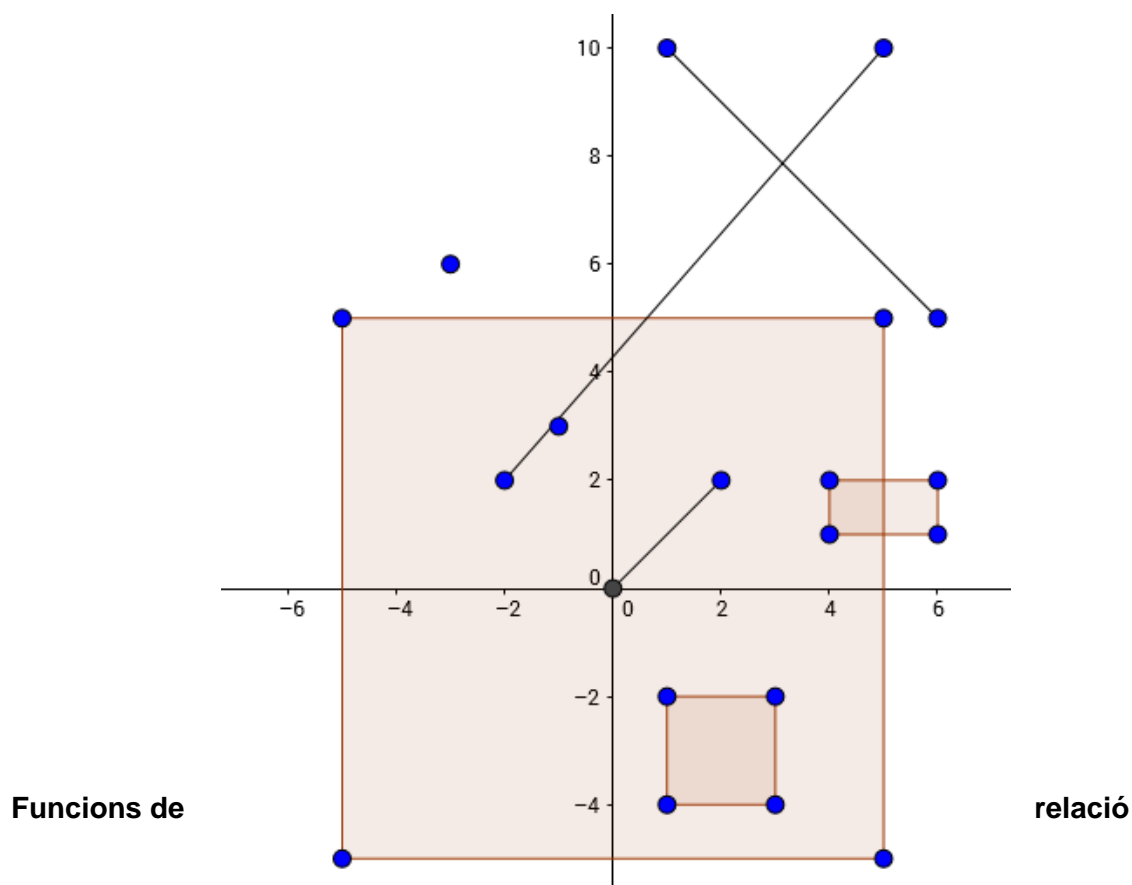
Aquestes funcions ens fan la feina molt més fàcil alhora de saber si un punt està o no dins d'un polígon, donats dos punts que ens retorni el polígon (caixa) que formen, etc....

Aquestes funcions és una de les gran millores que ha aportat la versió 5.7 i a continuació en detallarem unes quantes.

Important: Compte amb versions anteriors de 5.7 que els noms de les funcions a partir d'aquesta versió han quedat obsoletes

<https://dev.mysql.com/doc/refman/8.0/en/spatial-function-reference.html>

Els exemples de funcions mirarem de partir d'un espai amb les següents figures geomètriques:



<https://dev.mysql.com/doc/refman/8.0/en/spatial-relation-functions-object-shapes.html>

L'especificació d'OpenGIS defineix les següents funcions per comprovar la relació entre dos valors geomètrics (g1 i g2).

Normalment aquestes funcions retorna els valors 1 i 0 indicant cert(*true*) o fals(*false*) respectivament excepte ST_Distance(), que retornen la distància.

A partir de MySQL 5.7.5, aquestes funcions suporten com a argument els tipus definits per l'especificació de *Open Geospatial Consortium*.

Funció	Descripció
ST_Crosses(g1, g2)	Determina si dos formes geomètriques es creuen. Retornarà NULL si un argument és un polígon i l'altre un punt.
ST_Distance(g1, g2)	Retorna la distància entre dos elements geomètrics. A partir de MySQL 5.7.5, aquesta funció processa col·leccions geomètriques retornant la distància més curta entre tots els components.
ST_Equals(g1, g2)	Retorna 1 o 0 indicant si els dos elements geomètrics són iguals (a nivell espacial).
ST_Intersects(g1, g2)	Retorna 1 o 0 indicant si g1 encreua amb g2.
ST_Overlaps(g1, g2)	Es sobreposen

ST_Touches(g1, g2)	Es toquen
ST_Within(g1, g2)	Retorna 1 o 0 indicant si g1 està dins de g2. Podríem dir que és l'oposat de ST_Contains()

Funcions de Contingut / MBR (*Minimum Bounding Rectangles*)

<https://dev.mysql.com/doc/refman/8.0/en/spatial-relation-functions-mbr.html>

Aquestes funcions també ens comproven la relació entre dos objectes geomètrics tenint en compte el seu rectangle delimitador mínim.

