

Gestió de dades en bases de dades corporatives. Llenguatge SQL.

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

Operacions amb bases de dades ofimàtiques i corporatives

Setembre del 2007
© Isidre Guixà i Miranda
IES SEP Milà i Fontanals
C/. Emili Vallès, 4
08700 - Igualada

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

Cap part d'aquesta publicació, incloent-hi el disseny general i de la coberta, no pot ser copiada, reproduïda, emmagatzemada o tramesa de cap manera ni per cap mitjà, tant si és elèctric, com químic, mecànic, òptic, d'enregistrament, de fotocòpia, o per altres mètodes, sense l'autorització prèvia per escrit dels titulars del copyright.

Índex

Índex	3
Introducció.....	4
Objectius.....	7
1. Iniciació als SGBD corporatius de la ma de MySQL.	9
1.1. Orígens i evolució del llenguatge SQL de la ma dels SGBD.	9
1.2. Situació actual dels SGBD relacionals.....	10
1.2.1. SGBD de programari propietari.....	10
1.2.2. SGBD de programari lliure	11
1.3. MySQL	12
1.4. L'entorn d'un SGBD corporatiu	13
1.4.1. Programes incorporats en el servidor MySQL.....	13
1.4.2. Altres eines interessants pel servidor MySQL.....	20
1.4.3. MySQL Administrator	20
1.4.4. MySQL Query Browser	25
1.4.5. Altes, baixes i modificacions des de MySQL Query Browser....	28
2. Llenguatge SQL: Tipus de dades. Consultes simples.....	31
2.1. Tipus de sentències SQL.....	31
2.2. Tipus de dades	32
2.2.1. Tipus de dades numèriques	32
2.2.2. Tipus de dades alfanumèriques.....	35
2.2.3. Tipus de dades per emmagatzemar moments temporals	38
2.3. Consultes simples	39
2.3.1. Clàusules <code>SELECT</code> i <code>FROM</code>	39
2.3.2. Clàusula <code>WHERE</code>	43
3. Llenguatge SQL: Funcions incorporades. Resultats ordenats	49
3.1. Taula dual.....	49
3.2. Sensibilitat a majúscules/minúscules en cadenes	49
3.3. Consulta de columnes <code>BLOB</code>	53
3.4. Funcions incorporades per MySQL	54
3.5. Clàusula <code>ORDER BY</code>	62
3.6. Opció <code>DISTINCT</code>	63
4. Llenguatge SQL: Consultes complexes.....	65
4.1. Agrupaments de files.....	65
4.2. Combinacions entre taules	67
4.3. Subconsultes	71
4.4. Operacions amb sentències <code>SELECT</code>	76
5. Llenguatge SQL: Manipulació de dades.....	78
5.1. Sentència <code>INSERT</code>	78
5.2. Sentència <code>UPDATE</code>	87
5.3. Sentència <code>DELETE</code>	89
5.4. Transaccions	90

Introducció

El boom de la microinformàtica en la dècada dels 80 ha anat acompanyat de l'aparició de diverses eines informàtiques que es podien utilitzar en els equips microinformàtics. Així, doncs, van aparèixer els processadors de text, els fulls de càlcul i les bases de dades, eines que, en ocasions, podien estar agrupades dins un paquet informàtic, englobat sempre dins el món de la ofimàtica.

La vida diària ens presenta moltes situacions en les que la utilització d'una base de dades ofimàtica ens és suficient per la tasca d'emmagatzematge de la informació. Podem trobar molts exemples: base de dades per organitzar els nostres volums musicals (discos, CD's, ...), base de dades per portar la comptabilitat domèstica, base de dades per gestionar les fotos que tenim a casa, base de dades per gestionar els llibres de la meua biblioteca,...

Però les bases de dades ofimàtiques poden no tenir suficients recursos quan es tracta de gestionar grans volums d'informació que han de poder ser accedits per a molts usuaris simultàniament des de la xarxa local i també des de llocs de treball remots i, per aquest motiu, apareixen les bases de dades corporatives. Com a exemple d'aquestes noves situacions, pensem en:

- la base de dades necessària per a gestionar les infraccions de trànsit d'un estat a la qual possiblement si ha de poder accedir des de les diferents comissaries de trànsit, des de les entitats bancàries (per quan l'infractor fa efectiu el pagament de la multa) i des de la pròpia central de trànsit,
- la base de dades necessària per a gestionar la preinscripció i matrícula en els centres docents públics de Catalunya de primària i secundària (és a dir, tots els alumnes que cursen educació infantil, primària, secundària, batxillerat i formació professional) a la qual s'hi ha de poder accedir des de tots els centres docents i també des de les diferents delegacions educatives,
- la base de dades d'una entitat bancària a la qual s'hi ha de poder accedir des de totes les oficines distribuïdes pel territori.

Els SGBD que permeten gestionar bases de dades similars a les anomenades en els exemples precedents han de facilitar, entre d'altres, mecanismes per:

- permetre connexions simultànies accedint i/o modificant dades comunes,
- controlar l'accés dels usuaris i definir diferents nivells i permisos per així poder diferenciar les dades a les que té accés cada usuari,
- efectuar còpies de seguretat sense que la base de dades hagi d'aturar el seu funcionament,
- restaurar una base de dades davant d'errors.

Els actuals SGBD ofimàtiques permeten molts dels mecanismes anteriors, però no donen l'eficiència necessària per a grans volums d'informació i grans volums de transaccions. Per aquest motiu ja fa anys que existeixen els SGBD corporatives.

Per altra banda, tots els SGBD incorporen el llenguatge SQL (**Structured Query Language**) per a donar instruccions al sistema gestor i així poder efectuar altres, baixes, consultes, modificacions, creació de les estructures de dades (taules i índexs), creació d'usuaris per accedir a les bases de dades, concessió i revocació de permisos d'accés,... Doncs bé, anem a utilitzar el llenguatge SQL per a gestionar les dades (altes, baixes, consultes i modificacions) i l'utilitzarem en bases de dades corporatives.

Esteu a punt? En teniu ganes? Doncs som-hi!

Objectius

En acabar la unitat didàctica heu de ser capaços de:

- 1) Identificar els principals elements que formen part de les bases de dades corporatives.
- 2) Utilitzar el llenguatge SQL per a efectuar consultes, insercions, modificacions i eliminacions en les bases de dades relacionals.
- 3) Identificar els diferents tipus de dades permesos en les bases de dades.
- 4) Instal·lar, configurar i utilitzar eines clients que permetin l'accés a les bases de dades corporatives.

1. Iniciació als SGBD corporatius de la ma de MySQL.

El llenguatge SQL és un llenguatge que ens permet manipular les bases de dades gestionades pels SGBD relacionals i, per tant, per poder-ne comprovar el seu funcionament necessitem disposar d'algun SGBD. Certament que ho podríem fer en algun SGBD ofimàtiques dels que acostumem a tenir instal·lats en els ordinadors domèstics, però com que també ens correspon ser bons coneixedors dels SGBD corporatius, és un bon camí utilitzar els SGBD corporatius per l'aprenentatge del llenguatge SQL.

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

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

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

- IBM comercialitza diversos productes relacionals amb el llenguatge SQL: *System/38* en 1979, *SQL/DS* en 1981 i *DB2* en 1983.
- Relational Software, Inc. (actualment Oracle Corporation) desenvolupa la seva pròpia versió de SGBD relacional per la Marina dels E.E.U.U., la C.I.A. i d'altres i l'estiu del 1979 allibera *Oracle V2* (Versió 2) per a les computadores VAX (les grans competidores de l'època amb les computadores d'IBM) .

El llenguatge SQL va anar evolucionant (cada marca comercial seguia el seu propi criteri) fins que els principals organismes d'estandardització hi van ficar ma per tal d'obligar que els diferents SGBD relacionals implementessin una versió comuna del llenguatge i així, l'ANSI (**American National Standards Institute**) publica en 1986 l'estàndard

Per què SQL enlloc de SEQUEL?

La utilització de l'acrònim SQL enlloc de SEQUEL va ser motivada per què el mot *SEQUEL* ja estava registrat per la companyia anglesa d'avions Hawker-Siddeley.

Pronunciació de SQL

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

SQL-86, que en 1987 és ratificat per l'ISO (**I**nternational **O**rganization for **S**tandardization).

Des de 1986 s'han anat succeint diverses revisions de l'estàndard SQL:

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

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

1.2. Situació actual dels SGBD relacionals

L'oferta actual de SGBD relacionals és molt àmplia i es pot fer difícil prendre decisions sobre quin és el SGBD relacional a escollir per a la seva implantació en una empresa u organització. Per sort no és el nostre objectiu! Però hem de tenir algun coneixement sobre la gran oferta que actualment hi ha en aquest sector.

Nosaltres, en no tenir coneixements de les diferències tècniques que hi ha entre els diferents SGBD relacionals (eficiència, volums d'informació capaços de gestionar, ...) i classifiquem els SGBD relacionals en dos grans grups segons siguin de **programari propietari** o de **programari lliure**.

1.2.1. SGBD de programari propietari

Les empreses productores de programari propietari, en distribuir un producte únicament lliuren al comprador una còpia del programa executable junt amb l'autorització d'executar el programa en un número determinat de llocs de treball. En el contracte que subscriuen ambdues parts, comunament denominat "licència del producte" queda expressat clarament que el client únicament adquireix la facultat d'utilitzar el programa en determinada quantitat de llocs de treball i deixa ben clar que el programa segueix sent propietat de l'empresa productora del mateix i que l'usuari no està facultat a realitzar-hi cap canvi.

Alguns SGBD relacionals actuals d'aquesta tipologia són:

- 4th Dimension
- ADS
- Dataphor
- Daffodil database

- DB2
- FileMaker Pro
- FrontBase
- Informix
- Matisse
- Microsoft Access
- Microsoft SQL Server
- Microsoft Visual FoxPro
- Mimer SQL
- Netezza
- NonStop SQL
- Oracle
- Pyrrho DBMS
- Progress 4GL
- Sand Analytic Server (abans conegut com a Nucleus)
- Sybase Adaptive Server Anywhere (abans conegut com Watcom SQL)
- Sybase Adaptive Server Enterprise
- Sybase Adaptive Server IQ
- Teradata
- ThinkSQL
- VistaDB
- VMDS

1.2.2. SGBD de programari lliure

El programari lliure és aquell programari que una vegada obtingut pot ser utilitzat, copiat, estudiat, modificat i redistribuït amb total llibertat. Això no vol dir que la seva adquisició sigui lliure de cost i que no pugui ser comercialitzat.

Concretament un programari es defineix com a "lliure" si garanteix les quatre llibertat següents:

- Llibertat d'utilitzar el programa amb qualsevol propòsit (privat, educatiu, públic, comercial,...).
- Llibertat d'estudiar com funciona el programa i adaptar-lo a les teves necessitats. L'accés al codi font és una condició necessària.
- Llibertat de distribuir-ne còpies, amb el que es pot ajudar a altres usuaris interessats.
- Llibertat de millorar el programa i fer públiques les millores als altres, de manera que tota la comunitat se'n beneficiï.

Per tenir més informació sobre el programari lliure i els diferents matisos existents, podeu consultar els enllaços recomanats a la secció "Adreces d'interès" del web d'aquest crèdit

Alguns SGBD relacionals actuals d'aquesta tipologia són:

- Cloudscape
- Derby
- Firebird
- H2
- HSQLDB
- Ingres
- InterBase
- MaxDB
- MonetDB
- MySQL
- PostgreSQL
- SmallSQL
- SQLite
- tdbengin

1.3. MySQL

MySQL és la base de dades de codi font obert més utilitzada i és proporcionada per *MySQL AB*, empresa que té per negoci proporcionar serveis al voltant del SGBD MySQL.

El servidor MySQL va ser desenvolupat originàriament per gestionar grans bases de dades de manera molt més ràpida que les solucions existents i ha estat utilitzat amb èxit en ambients de producció molt exigents durant molts anys. Tot i que es troba en desenvolupament constant, el SGBD MySQL facilita avui un conjunt ric i útil de funcions en les principals plataformes existents (Windows, Mac OS X, Linux, BSD, Unix) cosa que no poden dir tot els competidors (els SGBD de Microsoft només són instal·lables en Windows).

La gran expansió i evolució d'aquest SGBD ha anat de la mà de l'expansió i l'evolució del desenvolupament de pàgines web dinàmiques, pel que és menester disposar d'un llenguatge de programació adequat (PHP n'era un bon candidat), d'un servidor web (Apache n'era un bon candidat) i un SGBD relacional (MySQL n'era un bon candidat).

PHP - Apache - MySQL

PHP és un llenguatge de servidor per a confeccionar pàgines web dinàmiques.

Els llenguatges de servidor (PHP com a projecte de codi obert, ASP de l'empresa d'informàtica Microsoft, JSP de l'empresa d'informàtica SUN, ColdFusion de l'empresa d'informàtica Allaire) permeten inserir instruccions (escrites en el corresponent llenguatge) dins el codi HTML de les pàgines web, de manera que quan un client sol·licita una pàgina web que conté codi de llenguatge servidor, el servidor web reacciona, llegeix el codi i executa les comandes (accedint normalment a bases de dades) per tal de confeccionar el definitiu codi HTML de la pàgina que s'envia al client. És a dir, la pàgina HTML s'ha construït en el moment present.

Per a que aquest procés sigui factible és menester l'existència d'un servidor web, que té allotjada la pàgina HTML i que ha de poder interpretar el codi de llenguatge servidor per a confeccionar la pàgina HTML definitiva. Apache és un servidor web de codi obert que sap interpretar el llenguatge PHP.

I per acabar de tancar el circuit necessitem un sistema gestor de bases de dades, com, per exemple, MySQL.

Els candidats que hem presentat eren "bons" per què donaven unes bones prestacions i pertanyien al grup del programari lliure, fet que ha provocat que gran part de la comunitat de desenvolupadors web els hagi adoptat, fins el punt que han aparegut diversos productes comercials que empaqueten les tres eines (llenguatge PHP + servidor web Apache + servidor de bases de dades MySQL) .

Com a conseqüència dels raonaments presentats és lògic escollir MySQL com a SGBD relacionals amb el que introduir-nos en el món dels SGBD corporatives i en l'aprenentatge del llenguatge SQL.

Instal·lem el SGBD MySQL seguint les indicacions de la secció "Recursos" del web d'aquest crèdit.

1.4. L'entorn d'un SGBD corporatiu

Un SGBD corporatiu es compon d'una gran quantitat de programes que podríem classificar en:

- 1) El **nucli del servidor**, que és el programa que està contínuament executant les instruccions dels usuaris i possiblement altres processos interns per a la bona marxa del sistema.
- 2) **Programa "escoltador" o "guaitador"** que està contínuament a l'espera de les instruccions dels usuaris.
- 3) **Programa consola** que permet obrir sessions de treball en el SGBD per tal d'executar instruccions d'usuari.
- 4) Diversos programes i **utilitats per a facilitar l'accés** dels usuaris a les dades i per a facilitar les tasques administratives. Alguns d'aquests programes i utilitats venen incorporats amb la instal·lació bàsica del propi SGBD i d'altres s'instal·len només si s'efectua una instal·lació completa o, fins i tot, poden ser instal·lacions de productes diferenciats del SGBD.

1.4.1. Programes incorporats en el servidor MySQL

En MySQL podem comprovar l'existència de quasi totes aquestes tipologies de programes si fem una visita al directori *bin* on s'han instal·lat els programes. Hi trobarem, més o menys (dependrà de la versió instal·lada), els programes següents:

Programa	Per a què serveix?
mysqld mysqld-nt mysqld-max mysqld-max-nt mysqld-debug	Són els diversos executables del nucli del servidor que aporta la instal·lació efectuada de MySQL. Les instal·lacions de MySQL en Windows com a servei deixen instal·lat l'executable "mysqld-nt". Es surt de l'objectiu d'aquest crèdit conèixer les diferències entre els diversos executables.
mysql	És la consola de MySQL.
mysqladmin	Permet efectuar tasques administratives des de la línia de comandaments del sistema operatiu.
mysqldump	Permet efectuar còpies de seguretat (<i>backup</i>) de la base de dades des de la línia de comandaments del sistema operatiu.
mysqlimport	Permet efectuar una càrrega de dades dins taules d'una bases de dades a partir d'arxius de text i des de la línia de comandaments del sistema operatiu.
mysqlmanager	Permet efectuar el seguiment de diverses instàncies MySQL i també la seva gestió.
mysqlshow	Permet llistar les bases de dades existents en el SGBD des de la línia de comandaments del sistema operatiu.

Programa	Per a què serveix?
mysqltest	Permet executar tests en el sistema que permetin detectar comportaments erronis.
myisamchk	Permet testear i reparar taules MyISAM de les bases de dades des de la línia de comandaments del sistema operatiu.
myisampack	Permet compactar taules MyISAM de les bases de dades des de la línia de comandaments del sistema operatiu.
mysqlbinlog	Permet generar arxius llegibles a partir dels arxius de log binaris que genera el servidor.
mysqlcheck	Permet analitzar, testear, reparar i optimitzar les taules de les bases de dades des de la línia de comandaments del sistema operatiu.

MySQL no té un programa específic per escoltar les instruccions dels usuaris. El propi nucli del servidor se n'encarrega. Altres SGBD com PostgreSQL i Microsoft SQLServer tenen el mateix funcionament. El SGBD Oracle, en canvi, manté programes diferenciats.

Més endavant veurem que MySQL distingeix diferents tipologies de taules. El tipus MyISAM és el més antic dels tipus actualment suportats i es mantenen algunes utilitats antigues que només eren vàlides per aquest tipus de taules.

Ara que ja tenim instal·lat el SGBD MySQL en la plataforma Windows, anem a comprovar algun funcionament d'aquestes utilitats. Partim de la base que tenim el servei MySQL en marxa (ho podem comprovar en la llista de serveis del sistema operatiu (panell de control):



Iniciado Automático Sistema local

Totes les actuacions següents les desenvoluparem des de la línia de comandaments del sistema operatiu (Inici – Executar... – cmd)

Si en executar qualsevol dels comandaments següents, per exemple, `mysql`, ens apareix un missatge similar a

```
"mysql" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.
```

és senyal que el *path* del sistema no té carregat el directori *bin* on resideixen els programes executables de MySQL. En tal situació ens caldrà modificar la variable d'entorn *path*.

Bé, anem a executar alguns dels programes abans esmentats:

1) Com accedir a l'ajuda que la majoria de programes aporten?

La majoria de programes permeten l'execució acompanyats dels paràmetres `--help` per a informar-nos de les opcions d'execució. Així, per exemple, podem executar:

```
C:\> mysql --help
C:\> mysqld --help
C:\> mysqladmin --help
```

```
C:\> mysqlshow --help
C:\> mysqlimport --help
```

Cadascuna d'aquestes execucions dona per resposta un munt d'informació referent a les opcions d'execució a poder utilitzar. Necessitarem conèixer aquestes opcions per poder executar els diferents programes.