Funció	Descripció
MBRContains(g1, g2)	Retorna 1 o 0 indicant si el MBR de g1 conté el MBR de g2. (g1 conté g2?)
MBRWithin(g1,g2)	Retorn 1 o 0 indicant si el MBR de g1 està a dins de g2.(g1 està a dins de g2?)
MBRDisjoint (g1, g2)	Retorna 1 o 0 indicant si els dos MBR de les dues figures geomètriques són disjunts, o sigui, no interseccionen.
MBRIntersects (g1, g2)	Retorna 1 o 0 indicant si els dos MBR de les dues figures geomètriques intersecccionen.
MBREquals (g1, g2)	Retorna 1 o 0 indicant si el MBR de les dues figures geomètriques són el mateix.
MBROverlaps(g1,g2)	Retorna 1 o 0 indicant si el MBR de les dues figures geomètriques es sobreposen. El fet de sobreposar-se és que el resultat de la intersecció doni la mateix dimensió.
MBRTouches(g1,g2)	Retorna 1 o 0 indicant si el MBR es toquen

Exemples:

```
-- MBRContains

mysql> SET @g1 = ST_GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = ST_GeomFromText('Point(1 1)');
mysql> SELECT MBRContains(@g1,@g2), MBRWithin(@g2,@g1);
+-----+-----+
| MBRContains(@g1,@g2) | MBRWithin(@g2,@g1) |
+-----+-----+
| 1 | 1 |
+-----+-----+
```

Altres funcions spaials útils

<https://dev.mysql.com/doc/refman/8.0/en/spatial-convenience-functions.html>

Funció	Descripció
ST_Distance_Sphere(g1, g2 [,radius])	Retorna la distància esfèrica mínima entre dos punts. Els càlculs utilitzen la forma d'una terra esfèrica. Per defecte el radi és 6.370.986 metres
ST_IsValid(g)	Comprova si una forma geomètrica passada per paràmetre és vàlida.
ST_MakeEnvelope(pt1, pt2)	Retorna el rectangle que es forma al voltant de dos punts. El càlcul es calcula mitjançant coordenades Cartesianes.
ST_Simplify(g,max_distance)	Redueix el número de punts en una línia aplicant l'algorisme de Douglas-Peucker

Altres funcions per tractament de XML

<https://dev.mysql.com/doc/refman/8.0/en/xml-functions.html>

Capítol 5. JSON

5.1. Introducció

Part introductòria d'un fitxer JSON

5.2. MySQL JSON

5.2.1. Tipus de dades en format JSON

A partir de la versió 5.7.8 MySQL incorpora de forma nativa el format de dades JSON definit per [RFC 7159](https://rfc7159.org/).

<https://dev.mysql.com/doc/refman/8.0/en/json.html>

Gràcies a aquesta incorporació tindrem:

- Validació automàtica de document JSON guardat en columnes de tipus JSON.
- Optimització en l'emmagatzematge. El document JSON són transformats en un format intern que permet un ràpid accés en els elements del document.

En el MySQL els valor JSON són escrits com cadenes de caràcters. MySQL realitza el parser d'aquesta cadena de caràcters.

Exemples:

```
mysql> CREATE TABLE t1 (jdoc JSON);
Query OK, 0 rows affected (0.20 sec)

mysql> INSERT INTO t1 VALUES('{"key1": "value1", "key2": "value2"}');
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO t1 VALUES('[1, 2, ');
ERROR 3140 (22032) at line 2: Invalid JSON text:
"Invalid value." at position 6 in value (or column) '[1, 2, '.
```

5.2.2. JSON Path Syntax

<https://dev.mysql.com/doc/refman/8.0/en/json-path-syntax.html>

5.2.3. Funcions JSON

Per tal de manipular aquests tipus de document MySQL incorpora una sèrie de funcions:

<https://dev.mysql.com/doc/refman/8.0/en/json-functions.html>

Funcions importants:

JSON_TYPE

Ens retorna el tipus de dades JSON passat per paràmetre. Si el paràmetre no és un JSON ens dóna una error.

Exemples:

```
mysql> SELECT JSON_TYPE('["a", "b", 1]');
+-----+
| JSON_TYPE('["a", "b", 1]') |
+-----+
| ARRAY                        |
+-----+

mysql> SELECT JSON_TYPE('"hello"');
+-----+
| JSON_TYPE('"hello"') |
+-----+
| STRING                |
+-----+

mysql> SELECT JSON_TYPE('hello');
ERROR 3146 (22032): Invalid data type for JSON data in argument 1
to function json_type; a JSON string or JSON type is required.
```

JSON_ARRAY

Ens crea un array JSON a partir d'una llista de valors.

Exemples:

```
mysql> SELECT JSON_ARRAY('a', 1, NOW());
+-----+
| JSON_ARRAY('a', 1, NOW()) |
+-----+
| ["a", 1, "2015-07-27 09:43:47.000000"] |
+-----+

mysql> SELECT JSON_ARRAY('Analista', 'Programador', 'Financer')
        AS Vector;
+-----+
| Vector |
+-----+
| ["Analista", "Programador", "Financer"] |
+-----+
```

JSON_OBJECT

Ens crea un objecte JSON a partir d'una llista de valors, indicant clau i valor.

Exemples:

```
mysql> SELECT JSON_OBJECT('key1', 1, 'key2', 'abc');
+-----+
| JSON_OBJECT('key1', 1, 'key2', 'abc') |
+-----+
| {"key1": 1, "key2": "abc"}           |
+-----+

SELECT JSON_OBJECT('nom', 'Robert'
                   , 'cognoms', 'Ventura Vall-llovera'
                   , 'edat', 22) AS persona;
+-----+
| persona
+-----+
| {"nom": "Robert", "edat": 22, "cognoms": "Ventura Vall-llovera"} |
+-----+
```

JSON_MERGE

Agafa dos o més documents JSON i retorna un document amb la combinació de tots:

```
mysql> SELECT JSON_MERGE(['a', 1], '{"key": "value"}');
+-----+
| JSON_MERGE(['a', 1], '{"key": "value"}') |
+-----+
| [a, 1, {"key": "value"}]                |
+-----+
```

JSON_SEARCH

[JSON_SEARCH](#)(json_doc, one_or_all, search_str[, escape_char[, path] ...])

Retorna el path

- [JSON_EXTRACT](#)(json_doc, path[, path] ...) / column->path

Capítol 6. Transaccions i concurrència

6.1. Gestió de transaccions

6.1.1. Introducció

Què passaria si enmig de l'execució d'una seqüència d'instruccions DML hi hagués alguna errada (de l'usuari o del sistema) que provoqués que les dades quedessin inconsistents?

Què passa si diferents usuaris o processos volen accedir o modificar les mateixes dades?

La resposta a aquesta pregunta no és senzilla. En tot cas el que cal fer sempre és intentar assegurar la consistència de les dades. Per això alguns SGBDR tenen mecanismes perquè això no succeeixi: **les transaccions**.

6.1.2. Transaccions

Una transacció és el conjunt d'instruccions SQL que s'executen consecutivament i es tracten com una sola instrucció que han de presentar les propietats anomenades ACID (**A**tomicity, **C**onsistency, **I**solation i **D**urability):

- **Atomicity** (atomicitat): La transacció s'ha d'executar com una unitat atòmica de treball, es a dir, o completa totes les modificacions de les dades o no en completa cap.
- **Consistency** (consistència): Les dades han de ser consistents abans de començar la transacció i després de la seva finalització. Per mantenir la consistència, durant la transacció s'han d'aplicar i complir els controls d'integritat, restriccions i desencadenadors establers. Si la transacció afecta a estructures internes del SGBD (metadades, índexs,) si la transacció es cancel·la cal retornar aquestes estructures a l'estat anterior.
- **Isolation**(aïllament): La transacció ha de estar aïllada dels canvis d'altres transaccions que poden actuar sobre les dades, per tal d'evitar l'ús de dades provisionals encara no confirmades. Això implica que la transacció ha de veure les dades en un estat previ o esperar fins la confirmació dels canvis realitzats per altres transaccions.
- **Durability** (durabilitat): Una vegada completada la transacció, els canvis a les dades seran permanents, amb independència de si hi hagués una fallada en el sistema. En altres paraules, quan una aplicació client rep la notificació de que una transacció s'ha completat es garanteix que els canvis són permanents.

Una transacció pot ser confirmada (**COMMIT**), si totes les operacions individuals s'han executat correctament o es tirarà endarrere (**ROLLBACK**) a la meitat de la seva execució si hi ha hagut algun problema.

<https://dev.mysql.com/doc/refman/8.0/en/sql-syntax-transactions.html>

Exemple d'una transacció:

Una transferència entre dos comptes bancaris ha d'incloure el dèbit a un compte corrent i el càrrec a un altre compte amb el mateix import.

Les dues accions s'han de realitzar correctament o incorrectament al mateix temps, de manera que el crèdit no es pot validar sense el dèbit.

```
UPDATE compte SET saldo = saldo + 500
  WHERE cuenta.id = 1;

UPDATE compte SET saldo = saldo - 500
  WHERE cuenta.id = 2;
```

Perquè varies operacions puguin ser considerades com a membres de la mateixa transacció és necessari determinar quins són els límits de la transacció. Indicant els punts d'inici i fi.

6.1.3. Control de transaccions en MySQL

En el SGBDR MySQL, una transacció s'inicia quan s'executa la sentència `BEGIN` o `START TRANSACTION` i finalitza quan es produeix alguna d'aquestes situacions:

- Es troba una instrucció `COMMIT` o `ROLLBACK`.
- Es troba una instrucció DDL (com per **Exemple**, un `CREATE/ALTER TABLE`).
- Es troba una instrucció DCL (*Data Control Language*).
- L'usuari abandona la sessió.
- Cau el sistema

Esquema:

```
BEGIN / START TRANSACTION
...
...instruccions DML...
....
COMMIT / ROLLBACK
```

Cal tenir present que qualsevol instrucció DDL o DCL produeix un `COMMIT` implícit, és a dir, totes les instruccions DML executades fins aquell moment passen a ser definitives.

El fet d'abandonar la sessió o que caigui el sistema produirà un `ROLLABCK` implícit.

Nota: altres SGBD com Oracle o SQL Server s'inicia una transacció cada vegada que s'executa una instrucció que necessita modificar dades (`INSERT`, `UPDATE` o `DELETE`). També es pot fer amb el SGBD MySQL si desactivem la variable `AUTOCOMMIT`.

COMMIT

La instrucció COMMIT confirma tots els canvis realitzats per la transacció i fa que siguin definitius i irrevocables.

Només cal utilitzar aquesta instrucció si estem d'acord amb els canvis. Cal estar completament segurs d'executar COMMIT ja que les instruccions executades poden afectar a milers de registres. A més a més el tancament correcte de la sessió produeix el COMMIT, tot i que sempre convé executar explícitament aquesta instrucció per tal d'assegurar-nos del que fem.

Hi ha alguns SGBDR (com MySQL/Sql Server/Oracle) que ofereixen mecanismes que permeten validar les transaccions de forma automàtica sense haver-ho d'indicar de forma explícita.

Amb MySQL, es pot configurar la variable AUTOCOMMIT. El valor d'aquesta es pot comprovar amb el SHOW, de la següent forma:

```
ORACLE> SHOW AUTOCOMMIT;
```

```
mysql> SHOW VARIABLES LIKE 'AUTOCOMMIT';
```

Per defecte en el MySQL el valor d'AUTOCOMMIT és ON, és a dir, qualsevol operació que es faci s'intentarà de fer un commit.

Podem canviar el valor d'aquesta variable mitjançant la sentència

```
mysql > SET AUTOCOMMIT = 0;
```

A partir d'aquest moment, qualsevol inserció, modificació i/o esborrat; **no** es validarà automàticament. **Compte! Només en la sessió activa.**

Per modificar de forma permanent o per defecte aquesta variable cal anar al fitxer de configuració del SGBD MySQL.