2) Posem en marxa la consola de treball `mysql`:

Si intentem executar directament el programa

```
C:\> mysql
```

obtenim l'error:

```
ERROR 1045 (28000): Access denied for user 'ODBC'@'localhost' (using password: NO)
```

Què està passant? Doncs que no hem indicat l'usuari de MySQL amb el que volem posar en marxa la consola. Si observem l'ajuda del programa `mysql` hi trobarem, entre molta informació, la següent:

```
mysql Ver 14.12 Distrib 5.0.19, for Win32 (ia32)
Copyright (C) 2002 MySQL AB
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL license
Usage: mysql [OPTIONS] [database]
  -?, --help          Display this help and exit.
.....
  -p, --password[=name] Password to use when connecting to server. If password is
                        not given it's asked from the tty.
.....
  -u, --user=name      User for login if not current user.
.....
```

L'ajuda ens diu que l'execució pot necessitar d'unes opcions i d'una base de dades on connectar-nos.

```
mysql [options] [database]
```

Respecte la base de dades, si no indiquem cap ens connectarem al servidor sense entrar en cap base de dades. Per tant, com que encara no hem creat cap base de dades, no indicarem res.

Respecte la resta d'opcions i tenint en compte l'error que ens hem trobat més amunt, sembla que haurem d'indicar l'usuari amb el que ens connectem i potser la seva contrasenya. Suposem que en instal·lar

Els símbols corxets []

La nomenclatura informàtica utilitza els corxets per indicar que allò que hi ha en el seu interior és optatiu i podem obviar-ne.

MySQL hem decidit que l'usuari administrador (*root*) quedés sense contrasenya. Com que l'únic usuari que sabem que existeix és l'usuari *root*, tenint en compte que no deu tenir contrasenya i considerant les opcions més amunt indicades, haurem d'executar

```
C:\> mysql -u root
```

```
o
```

```
C:\> mysql --user=root
```

Si l'usuari *root* tingués una contrasenya, executariem qualsevol de les dues instruccions anteriors afegint l'opció `-p` o `--password` de qualsevol de les maneres següents:

```
-p <contrasenya>
--password=<contrasenya>
-p
--password
```

on `<contrasenya>` es refereix al mot que correspon a la contrasenya.

En les dues primeres possibilitats l'usuari escriu la contrasenya darrera de l'opció `-p` o `--password` i el mot es veu per pantalla. En canvi, en les dues darreres possibilitats, MySQL demana posteriorment la contrasenya a l'usuari, i en aquest cas, el text que l'usuari escriu es mostra emmascarat amb asteriscos per pantalla.

En qualsevol cas obtindrem:

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 4 to server version: 5.0.19-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Ja hem establert una connexió amb el servidor MySQL mitjançant la consola de treball. Ens està informant sobre:

- Els comandaments que vulguem executar cal finalitzar-los amb `;` o `\g`.
- La versió del sistema MySQL al que estem connectats i l'identificador de la nostra connexió (doncs un servidor MySQL admet molts connexions simultàniament).

- Com podem cercar ajuda sobre els comandaments executables des de la consola.
- Finalment, el prompt `mysql>` té una funció similar al prompt `C:\>` del sistema operatiu: espera els comandaments a executar.

Si us sembla podem mirar l'ajuda. Al llarg d'aquest crèdit veurem molts comandaments que podrem executar des de la consola. Necessitem, però, el comandament per sortir i tancar la consola de treball: `exit` o `quit`.

Abans de sortir, però, pot ser interessant utilitzar el comandament que permet veure les bases de dades en el servidor MySQL aptes per a ser utilitzades.. Si no hem creat cap potser no hauríem de trobar-hi res, no? Executem

```
mysql> show databases;
```

i obtenim

```
+-----+
| Database          |
+-----+
| information_schema |
| mysql              |
| test               |
+-----+
5 rows in set (0.22 sec)
```

```
mysql >
```

Sorpresa! Resulta que tenim 3 bases de dades en el nostre servidor! Més endavant explicarem quines són aquestes bases de dades.

3) Utilitzem el programa `mysqladmin` per aturar el servei

El programa `mysqladmin` facilita diverses funcionalitats. Una d'elles, per exemple, la d'aturar el servei MySQL amb l'opció `shutdown`:

```
C:\> mysqladmin -u root shutdown
```

Si anem a la llista de serveis del sistema operatiu (panell de control) hauríem d'observar que el servei MySQL ja no és iniciat.

Aquesta no és la única manera d'aturar el servei MySQL. Com molt bé sabem d'altres crèdits, els serveis en el sistema operatiu es poden aturar i engegar amb els comandaments del sistema operatiu `net stop` i `net start`.

Així, si executem:

```
C:\> net start MySQL
```

hauríem d'aconseguir posar en marxa altra vegada el servei MySQL i l'execució de:

```
C:\> net stop MySQL
```

hauria de suposar l'aturada del servei MySQL.

Els comandaments `net` són facilitats pel sistema operatiu Windows, mentre que el programa `mysqladmin` és una eina facilitada per servidor MySQL.

El programa `mysqladmin` no només serveix per aturar el servei MySQL. Té altres funcionalitats interessants:

- Per conèixer la versió del servidor

```
C:\> mysqladmin -u root version
```

- Per conèixer si el servidor és "viu"

```
C:\> mysqladmin -u root ping
```

- Per crear una base de dades (tenint el servidor MySQL engegat)

```
C:\> mysqladmin -u root create <nomBD>
```

Podem comprovar que s'ha creat una base de dades de nom `<nomBD>` posant en marxa una consola de treball i veient les bases de dades existents amb el comandament `show databases`.

- Per eliminar una base de dades

```
C:\> mysqladmin -u root drop <nomBD>
```

Eliminar una base de dades implica eliminar tota la informació que conté i, per precaució, el servidor MySQL demana confirmació.

Igual que abans podem comprovar que s'ha eliminat la base de dades de nom `<nomBD>` amb el comandament `show database` des d'una consola de treball.

Estat dels serveis des del panell de control

Si engeguem i aturem serveis del sistema amb comandaments i volem comprovar la bona execució observant la llista de serveis des del panell de control, recordem de prémer F5 per refrescar la llista.

Importància de la base de dades anomenada "mysql"

La base de dades `mysql` conté informació fonamental sobre la resta de bases de dades gestionades pel SGBD. Per tant, ni se'n passi pel cap la seva eliminació.

La base de dades `information_schema` és una base de dades virtual que serveix per proporcionar informació sobre la resta de bases de dades. En aquest cas no hi ha perill d'eliminació doncs no és una veritable base de dades i el comandament `drop` fallaria.

4) Posem en marxa el servidor MySQL com a procés enlloc de com a servei amb el programa `mysqld` o qualsevol `mysqld-versio`.

Suposant que tenim aturat el servei de Windows MySQL, podem posar en marxa el servidor MySQL com a procés executant qualsevol de les instruccions:

```
C:\> mysqld --standalone
C:\> mysqld-nt --standalone
C:\> mysqld-max --standalone
C:\> mysqld-max-nt --standalone
```

Qualsevol d'aquestes execucions provoca que el servidor MySQL es posi en marxa com a procés. Si comprovem el servei MySQL per la llista de serveis del sistema operatiu (panell de control) veurem que no està iniciat (dons no hem iniciat el servei, sinó el servidor MySQL com a procés).

En engegar el servidor MySQL com a procés, la consola del sistema operatiu utilitzada queda "penjada" doncs en ella s'està executant el servidor com a procés. Podem utilitzar igualment el programa `mysqladmin version`, des d'una segona consola del sistema, per a comprovar que tenim el servidor MySQL en marxa i verificar que ens informa de la versió que hem engegat. Així mateix, podem aturar el procés amb el programa `mysqladmin shutdown` des de la segona consola del sistema i veurem com la primera consola deixar de estar "penjada".

5) Utilitzem el programa `mysqlshow` per veure les bases de dades existents.

Més amunt hem constatat que el programa `mysql` (consola de treball) facilita el comandament `show databases` per veure les bases de dades existents en el servidor MySQL. El programa `mysqlshow` permet obtenir la mateixa informació sense obrir una consola de treball.

```
C:\> mysqlshow -u root
```

1.4.2. Altres eines interessants pel servidor MySQL

L'empresa *MySQL AB* facilita diversos programes interessants d'utilitzar per tal de treballar amb el servidor MySQL. La seva instal·lació no és imprescindible però seria una tonteria no aprofitar-los. En concret, ens estem referint a:

- *MySQL Administrator* (incorpora *MySQL System Tray Monitor*)
- *MySQL Query Browser*

Però no només l'empresa *MySQL AB* proporciona eines per a gestionar MySQL. Així, per exemple, trobem:

- *PHPMyAdmin*

Es tracta d'una interfície de gestió exclusiva per al MySQL creada inicialment per Tobias Ratschiller amb el llenguatge PHP i continuada per molts col·laboradors. Consisteix en un conjunt de programes PHP per administrar les bases de dades del MySQL sota entorn web. Aquesta particularitat de funcionar sota entorn web permet administrar bases de dades en remot; simplement necessitem una connexió a Internet i un navegador qualsevol.

Instal·lem aquestes eines seguint les indicacions de la secció "Recursos" del web d'aquest crèdit.

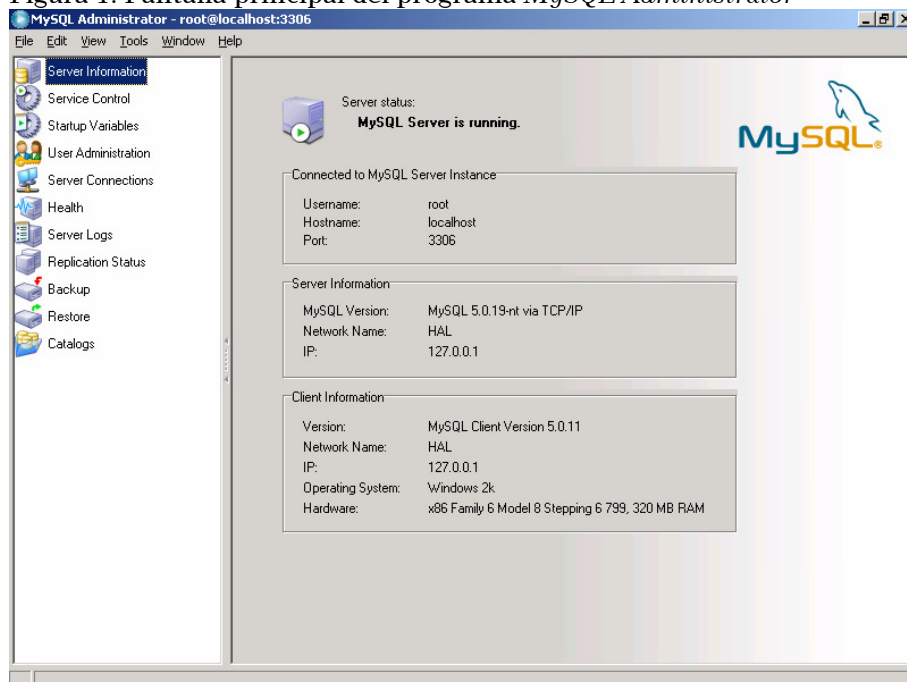
1.4.3. MySQL Administrator

MySQL Administrator és un programa per a executar tasques administratives com configurar el servidor MySQL, monitoritzar el seu estat i rendiment, engregar-lo i aturar-lo, gestionar usuaris i connexions, planificar i executar còpies de seguretat,...

Moltes d'aquestes tasques (per no dir totes) es poden executar des de la consola de sistema operatiu utilitzant els programes `mysqladmin` i `mysql`, però *MySQL Administrator* és millor en els següents aspectes:

- La seva interfície gràfica ens fa molt intuïtiva la seva utilització.
- Facilita una ràpida ullada als punts claus a tenir en compte pel rendiment, precisió i seguretat dels servidors MySQL.
- Mostra indicadors gràfics de rendiment que faciliten la posta a punt del servidor.

En posar en marxa l'eina *MySQL Administrator* ens trobem amb la pantalla principal (veure la figura 1). La pantalla principal conté una barra lateral (esquerra), una zona de treball i menús i submenús.

Figura 1. Pantalla principal del programa *MySQL Administrator*

Anem a fer una ullada a les diferents funcionalitats d'aquesta pantalla.

1) **La barra de lateral** (esquerra) té les seccions que ara comentem. Més endavant ens caldrà efectuar tasques administratives que podrem efectuar mitjançant aquestes seccions o mitjançant els programes `mysqladmin` i `mysql`.

a) *Server Information*

Conté informació sobre la instància MySQL a la que estem connectats, la màquina servidora (on s'està executant MySQL) i la màquina client (des d'on estem connectats).

En aquest moment la màquina servidora i la màquina client coincideixen, però si ens connectéssim des d'una altra màquina (on hi hagués instal·lades eines client MySQL), la informació seria diferent.

b) *Service Control*

Aquesta opció ens permet engegar i aturar la instància MySQL a la que estem connectats. Si la màquina servidora està en plataforma Windows, fins i tot ens permet configurar el servei.

c) *Startup Variables*

Permet configurar variables d'inici del servidor MySQL.

d) *User Administration*

Permet administrar els usuaris existents, crear nous usuaris i eliminar usuaris existents.

e) *Server Connections*

Permet visualitzar i/o matar les connexions establertes amb el servidor MySQL.

f) *Health*

Mostra gràfiques referents a diversos valors d'utilització del servidor i també una vista jeràrquica de les variables de sistema i d'estat.

g) *Server Logs*

Mostra les entrades existents en els diversos arxius de log.

h) *Replication Status*

Mostra informació sobre l'estat de replicació, en cas d'estar activat.

Arxius de log

Els arxius de log són arxius on el SGBD enregistra els diversos esdeveniments que hi tenen lloc (quan s'arrenca, quan s'atura, errors que es produeixen,...)

Replicació

La replicació és un mecanisme consistent en tenir part o totalitat de les dades emmagatzemades en un SGBD copiades (replicades) en altres màquines (potser molt llunyanes) de manera que s'actualitzin simultàniament (o periòdicament).

i) *Backup*

Permet planificar i administrar còpies de seguretat, seleccionar les bases de dades sobre les que efectuar còpies i iniciar els processos de còpia.

j) *Restore*

Permet restaurar les dades de les bases de dades a partir de còpies de seguretat efectuades.

k) *Catalogs*

Mostra informació sobre les bases de dades emmagatzemades en el servidor amb llurs taules, columnes, índexs i files. Permet també, de cada taula, fer-ne el seu manteniment (optimitzar, testejar i reparar).

Instal·lem en el nostre servidor una versió d'una base de dades música que hem estudiat en anteriors unitats didàctiques a partir de la recuperació d'una còpia de seguretat, seguint les indicacions de la secció "Recursos" del web d'aquest crèdit.

Si no hem creat ni eliminat bases de dades (programa `mysqladmin` amb els comandaments `create / drop`) la secció *Catalogs* ens hauria de mostrar les bases de dades: `information_schema`, `mysql`, `test`.

2) **La zona de treball** va mostrant la informació que correspon a la secció escollida en la barra lateral esquerra. En algunes seccions, l'àrea de treball és formada per dues o més pestanyes.

3) Anem a comentar, ara, les diverses opcions dels **menús**.

a) *File*

- *New Instance Connection ...*

Obre la finestra de connexió (la mateixa que quan posem en marxa *MySQL Administrator*) permetent obrir una altre connexió al servidor MySQL (amb el mateix o diferent usuari). es poden obrir un número qualsevol de connexions. Podem, per tant, tenir diferents instàncies del programa *MySQL Administrator* en execució, connectades amb un o varis servidors MySQL.

- *Reconnect...*

Tanca la connexió actual i obre la finestra de connexió per establir una nova connexió a un servidor MySQL.

- *Close*

Tanca el programa *MySQL Administrator* i finalitza la connexió amb el servidor MySQL. Només tanca la instància de *MySQL Administrator* des d'on s'executa *Close*.

b) *Edit*

Els típics comandaments *Cut*, *Copy* i *Paste* permeten tallar, copiar i enganxar el texts seleccionat de l'àrea de treball. El comandament *Select All* permet seleccionar tot el text.

c) *View*

Aquesta menú permet situar-nos en qualsevol de les seccions de la barra lateral esquerra.

d) *Tools*

Aquest menú permet engegar altres eines subministrades per l'empresa *MySQL AB* per a treballar amb MySQL juntament amb eines que aporta el propi *MySQL Administrator*. Així, hi podem trobar:

- *MySQL Command Line Client*

Aquesta opció serà presentada si *MySQL Administrator* troba l'executable `mysql` (consola de treball) instal·lat en l'ordinador. En executar aquesta opció, *MySQL Administrator* posa en marxa una consola de treball utilitzant els mateixos paràmetres que s'ha introduït en la finestra de connexió per establir la connexió des de *MySQL Administrator*.

- *MySQL System Tray Monitor*

Aquesta opció posa en execució un programa que serveix per monitoritzar l'estat dels serveis MySQL en execució. Aquest monitor el podem trobar al costat del rellotge a la barra de tasques de Windows

MySQL System Tray Monitor

En el procés d'instal·lació indicat en el material web d'aquest crèdit es fa una breu explicació de les opcions d'aquest programa.

- *MySQL Query Browser*

Programa pel que comentarem més endavant les seves funcionalitats.

- *Windows Command Line*

Obre una consola del sistema operatiu Windows.

- *Manage Connections ...*

Obre la finestra de diàleg de les opcions de configuració de *MySQL Administrator* situant-nos en l'apartat referent a les connexions emmagatzemades en *MySQL Administrator*. Així, quan posem en marxa *MySQL Administrator*, observem que el primer camp *Stored Connection* ens permet seleccionar una connexió emmagatzemada i així ens estalviem d'introduir la informació de la resta de camps. Aquesta finestra també és accessible des de la finestra de connexió prement el botó que té per etiqueta "...".

- *Save current Connection ...*

Obre la finestra de diàleg de les opcions de configuració de *MySQL Administrator* situant-nos en l'apartat referent a les connexions i ens permet emmagatzemar una nova connexió per facilitar la seva utilització en posar en marxa *MySQL Administrator*.

- *Options*

Obre la finestra de diàleg de les opcions de configuració de *MySQL Administrator*.

e) *Help*

- *Help*

Posa en marxa l'ajuda de MySQL Administrator.

- *Launch MySQL.com Website*

Estableix una connexió amb la pàgina principal de la web de *MySQL AB*.

- *List reported Bugs*

Estableix una connexió amb la pàgina de la web de *MySQL AB* que conté la informació dels errors de funcionament detectats en *MySQL Administrator*.

- *Report a new Bug to MySQL*

Estableix una connexió amb la pàgina web de *MySQL AB* destinada a recollir els errors de funcionament que els usuaris detectem en *MySQL Administrator*.