ROLLBACK

Aquesta instrucció permet tornar a l'estat anterior a l'inici de la transacció, normalment l'últim COMMIT, la última instrucció DDL o DCL o a l'inici de sessió.

Anul·la definitivament els canvis, motiu pel qual cal estar també molt segur d'executar aquesta operació.

Exemple de transacció:

```
SET AUTOCOMMIT=0;

BEGIN;

INSERT INTO empl(empl_num, empl_nom, empl_dept_num)
VALUES(1000, 'Pere', 10);

INSERT INTO empl(empl_num, empl_nom, empl_dept_num)
VALUES(2000, 'Joan', 20);

COMMIT;

INSERT INTO empl(empl_num, empl_nom, empl_dept_num)
VALUES(3000, 'Albert', 10);

INSERT INTO empl(empl_num, empl_nom, empl_dept_num)
VALUES(4000, 'Manel', 40);

ROLLBACK;
```

En l'**Exemple** anterior s'afegeixen els empleats Pere i Joan, però **no** l'Albert i en Manel, ja que la instrucció de validació COMMIT s'executa abans de la instrucció d'inserció de l'empleat Albert.

SAVEPOINT

Aquesta instrucció permet establir un punt de retorn dins una transacció. El problema d'utilitzar conjuntament ROLLBACK/COMMIT és que un COMMIT ho accepta tot i un ROLLBACK ho anul·la tot.

SAVEPOINT permet senyalar un punt entre l'inici i el final de la transacció. La seva sintaxi és:

```
...instruccions DML...  
SAVEPOINT nom  
...instruccions DML...
```

Exemple:

```
INSERT INTO empleats(empleat_id, nom, departament_id)  
VALUES(1000, 'Pere', 10);  
  
SAVEPOINT Pere;  
  
INSERT INTO empleats (empleat, nom, departament_id)  
VALUES(2000, 'Joan', 20);  
  
SAVEPOINT Joan;  
  
INSERT INTO empleats (empleat, nom, departament_id)  
VALUES(3000, 'Albert', 10);  
  
ROLLBACK TO Pere;  
  
INSERT INTO empleats (empleat, nom, departament_id)  
VALUES(4000, 'Manel', 40);  
  
COMMIT;
```

En l'exemple anterior s'afegiran els empleats Pere i Manel. **Per què?**

Perquè el primer ROLLBACK to Pere; el que fa és realitzar un ROLLBACK des de la instrucció ROLLBACK fins al SAVEPOINT.

Important Estat de les dades dins d'una transacció abans d'un COMMIT o ROLLBACK

Si s'inicia una transacció utilitzant instruccions DML cal tenir en compte que:

- Les instruccions de consulta SELECT realitzades **per l'usuari que ha iniciat** la transacció mostren les dades ja modificades per les instruccions DML.
- La resta d'usuaris veuen les dades tal com estaven abans de la transacció depenent del nivell d'aïllament (*isolation level*) configurat en el nostre sistema.
- Els registres afectats per la transacció apareixen bloquejats fins que la transacció no finalitza. La resta d'usuaris no podran modificar els valors d'aquests registres.

Estat de les dades després d'un COMMIT

- Els canvis de dades es fan permanents a la base de dades.
- L'estat anterior de les dades es perd per sempre (no es pot tornar enrere).
- Tots els usuaris poden veure els resultats.
- Els bloquejos de les files afectades s'alliberen. Aquestes files estan disponibles perquè altres usuaris les puguin manipular.
- Tots els punts de retorn (SAVEPOINT) s'esborren.

Estat de les dades després d'un ROLLBACK

- Els canvis de dades es perden.
- Es recupera l'estat anterior de les dades.
- S'alliberen els bloquejos de les files afectades.

Transaccions aniuades

- L'ús de transaccions aniuades **no** es consideren una bona pràctica de programació, però es poden utilitzar en MySQL.
- Les transaccions aniuades es consideren com una de sola, que comença amb la primera instrucció BEGIN i acaba en l'últim COMMIT o el primer ROLLBACK.

Exemple 1:

```
SET AUTOCOMMIT=0;

BEGIN;

    INSERT INTO empleats(empleat_id, nom, departament_id)
    VALUES (1000, 'Pere', 10);

    BEGIN;
        INSERT INTO empleats(empleat_id, nom, departament_id)
        VALUES (2000, 'Joan', 20);

        INSERT INTO empleats(empleat_id, nom, departament_id)
        VALUES (3000, 'Albert', 10);

        INSERT INTO empleats(empleat_id, nom, departament_id)
        VALUES (4000, 'Manel', 20);

    ROLLBACK;

COMMIT;
```

En aquest cas només s'inserirà l'empleat Pere de forma permanent.