- *About ...*

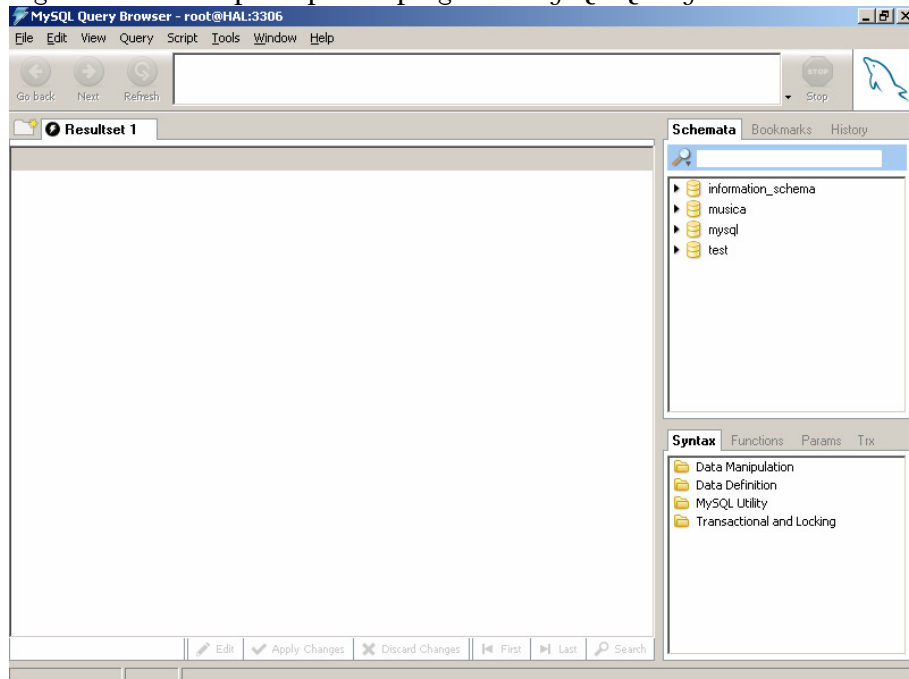
Mostra informació sobre *MySQL Administrator*.

1.4.4. MySQL Query Browser

MySQL Query Browser es una eina gràfica subministrada per *MySQL AB* per crear, executar i optimitzar consultes en un entorn gràfic. Mentre que *MySQL Administrator* està pensat per administrar un servidor MySQL, *MySQL Query Browser* està dissenyat per ajudar-nos en la consulta i l'anàlisi de les dades emmagatzemades en les bases de dades MySQL.

Mentre totes les consultes executades en *MySQL Query Browser* poden ser també executades des de la consola de treball (`mysql`), *MySQL Query Browser* permet treballar d'una manera més intuïtiva en la consulta i edició de dades.

En posar en marxa l'eina *MySQL Query Browser* ens trobem amb la pantalla principal (veure la figura 2). Totes les funcionalitats de l'aplicació són accessibles des d'aquesta pantalla.

Figura 2. Pantalla principal del programa *MySQL Query Browser*

ud3_03

La pantalla principal està dividida en diferents seccions:

1) Barra d'eines de consulta

És la barra d'eines superior i en ella es creen i executen les instruccions de llenguatge SQL que ens permetran consultar i modificar les dades. Té tres botons de navegació (*Go Back*, *Next*, *Refresh*), l'àrea de consulta (on escriurem les sentències SQL, un botó d'acció (*Stop*) i una petita pestanya amb tres opcions d'execució (*Execute*, *Execute in New Tab*, *Split Tab and Execute*).

2) Àrea de resultats

És la gran àrea central i en ella s'hi visualitzen els resultats de totes les consultes. Està organitzada en diferents pestanyes. A la figura 2 se n'observa només una, de nom *Resultset1* però podem tenir múltiples pestanyes actives al mateix temps, fet que ens permet treballar simultàniament amb diferents consultes. Aquestes pestanyes poden dividir-se (*split*) verticalment i horitzontal per recollir els resultats de diferents consultes (prement el botó secundari del ratolí sobre la zona d'una pestanya ens apareixeran les diferents opcions de divisió (*split*)).

3) Visualitzador d'objectes

És la part superior de la barra lateral dreta i consta de tres pestanyes:

a) *Schemata*

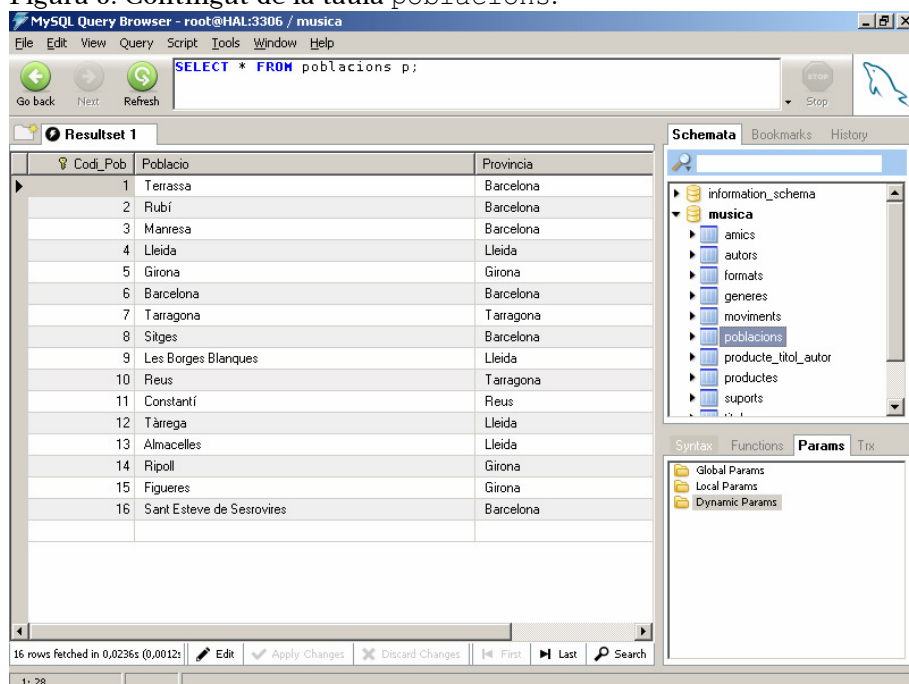
Ens mostra les bases de dades i ens permet seleccionar aquella amb la que vulguem treballar. Si en posar en marxa *MySQL Query Browser* hem indicat una base de dades (*Default Schema*) aquesta és la base de dades activa en l'*Schemata*. Dins la base de dades seleccionada, podem escollir qualsevol de les taules i executar-hi diferents tasques. Per exemple:

- Amb el botó secundari del ratolí tenim accés a tasques administratives que normalment es fan des de *MySQL Administrator*: modificar l'estructura de la taula (*edit*), eliminar la taula (*drop*), ...
- Si fem doble clic damunt la icona de la taula que ens interessi, veurem que ens apareix una sentència "select..." a l'àrea de consulta. Si repetim altra vegada doble clic damunt la icona de la mateixa taula, veurem el contingut de la taula (equivalent a l'execució de la sentència amb <ctrl> + <enter>).
- Així, per exemple, si fem doble clic damunt la taula *poblacions* de la base de dades *musica*, tenim el primer contacte amb el llenguatge SQL, doncs ens apareix la sentència SQL:

```
SELECT * FROM poblacions p;
```

i si tornem a fer doble clic damunt la taula se'ns visualitza el contingut de la taula *poblacions*, tal i com podem observar a la figura 3.

Figura 3. Contingut de la taula *poblacions*.



b) Bookmarks

Espai on emmagatzemem les consultes que utilitzem més sovint per tal de trobar-les ràpidament i executar-les quan sigui menester.

Per tal d'emmagatzemar la consulta existent a l'àrea de consulta, utilitzarem el menú *Query* i l'opció *Add Bookmark* o directament `<ctrl>+<B>`.

c) *History*

Enregistra automàticament la informació de totes les sentències SQL executades.

4) Visualitzador d'informació

És la part inferior de la barra lateral dreta i conté informació tècnica sobre diferents aspectes del llenguatge SQL que podem necessitar. Conté 4 pestanyes:

- *Syntax*, amb informació sobre la sintaxi de sentències SQL
- *Functions*, amb informació sobre la sintaxi de funcions incorporades en MySQL
- *Params*, amb informació sobre els paràmetres definits en la sessió actual
- *Trx*, amb informació sobre la transacció actual

En l'apartat 4.3.4. veurem les funcions que incorpora MySQL i en l'apartat 5.5.4. introduïrem el concepte de transacció.

5) Menús i submenús

No farem una explicació de les opcions dels menús i submenús donat que n'hi ha de molt intuïtives (sense problema), d'altres es corresponen a aspectes que no estudiem en aquest crèdit (programació) i d'altres s'aniran comentant a mida que avancem en el crèdit.

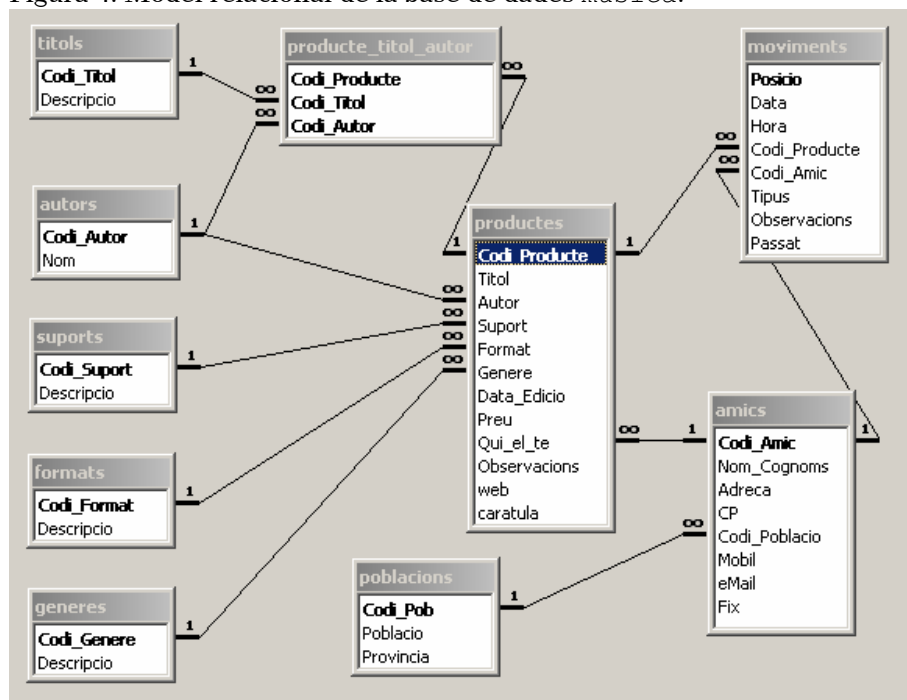
1.4.5. Altes, baixes i modificacions des de MySQL Query Browser

En l'anterior apartat hem presentat *MySQL Query Browser* i l'hem utilitzat per a fer una primera consulta sobre la base de dades *música* per la qual tenim el model relacional a la figura 4.

Observem que els noms de les taules i dels camps no contenen caràcters estranys (caràcters accentuats, 'ç', espais en blanc, 'ñ', ...) degut a que hi ha SGBD que no permeten la utilització d'aquests caràcters en els noms de taules i de camps. No ens cal conèixer si MySQL els permet o no!

Pensem que aquests caràcters poden portar-nos problemes en molts SGBD i tinguem en compte que en ocasions ens caldrà accedir a taules d'un SGBD des d'un altre SGBD. Per tant ens convé acostumar-nos a utilitzar caràcters "no estranys".

Figura 4. Model relacional de la base de dades música.



En les bases de dades ofimàtiques és molt fàcil consultar les dades així com efectuar altes, baixes i modificacions de registres des de l'entorn gràfic que incorporen. En realitat, qualsevol de les operacions esmentades (consulta, alta, baixa, modificació) així com les operacions de definició de les dades (creació/modificació de taules i establiment de relacions entre taules) i les operacions de definició d'usuaris i concessió de permisos d'accés, tot i que s'indiquin des d'una eina gràfica, aquesta les executa invocant sentències del llenguatge SQL, les quals coneixerem en els propers capítols.

Les primeres versions de SGBD no incorporaven interfícies gràfiques (és més, no existien encara aquestes interfícies) i, per tant, la única manera d'executar les operacions més amunt indicades era utilitzant directament el llenguatge SQL.

Amb l'aparició de les interfícies gràfiques, els SGBD han anat incorporant pantalles que permeten desenvolupar totes aquestes operacions. Així, ja hem vist que pel SGBD MySQL tenim *MySQL Administrator* i *MySQL Query Browser* facilitats per *MySQL AB* i *PHPMysqlAdmin* facilitat per la comunitat PHP. Les versions actuals del SGBD Oracle incorporen l'*Enterprise Manager Console* i així amb la majoria dels SGBD actuals.

En MySQL podem utilitzar *MySQL Query Browser* per a consultar les dades però també per efectuar altes, baixes i modificacions.

2. Llenguatge SQL: Tipus de dades. Consultes simples.

Iniciem, ara sí, l'estudi del llenguatge SQL i ho farem executant consultes senzilles a la base de dades. Abans, però, ens convé conèixer els diferents tipus de sentències SQL (subdivisió del llenguatge SQL) així com els diferents tipus de dades (numèriques, dates, cadenes,...) que ens podem trobar emmagatzemats a les bases de dades.

2.1. Tipus de sentències SQL.

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

- a) **Sentències destinades a la definició de les dades (LDD)**, que permeten definir els objectes (taules, camps, valors possible, regles d'integritat referencial, restriccions,...).
- b) **Sentències destinades al control sobre les dades (LCD)**, que permeten concedir i retirar permisos sobre els diferents objectes de la base de dades.
- c) **Sentències destinades a la consulta de les dades (LC)**, que permeten accedir a les dades en mode consulta.
- d) **Sentències destinades a la manipulació de les dades (LMD)**, que permeten actualitzar la base de dades (altes, baixes i modificacions).

Termes en anglès

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

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

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

SQL hostatjat

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

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

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

No forma part dels objectius d'aquest crèdit el fet de desenvolupar unitats de programació dins els SGBD; aquesta és una de les tasques reservades als administradors dels SGBD.

2.2. Tipus de dades

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

De tot això hem de deduir que per treballar amb un SGBD hem de conèixer els tipus de dades que facilita. Vegem els principals tipus de dades admesos en el SGBD MySQL 5.x.

2.2.1. Tipus de dades numèriques

Els principals tipus de dades numèriques que aporta MySQL els podem classificar en:

1) Tipus de dades per a emmagatzemar números enters:

- `TINYINT[(M)] [UNSIGNED] [ZEROFILL]`
- `SMALLINT[(M)] [UNSIGNED] [ZEROFILL]`
- `MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]`
- `INT[(M)] [UNSIGNED] [ZEROFILL]`
- `INTEGER[(M)] [UNSIGNED] [ZEROFILL]`

Cadascun d'aquests tipus ocupa un número diferent de bytes i, per tant, els valors que poden emmagatzemar són diferents. El tipus `INTEGER` és sinònim del tipus `INT`.

L'atribut optatiu `UNSIGNED` indica que només es permeten valors sense signe (zero i positius). Per defecte s'emmagatzemen valors amb signe.

La següent taula mostra els bytes que ocupa cada tipus i els intervals de valors permesos.

Tipus de dada	Bytes	Valors permesos per a tipus amb signe	Valors permesos per a tipus sense signe (UNSIGNED)
TINYINT	1	Entre -128 i 127	Entre 0 i 255
SMALLINT	2	Entre -32768 i 32767	Entre 0 i 65535
MEDIUMINT	3	Entre -8388608 i 8388607	Entre 0 i 16777215
INT	4	Entre -2147483648 i 2147483647	Entre 0 i 4294967295
BIGINT	8	Entre -9223372036854775808 i 9223372036854775807	Entre 0 i 18446744073709551615

L'atribut numèric optatiu **M** serveix per acotar el número màxim de dígit de la dada independentment dels valors permesos. Vegem-ne alguns exemples:

Definició del camp	Valors permesos
smallint (4) unsigned	Entre 0 i 9999
tinyint (2)	Entre -99 i 99
int (7)	Entre -9999999 i 9999999

L'atribut optatiu **ZEROFILL** serveix per indicar que la visualització dels valors d'aquests camps es mostri ple amb zeros per l'esquerra fins a l'amplada definida. En indicar **ZEROFILL**, MySQL defineix automàticament el camp de tipus **UNSIGNED**.

2) Tipus de dada **BOOL**, **BOOLEAN**

Aquest tipus és sinònim **TINYINT(1)** i serveix per a emmagatzemar els valors lògics CERT/FALS (valors SI/NO en alguns SGBD). Ara bé, enlloc de gestionar valors CERT/FALS o SI/NO, gestiona valors numèrics segons el següent conveni:

- El valor **ZERO** és considerat **FALS**
- Qualsevol valor diferent de **ZERO** és considerat **CERT**

En el futur, possiblement aquest tipus tindrà un funcionament d'acord amb l'estàndard SQL (valors **TRUE/FALSE**).

3) Tipus de dades per a emmagatzemar números decimals en punt flotant

Els principals tipus de dades que proporciona MySQL per emmagatzemar números decimals en punt flotant són:

- `FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]` (simple precisió)
- `DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]` (doble precisió)

De manera similar als tipus de dades enters, el que diferencia els diferents tipus de dades decimals és el número de bytes que ocupen i, per tant, l'interval de valors a emmagatzemar és diferent.

La següent taula mostra els intervals de valors permesos:

Tipus	Valors permesos
FLOAT	Entre 3.402823466E+38 i -1.175494351E-38
	0
	Entre 1.175494351E-38 i 3.402823466E+38
DOUBLE	Entre -1.7976931348623157E+308 i -2.2250738585072014E-308
	0
	Entre 2.2250738585072014E-308 i 1.7976931348623157E+308

L'atribut optatiu `M` permet acotar el número total de díigits decimals (part sencera + part decimal) mentre que l'atribut optatiu `D` permet acotar el número total de díigits decimals que segueixen al punt decimal (part decimal). Si `M` i `D` no s'indiquen, els valors s'emmagatzemen segons els valors permesos pel maquinari (processador).

MySQL permet altres tipus de dades decimals en punt flotant que no deixen de ser sinònims dels ja indicats:

- `DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL]`

Sinònim del tipus `DOUBLE`.

- `REAL[(M,D)] [UNSIGNED] [ZEROFILL]`

Sinònim del tipus `DOUBLE` a no ser que MySQL s'estigui executant en mode `REAL_AS_FLOAT`, fet que provocaria que es comportés com a sinònim del tipus `FLOAT`.

Modes de funcionament del servidor MySQL i dels clients

El servidor MySQL pot operar en diferents modes i és possible aplicar diferents modes als diferents clients connectats al servidor. Els modes en els que opera el servidor es poden indicar en posar-lo en marxa amb el comandament `mysqld --sql_mode="modes"` on modes es refereix a una llista dels diferents modes separats per coma. Així, per exemple, si poséssim en marxa el servidor amb `mysqld --sql_mode="REAL_AS_FLOAT"` els camps de tipus `REAL` es comportarien com a `FLOAT`.

Si es vol canviar algun mode de funcionament mentre el servidor és en execució, caldrà utilitzar els comandaments `set global sql_mode='modes'` per modificar el mode de funcionament del servidor o `set session sql_mode="modes"` per a modificar el funcionament per la sessió des d'on s'executa el comandament `set`.