Exemple 2:

```
SET AUTOCOMMIT=0;

BEGIN;

INSERT INTO empleats(empleat_id, nom, cognoms, departament_id)
VALUES (1000, 'Pere', 'Prova', 10);

BEGIN;

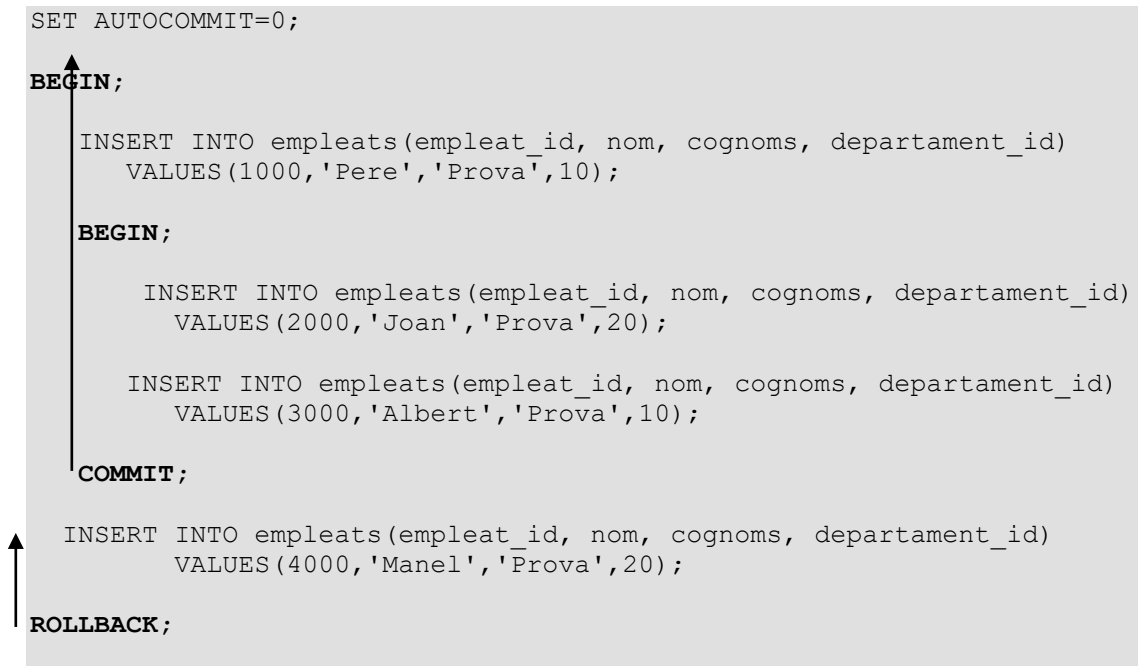
INSERT INTO empleats(empleat_id, nom, cognoms, departament_id)
VALUES (2000, 'Joan', 'Prova', 20);

INSERT INTO empleats(empleat_id, nom, cognoms, departament_id)
VALUES (3000, 'Albert', 'Prova', 10);

COMMIT;

INSERT INTO empleats(empleat_id, nom, cognoms, departament_id)
VALUES (4000, 'Manel', 'Prova', 20);

ROLLBACK;
```



En aquest segon no afegirà en Manel. Perquè el COMMIT fa finalitzar el primer BEGIN.

Compte!: Si no haguéssim desactivat l'AUTOCOMMIT s'inseririen els 4 empleats perquè el COMMIT fa finalitzar el primer BEGIN i per tan quan s'executa l'INSERT d'en Manel es fa sense cap transacció iniciada i es realitza amb AUTOCOMMIT.

```
SET AUTOCOMMIT=0;

BEGIN;

INSERT INTO empleats(empleat_id, nom, cognoms, departament_id)
VALUES (1000, 'Pere', 'Prova', 10);

BEGIN;

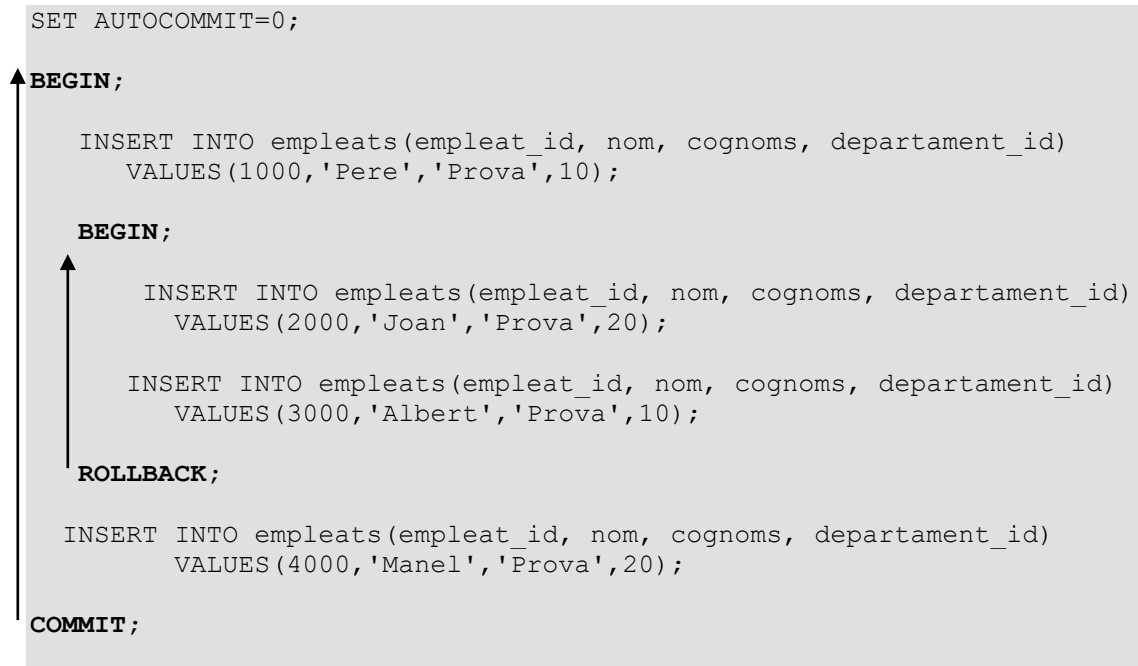
INSERT INTO empleats(empleat_id, nom, cognoms, departament_id)
VALUES (2000, 'Joan', 'Prova', 20);

INSERT INTO empleats(empleat_id, nom, cognoms, departament_id)
VALUES (3000, 'Albert', 'Prova', 10);

ROLLBACK;

INSERT INTO empleats(empleat_id, nom, cognoms, departament_id)
VALUES (4000, 'Manel', 'Prova', 20);

COMMIT;
```



En canvi aquest segon cas afegirà en Pere i en Manel (El primer i l'últim INSERT). Perquè el ROLLBACK només afecta en el segon i tercer INSERT.

6.2. Gestió de la concurrència

6.2.1. Bloquejos

Normalment els bloquejos són mecanismes que no requereixen intervenció de l'usuari. Prevenen la interacció destructiva que hi pot haver entre dues transaccions concurrents. Un usuari no pot modificar les mateixes dades que un altre usuari.

Quan un usuari inicia una transacció i seguidament modifica certs registres. Aquests queden bloquejats pels altres usuaris per tal de garantir la no interferència d'altres usuaris sobre aquests registres.

Bloqueig a nivell de taula

Els usuaris també poden bloquejar de forma explícita una taula i no permetre l'accés d'escriptura dels registres d'aqueta taula.

Per exemple, estem fent un procés de migració de dades d'un altre sistema o taula o estem fent un procés de recàlcul de salaris de tots els empleats i no volem que durant aquest procés cap usuari ens pugui modificar les dades.

MySQL disposa d'un mecanisme per bloquejar tots els registres d'una taula

Sintaxi:

```
LOCK TABLE <nom_taula> [READ|WRITE]
```

Amb les paraules READ I WRITE indiquem quin tipus de bloqueig volem.

READ

Hem bloquejat la taula només per lectura.

Això provocarà que la nostra sessió i les altres sessions no puguin realitzar operacions d'escriptura sobre la taula, però si que podran realitzar operacions de lectura.

WRITE

La sessió que ha realitzat el LOCK podrà llegir i escriure sobre la taula.

Altres sessions no podran ni llegir ni escriure sobre la taula.

Per alliberar els bloquejos de la taula tenim la instrucció `UNLOCK`

Sintaxi:

```
UNLOCK TABLES;
```

Només podem bloquejar una taula cada vegada. Fer un segon `LOCK TABLES` significa que es desbloquejarà la taula a la que s'ha aplicat el LOCK inicial i es passarà a bloquejar l'actual.

Exemple: Només volem permetre lectures sobre la taula de clients i per tant volem que els altres usuaris no hi puguin realitzar operacions de INSERT, UPDATE, DELETE

```
LOCK TABLE empleats READ;

UPDATE empleats
  SET data_contactacio = NOW()
 WHERE data_contractacio IS NULL;

Error Code: 1099. Table 'clients' was locked with a READ lock and
can't be updated 0.000 sec
```

Exemple: Hem de modificar una clau forana d'una taula i per tant durant un cert període de temps no volem que ningú pugui afegir registres en aquesta taula per tal de que no es produeixin operacions DML que comportin inconsistència de dades.

```
LOCK TABLE empleats WRITE;
/* Com que no es pot modificar una FK, l'esborro i la torno a crear
*/
ALTER TABLE empleats
  DROP FOREIGN KEY FK_EMPLEATS_DEPARTAMENTS;

ALTER TABLE empleats
  ADD CONSTRAINT FK_EMPLEATS_DEPARTAMENTS FOREIGN KEY (departament_id)
    REFERENCES departaments(departament_id) ON DELETE CASCADE;

UNLOCK TABLES;
```

Exemple:

TX SESSIÓ 1	TX SESSIÓ2
BEGIN;	BEGIN;
INSERT INTO empleats (nom, edat) VALUES ('Marta',20);	
	LOCK TABLE empleats WRITE; Es produeix un bloqueig fins que la tx de sessió1 finalitzi
COMMIT;	La sessió bloqueja tota la taula

Bloqueig a nivell de registre

Exemple de bloqueig

TX SESSIÓ 1	TX SESSIÓ2 (READ COMMITTED)
	BEGIN;
BEGIN;	
	SELECT * FROM empleats; (Steven)
UPDATE empleats SET nom = 'Joan' WHERE nom = 'Steven';	
SELECT * FROM empleats; (Joan)	
	SELECT * FROM empleats; (Steven)
	UPDATE empleats SET nom = 'Pere' WHERE nom = 'Steven'; (Bloqueig)

Com es pot veure en aquest en aquest cas fins que la transacció de la sessió 1 no finalitzi la transacció de la sessió 2 no podrà llegir la dada.

6.2.2. Consistència de lectura

Els usuaris de la base de dades executen bàsicament dos tipus d'operacions DML:

- Operacions de lectura (SELECT)
- Operacions d'escriptura (INSERT, UPDATE, DELETE)

Cal que hi hagi una consistència de lectura perquè:

1. Els qui escriuen i els qui llegeixen s'assegurin una visualització consistent de les dades.
2. Els qui llegeixen no vegin les dades que s'estan modificant.
3. Els qui escriuen s'assegurin que els canvis a la base de dades es fan de forma consistent.
4. Els canvis fets per un escriptor no interfereixin als que està fent un altre escriptor ni entrin en conflicte entre ells.

Per tant, la consistència de la lectura pretén assegurar que cada usuari vegi les dades tal com eren a l'última validació (*commit*), abans de començar una nova transacció.

Nivell d'aïllament i accés concurrent de les dades

Quan s'utilitzen transaccions, poden aparèixer problemes de concurrència en l'accés o lectura de les dades, és a dir, problemes ocasionats per l'accés a la mateixa informació per dues transaccions diferents. Aquests problemes es coneixen com:

- Dirty Read (Lectura Bruta)
- Non-repeatable Read (Lectura No Repetible)
- Phantom Read (Lectura Fantasma)

Aquests problemes estan molt relacionats amb el nivell d'aïllament que hàgim configurat en el nostre SGBD.

MySQL, igual que molts SGBDR té la capacitat de tenir diferents nivells d'aïllament de les modificacions d'altres transaccions

Gràcies a aquests nivell d'aïllament ens permeten treballar dins d'una transacció de forma més o menys aïllada respecte les altres depenent del nivell d'aïllament configurat.

Nivells d'aïllament (isolation levels) de menys a més:

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE

Treballar amb nivells d'aïllament baixos implica que les modificacions que altres transaccions m'afecten.