- `FLOAT(p) [UNSIGNED] [ZEROFILL]`

Segons el valor numèric *p*, MySQL tracta el camp de diferent manera: si *p* està entre 0 i 24, el tipus esdevé `FLOAT` i si *p* està entre 25 i 53, el tipus esdevé `DOUBLE`. En qualsevol cas, no es defineixen els valors *M* i *D*.

MySQL facilita el tipus `FLOAT(p)` per compatibilitat amb ODBC.

ODBC

L'*Open Database Connectivity* permet establir connexions entre diferents SGBD.

4) Tipus de dades per a emmagatzemar números decimals en punt fix

- `DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]`

Aquest tipus de dada permet emmagatzemar valors en els que *M* és el número total de dígitos decimals (precisió, que correspon a part sencera + part decimal) i *D* és el número total de dígitos després del punt decimal (escala).

Les posicions ocupades pel punt decimal i el signe (pels números negatius) no són comptabilitzades en *M*. Si *D* és zero, els valors no tenen part decimal. El valor màxim per a *M* és 65 i el valor màxim per a *D* és 30. El valor per defecte per a *D* és 0 i per a *M* és 10.

Algun dels valors comentats pot ser lleugerament diferent segons la versió de MySQL.

Sinònims del tipus de dada `DECIMAL` són els següents:

- `DEC[(M[,D])] [UNSIGNED] [ZEROFILL]`
- `NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL]`
- `FIXED[(M[,D])] [UNSIGNED] [ZEROFILL]`

2.2.2. Tipus de dades alfanumèriques

Els principals tipus de dades alfanumèriques que aporta MySQL es poden classificar en:

1) Tipus de dades per a emmagatzemar cadenes de caràcters

Bàsicament, els tipus de dades que utilitzarem per a emmagatzemar cadenes de caràcters són `CHAR` i `VARCHAR`:

- `CHAR(M) [BINARY | ASCII | UNICODE]`

Permet emmagatzemar una cadena de caràcters que és emplenada amb espais per la dreta fins a la longitud indicada per l'atribut *M*, valor que pot anar entre 0 i 255.

L'atribut optatiu `BINARY` provoca que les ordenacions i les comparacions s'efectuïn segons el valor numèric del caràcter.

L'atribut optatiu `ASCII` provoca que les ordenacions s'efectuïn segons el conjunt de caràcters `latin1`, mentre que l'atribut optatiu `UNICODE` provoca que les ordenacions s'efectuïn segons el conjunt de caràcters `ucs2`.

- `VARCHAR(M) [BINARY]`

Permet emmagatzemar una cadena de caràcters de llargada màxima la indicada per l'atribut `M`, valor que pot anar entre 0 i 65535.

A diferència del tipus `CHAR(M)`, aquest tipus no emplena amb espais per la dreta i només enregistra els caràcters que s'ha introduït.

Per tant, caldria concloure que per a camps que hagin d'emmagatzemar cadenes de caràcters inferiors a 255 caràcters, per les que podem utilitzar `CHAR(M)` o `VARCHAR(M)`, seria millor utilitzar `CHAR(M)` si les cadenes tenen sempre una mateixa longitud (per exemple, el NIF d'una persona), mentre que seria millor utilitzar `VARCHAR(M)` si les cadenes poden tenir diferent longitud (nom i cognoms de persones, nom de les poblacions,...).

- `CHAR`

Aquest tipus s'utilitza per a enregistrar 1 caràcter i és equivalent a `CHAR(1)`.

2) Tipus de dades per a emmagatzemar cadenes de bytes

MySQL proporciona dos tipus de dades per a emmagatzemar cadenes de bytes:

- `BINARY(M)`, similar a `CHAR(M)`.
- `VARBINARY(M)`, similar a `VARCHAR(M)`.

3) Tipus de dades per a emmagatzemar grans quantitats d'informació.

MySQL proporciona quatre tipus de dades `BLOB` i `TEXT` per a emmagatzemar grans quantitats d'informació. Els 4 tipus només es diferencien en la llargada de la informació que poden emmagatzemar.

La taula següent ens mostra els quatre tipus de dades:

Blob	Text	Llargades possibles
TINYBLOB	TINYTEXT	Fins a 255 ($2^8 - 1$)
BLOB [(M)]	TEXT [(M)]	Fins a 65.535 o 64KB ($2^{16} - 1$)
MEDIUMBLOB	MEDIUMTEXT	Fins a 16.777.215 o 16MB ($2^{24} - 1$)
LONGBLOB	LONGTEXT	Fins a 4.294.967.295 o 4GB ($2^{32} - 1$)

L'atribut opcional **M** pels tipus **BLOB** i **TEXT** permet indicar una llargada opcional. MySQL crea el camp d'un dels 4 tipus indicats prenent el que més s'apropa per damunt al valor indicat per **M**.

Els camps **BLOB** no tenen un conjunt de caràcters assignat i, per tant, les ordenacions i les comparacions s'efectuen segons els valors numèrics dels bytes emmagatzemats. En canvi, els camps **TEXT** sí tenen un conjunt de caràcters assignat.

En molts aspectes, les columnes **BLOB** es poden veure com a columnes **VARBINARY** i les columnes **TEXT** com a columnes **VARCHAR**. Però hi ha diferències que s'escapen de l'àmbit d'aquest crèdit i que es poden consultar en la documentació del SGBD MySQL.

En principi utilitzarem columnes de tipus **BLOB** per a emmagatzemar dades binàries com les imatges i columnes de tipus **TEXT** per a emmagatzemar grans textos pels que no s'acostuma a efectuar recerques ni ordenacions (tot i que MySQL ho permet fer).

Similitud entre **TEXT** i

MEMO

Les columnes **TEXT** de MySQL es poden considerar similars a les columnes **MEMO** de MsAccess.

4) Tipus de dades per enregistrar enumeracions i conjunts de valors

MySQL facilita també aquests dos tipus de dades que ens poden facilitar l'enregistrament de dades en certs casos:

- `ENUM ('value1', 'value2', ...)`

Els camps d'aquest tipus (enumeració) només poden emmagatzemar un valor dels indicats a la llista de valors `'value1', 'value2', ...` o `NULL` si la columna ho permet.

Una columna de tipus **ENUM** pot tenir fins a 65.535 valors diferents.

- `SET ('value1', 'value2', ...)`

Els camps d'aquest tipus (conjunt) poden emmagatzemar zero o varis valors dels indicats a la llista de valors `'value1', 'value2', ...`.

Una columna de tipus **SET** pot tenir fins a 64 valors diferents.

2.2.3. Tipus de dades per emmagatzemar moments temporals

MySQL suporta els següents tipus de dades per a gestionar moments temporals (data i hora):

Tipus de dada	Observacions
DATE	Per valors de dates entre '01-01-1000' i '31-12-9999'
DATETIME	Combinació de data i hora que permet emmagatzemar moments temporals entre '00:00:00 01-01-1000' i '23:59:59 31-12-9999'
TIMESTAMP	Combinació de data i hora que permet emmagatzemar moments temporals entre '00:00:00 01-01-1970' i '23:59:59 31-12-2037'. Normalment aquest tipus de dada s'utilitza per enregistrar el moment temporal en el que s'ha efectuat l'alta o la darrera actualització de la fila. El SGBD permet assignar un moment temporal dins dels valors permesos, però en cas de no assignar valor o assignar valor NULL, MySQL assigna el moment temporal en el que s'ha produït la modificació.
TIME	Per valors d'hora entre '-838:59:59' i '838:59:59'
YEAR[(2 4)]	Per valors d'anys en 2 o 4 (per defecte) dígits.

La següent taula mostra els formats en que MySQL tracta aquests tipus de dades:

Tipus de dada	Format de tractament (visualització i assignació)
DATE	'YYYY-MM-DD'
DATETIME	'YYYY-MM-DD HH:MM:SS'
TIMESTAMP	'YYYY-MM-DD HH:MM:SS'
TIME	'HH:MM:SS'

Les columnes de tipus `DATE`, `DATETIME` i `TIMESTAMP` es poden visualitzar numèricament (sense guions) afegint `+0` al nom de la columna. Així mateix, per tal d'assignar valors a aquests tipus de columnes es pot utilitzar expressions de cadenes o expressions numèriques.

A diferència d'altres SGBD, l'actual versió de MySQL no permet configurar el servidor de manera que el tractament per defecte d'aquests tipus de dades s'adeqüi a la configuració regional del lloc on és instal·lat el servidor ni del lloc des d'on s'estableixen les connexions. Igual en futures versions... Podrem, però, utilitzar algunes de les funcions que aporta MySQL per a visualitzar aquests tipus de columnes en els formats que ens sembli més adequats.

2.3. Consultes simples

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

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

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

2.3.1. Clàusules `SELECT` i `FROM`.

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

```
SELECT <expressió/columna>, <expressió/columna>, ...  
FROM <taula>, <taula>, ...;
```

- `SELECT` permet escollir columnes i/o valors derivats d'elles (resultats d'expressions en els que acostumen a intervenir les columnes); 
- `FROM` permet especificar les taules on cal anar a cercar les columnes o sobre les que es calcularan els valors resultats de les expressions. 

Observacions:

- Totes les instruccions que apareguin a partir d'ara les podem posar en pràctica per mitjà de la consola de treball `mysql` o des de l'àrea de consulta de l'aplicatiu *MySQL Query Browser*. 

Recordem que caldrà seleccionar la base de dades amb la que treballar abans de procedir a executar cap tipus de sentència SQL. Des de *MySQL Query Browser* és molt senzill escollir la base de dades des de la pestanya *Schemadata* de la barra lateral dreta. Però des de la consola de treball `mysql` cal utilitzar el comandament `use`:

```
mysql> use <nom_base_dades>;
```

Totes les instruccions SQL que segueixin s'efectuaran a la base de dades *musica* ja instal·lada. Així doncs, des de la consola de treball escriuríem:

```
mysql> use musica;
```

A més del comandament `use`, altres comandaments interessants d'utilitzar des de la consola de treball són:

```
mysql> show tables;
```

que permet visualitzar les taules existents a la base de dades seleccionada;

```
mysql> desc <nom_taula>;
```

que permet veure la descripció de les diferents columnes de la taula.

La informació sobre les taules també es pot visualitzar des de l'aplicació *MySQL Administrator* seleccionant la base de dades per la secció *Catalogs*.

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

Exemple: Mostrar els codis, nom i província de les poblacions existents a la bases de dades.

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

```
select codi_pob, poblacio, provincia  
from poblacions;
```

Exemple: Mostrar codi, títol, data d'edició i preu -en pessetes- dels productes enregistrats a la base de dades, tenint en compte que els preus enregistrats a la base de dades estan en euros.

En aquest cas la columna "preu en pessetes" cal obtenir-la en base a un càlcul en el que intervé una de les columnes de la taula "productes".

```
SELECT codi_producte, titol, data_edicio, preu*166.386  
FROM productes;
```

En executar aquesta consulta podem comprovar el que comentàvem en un apartat anterior sobre el format de visualització de les columnes de tipus DATE.

Observacions:

- En la clàusula `SELECT` es pot utilitzar un asterisc per a indicar que es volen totes les columnes de les taules seleccionades en la clàusula `FROM`. 
- El llenguatge SQL utilitza els noms reals de les columnes com a títols en la presentació del resultat. Es permet donar un nom alternatiu a cada columna, és a dir, donant un **àlies** a cada columna. Per aconseguir-ho, cal escriure l'alias tancat entre dobles cometes i després de la corresponent columna i abans de la coma que separa amb la següent columna. 

Exemple: Mostrar codi, títol i preus en euros i en pessetes utilitzant per títols de les columnes: "Codi", "Títol", "Preu en Ptes." i "Preu en Eur."

```
SELECT codi_producte "Codi", titol "Títol",  
       preu "Preu en Eur.", preu*166.386 "Preu en Ptes."  
FROM productes;
```

Exemple: Mostrar tota la informació existent sobre els nostres amics.

```
select * from amics;
```

Observacions:

- La clàusula `FROM` pot fer referència a taules d'una altra base de dades. Llavors la taula s'indica com: `<nom_esquema>.<nom_taula>` 
- El llenguatge SQL efectua el producte cartesià de totes les taules que troba a la clàusula `FROM`. En aquest cas, si hi ha columnes amb mateix nom a diferents taules i hem de referir-nos-hi, utilitzem la nomenclatura anterior `<nom_taula>.<nom_columna>`. I fins i tot utilitzem `<nom_esquema>.<nom_taula>.<nom_columna>` per accedir a una columna d'una taula d'una altra base de dades. 
- El llenguatge SQL permet definir **àlies** per a una taula. Per aconseguir-ho, cal escriure l'alias en la clàusula `FROM` després del nom de la taula i abans de la coma que separa amb la següent taula. 
- De la mateixa manera que es per diferenciar les taules s'utilitza la sintaxi `<nom_taula>.<nom_columna>`, si ens cal referenciar una

taula d'una altra base de dades gestionada pel servidor MySQL a la qual tinguem permís d'accés, caldrà utilitzar la sintaxis `<nom_base_dades>.<nom_taula>.<nom_columna>`. 

Exemple: Tenint activada la base de dades `musica` mostrar els usuaris donats d'alta en el SGBD.

La informació que se'ns demana resideix a la taula `user` de la base de dades `mysql` que es crea en el procés d'instal·lació de MySQL. La base de dades `mysql` enregistra informació sobre usuaris i privilegis d'aquests usuaris. L'usuari `root` té accés a totes les informacions emmagatzemades en les taules de `mysql`.

La taula `user` conté molta informació. Anem a visualitzar únicament les columnes `host`, `user` i `password`.

```
select host, user, password from mysql.user;
```

Observem que si no hem afegit encara cap usuari des de la instal·lació del SGBD, la informació que ens dona (execució des de la consola `mysql`) és:

```
+-----+-----+-----+
| host      | user | password |
+-----+-----+-----+
| localhost | root |          |
+-----+-----+-----+
1 row in set (0.00 sec)
```

És a dir, el nostre SGBD té un únic usuari de nom `root` que es pot connectar únicament des de la pròpia màquina que té el SGBD instal·lat (`localhost`) i no té contrasenya assignada. La gestió d'usuaris és una tasca administrativa a la base de dades.

Exemple: Mostrar totes les combinacions possibles dels diferents formats amb els diferents suports.

Es tracta d'efectuar el producte cartesià de les taules `suports` i `formats`.

```
select * from suports, formats;
```

Exemple: Del producte cartesià de l'exemple anterior, mostrar-ne únicament la descripció del suport i la descripció del format

```
select formats.descripcio, suports.descripcio
from formats, suports;
```

Una altra possibilitat és, amb la utilització d'àlies per les taules i de títols més entenedors per les columnes (doncs en l'anterior consulta ambdues columnes tenien el mateix nom):

```
select f.descripcio "Format", s.descripcio "Suport"
from formats f, suports s;
```

2.3.2. Clàusula WHERE

La clàusula WHERE permet establir els criteris de recerca sobre les files generades per la clàusula FROM.

Aquesta clàusula s'afegeix darrera de la clàusula FROM:

```
SELECT <expressió/columna>, <expressió/columna>, ...
FROM <taula>, <taula>, ...
WHERE <condició_de_recerca>;
```

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

1) Operadors aritmètics

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

També tenim l'operador DIV per a la divisió entera i l'operador MOD o % per al mòdul.

Divisió entera

La divisió entera entre dos números enters és el quocient enter de la divisió (sense baixar decimals), mentre que el mòdul entre dos números enters és el residu de la divisió entera. Exemples:

```
234 MOD 10 = 4
234 DIV 10 = 23
234 % 10 = 4
```

2) Operadors de comparació

Disposem de diferents operadors per efectuar comparacions:

- Els típics operadors = (igualtat), != o <> (desigualtat), > (major que), < (menor que), >= (major o igual que), <= (menor o igual que) per a

efectuar comparacions entre dades obtenint un resultat lògic: cert o fals.

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

'%' per indicar qualsevol cadena de zero o més caràcters

'_' per indicar qualsevol caràcter

Així:

`LIKE 'Torres'` Compara amb la cadena 'Torres'

`LIKE 'Torr%'` Compara amb qualsevol cadena iniciada per 'Torr'

`LIKE '%S%'` Compara amb tota cadena que contingui una 'S'

`LIKE '_o%'` Compara amb qualsevol cadena que tingui una 'o' per segon caràcter

`LIKE '___'` Compara amb qualsevol cadena de dos caràcters

- Un últim conjunt d'operadors lògics:

`[NOT] BETWEEN <valor_1> AND <valor_2>`

que permet efectuar la comparació entre dos valors

`[NOT] IN (llista_valors)`

que permet comparar amb una llista de valors

`IS [NOT] NULL`

que permet reconèixer si estem davant d'un valor NUL

`<comparador genèric> ANY (llista_valors_de_SELECT)`

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

`<comparador genèric> ALL (llista_valors_de_SELECT)`

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

Observem que: `=ANY` és equivalent a `IN`
`!=ALL` és equivalent a `NOT IN`

Una llista de valors es pot construir a partir de (`valor1`, `valor2`, `valor3`, ...) o a partir d'una consulta `SELECT` (anomenada subconsulta doncs és una consulta dins una altra consulta). En MySQL els operadors `ALL` i `ANY` només poden anar seguits d'una llista de valors construïda a

partir d'una subconsulta, però en altres SGBD seria possible utilitzar-los seguits d'una llista de valors indicats explícitament.

3) Operadors lògics

Les operacions lògiques són aquelles que s'executen sobre valors lògics (cert i/o fals) per tal d'aconseguir un altre valor lògic.

MySQL facilita els 4 operadors lògics següents:

- **Negació** (indicada amb el prefix `NOT` o `!`) que canvia el valor de l'operand.
- **Conjunció** (indicada amb l'operador `AND` o `&&` entre els dos operands) que pren per resultat el valor cert si i només si els dos operands tenen el valor cert i pren el valor fals en qualsevol altre cas.
- **Disjunció no exclusiva** (indicada amb l'operador `OR` o `||` entre els dos operands) que pren per resultat el valor fals si i només si els dos operands tenen el valor fals i pren el valor cert en qualsevol altre cas.
- **Disjunció exclusiva** (indicada amb l'operador `XOR` entre els dos operands) que pren per resultat el valor cert quan els 2 operands tenen valor diferent (cert i fals) i pren el valor fals quan els 2 operands tenen el mateix valor.

Per entendre millor el comportament dels operadors lògics ens pot ajudar la utilització de les **taules de veritat**. No són altra cosa que un procés que ens permet esbrinar, amb claredat i fiabilitat, tots els possibles resultats d'una operació lògica a partir dels valors dels operands. Això es factible degut a que no hi ha un gran nombre de valors inicials possibles. Vegem-ho.

Suposem que tenim dos valors lògics *A* i *B* els quals volem operar amb les operacions binàries *conjunció* i *disjunció*. *A* i *B* només poden tenir dos valors; per tant, la combinació de les dues dades només pot considerar quatre possibles situacions, és a dir, un nombre petit de situacions fàcilment representable.

A	B	A and B	A or B	A xor B
cert	cert	cert	cert	fals
cert	fals	fals	cert	cert
fals	cert	fals	cert	cert
fals	fals	fals	fals	fals

Pel cas de la *negació* també podem utilitzar una taula de veritat, però en aquest cas, s'aplica a una única dada lògica ja que la negació és una operació que s'aplica sobre un únic operand.