Podem modificar el nivell d'aïllament de la sessió actual:

SET TRANSACTION [GLOBAL SESSION] ISOLATION LEVEL [READ-UNCOMMITTED READ-COMMITTED REPEATABLE-READ SERIALIZABLE]

- GLOBAL
 - La sentència s'aplicarà a totes les sessions posteriors.
 - Les sessions existents no es veuran afectades .
 - Cal el privilegi **CONNECTION_ADMIN** per poder realitzar aquesta acció.
- SESSION
 - La sentència s'aplica a totes les transaccions posteriors realitzades dins de la sessió actual.
 - La sentència està permesa dins de les transaccions, però no afecta la transacció en curs.

- Sense especificar GLOBAL o SESSION
 - Si **no especifiquem** GLOBAL o SESSION **s'aplicarà només** a la **pròxima transacció** .
 - Les transaccions posteriors tornaran a utilitzar el valor de sessió
 - Aquesta declaració no està permesa dins de transaccions

A través del fitxer de configuració:

```
[mysqld]  
transaccion-isolation = <NIVELL>
```

Per comprovar quin nivell d'aïllament tenim podem consultar les variables :

```
SELECT @@GLOBAL.transaction_isolation;  
  
SELECT @@SESSION.transaction_isolation;
```

READ UNCOMMITTED (Lectures no confirmades)

És el **nivell d'aïllament més baix** i podem recuperar dades modificades i no confirmades d'altres transaccions. És el nivell a on es permet el nivell més alt de concurrència.

Les sentències SELECT són executades sense realitzar bloquejos.

Pot donar lloc a "lectures brutes" (**DIRTY READ**): Són les lectures que realitza una transacció d'unes dades que s'estan modificant per una altra transacció. Se'n diuen brutes perquè les dades que s'estan modificant encara no estan confirmades ni revocades. En canvi una altra transacció les pot llegir.

Exemple de lectures brutes: Suposem que tenim un empleat a la BD que es diu 'Pere'

TX SESSIÓ 1	TX SESSIÓ2 (READ UNCOMMITTED)
	SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
BEGIN;	BEGIN;
	SELECT * FROM empleats; (Pere)
UPDATE empleats SET nom = 'Marta' WHERE nom = 'Pere';	
SELECT * FROM empleats; (Marta)	
	SELECT * FROM empleats; (Marta)

Com es pot veure en aquest exemple els canvis que fa la transacció de la sessió 1 estan afectant a les dades de la sessió 2. Això no és desitjable perquè en aquell moment la transacció de la sessió 1 no ha finalitzat i per tant no sabem si s'acabarà amb un COMMIT o amb un ROLLBACK.

Per tant pel motiu que sigui si la sessió 1 decideix fe un ROLLBACK l'usuari de la sessió 2 ha llegit *dades brutes*, dades que no haurien d'haver existit mai.

Amb aquest nivell d'aïllament veiem tot els canvis que estan fent les altres transaccions i per tant estem llegint dades que potser no es validaran (commit) mai.

Important: No hi ha nivells d'aïllament més bons que altres.

Això no significa que aquest nivell d'aïllament no s'hagi d'utilitzar. Tot dependrà del moment i context a on estem.

Per exemple si tenim un procés que es dedica a obtenir un valor de quantes visites té una pàgina web, "No passa res" si a vegades ens apareix 1.567.999 o 1.567.998 perquè resulta que hi ha una transacció que s'havia equivocat d'afegir el registre de la visita d'un usuari.

O bé tenim un procés que es dedica a mostrar els "Likes" d'un comentari al Facebook. Tal i fa si en algun moment el comentari té 6 o 8 Likes.

Aquest tipus d'aïllament allibera molt la base de dades de recursos i comprovacions per tal de mantenir aïllades les transaccions.

Per tant quan més baix tinguem el nivell d'aïllament més descarregat anirà el nostre SGBDR.

READ COMMITTED (Lectures confirmades)

Aquest nivell afegeix un grau d'aïllament a les transaccions i ja no es produeixen les "lectures brutes" (dirty read).

Amb aquest configuració no permetrem que altres transaccions llegeixin dades modificades que no han estat confirmades per altres transaccions.

És el nivell d'aïllament que utilitzen per defecte els SGBD d'Oracle o SQL Server.

En aquest nivell apareix el problema de "**lectures no repetibles**" (**NONREPEATABLE READ**): Dins d'una mateixa transacció si llegim les mateixes dades resulta que aquestes canvien al llarg de la nostra transacció sense que nosaltres l'hi hàgim aplicat cap canvi.

Per exemple imaginem que estem davant d'un procés (transacció) de llarga durada com per exemple un informe mensual. Poden ocasionar certes inconsistències degudes a canvis que es produeixen a la base de dades entre l'inici i el final del procés de la creació de l'informe.

Exemple de lectures no repetibles.

TX SESSIÓ 1	TX SESSIÓ2 (READ COMMITTED)
	SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
	BEGIN;
	SELECT * FROM empleats; (Pere)
BEGIN;	
UPDATE empleats SET nom = 'Marta' WHERE nom = 'Pere';	Fem un sèrie de càlculs refiant-nos del valors que hem llegit.
COMMIT;	
	SELECT * FROM empleats; (Marta)

En aquest cas veiem que no es produeixen lectures brutes, però la transacció de la sessió 2 es pot està refiant de les dades que ha llegit a través de la primera lectura i per tant, quan la llegeix per segona vegada la dada ha canviat per culpa de la transacció de la sessió 1.

En aquest cas si estem al davant d'una sèrie de càlculs que volem que els valors no canviïn perquè estem realitzant una sèrie de balanços, saldos o càlculs importants que necessitem que no ens toquin les dades mentre els estic realitzant, aleshores aquest nivell d'aïllament no és adequat.

REPEATABLE READ (lectures repetibles)

Aquest nivell d'aïllament és el que utilitza el SGBD MySQL per defecte. Soluciona el tema de les lectures no repetibles ja que cada transacció guarda un "snapshot" de la informació llegida en tota la transacció. D'aquesta manera no permet que altres transaccions llegeixin dades modificades d'altres transaccions no confirmades.

Soluciona el problema de les lectures brutes (*Dirty reads*) i de les lectures no repetibles (*Non repeatable reads*).

Exemple

TX SESSIÓ 1	TX SESSIÓ2 (REPEATABLE READ)
	SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
	BEGIN;
	SELECT * FROM empleats; (Pere)
BEGIN;	
UPDATE empleats SET nom = 'Marta' WHERE nom = 'Pere';	Fem un sèrie de càlculs refiant-nos del valors que hem llegit.
COMMIT;	
	SELECT * FROM empleats; (Pere)

Veiem que la transacció de la sessió 1 ja té el nou valor de l'empleat 'Marta', però la transacció de la sessió 2 encara té el nom 'Pere' fins que finalitzi la transacció.

Tal i com hem dit això es soluciona mitjançant "snapshot", tal i com també fa Oracle, de la informació llegida en tota la transacció i d'aquesta manera les transaccions poden treballar de forma aïllada. En el cas de **SQL Server** la transacció de la sessió1 (UPDATE) donaria un bloqueig (s'esperaria) fins que la transacció de la sessió 2 no finalitzés perquè no utilitza 'snapshots'.

Aquest nivell d'aïllament no evita el problema de les "lectures fantasmes" (PHANTOM READ): Altres transaccions poden afegir registres i una altra transacció les llegirà.

Exemple de lectures fantasma en SQL Server i Oracle.

En aquest cas suposem que a la BD tenim a la taula un empleat 'Pere' i en volem inserir un altre anomenat 'Paula'

TX SESSIÓ 1	TX SESSIÓ2 (REPEATABLE READ)
	SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
	BEGIN;
	SELECT * FROM empleats; (Pere)
BEGIN;	
INSERT INTO empleats (nom) VALUES ('Marta');	
COMMIT;	
	SELECT * FROM empleats; (Pere, Marta) → SQL Server/Oracle

En aquest exemple veiem que la transacció de la sessió 2 no està aïllada completament perquè l'INSERT de la transacció de la sessió 1 l'ha afectat i s'està llegint.

Important! Aquest tipus de problema no apareix en InnoDB perquè utilitza *Multiversion concurrency control* – per cada registre. MVCC engine sap el número de transacció que la va inserir, esborrar i pot reproduir l'històric de les modificacions de la fila.

Exemple de lectures fantasma en MySQL.

En aquest cas suposem que a la BD hi tenim un empleat anomenat 'Pere'.

TX SESSIÓ 1	TX SESSIÓ2 (REPEATABLE READ)
	SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
	BEGIN;
	SELECT * FROM empleats; (Pere)
BEGIN;	
INSERT INTO empleats (nom) VALUES ('Marta');	
COMMIT;	
	SELECT * FROM empleats; (Pere)
	UPDATE empleats SET edat = 30; 2 rows affected
	SELECT * FROM empleats; (Pere, Marta)

En aquest cas veiem que en MySQL també apareixen registres fantasma. En aquest cas apareixen sempre i quan hi realitzem alguna modificació dins de la transacció.

En aquest cas s'ha realitzat una modificació de l'edat de l'empleada Marta (UPDATE empleats SET edat=30) i això ha afectat a la sessió 2.

SERIALIZABLE

Per tal de que no apareguin lectures fantasma hi ha el nivell *serializable* que **bloquejarà les dades llegides**.

En aquest cas suposem que a la BD hi tenim un empleat anomenat 'Pere'.

TX SESSIÓ 1	TX SESSIÓ2 (<i>SERIALIZABLE</i>)
	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
	BEGIN;
	SELECT * FROM empleats; (Pere)
BEGIN;	
INSERT INTO empleats (nom) VALUES ('Marta');	
(Bloqueig)	
COMMIT;	
	SELECT * FROM empleats;

En aquest moment veiem que la transacció de la sessió es queda bloquejada perquè la transacció de la sessió 2 té el conjunt de dades que vol inserir la transacció 1 i per tant no li deixa afegir.

Això succeeix perquè la sessió 2 ha fet una SELECT de **tota** la taula. Per tant bloqueja tota la taula.

Ara bé si el conjunt de dades amb el que està treballant la transacció de la sessió 2 no té res a veure amb el conjunt de dades que necessita la transacció de la sessió 1 aquesta podrà treballar-hi. Seguidament podem veure'n un exemple si suposem que l'empleat Pere té per exemple 5 anys.

TX SESSIÓ 1	TX SESSIÓ2 (<i>SERIALIZABLE</i>)
	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
	BEGIN;
	SELECT * FROM empleats WHERE edat <10; (Pere)
BEGIN;	
INSERT INTO empleats (nom, edat) VALUES ('Marta',20);	
COMMIT;	
	SELECT * FROM empleats; (Pere)
	UPDATE empleats SET edat = 30; 2 rows affected
	SELECT * FROM empleats; (Pere, Marta)

També veiem, però que si fem un canvi en les dades aquestes ens apareixeran de forma *fantasma*.

A partir d'aquest moment podem establir una taula de relació entre el nivell d'aïllament que utilitzi la nostra sessió i la problemàtica de lectura:

Nivell Aïllment vs. Problemàtica	Lectura Bruta (<i>Dirty Read</i>)	Lectures no repetibles (<i>Non-repeatable reads</i>)	Lectura Fantasma (<i>Phantoms</i>)
READ UNCOMMITTED	ES POT PRODUÏR	ES POT PRODUÏR	ES POT PRODUÏR
READ COMMITED	-	ES POT PRODUÏR	ES POT PRODUÏR
REPEATABLE READ	-	-	ES POT PRODUÏR
SERIALIZABLE	-	-	-

[https://en.wikipedia.org/wiki/Isolation_\(database_systems\)](https://en.wikipedia.org/wiki/Isolation_(database_systems))