A	no A
cert	fals
fals	cert

Exemple: Mostrar els productes (codi, títol, preu) que tenen un preu superior als 20 €.

```
select codi_producte, titol, preu
from productes
where preu>20;
```

Exemple: Mostrar els productes (codi, títol, data d'edició) editats a partir de l'any 2003.

```
select codi_producte, titol, data_edicio
from productes
where data_edicio>="2003-01-01";
```

Comparació de dades de tipus temporal

Aquest exemple ens serveix també per a veure com es comparen dades de tipus temporal en MySQL. Cal tenir present que cada SGBD té la seva pròpia forma de tractar les comparacions amb dades de tipus temporal.

A tall d'exemple la condició de l'anterior clàusula WHERE en altres SGBD seria:

```
data_edicio>= to_date ("01-01-2003", "dd-mm-yyyy") en el SGBD Oracle
data_edicio>= #01-01-2003# en el SGBD MsAccess
```

Exemple: Mostrar els productes resultat de la intersecció dels dos darrers exemples, és a dir, productes que tenen un preu superior als 20€ i que han estat editats a partir de l'any 2003.

```
select codi_producte, titol, preu
from productes
where preu>20
and data_edicio>="2003-01-01";
```

Observem que utilitzem l'operador `and` degut a que volem que es compleixin les dues condicions simultàniament.

Exemple: Mostrar els productes que tenen un preu superior als 20€ o que han estat editats a partir de l'any 2003 (o les dues condicions).

```
select codi_producte, titol, preu
from productes
where preu>20
or data_edicio>="2003-01-01";
```

Utilitzem l'operador `or` degut a que volem els productes que compleixin qualsevol de les dues condicions (o les dues simultàniament).

Exemple: Mostrar els productes que, o tenen un preu superior als 20€, o que han estat editats a partir de l'any 2003 (només una de les dues condicions).

```
select codi_producte, titol, preu
from productes
where preu>20
xor data_edicio>="2003-01-01";
```

Observem que utilitzem l'operador `xor` degut a que volem que els productes que compleixin una i només una de les dues condicions.

Exemple: Mostrar els productes que tenen un preu entre 15 i 25 €.

```
select codi_producte, titol, preu
from productes
where preu >= 15 and preu <= 25;
```

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

```
select codi_producte, titol, preu
from productes
where preu between 15 and 25;
```

Exemple: Mostrar els amics de les poblacions 2 i 15.

```
select codi_amic, nom_cognoms
from amics
where codi_poblacio=2 or codi_poblacio=15;
```

Podem, però, utilitzar els operadors `IN` :

```
select codi_amic, nom_cognoms
from amics
where codi_poblacio in (2,15);
```

En molts SGBD també es podria utilitzar l'operador `=ANY`:

```
select codi_amic, nom_cognoms
```

```
from amics
where codi_poblacio = ANY (2,15);
```

Però en aquesta versió de MySQL això no és factible. Ara bé, "feta la llei, feta la trampa", i podem utilitzar =ANY seguit d'una subconsulta:

```
select codi_amic, nom_cognoms
from amics
where codi_poblacio = ANY (select codi_pob
                           from poblacions
                           where codi_pob=2 or codi_pob=15);
```

Exemple: Mostrar els amics que no són de les poblacions 2 ni 15.

```
select codi_amic, nom_cognoms
from amics
where codi_poblacio <> 2 and codi_poblacio <> 15;
```

Podem, però, utilitzar els operadors NOT IN :

```
select codi_amic, nom_cognoms
from amics
where codi_poblacio not in (2,15);
```

En molts SGBD també es podria utilitzar l'operador <>ALL:

```
select codi_amic, nom_cognoms
from amics
where codi_poblacio <> ALL (2,15);
```

Però en aquesta versió de MySQL això no és factible. i de forma similar a = ANY podem utilitzar <> ALL seguit d'una subconsulta:

```
select codi_amic, nom_cognoms
from amics
where codi_poblacio <> ALL (select codi_pob
                           from poblacions
                           where codi_pob=2 or codi_pob=15);
```

3. Llenguatge SQL: Funcions incorporades. Resultats ordenats.

Continuem l'aprenentatge del llenguatge SQL per tal d'aconseguir millors consultes i per això ens cal utilitzar els diferents recursos que facilita, com són les funcions que incorpora i les possibilitats d'ordenació.

3.1. Taula dual

Alguns SGBD incorporen una taula fictícia per a efectuar càlculs independents de cap taula de la base de dades aprofitant la potència de la sentència `SELECT`. En tal cas la sintaxi emprada és:

```
select expressió/variable_sistema,... from <taula_fictícia>;
```

MySQL incorpora també aquesta taula fictícia anomenada `dual`, però la sintaxi SQL de MySQL permet no indicar la clàusula `FROM`.

Exemple: Mostrar la data i l'hora del sistema.

```
select current_date, current_time from dual;
```

o sense utilitzar la taula `dual`:

```
select current_date, current_time;
```

En aquest exemple hem utilitzat dues variables del sistema que contenen en tot moment la data i l'hora actuals en el servidor MySQL. La majoria de SGBD incorporen variables de sistema similars (no tenen per què coincidir en el nom) i d'altres que ens poden donar informació important.

3.2. Sensibilitat a majúscules/minúscules en cadenes

En l'apartat anterior hem conegut la clàusula `WHERE` i hem vist exemples en els que construïem diferents condicions a utilitzar en dita clàusula. Fixem-nos, però, que no hem vist cap exemple en el que comparéssim

dades alfanumèriques i això és degut a que el tema es mereix un apartat específic.

Exemple: Mostrar els amics que tenen per inicial del seu nom una 'M'.

```
select nom_cognoms from amics where nom_cognoms like 'M%';
```

Aquesta instrucció mostra els amics tals que el seu nom comença per la lletra 'M'. Observem que si haguéssim escrit la 'M' en minúscula el resultat obtingut seria el mateix. Això és degut a que MySQL és no sensible a majúscules/minúscules (no *case sensitive*) i, per tant, a l'hora de cercar amics que tinguin nom iniciat per la lletra 'M', és indiferent escriure

```
where nom_cognoms like 'M'%'
```

o

```
where nom_cognoms like 'm%'
```

CASE SENSITIVE

En informàtica s'utilitza aquest terme per a indicar que les operacions amb dades alfanumèriques (comparacions, recerques, ordenacions) diferencien els caràcters en majúscula dels caràcters en minúscula.

Aquest funcionament no és idèntic en tots els SGBD i pot portar a greus problemes de compatibilitat d'instruccions SQL en diferents SGBD.

Sensibilitat a majúscules/minúscules en el SGBD Oracle

A tall d'exemple, podem comentar que les diferents versions aparegudes del SGBD Oracle han estat sempre *case sensitive* fins la versió 10 en la que es pot configurar el funcionament del SGBD com a *case sensitive* o no *case sensitive*.

MySQL permet indicar quines columnes de quines taules han de ser tractades com a *case sensitive* utilitzant l'opció `BINARY` en definir la columna. Encara no hem vist com crear les taules, però sí hem conegut l'opció `BINARY` en els tipus de dades alfanumèriques.

Les columnes alfanumèriques de les taules de la base de dades `musica` no incorporen l'opció `BINARY`, per la qual cosa totes les comparacions, recerques i ordenacions que efectuem sobre aquestes taules en el seu estat inicial (sense l'opció `BINARY`) no seran sensibles a majúscules/minúscules.

Ens podem preguntar si en MySQL, donada una columna alfanumèrica sense l'atribut `BINARY`, és possible tractar-la com si fos `BINARY`. La resposta és afirmativa utilitzant l'operador `BINARY` davant el nom de la columna. És a dir, donada una columna alfanumèrica, si en utilitzar-la en una sentència SQL la precedim de l'operador `BINARY`, el seu comportament passa a ser *case sensitive*. Aquest operador es pot utilitzar davant de columnes alfanumèriques i també davant cadenes constants.

Així, per exemple,

```
select nom_cognoms from amics where nom_cognoms like binary 'M%';
```

mostrarà els amics tals que el seu nom comença per 'M' en majúscula.

Cal tenir present que per MySQL la comparació de cadenes amb l'operador '=' dona resultat cert si les dues cadenes (eliminant els espais per la dreta) són iguals. És a dir:

```
select 'x' = 'x';
```

té resultat cert, mentre que

```
select 'x' like 'x';
```

té resultat fals. En canvi, si utilitzem l'operador `BINARY` el comportament és molt diferent:

```
select binary 'x' = binary 'x';
```

té resultat fals.

Així, doncs, ja coneixem com efectuar comparacions en MySQL, però en altres SGBD haurem d'esbrinar el comportament que facilita. En cas de SGBD que sempre actuïn de forma *case sensitive* (Oracle fins les versions 9.x), si en algun moment no hem de distingir entre majúscules i minúscules haurem d'utilitzar funcions de conversió a majúscules (`upper()`) o a minúscules (`lower()`). Així, si volguéssim assegurar que estem agafant tots els amics que el seu nom comença per 'M' o 'm', podríem fer:

```
select nom_cognoms from amics
where upper(nom_cognoms) like 'M%';
```

Clar que també haguéssim pogut escriure:

```
select nom_cognoms from amics
where nom_cognoms like 'M%' or nom_cognoms like 'm%';
```

Exemple: Mostrar els amics que tenen alguna 'R' majúscula en el seu cognom.

En MySQL una solució seria:

Funcions

Les funcions en els llenguatges informàtics són similars a les funcions estudiades en matemàtiques: són càlculs que s'efectuen a partir d'unes dades inicials (indicades entre els parèntesis que acompanyen al nom de la funció) per tal d'assolir un resultat final.

En el proper apartat veurem les funcions que facilita el llenguatge SQL de MySQL.

```
select codi_amic, nom_cognoms from amics where nom_cognoms like binary '%R%';
```

En SGBD que actuïn sempre en mode *case sensitive* n'hi hauria prou amb:

```
select codi_amic, nom_cognoms
from amics
where nom_cognoms like '%R%';
```

Exemple: Mostrar els amics que no tenen la 'R' (ni majúscula ni minúscula) com a tercera lletra del seu nom.

En MySQL una solució seria:

```
select codi_amic, nom_cognoms
from amics
where nom_cognoms not like '___R%';
```

En canvi, en SGBD que actuïn sempre en mode *case sensitive*:

```
select codi_amic, nom_cognoms
from amics
where upper(nom_cognoms) not like '___R%';
```

Exemple: Mostrar el codi i la data d'edició d'un producte anomenat 'La Macarena'.

En MySQL, tenint en compte que la columna `títol` no és `BINARY`, podem efectuar la comparació escrivint 'La Macarena' de qualsevol manera en el referent a majúscules/minúscules:

```
select codi_producte, data_edicio
from productes
where titol='La Macarena';
```

Ara bé, si la columna `títol` hagués estat `BINARY` o en un SGBD amb funcionament *case sensitive*, per assegurar-nos de trobar el producte, hauríem de posar:

```
select codi_producte, data_edicio
from productes
where upper(titol)='LA MACARENA';
```

I si es desitja cercar alguna cadena que contingui els caràcters '_' o '%' ?
En tal cas hem de poder distingir el caràcter '_' com a caràcter comodí de

recerca del caràcter '_' com a caràcter a cercar dins la cadena. Per aconseguir-ho utilitzarem la següent sintaxis de l'operador `LIKE`: 

```
LIKE cadena ESCAPE 'caracter'
```

El caràcter que segueix la paraula `ESCAPE` s'utilitza per precedir als caràcters '_' o '%' quan es volen cercar dins la cadena.

Exemple: Mostrar codi i descripció dels títols que tenen algun caràcter '_' en la seva descripció.

```
select codi_titol, descripcio
from titols
where descripcio like '%@_%' escape '@';
```

En aquest cas el caràcter d'escapament '@', en utilitzar-lo com a caràcter precedent del '_' en la cadena que acompanya l'operador `LIKE`, fa que el resultat sigui cert pels títols que tenen per descripció qualsevol número de caràcters (primer '%') seguit del caràcter '_' seguit de qualsevol número de caràcters (segon '%').

Es pot utilitzar qualsevol caràcter com a caràcter `ESCAPE`. Si precisament es necessita cercar el caràcter `ESCAPE`, s'escriu dues vegades. Així, per exemple,

```
select codi_titol, descripcio
from titols
where descripcio like '%&&%' escape '&';
```

mostra tots els títols que tenen algun '&' dins la seva descripció. Clar que s'hagués obtingut el mateix resultat fent:

```
select codi_titol, descripcio
from titols
where descripcio like '%&%';
```

però la penúltima sentència és un exemple de com cercar el caràcter `ESCAPE` si és menester.

3.3. Consulta de columnes BLOB

Les columnes `BLOB` permeten emmagatzemar informació de qualsevol tipologia i s'utilitzen, sobretot, per emmagatzemar-hi informació multimèdia.

La taula `productes` de la nostra base de dades incorpora la columna `caràtula` de tipus `MEDIUMBLOB` pensada per enregistrar-hi la imatge de la caràtula de cada producte. En el seu estat inicial, la taula `productes` porta incorporada les caràtules d'un parell de productes.

Cal anar en compte en les sentències `SELECT` sobre taules que tenen columnes `BLOB`, doncs no es pot visualitzar, via `SELECT`, la informació que contenen.

Si des de la consola `mysql` executem la sentència:

```
select * from productes where caratula is null;
```

visualitzarem totes les columnes de la taula `productes` i, fins i tot, la columna `caràtula` sense cap valor (doncs hem seleccionat les files que no tenen valor per aquesta columna).

Però si des de la consola `mysql` executem la sentència contrària:

```
select * from productes where caratula is not null;
```

el resultat consisteix en el bolcat per pantalla de tots els caràcters binaris que corresponen a les imatges dels productes seleccionats i, evidentment, tot és intel·ligible.

La conclusió és, per tant, que si estem utilitzant columnes `BLOB` per emmagatzemar informació no textual (imatges, so,...) haurem d'anar en compte d'incloure aquestes columnes en les sentències `SELECT`.

La gestió de les columnes `BLOB` per emmagatzemar informació no textual s'acostuma a fer des d'aplicacions informàtiques que interactuen amb la base de dades. Així, per exemple, nosaltres utilitzarem *MySQL Query Browser* per a gestionar les caràtules de la taula `productes`.

Anem a utilitzar *MySQL Query Browser* per a gestionar columnes `BLOB` seguint les indicacions de la secció "Recursos" del web d'aquest crèdit.

3.4. Funcions incorporades per MySQL

Els SGBD acostumen a incorporar funcions utilitzables des del llenguatge SQL. El llenguatge SQL, per sí mateix, no incorpora funcions genèriques. Únicament incorpora les anomenades funcions d'agrupament que veurem en l'estudi de la clàusula `GROUP BY`.

Les funcions es poden utilitzar dins d'expressions i actuen amb els valors de les columnes, variables o constants en les clàusules `SELECT`, `WHERE` i `ORDER BY` (clàusula pendent de veure). També és possible utilitzar funcions de funcions, és a dir, es permet el niament.

Vegem, a continuació, un recull de les principals funcions que facilita el SGBD MySQL. No hi són totes. Sempre caldrà visitar la documentació de MySQL.

La versió actual de MySQL té un error en executar certes funcions incorporades si es deixa algun espai entre el nom de la funció i el parèntesi d'obertura. Per tant, ull amb aquest problema i no hi deixem l'espai en blanc.

1) Matemàtiques

Funció	Descripció	Exemples
<code>ABS (n)</code>	Retorna el valor absolut de n	<code>ABS (-10)</code> retorna 10
<code>CEIL (n)</code> , <code>CEILING (n)</code>	Retorna el valor enter immediatament superior o igual a n	<code>CEIL (10.6)</code> retorna 10
<code>FLOOR (n)</code>	Retorna el valor enter immediatament inferior o igual a n	<code>FLOOR (10.6)</code> retorna 10
<code>MOD (m, n)</code>	Retorna el residu de la divisió entera entre m i n És equivalent a l'operador MOD vist més amunt	<code>MOD (15, 2)</code> retorna 1
<code>POWER (m, n)</code> , <code>POW (m, n)</code>	Retorna el valor m elevat a la potència n	<code>POWER (4, 2)</code> retorna 16
<code>ROUND (n [, m])</code>	Retorna el valor n arrodonit a m decimals. Si m no existeix, es suposa zero. Si m és negatiu, l'arrodoniment de dígit s'efectua per l'esquerra del punt decimal.	<code>ROUND (1.6724, 1)</code> retorna 1.7 <code>ROUND (1.6724)</code> retorna 2 <code>ROUND (5462.67, -2)</code> retorna 5500
<code>SQRT (n)</code>	Retorna l'arrel quadrada de n, que no pot ser negatiu	<code>SQRT (5)</code> retorna 2.23606798
<code>TRUNCATE (n [, m])</code>	Retorna n truncat a m decimals. Si m no existeix, es suposa zero. Si m és negatiu, el truncament de dígit s'efectua per l'esquerra del punt decimal.	<code>TRUNCATE (1.6724, 1)</code> retorna 1.6 <code>TRUNCATE (1.6724)</code> retorna 1 <code>TRUNCATE (5462.67, -2)</code> retorna 5400

Aquestes no són les úniques. Hi ha també funcions per a càlculs trigonomètrics i logarítmics.

2) De cadenes de caràcters

Funció	Descripció	Exemples
<code>ASCII (cad)</code>	Retorna el valor numèric del primer caràcter de la cadena. Retorna 0 si es tracta de la cadena buida i retorna NULL si es tracta de la cadena NULL.	<code>ASCII ('2')</code> retorna 50
<code>CONCAT (cad1, cad2)</code>	Retorna la concatenació de les cadenes cad1 i cad2. Retorna NULL si algun argument és NULL	<code>CONCAT ('Avui és', 'Diumenge')</code> retorna 'Avui és Diumenge'
<code>FORMAT (X, D)</code>	Retorna el número X en una cadena amb un format similar a <code>#,###,###.##</code> arrodonit a D llocs decimals.	<code>FORMAT (2321.423, 2)</code> retorna "2,321.42"
<code>INSERT (cad, pos, lon, aux)</code>	Retorna la cadena cad substituint subcadena a partir de la posició pos i de llargada lon amb el contingut de la cadena aux.	<code>INSERT ("Diumenge", 3, 2, "Hola")</code> retorna "DiHOLAenge"

Funció	Descripció	Exemples
INSTR(cad, subcad) LOCATE(subcad, cad[, n]) POSITION(subcad IN cad)	Retorna la posició de la primera ocurrència de subcad dins cad. Retorna 0 si no es troba subcad dins cad. En LOCATE, el paràmetre n permet indicar la posició a partir d'on cal cercar subcad dins cad.	INSTR("Diumenge", "me") retorna 4 INSTR("Diumenge", "ma") retorna 0 LOCATE("me", "Diumenge") retorna 4
LCASE(cad), LOWER(cad)	Retorna la cadena cad amb totes les lletres convertides a minúscules	LOWER('Avui') retorna 'avui'
LEFT(cad, lon)	Retorna la subcadena de cad de llargada lon	LEFT("Diumenge", 4) retorn "Dium"
LENGTH(cad)	Retorna la longitud de la cadena cad	LENGTH("Diumenge") retorna 8
LPAD(cad1, n[, cad2])	Retorna cad1 amb longitud n i justificada per la dreta. La cadena cad2 s'utilitza per omplir per l'esquerra.	LPAD('R', 7, 'xs') retorna 'xsxsxsR' LPAD('R', 3, 'Hola') retorna 'HoR'
LTRIM(cad)	Retorna cad sense espais per l'esquerra.	LTRIM(' Hola') retorna 'Hola'
MID(cad, pos, ll) SUBSTRING(cad, pos[, ll])	Retorna la subcadena de la cadena cad a partir de a la posició pos i fins a ll caràcters. El valor de ll pot ser inferior a 1. El valor de pos pot ser negatiu i s'interpreta com a la posició començant pel final.	SUBSTRING('Catalunya', 3, 3) retorna 'tal' SUBSTRING('Catalunya', -5, 2) retorna 'lu' MID('Catalunya', 3, 3) retorna 'tal'
REPEAT(cad, n)	Repeteix la cadena cad tantes vegades com indica n	REPEAT("la", 3) retorna "lalala"
REPLACE(cad, t[, subst])	Retorna cad amb cada aparició de la cadena t substituïda per la cadena subst. Si no hi ha cadena substituïda, es procedeix a eliminar cada aparició de la cadena tros.	REPLACE('Alibaba', 'ba', 'je') retorna 'Alijeje' REPLACE('Alibaba', 'ba') retorna 'Ali'
REVERSE(cad)	Retorna la cadena cad invertida	REVERSE("zxcv") retorna "vcxz"
RIGHT(cad, lon)	Retorna els darrers lon caràcters de cad	RIGHT("Diumenge", 4) retorna "enge"
RPAD(cad1, n[, cad2])	Retorna cad1 amb longitud n i justificada per l'esquerra. La cadena cad2 s'utilitza per omplir per la dreta.	RPAD('R', 7, 'xs') retorna 'Rxsxsxs' RPAD('R', 3, 'Hola') retorna 'RHo'
RTRIM(cad)	Retorna la cadena sense els espais per la dreta	RTRIM('Hola ') retorna 'Hola'
SPACE(n)	Retorna una cadena formada per n espais	SPACE(3) retorn ' '
TRIM(cad)	Elimina els espais per la dreta i per l'esquerra	TRIM(' Hola ') retorna 'Hola'
UCASE(cad), UPPER(cad)	Retorna la cadena cad amb totes les lletres convertides a majúscules	UPPER('Avui') retorna 'AVUI'

3) De gestió de moments temporals (dates i hores)

Hi ha SGBD que incorporen els operadors + i - per al tractament de dates, de manera que:

- $d1 + n$, on $d1$ és data i n és un número natural, calcula la data posterior a $d1$ en n dies
- $d1 - n$, on $d1$ és data i n és un número natural, calcula la data anterior a $d1$ en n dies
- $d1 - d2$ on $d1$ i $d2$ són dates calcula el número de dies entre $d1$ i $d2$.

Però el SGBD MySQL no facilita aquests tractaments i, per tant, haurem d'utilitzar forçosament les funcions incorporades per a la gestió de dates.

Funció	Descripció	Exemples
DATE_ADD(d, INTERVAL tipus)	n Retorna la data d incrementada en n unitats indicades pel tipus (més avall hi ha la taula amb els tipus possibles)	DATE_ADD("2006-04-23", INTERVAL 10 DAY) retorna "2006-05-03"
DATE_SUB(d, INTERVAL tipus)	n Retorna la data d decrementada en n unitats indicades pel tipus (més avall hi ha la taula amb els tipus possibles)	DATE_SUB("2006-04-23", INTERVAL 1 YEAR) retorna "2005-04-23"
ADDDATE(d, INTERVAL n tipus) ADDDATE(d, dies)	Utilitzant INTERVAL en el segon argument és igual a la funció DATE_ADD Afegeix dies a la data d	ADDDATE("2006-04-23", 10) retorna "2006-05-03"
SUBDATE(d, INTERVAL n tipus) SUBDATE(d, dies)	Utilitzant INTERVAL en el segon argument és igual a la funció DATE_SUB Resta dies a la data d	SUBDATE("2006-04-23", 10) retorna "2006-04-13"
ADDTIME(expr1, expr2) SUBTIME(expr1, expr2)	Suma/Resta expr2 a expr1 on es suposa que expr1 és un dada de tipus TIME o DATETIME i expr1 és una dada de tipus TIME.	ADDTIME('2005-12-31 23:59:59.999999', '1 1:1:1.000002') retorna '2006-01-02 01:01:01.000001' doncs a les 23h59m59.999999s del dia 31/12/2005 hem sumat 1dia 1hora 1minut i 1.000002 segons i hem obtingut les 01h01m01.000001s del dia 2/1/2006
CURDATE() CURRENT_DATE() CURRENT_DATE	Retorna la data actual	Si CURDATE() retorna "2006-04-23" llavors CURDATE()+0 retorna 20060423
CURTIME() CURRENT_TIME() CURRENT_TIME	Retorna l'hora actual	Si CURTIME() retorna "08:30:45" llavors CURTIME()+0 retorna 83045
CURRENT_TIMESTAMP() CURRENT_TIMESTAMP NOW() LOCALTIME() LOCALTIME LOCALTIMESTAMP() LOCALTIMESTAMP SYSDATE()	Retornen el moment temporal actual (data i hora)	NOW() pot retornar un valor com "2006-04-23 12:30:45" i llavors NOW()+0 retornaria 20060423123045
DATE(dt)	Retorna la data d'una dada DATETIME	DATE("2006-04-23 12:30:45") retorna "2006-04-23"
DATEDIFF(d1, d2)	Retorna el número de dies entre d1 i d2	DATEDIFF("2006-05-10", "2006-04-23") retorna 17
DATE_FORMAT(d, format)	Retorna la data com a cadena amb un determinat format (més avall hi ha la taula amb els formats possibles)	DATE_FORMAT("2006-05-10", "%d-%m-%Y") retorna "10-05-2006"
DAY(d) o DAYOFMONTH(d)	Retorna el dia del mes	DAY("2006-05-10") retorna 10
DAYNAME(d)	Retorna el nom del dia de la setmana	DAYNAME("2006-04-25") retorna Tuesday
DAYOFWEEK(d)	Retorna el dia numèric dins la setmana, tenint en compte que 1=Diumenge, 2=Dilluns,..., 7=Dissabte	DAYOFWEEK("2006-04-25") retorna 3
DAYOFYEAR(d)	Retorna el dia numèric dins l'any	DAYOFYEAR("2006-04-25") retorna 115
EXTRACT(tipus FROM d)	Extreu el tros de d segons indica tipus (més avall hi ha la taula amb els tipus possibles)	EXTRACT(YEAR_MONTH, "2006-04-25") retorna "200604"

Funció	Descripció	Exemples
hour(moment_temporal)	Retorna l'hora	Si ara són les 7:25:40, hour(current_timestamp) retorna 7
last_day(d)	Retorna la data del darrer dia del mes al que pertany d	last_day("2008-02-15") retorna "2008-02-29"
make_date(any, dia_dins_any)	Retorna la data que correspon al dia indicat dins l'any	make_date(2006,115) retorna "2006-04-25"
make_time(hora,minut,segon)	Retorna el moment temporal format per l'hora, minut i segon indicats	make_time(12,25,30) retorna "12:25:30"
microsecond(moment_temporal)	Retorna el microsegon del moment temporal indicat	microsecond("12:00:00.123456") retorna 123456
minute(moment_temporal)	Retorna el minut del moment temporal	Si ara són les 7:25:40, minute(current_timestamp) retorna 25
month(d)	Retorna el número del mes	month("2006-04-25") retorna 4
monthname(d)	Retorna el nom del mes	monthname("2006-04-25") retorna April
quarter(d)	Retorna el trimestre	quarter("2006-04-25") retorna 2
second(moment_temporal)	Retorna el segon del moment temporal	Si ara són les 7:25:40, second(current_timestamp) retorna 40
sec_to_time(segons)	Retorna l'argument segons en moment temporal	sec_to_time(50000) retorna "13:53:20"
str_to_date(cad,format)	Funció inversa de date_format, i construeix una data a partir d'una cadena segons el format indicat (més avall hi ha la taula amb els formats possibles)	str_to_date("25/04/2006", "%d/%m/%Y") retorna la data "2006-04-25"
time(dt)	Retorna el moment temporal d'una dada de tipus data_temps	time("2006-04-25 07:12:45") retorna "07:12:45"
timediff(dt1,dt2)	Retorna el temps entre els dos moments dt1 i dt2	timediff("2006-04-25 01:45:23", "2006-01-03 23:12:45") retorna 2666:32:38
timestampdiff(tipus,dt1,dt2)	Retorna la diferència en les unitats expressades per tipus entre els dos moments temporals dt1 i dt2 (més avall hi ha la taula amb els tipus possibles)	timestampdiff(month, "2006-04-25 01:45:23", "2006-01-03 23:12:45") retorna -3
time_format(temps,format)	Similar a la funció date_format, però temps només pot correspondre a una expressió de temps (més avall hi ha la taula amb els formats possibles)	time_format("23:40:21", "%r") retorna "11:40:21 PM"
time_to_sec(temps)	Inversa de la funció sec_to_time, retorna el temps en segons	time_to_sec("13:53:20") retorna 50000
weekday(d)	Retorna el dia numèric de la setmana (0:Dilluns, 1:Dimarts,...,6:Diumenge)	weekday("2006-04-25") retorna 1
weekofyear(d)	Retorna el número de la setmana dins l'any, tenint en compte que la primera setmana és aquella que té més de 3 dies dins l'any	weekofyear("2005-04-26") retorna 17. El dia 1 de l'any 2005 va ser dissabte i la primera setmana del 2005 va començar el dia 3 de gener. weekofyear("2004-04-26") retorna 18. El dia 1 de l'any 2004 va ser dijous i la primera setmana del 2004 va començar el dia 29 de desembre del 2003.
year(d)	Retorna l'any de la data	year("2006-04-25") retorna 2006

Aquestes no són les úniques funcions per al tractament de moments temporals que facilita MySQL.

Referent a l'argument `TIPUS` que apareix en `DATE_ADD`, `DATE_SUB`, `ADDDATE`, `SUBDATE` i `EXTRACT` cal tenir en compte que es pot utilitzar qualsevol dels tipus indicats a la següent taula:

Valors possibles per a TIPUS	Significat	Format a utilitzar
MICROSECOND	Microsegons	'Microsegons'
SECOND	Segons	'Segons'
MINUTE	Minuts	'Minuts'
HOURL	Hores	'Hores'
DAY	Dies	'Dies'
WEEK	Setmanes	'Setmanes'
MONTH	Mesos	'Mesos'
QUARTER	Trimestres	'Trimestres'
YEAR	Anys	'Anys'
SECOND_MICROSECOND	Segons amb microsegons	'Segons.Microsegons'
MINUTE_MICROSECOND	Minuts amb microsegons	'Minuts.Microsegons'
MINUTE_SECOND	Minuts amb segons	'Minuts:Segons'
HOURL_MICROSECOND	Hores amb microsegons	'Hores.Microsegons'
HOURL_SECOND	Hores-minuts-segons	'Hores:Minuts:Segons'
HOURL_MINUTE	Hores-minuts	'Hores:Minuts'
DAY_MICROSECOND	Dies amb microsegons	'Dies.Microsegons'
DAY_SECOND	Dies-hores-minuts-segons	'Dies Hores:Minuts:Segons'
DAY_MINUTE	Dies-hores-minuts	'Dies Hores:Minuts'
DAY_HOURL	Dies-hores	'Dies Hores'
YEAR_MONTH	Anys-mesos	'AnysMesos'

En l'inici d'aquest subapartat sobre les funcions per la gestió de tipus de dades corresponents a moments temporals hem comentat que MySQL no facilita els operadors `+` i `-` per sumar i restar dies a dates. Bé, ara que ja hem vist els valors possibles per a l'especificador `TIPUS`, podem explicar que sí són possibles les següents operacions:

```
"2006-04-25" + INTERVAL 35 DAY calcula "2006-05-30"
```

```
"2006-04-25 15:30:45" + INTERVAL "10 5:10:25" DAY_SECOND  
calcula "2006-05-05 20:41:10"
```

```
"2006-05-05 20:41:10" - INTERVAL "10 5:10:25" DAY_SECOND  
calcula "2006-04-25 15:30:45"
```

Referent a l'argument `FORMAT` que apareix en `DATE_FORMAT` cal tenir en compte que es pot utilitzar qualsevol dels especificadors indicats a la següent taula:

Especificador	Descripció
%a	Nom abreviat del dia de la setmana (Sun..Sat)
%b	Nom abreviat del mes (Jan..Dec)
%c	Mes en numèric utilitzant els dígit necessaris (0..12)
%D	Dia numèric del mes amb sufix anglès (0th, 1st, 2nd, 3rd, ...)
%d	Dia numèric del mes utilitzant dos dígit (00..31)
%e	Dia numèric del mes utilitzant els dígit necessaris (0..31)
%f	Microsegons (000000..999999)
%H o %I	Hora en format 00-23 (00..23)
%h	Hora en format 01-12 (01..12)
%i	Minuts (00..59)
%j	Dia de l'any (001..366)
%k	Hora en format 0-23 (0..23)
%l	Hora en format 1-12 (1..12)
%M	Nom del mes (January..December)
%m	Mes en numèric utilitzant dos dígit (00..12)
%p	AM (matí) o PM (tarda)
%r	Temps en format 12 hores (hh:mm:ss seguit per AM o PM)
%S o %s	Segons (00..59)
%T	Temps en format 24 hores (hh:mm:ss)
%U	Setmana dins l'any (00..53), considerant que el Diumenge és el primer dia de la setmana.
%u	Setmana dins l'any (00..53), considerant que el Dilluns é el primer dia de la setmana.
%V	Setmana dins l'any (00..53), considerant que el Diumenge és el primer dia de la setmana, utilitzat amb %X
%v	Setmana dins l'any (00..53), considerant que el Dilluns é el primer dia de la setmana, utilitzat amb %x
%W	Nom del dia de la setmana (Sunday..Saturday)
%w	Número de dia dins la setmana (0=Sunday..6=Saturday)
%X	Any per la setmana, on Diumenge és el primer dia de la setmana, utilitzat amb %V
%x	Any per la setmana, on Dilluns és el primer dia de la setmana, utilitzat amb %v
%Y	Any en 4 dígit
%y	Any en 2 dígit

Especificador	Descripció
%%	Un caràcter '%'
%x	Un caràcter x per qualsevol 'x' no llistat més amunt.

Anem a veure alguns exemples de consultes en el que és necessària la utilització d'algunes de les funcions esmentades.

Exemple: Mostrar l'antiguitat en dies que té cada producte.

Per a poder efectuar aquesta consulta utilitzem la variable del sistema `current_date` ja presentada anteriorment i ens cal calcular els dies que hi ha entre aquesta data i la columna `data_edició` de la taula `productes`.

```
select codi_producte, titol, data_edicio, current_date,
       datediff(current_date, data_edicio)
from productes;
```

En l'anterior sentència hem indicat que es mostri les columnes `data_edició` i `current_date` (variable del sistema), per a poder comprovar, si volem, que els càlculs en la darrera columna són correctes, però podem no visualitzar-les (és a dir, no és necessari mostrar-les per a poder efectuar el càlcul). Per tant, haguéssim pogut escriure:

```
select codi_producte "Codi", titol "Titol",
       datediff(current_date, data_edicio) "Dies Antiguitat"
from productes;
```

Ara bé, molt de compte en utilitzar en MySQL una sentència com:

```
select codi_producte, titol, current_date - data_edicio
from productes;
```

doncs el càlcul que efectua `current_date - data_edició` no és altra cosa que la resta entre els valors numèrics `yyyymmdd` de `current_date` i `yyyymmdd` de `data_edició`. És a dir, si avui és 15/05/2006 i tenim un producte que té per `data_edició` el 8/9/2003, el càlcul `current_date - data_edició` donaria 29606, resultat de 20060515 - 20060908.

En alguns SGBD és possible restar dates per obtenir els dies que hi ha entre elles (per exemple Oracle), però això no és possible en MySQL. 

3.5. Clàusula ORDER BY

La clàusula ORDER BY permet especificar el criteri de classificació del resultat de la consulta i s'afegeix darrera de la clàusula WHERE si n'hi ha:

```
SELECT <expressió/columna>, <expressió/columna>, ...
FROM <taula>, <taula>, ...
WHERE <condició_de_recerca>
ORDER BY <expressió/columna> [ASC/DESC],
        <expr/col>, [ASC/DESC], ...;
```

Com es pot observar, la clàusula ORDER BY permet ordenar la sortida segons diferents expressions i/o columnes, que han de ser calculables a partir dels valors de les columnes de les taules de la clàusula FROM encara que no apareguin en les columnes de la clàusula SELECT.

Les expressions i/o columnes de la clàusula ORDER BY que apareixen a la clàusula SELECT es poden referenciar pel número ordinal de la posició que ocupen a la clàusula SELECT enlloc d'escriure el seu nom.

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

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

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

Exemple: Mostrar els codis, títols i preus dels productes ordenats de forma ascendent pel seu preu i, ordenats pel títol quan tinguin el mateix preu.

```
select codi_producte, titol, preu
from productes
order by preu, titol;
```

O també referenciant el número de columna enlloc del seu nom:

```
select codi_producte, titol, preu
from productes
order by 3, 2;
```

O també, si algunes de les columnes de la clàusula `SELECT` tenen àlies definits, aquests es poden utilitzar en els criteris d'ordenació:

```
select codi_producte "Codi", titol "Titol", preu "PVP"
from productes
order by PVP, Titol;
```

Exemple: Mostrar el codi i el títol dels productes ordenats de forma descendent per la longitud del seu títol.

```
select codi_producte, titol
from productes
order by length(titol) DESC;
```

3.6. Opció `DISTINCT`

La clàusula `SELECT` permet decidir quines columnes seleccionar del producte cartesià de les taules especificades en la clàusula `FROM` i la clàusula `WHERE` en filtra les files corresponents. El resultat, però, pot tenir files repetides, per les quals ens interressi tenir només un exemplar.

L'opció `DISTINCT`, acompanyant a la clàusula `SELECT` permet especificar que es vol un únic exemplar per les files repetides. La sintaxis és:

```
SELECT DISTINCT <expressió/columna>, <expressió/columna>, ...
FROM <taula>, <taula>, ...
WHERE <condició_de_recerca>
ORDER BY <expressió/columna> [ASC/DESC],
        <expressió/columna>, [ASC/DESC], ...;
```

La utilització de l'opció `DISTINCT` implica que el SGBDR executi obligatòriament un `ORDER BY` sobre totes les columnes seleccionades (encara que no s'especifiqui la clàusula `ORDER BY`), fet que implica un cost d'execució addicional. Per tant, l'opció `DISTINCT` caldria utilitzar-la en cas que hi pugui haver files repetides i interressi un únic exemplar i s'hauria d'evitar quan es tingués la seguretat que no hi pot haver files repetides.

Exemple: Mostrar els codis dels autors pels que tenim algun producte.

Per obtenir aquesta informació cal quedar-nos, de la taula `productes`, els diferents autors que hi apareixen. No es pot efectuar sobre la taula `autors`, ja que hi podria haver algun autor introduït a la base de dades sense cap producte. Si fem:

```
select autor from productes;
```

obtindrem els codis de tots els autors, tants com productes hi hagi a la taula. I, evidentment, si un autor ho és de n productes, apareixerà n vegades. Per a evitar que un mateix autor aparegui varies vegades, utilitzarem l'opció `DISTINCT`:

```
select distinct autor from productes;
```

4. Llenguatge SQL: Consultes complexes.

El llenguatge SQL es veu àmpliament potenciat amb la possibilitat d'efectuar consultes complexes que continguin agrupaments de files, combinacions entre diferents taules i subconsultes niades, possibilitats que veiem a continuació.

4.1. Agrupaments de files

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

La clàusula `GROUP BY` permet agrupar les files resultat de les clàusules `SELECT`, `FROM` i `WHERE` segons una o vàries de les columnes seleccionades.

La clàusula `HAVING` permet especificar condicions de filtrat sobre els grups assolits per la clàusula `GROUP BY`.

La sintaxi és:

```
SELECT [distinct] <expressió/columna>, <expressió/columna>, ...
FROM <taula>, <taula>, ...
WHERE <condició_de_recerca>
GROUP BY <àlias/columna>, <àlias/columna>, ...
HAVING <condició_sobre_grups>
ORDER BY <expressió/columna> [ASC/DESC], <expressió/columna> [ASC/DESC], ...;
```

En les sentències `SELECT` d'agrupament de files es pot utilitzar les funcions d'agrupament següents:

Funció	Descripció	Exemples
<code>AVG (n)</code>	Retorna el valor mig de la columna n ignorant els valors nuls.	<code>AVG (preu)</code> (sobre la taula productes) retorna el salari mig de tots els productes seleccionats que tenen preu (els nuls s'ignoren). Retorna NUL si no hi ha cap registre seleccionat.
<code>COUNT ([* expr])</code>	Retorna el número de vegades que expr avalua alguna dada amb valor no nul. L'opció * comptabilitza totes les files seleccionades.	<code>COUNT (codi_producte)</code> (sobre la taula productes) compta quants productes hi ha a la taula.
<code>MAX (expr)</code>	Retorna el valor màxim d'expr.	<code>MAX (preu)</code> retorna el preu més alt
<code>MIN (expr)</code>	Retorna el valor mínim d'expr	<code>MIN (preu)</code> retorna el preu més baix.

Funció	Descripció	Exemples
STDDEV (expr)	Retorna la desviació típica d'expr sense tenir en compte els valors nuls.	STDDEV (preu) retorna la desviació típica dels preus.
SUM (expr)	Retorna la suma dels valors d'expr sense tenir en compte els valors nuls	SUM (preu) retorna la suma de tots els preus
VARIANCE (expr)	Retorna la variància d'expr sense tenir en compte els valors nuls.	VARIANCE (preus) retorna la variància dels preus.

Les funcions d'agrupament AVG, MAX, MIN, SUM i COUNT es poden utilitzar amb les opcions DISTINCT en el seu interior per a comptabilitzar només els valors diferents.

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

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

Exemple: Comptar quants productes hi ha a la base de dades.

```
select count(*) "Quants productes" from productes;
```

Exemple: Mostrar quants autors hi ha que tinguin productes a la base de dades.

Aquesta informació la cercarem a la taula productes i no pas a la taula autors, ja que podem tenir autors a la base de dades sense cap producte introduït.

```
select count(distinct autor) "Quants autors amb productes" from productes;
```

Exemple: Mostrar quants productes hi ha de cada tipus de suport.

Igualment cercarem aquesta informació a la taula productes.

```
select suport "Suport", count(*) "Quants productes"
from productes
group by suport;
```

STDDEV - VARIÀNCIA

Aquests són dos conceptes utilitzats en estadística. Ens cal saber que SQL facilita la possibilitat de calcular-los, però no entrarem en el seu estudi.

Exemple: Mostrar els suports pels que hi ha productes a la base de dades, amb el preu de producte més alt.

La informació ens cal cercar-la a la taula productes:

```
select suport "Suport", max(preu) "Preu Màxim"
from productes
group by suport;
```

Exemple: Mostrar els suports pels que hi ha productes a la base de dades, amb el preu de producte més alt, i ordenada pel màxim dels preus.

```
select suport "Suport", max(preu) "Preu Màxim"
from productes
group by suport
order by "Preu Màxim";
```

Exemple: Mostrar quants productes hi ha de cada tipus de suport per a cada autor que tingui productes a la base de dades.

```
select autor "Autor", suport "Suport", count(*)
from productes
group by 1,2;
```

Exemple: Mostrar quants productes hi ha de cada tipus de suport per l'autor de codi 303.

```
select autor "Autor", suport "Suport", count(*)
from productes
where autor=303
group by 1,2;
```

Exemple: Mostrar el número de productes que hi ha de cada tipus de suport pels suports que tenen més de 10 productes.

```
select suport "Suport", count(*) "Quants productes"
from productes
group by suport
having count(*) > 10;
```

4.2. Combinacions entre taules

En l'apartat en què hem presentat la clàusula FROM hem comentat que el llenguatge SQL efectua el producte cartesià de totes les taules que hi apareixen. La majoria de les ocasions no ens interessarà el producte

cartesià, sinó únicament un subconjunt del producte cartesià. Anem a veure com podem aconseguir-ho.

1) Efectuant filtres sobre les files del producte cartesià

Es tracta d'utilitzar la clàusula `WHERE` per a filtrar files del producte cartesià.

Exemple: Mostrar els amics, ordenats alfabèticament, junt amb la descripció de la seva població.

Tenim clar que els amics els hem d'anar a cercar a la taula `amics`, però cada amic està acompanyat del codi de la població i nosaltres volem saber el nom de la població, informació que resideix en la taula `poblacions`. Per aquest motiu haurem de posar les dues taules a la clàusula `FROM`, fet que provocarà que MySQL en calculi el producte cartesià (és a dir, totes les files de la primera taula per totes les files de la segona taula). Però aquest càlcul barreja amics d'una població amb totes les poblacions i només ens interessen les combinacions d'amics amb poblacions que tinguin coincidència pel codi de població. Per tant, utilitzarem la clàusula `WHERE` per aconseguir aquest filtrat.

```
select codi_amic "Codi", nom_cognoms "Amic", poblacio "Poblacio"
from amics , poblacions
where amics.codi_poblacio = poblacions.codi_pob
order by nom_cognoms;
```

Si volguéssim visualitzar també el codi de la població, hauríem d'escollir el codi de la taula `amics` o el codi de la taula `poblacions`.

```
select codi_amic "Codi Amic", nom_cognoms "Amic",
       codi_poblacio "Codi Població", poblacio "Poblacio"
from amics , poblacions
where amics.codi_poblacio = poblacions.codi_pob
order by nom_cognoms;
```

Recordem que podem utilitzar àlies per les taules:

```
select codi_amic "Codi Amic", nom_cognoms "Amic",
       codi_poblacio "Codi Població", poblacio "Poblacio"
from amics a, poblacions p
where a.codi_poblacio = p.codi_pob
order by nom_cognoms;
```

Exemple: Mostrar els productes (codi i títol) acompanyats de la descripció dels suports, formats i gèneres.

```
select codi_producte "Codi", titol "Títol", s.descripcio "Suport",  
       f.descripcio "Format", g.descripcio "Gènere"  
from productes p, suports s, formats f, generes g  
where p.suport = s.codi_suport  
and p.format = f.codi_format  
and p.genere = g.codi_genere;
```

2) Combinacions JOIN

Els dos exemples anteriors ens serviran per introduir aquest concepte.

En el penúltim exemple es veu de forma força clara que la informació que ens interessa resideix a la taula `amics` i que la combinació amb la taula `poblacions` únicament la necessitem per a tenir accés al nom de la població.

Per tant, seria interessant poder indicar que la informació principal que ens interessa resideix a la taula `amics` i que a partir d'alguna(es) columna(es) d'aquesta taula ens cal accedir a altra(es) taula(es) per a complementar la informació.

El llenguatge SQL facilita, per aconseguir el nostre propòsit, les operacions `JOIN` en la clàusula `FROM` indicant la condició de combinació, la qual no s'haurà d'indicar ja dins la clàusula `WHERE`.

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

```
...  
FROM taula1, taula2  
WHERE <condició combinació entre taula1 i taula2>  
...
```

a

```
...  
FROM taula1 [ INNER | LEFT | RIGHT ] JOIN taula2  
ON <condició combinació entre taula1 i taula2>  
WHERE...
```

Així, les dues darreres sentències `SELECT` es poden reescriure com:

```
select codi_amic "Codi", nom_cognoms "Amic", poblacio "Poblacio"  
from amics join poblacions on amics.codi_poblacio = poblacions.codi_pob  
order by nom_cognoms;
```

```
select codi_producte "Codi", titol "Títol", s.descripcio "Suport",  
       f.descripcio "Format", g.descripcio "Gènere"  
from productes p join suports s on p.suport = s.codi_suport
```

```
join formats f on p.format = f.codi_format  
join generes g on p.genere = g.codi_genere;
```

En ambdós casos ha desaparegut la clàusula `WHERE`, però això ha passat en aquestos casos doncs la clàusula `WHERE` només s'utilitzava per les condicions de filtrat sobre el producte cartesià.

Exemple: Mostrar els productes (codi i títol) de preu superior a 25 €, acompanyats de la descripció dels suports, formats i gèneres.

```
select codi_producte "Codi", titol "Títol", s.descripcio "Suport",  
       f.descripcio "Format", g.descripcio "Gènere"  
from productes p join suports s on p.support = s.codi_suport  
join formats f on p.format = f.codi_format  
join generes g on p.genere = g.codi_genere  
where preu > 25;
```

En la sintaxi presentada per l'operació `JOIN`, aquesta paraula ve precedida de les paraules `INNER`, `LEFT`, `RIGHT` que corresponen a tres tipus diferents de `JOIN`.

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

En ocasions, ens interessarà que apareguin totes les files d'alguna de les dues taules malgrat que no hi hagi cap fila de l'altra taula amb valors coincidents pel(s) camp(s) de combinació. En aquest cas es parla de combinació externa i cal utilitzar les operacions `LEFT JOIN` si volem combinar totes les files de la taula de l'esquerra del `JOIN` amb les files amb valors coincidents de la taula de la dreta o `RIGHT JOIN` si volem combinar totes les files de la taula de la dreta amb les files amb valors coincidents de la taula de l'esquerra.

Exemple: Mostrar tots els autors (codi i nom) acompanyats dels seus productes.

```
select codi_autor "Codi Autor", nom "Autor", codi_producte "Producte", titol "Títol"  
from autors a left join productes p on p.autor = a.codi_autor;
```

Amb aquesta sentència ens assegurem que apareguin tots els autors malgrat que algun d'ells no tingui encara cap producte introduït a la base de dades. En tal situació, les columnes `codi_producte` i `titol` de la sentència `SELECT` queden plenes amb valor `NUL`.

Per últim comentar que és possible construir sentències `SELECT` amb `JOINS` niats, seguint la sintaxi:

```
...  
FROM taula1 <tipus> JOIN  
    ( taula2 <tipus> JOIN taula3 on <combinació taula2 amb taula3> )  
ON <condició combinació entre taula1 i taula2>  
WHERE...
```

I encara es podria niar més nivells!

Les operacions `LEFT JOIN` o `RIGHT JOIN` poden estar niades dins una operació `INNER JOIN`, però una operació `INNER JOIN` no pot estar niada dins les operacions `LEFT JOIN` o `RIGHT JOIN`.

4.3. Subconsultes

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

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

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

Exemple: Mostrar els productes que tenen preu igual o superior al preu mig dels productes actuals.

```
select codi_producte "Codi", titol "Titol"  
from productes  
where preu > (select avg(preu) from productes);
```

Observacions:

- En certes situacions pot ser necessari accedir des de la subconsulta als valors de les columnes seleccionades en la consulta. El llenguatge SQL ho permet sense problemes i, en cas que els noms de les columnes coincideixen, es poden utilitzar àlies. 
- Els noms de columnes que apareixen en les clàusules d'una subconsulta s'intenten avaluar, en primer lloc, com a columnes de les taules definides en la clàusula `FROM` de la subconsulta, a no ser que

vagin acompanyades d'àlies que les identifiquin com a columnes d'una taula en la consulta contenidora. !

Exemple: Mostrar els productes de cada gènere que tenen preu menor que el preu mig dels productes del gènere.

```
select codi_producte "Codi", titol "Titol", genere "Gènere"
from productes p1
where preu < (select avg(preu) from productes p2
              where p1.genere=p2.genere);
```

Observacions:

- Els valors retornats per les subconsultes s'utilitzen en les clàusules WHERE com a part dreta d'operacions de comparacions en les que intervenen els operadors =, !=, <, <=, >, >=, [NOT] IN, <op> ANY i <op> ALL. !
- Les subconsultes també poden vincular-se a la consulta contenidora per la partícula [NOT] EXISTS: !

```
...
where [NOT] EXISTS (subconsulta)
```

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

- Les consultes que poden retornar un únic valor o cap, poden actuar com a subconsultes en expressions on el valor que es retorna es compara amb qualsevol operador de comparació. !
- Les consultes que poden retornar més d'un valor (encara que en execucions concretes només en retornin un) mai poden actuar com a subconsultes en expressions on els valors que es retornen es comparen amb l'operadora =, ja que quan en retornin més d'un, el SGBDR no sabrà amb quin d'elles efectuar la comparació d'igualtat i donarà l'error. !

Així, per exemple, l'execució de la següent consulta

```
select codi_producte "Codi", titol "Titol"
from productes
where codi_producte = (select codi_producte from productes);
```

produeix l'error:

ERROR 1242 (21000): Subquery returns more than 1 row

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

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

Exemple: Mostrar els productes del mateix gènere que el gènere del producte que té per títol "Meteora".

```
select codi_producte "Codi", titol "Títol"  
from productes  
where genere = (select genere from productes  
                where upper(titol)="METEORA")  
and upper(titol)<>"METEORA";
```

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

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

```
select codi_producte "Codi", titol "Títol"  
from productes  
where genere in (select genere from productes  
                where upper(titol)="METEORA")  
and upper(titol)<>"METEORA";
```

Però també coneixem altres maneres d'aconseguir el mateix resultat:

```
select p1.codi_producte "Codi", p1.titol "Títol"  
from productes p1, productes p2  
where p1.genere = p2.genere  
and upper(p1.titol)<>"METEORA"  
and upper(p2.titol)="METEORA";
```

O també:

```
select p1.codi_producte "Codi", p1.titol "Títol"
```

```
from productes p1 inner join productes p2 on p1.genere = p2.genere
                                         and upper(p2.titol)="METEORA"
where upper(p1.titol)<>"METEORA"
```

Exemple: Mostrar els autors d'alguna cançó de "pop".

```
select codi_autor "Codi", nom "Autor" from autors
where codi_autor IN
    (select codi_autor from producte_titol_autor pta, productes pro, generes gen
     where pta.codi_producte = pro.codi_producte
       and pro.genere = gen.codi_genere
       and upper(gen.descripcio)="POP")
order by 1,2;
```

O també, utilitzant =ANY:

```
select codi_autor "Codi", nom "Autor" from autors
where codi_autor =ANY
    (select codi_autor from producte_titol_autor pta, productes pro, generes gen
     where pta.codi_producte = pro.codi_producte
       and pro.genere = gen.codi_genere
       and upper(gen.descripcio)="POP")
order by 1,2;
```

O també, utilitzant EXISTS:

```
select codi_autor "Codi", nom "Autor" from autors a
where EXISTS
    (select * from producte_titol_autor pta, productes pro, generes gen
     where pta.codi_autor = a.codi_autor
       and pta.codi_producte = pro.codi_producte
       and pro.genere = gen.codi_genere
       and upper(gen.descripcio)="POP")
order by 1,2;
```

En aquesta darrera sentència observem que utilitzem la subconsulta per comprovar si hi ha algun registre per l'autor seleccionat en la SELECT més externa i, per tant, les columnes a seleccionar en la subconsulta són indiferents. Però si mantenim '*' estem obligant a que el SGBD internament les seleccioni totes... Oi que no cal? Doncs per millorar l'eficiència podem posar un valor constant qualsevol, com per exemple el número 1:

```
select codi_autor "Codi", nom "Autor" from autors a
where EXISTS
    (select 1 from producte_titol_autor pta, productes pro, generes gen
     where pta.codi_autor = a.codi_autor
       and pta.codi_producte = pro.codi_producte
       and pro.genere = gen.codi_genere
       and upper(gen.descripcio)="POP")
order by 1,2;
```

Tenim, encara més versions possibles:

```
select distinct a.codi_autor "Codi", a.nom "Autor"
from autors a, producte_titol_autor pta, productes pro, generes gen
where a.codi_autor = pta.codi_autor
      and pta.codi_producte = pro.codi_producte
      and pro.genere = gen.codi_genere
      and upper(gen.descripcio)="POP"
order by 1,2;
```

Observem que cal utilitzar l'opció `DISTINCT` per eliminar les files repetides, doncs del contrari cada autor ens sortiria tantes vegades com cançons seves hi hagués en els diferents productes existents a la base de dades.

Però encara tenim més versions si utilitzem les operacions `JOIN`:

```
select distinct codi_autor "Codi", nom "Autor"
from autors a join (producte_titol_autor pta
                  join (productes pro
                      join generes gen on pro.genere = gen.codi_genere
                      and upper(gen.descripcio)="POP"
                  )
                  on pta.codi_producte = pro.codi_producte
              )
on a.codi_autor = pta.codi_autor
order by 1,2;
```

Observem la sentència següent:

```
select codi_autor "Codi", nom "Autor" from autors
where codi_autor =ANY
      (select codi_autor from producte_titol_autor
       where codi_producte =ANY
         (select codi_producte from productes
          where genere =ANY (select genere from generes
                           where upper(descripcio)="POP")))
order by 1,2;
```

Sembla correcta per a obtenir els autors d'alguna cançó de "pop"? Doncs no ho és. Observem que aquesta consulta segueix l'esquema:

```
select ... from A
where <camp> =ANY (select ... from B
                  where <camp> = ANY (select ... from C
                                      ...
                                )
                )
```

En aquesta consulta, per cada fila d'A es mira si està relacionada amb alguna fila de B i per cada fila de B es mira si està relacionada amb

alguna fila de C, però que aquest darrer fet tingui lloc no implica que la fila d'A estigui relacionada amb cap fila de C. Aquí radica l'error!

En canvi, sí és correcta la versió:

```
select codi_autor "Codi", nom "Autor" from autors a
where exists (select * from producte_titol_autor pta
              where pta.codi_autor = a.codi_autor
              and exists (select * from productes pro
                          where pro.codi_producte = pta.codi_producte
                          and exists (select * from generes
                                      where codi_genere = pro.genere
                                      and upper(descripcio)="POP")) );
```

4.4. Operacions amb sentències **SELECT**

L'execució de les sentències **SELECT** dona un conjunt de resultats (files) i ens preguntem si diferents conjunts de files els podem operar entre sí amb les conegudes operacions de conjunts: unió, intersecció i diferència.

Operacions de conjunts

La unió de dos conjunts A i B és el conjunt format per tots els elements d'A i tots els elements de B considerant una única vegada els elements que són simultàniament a A i a B.

La intersecció de dos conjunts A i B és el conjunt format per les files comunes d'A i de B (és a dir, les files que formen part simultàniament d'A i de B).

La diferència del conjunt A menys el conjunt B està formada per les files d'A que no són a B.

Així, si tenim els conjunts $A=\{ 'a', 'b', 'c', 'd', 'e' \}$ i $B=\{ 'a', 'e', 'i', 'o', 'u' \}$, resulta que:

Unió:	$A \cup B = \{ 'a', 'b', 'c', 'd', 'e', 'i', 'o', 'u' \}$
Intersecció:	$A \cap B = \{ 'a', 'e' \}$
Diferència:	$A - B = \{ 'b', 'c', 'd' \}$

MySQL facilita l'operació **unió** amb l'obligatorietat que les sentències **SELECT** a unir han de ser compatibles (igual quantitat de columnes i columnes compatibles -tipus de dades equivalents- dos a dos).

La sintaxis és:

```
sentència_select_sense_order_by
UNION
sentència_select_sense_order_by
[order by ...]
```

Exemple: Mostrar conjuntament els suports, els formats i els gèneres existents a la base de dades en un format similar al de la taula següent, on 'S' indica suport, 'G' indica gènere i 'F' indica format.

Tipus	Codi	Descripció
F	1	single
F	2	lp
G	3	pop
G	4	techno
S	4	dvd
S	10	MP3

```
select "S" as "Tipus", codi_suport "Codi", descripcio "Descripció"
from suports
union
select "F", codi_format, descripcio from formats
union
select "G", codi_genere, descripcio from generes
order by 1,2;
```

Observem que els àlies de les columnes cal definir-los en la primera sentència `SELECT`.

Altres SGBD proporcionen també els operadors per efectuar la intersecció i la diferència de conjunts, però no és el cas de MySQL. No obstant, els resultats es poden obtenir utilitzant les diferents possibilitats de sentències `SELECT` ja presentades.

5. Llenguatge SQL: Manipulació de dades.

El llenguatge SQL incorpora diferents tipus de sentències per a treballar amb les bases de dades: llenguatge per consultar les dades (LC), llenguatge de manipular les dades (LMD), llenguatge per definir les dades (LDD) i llenguatge per al control de les dades (LCD). Anem a conèixer les sentències corresponents a la manipulació de dades (LMD).

Per a la manipulació de les dades (instruccions LMD), el llenguatge SQL proporciona tres sentències: `INSERT` per la introducció de noves files, `UPDATE` per la modificació de files i `DELETE` per l'esborrat de files.

5.1. Sentència `INSERT`

Aquesta és la sentència facilitada pel llenguatge SQL per a inserir noves files a les taules. Admet dues possibles sintaxis:

- 1) Els valors a inserir s'expliciten a la pròpia instrucció en la clàusula `VALUES`:

```
INSERT INTO <nom_taula> [(col1, col2, ...)]  
VALUES (val1, val2, ...);
```

- 2) Els valors a inserir s'aconsegueixen per via d'una sentència `SELECT`:

```
INSERT INTO <nom_taula> [(col1, col2, ...)]  
SELECT ...;
```

En qualsevol cas es pot especificar les columnes de la taula a omplir i l'ordre en què es subministren els diferents valors. En cas que no s'especifiquin les columnes, el SGBDR entén que els valors es subministren per totes les columnes de la taula i, a més, en l'ordre en què estan definits a la taula.

La llista de valors de la clàusula `VALUES` i la llista de resultats de la sentència `SELECT` han de coincidir en número, tipus i ordre amb la llista de columnes a omplir.

Exemple: Inserir l'autor Barry White.

Visualitzem el descriptor de la taula `autors`:

```
mysql> desc autors;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Codi_Autor | int(11)        | NO   | PRI | 0        |       |
| Nom        | varchar(40)    | NO   |     |          |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.03 sec)
```

on el codi d'autor no és autoincremental (de ser-ho apareixeria a la columna *extra*).

Per tant hem de decidir el codi d'autor que volem assignar a Barry White. Una possibilitat seria anar provant amb diferents números fins que n'encertem un que encara no sigui donat d'alta... Això no sembla molt professional, no? Una segona possibilitat és cercar el major número actual per assignar el següent. Com cercar-lo?

Una manera no gens eficient seria:

```
select codi_autor from autors order by 1;
```

doncs això ens mostrarà tots els codi d'autors quan només ens interessa el major. Per tant, una manera molt més eficient és:

```
select max(codi_autor) from autors;
```

Però és clar, en aquest cas ens mostra el codi 999 que sembla ser un codi de prova (ho veiem si fem una ullada a la taula *autors*) . Doncs bé,

```
select max(codi_autor) from autors where codi_autor<999;
```

ens dona el major codi d'autor existent a la taula per sota del 999. Suposem que és el codi 663 i, per tant, volem assignar el següent número a Barry White. Farem:

```
insert into autors (codi_autor, nom)
values (664, "Barry White");
```

En aquest cas hem indicat el nom de les columnes, però sabent que a la taula estan en l'ordre *codi_autor* i *nom*, podríem fer:

```
insert into autors
values (664, "Barry White");
```

Però també podem fer:

```
insert into autors (nom, codi_autor)
values ("Barry White", 664);
```

Exemple: Suposem que teniu una amiga de nom Maria Teresa Bernal Catal que voleu inserir a la vostra base de dades. Sabeu que viu a Igualada i teniu el seu telèfon fix: 93 805 00 00.

Per poder inserir aquesta informació necessitem saber les columnes de la taula `amics`.

```
mysql> desc amics;
```

Field	Type	Null	Key	Default	Extra
Codi_Amic	int(11)	NO	PRI	0	
Nom_Cognoms	varchar(40)	NO			
Adreca	varchar(40)	YES			
CP	varchar(5)	YES			
Codi_Poblacio	int(11)	NO	MUL	0	
Mobil	varchar(15)	YES			
eMail	varchar(40)	YES			
Fix	varchar(15)	YES			

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

Veiem que els camps adreça, codi postal, mòbil i e-mail permeten valor NULL. Per tant, amb la informació que tenim ja n'hi ha prou... o no?

Si reflexionem una mica ens adonarem que no sabem el codi de la població Igualada... Cerquem-lo !

```
select * from poblacions
where upper(poblacio)='IGUALADA';
```

I resulta que no tenim la població Igualada inserida a la base de dades. Observem el descriptor de la taula `poblacions`:

```
mysql> desc poblacions;
```

Field	Type	Null	Key	Default	Extra
Codi_Pob	int(11)	NO	PRI		auto_increment
Poblacio	varchar(40)	NO			
Provincia	varchar(30)	YES			

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

En aquest cas observem que el codi de població és autoincremental. així doncs, per inserir la nova població podem fer:

```
insert into poblacions (poblacio, provincia)
values ('Igualada', 'Barcelona');
```

Atenció en introduir el nom de la província de la mateixa manera que estan introduïdes les altres poblacions de la mateixa província, és a dir, no se'ns acudeixi posar 'BCN' o 'Barna', doncs si ho féssim tindríem greus problemes per aconseguir cercar tots els amics i amigues d'una determinada província... Això no passaria si tinguéssim les províncies codificades en una taula, no?

Per veure el codi que li ha assignat podem executar altra vegada:

```
select * from poblacions
where upper(poblacio)='IGUALADA';
```

Suposem que ha estat el codi 17. Decidim també el codi d'amic que li volem assignar (suposem el 4) i ja podem inserir la nostra amiga:

```
insert into amics (codi_amic, nom_cognoms, codi_poblacio, fix)
values (4, 'Maria Teresa Bernal Catal', 17, '938050000');
```

Si executem una sentència per a comprovar el contingut actual de la taula `amics`, trobarem la nova fila amb molts camps en blanc (valors `NULL`).

Observacions:

- Podem obtenir el valor a inserir en una columna a partir del resultat d'una `SELECT` sempre i quan el resultat d'aquesta sigui una i només una fila. La sentència `SELECT` ha d'anar tancada entre parèntesis.



- L'observació anterior té, però, una excepció: la taula sobre la que s'efectua la consulta `SELECT` per obtenir el valor no pot ser la mateixa taula on s'insereix la fila.

Exemple: La inserció de la informació de l'amiga Maria Teresa a la taula `amics` hagués pogut incloure la sentència per a calcular el codi de la població Igualada.

```
insert into amics (codi_amic, nom_cognoms, codi_poblacio, fix)
values (4, 'Maria Teresa Bernal Catal',
        (select codi_pob from poblacions where upper(poblacio)='IGUALADA'),
        '938050000');
```

Exemple: La inserció de Barry White a la taula `autors` no pot calcular el codi a utilitzar en la pròpia instrucció, doncs és una informació a calcular sobre la pròpia taula on s'està inserint la fila. És a dir, la sentència:

```
insert into autors
values ((select max(codi_autor)+1 from autors where codi_autor<999), 'Barry White');
```

produeix l'error:

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

que ve a dir que no es pot utilitzar la taula `autors` que volem modificar en la clàusula `FROM`.

Exemple: Ha arribat a les nostres mans el producte "Greatest Hits Barry White" que consta de 2 volums i volem inserir-lo a la base de dades. La informació rellevant per nosaltres és:

Volum 1	Volum 2
Loves theme	Fragile - handle with care
You're the first, the last, my everything	Your heart & soul
What amb I gonna do with you	Big man
Let the music play	My Buddy
Can't get enough of your love babe	I owe it all to you
Wout of the shadows	Lady sweet lady
L've got the world to hold me up	I like you you like me
Under the influence of love	Change
Where can I turn to	I found love
Long black veil	Long black veil
All in the run of a day	Sho' you right
Come on in love	Come on

En Barry White ja el tenim inserit a la taula d'autors (codi 664). Per inserir la nova informació hem de seguir un ordre adequat Així, podem ja donar d'alta el producte, però no ho podríem fer si en Barry White no estés a la base de dades. En primer lloc ens informem dels camps de la taula `productes`:

```
mysql> desc productes;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Codi_Producte  | varchar(13)   | NO   | PRI |          |       |
| Titol          | varchar(30)   | NO   |     |          |       |
| Autor          | int(11)       | NO   | MUL | 0        |       |
| Suport         | int(11)       | NO   | MUL | 0        |       |
```

Format	int(11)	NO	MUL	0		
Genere	int(11)	NO	MUL	0		
Data_Edicio	date	YES				
Preu	float	YES				
Qui_el_te	int(11)	YES	MUL			
Observacions	mediumtext	YES				
web	mediumtext	YES				
caratula	mediumblob	YES				

```
12 rows in set (0.02 sec)
```

Veiem que ens mancarà saber els codis de suport, format i gènere. Executem tres SELECT sobre les respectives taules i obtenim que el suport de codi 3 correspon al CD, el format de codi 2 correspon al LP i que el gènere de codi 21 correspon a les balades. Amb tota aquesta informació i considerant cada CD com a diferent producte amb el seu codi de barres com a codi de producte, inserim:

```
insert into productes (codi_producte, titol, autor, suport, format, genere)
values ('8435108611742', 'Greatest Hits Barry White Vol .1', 664, 3, 2, 21);
insert into productes (codi_producte, titol, autor, suport, format, genere)
values ('8435108611759', 'Greatest Hits Barry White Vol .2', 664, 3, 2, 21);
```

Ara ja podem anar inserint els diversos títols dels temes a la taula títols. Si visualitzem la descripció d'aquesta taula veiem que:

```
mysql> desc titols;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Codi_Titol | int(11)       | NO   | PRI | 0       |       |
| Descripcio | varchar(30)   | NO   |     |         |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

Observem que el codi de títol no és autoincremental. Però si ho fos també ens posaria en un problema, doncs per cada títol inserit a la taula títols hem de saber-ne el seu codi per posteriorment inserir la informació a la taula producte_títol_autor (on es recull a cada producte els temes que el formen i l'autor de cada tema -que pot ser diferent de l'autor del producte-).

Així que hem d'executar tot un seguit d'instruccions per anar omplint les dues taules títols i producte_títol_autor:

```
insert into titols (codi_titol,descripcio) values (13217,"Loves theme");
insert into producte_titol_autor values ('8435108611742',13217,664);
insert into titols (codi_titol,descripcio)
values (13218,"You're the first, the last, my everrything");
insert into producte_titol_autor values ('8435108611742',13218,664);
insert into titols (codi_titol,descripcio)
```

```
values (13219,"What amb I gonna do with you");
insert into producte_titol_autor values ('8435108611742',13219,664);
insert into titols (codi_titol,descripcio) values (13220,"Let the music play");
insert into producte_titol_autor values ('8435108611742',13220,664);
insert into titols (codi_titol,descripcio)
values (13221,"Can't get enough of your love babe");
insert into producte_titol_autor values ('8435108611742',13221,664);
insert into titols (codi_titol,descripcio) values (13222,"Wout of the shadows");
insert into producte_titol_autor values ('8435108611742',13222,664);
insert into titols (codi_titol,descripcio)
values (13223,"L've got the world to hold me up");
insert into producte_titol_autor values ('8435108611742',13223,664);
insert into titols (codi_titol,descripcio)
values (13224,"Under the influence of love");
insert into producte_titol_autor values ('8435108611742',13224,664);
insert into titols (codi_titol,descripcio) values (13225,"Where can I turn to");
insert into producte_titol_autor values ('8435108611742',13225,664);
insert into titols (codi_titol,descripcio) values (13226,"Long black veil");
insert into producte_titol_autor values ('8435108611742',13226,664);
insert into producte_titol_autor values ('8435108611759',13226,664);
insert into titols (codi_titol,descripcio) values (13227,"All in the run of a day");
insert into producte_titol_autor values ('8435108611742',13227,664);
insert into titols (codi_titol,descripcio) values (13228,"Come on in love");
insert into producte_titol_autor values ('8435108611742',13228,664);
insert into titols (codi_titol,descripcio) values (13229,"Fragile - hndle with care");
insert into producte_titol_autor values ('8435108611759',13229,664);
insert into titols (codi_titol,descripcio) values (13230,"Your heart & soul");
insert into producte_titol_autor values ('8435108611759',13230,664);
insert into titols (codi_titol,descripcio) values (13231,"Big man");
insert into producte_titol_autor values ('8435108611759',13231,664);
insert into titols (codi_titol,descripcio) values (13232,"My Buddy");
insert into producte_titol_autor values ('8435108611759',13232,664);
insert into titols (codi_titol,descripcio) values (13233,"I owe it all to you");
insert into producte_titol_autor values ('8435108611759',13233,664);
insert into titols (codi_titol,descripcio) values (13234,"Lady sweet lady");
insert into producte_titol_autor values ('8435108611759',13234,664);
insert into titols (codi_titol,descripcio) values (13235,"I like you you like me");
insert into producte_titol_autor values ('8435108611759',13235,664);
insert into titols (codi_titol,descripcio) values (13236,"Change");
insert into producte_titol_autor values ('8435108611759',13236,664);
insert into titols (codi_titol,descripcio) values (13237,"I found love");
insert into producte_titol_autor values ('8435108611759',13237,664);
insert into titols (codi_titol,descripcio) values (13238,"Sho' you right");
insert into producte_titol_autor values ('8435108611759',13238,664);
insert into titols (codi_titol,descripcio) values (13239,"Come on");
insert into producte_titol_autor values ('8435108611759',13239,664);
```

Observem que incomprensiblement hi ha una cançó (Long black veil) repetida en ambdós CD's. Per tant no l'hem donat d'alta dues vegades a la taula `titols`, però sí que l'hem donat d'alta dues vegades a la taula `producte_titol_autor` doncs forma part de dos productes diferents.

Adonem-nos, també, que primer hem afegit el tema a la taula `titols` i posteriorment hem introduït la informació a la taula `producte_titol_autor`. Si intentéssim introduir una fila en aquesta taula sense que el corresponent `codi_producte` existís a la taula

productes o sense que el corresponent `codi_títol` existís a la taula `títols` o sense que el corresponent `codi_autor` existís a la taula `autors`, MySQL ens retornaria un error com:

```
ERROR 1452 (23000): Cannot add or update a child row: ...
```

que ens indica que no és possible afegir o modificar una fila filla (s'entén que les files de la taula `producte_títol_autor` són filles de les taules `productes`, `títols` i `autors` degut a la integritat referencial definida. De fet, en l'espai indicat pels punts suspensius en l'anterior missatge d'error, MySQL ens informaria de la restricció d'integritat referencial violada.

I, un darrer comentari: oi que és un pal introduir les dades a la base de dades tal i com ho hem fet? oi que és d'agrair una aplicació informàtica com *MySQL Query Browser* que ens facilita la introducció de dades? Doncs tingueu en compte que totes les aplicacions informàtiques que gestionen dades emmagatzemades en bases de dades utilitzen el llenguatge SQL per a consultar (`SELECT`) i actualitzar (`INSERT`, `UPDATE`, `DELETE`) les dades.

Bé, una vegada executades totes les instruccions d'inserció anteriors, podem comprovar que la informació ha quedat ben introduïda i de pas exercitem les possibilitats de les sentències `SELECT`.

Example: Productes de l'autor Barry White existents a la base de dades.

```
select codi_producte "Producte", titol "Títol"
from productes where autor = 664;
```

I si no sabem que Barry White és l'autor de codi 664:

```
select codi_producte "Producte", titol "Títol" from productes
where autor = (select codi_autor from autors where upper(nom)="BARRY WHITE");
```

El resultat ha de ser:

```
+-----+-----+
| Producte          | Títol                               |
+-----+-----+
| 8435108611742     | Greatest Hits Barry White Vol .1 |
| 8435108611759     | Greatest Hits Barry White Vol .2 |
+-----+-----+
2 rows in set (0.01 sec)
```

Exemple: Productes de l'autor Barry White existents a la base de dades amb el número de temes que hi ha a cada producte.

```
select pro.codi_producte "Producte", titol "Títol", count(*) "N. Temes"
from productes pro, producte_titol_autor pta
where autor = (select codi_autor from autors where upper(nom)="BARRY WHITE")
and pro.codi_producte = pta.codi_producte
group by 1,2;
```

O també:

```
select pro.codi_producte "Producte", titol "Títol", count(*) "N. Temes"
from productes pro inner join producte_titol_autor pta
on pro.codi_producte = pta.codi_producte
where autor = (select codi_autor from autors where upper(nom)="BARRY WHITE")
group by 1,2;
```

El resultat ha de ser:

```
+-----+-----+-----+
| Producte          | Títol                                | N. Temes |
+-----+-----+-----+
| 8435108611742    | Greatest Hits Barry White Vol .1 | 12       |
| 8435108611759    | Greatest Hits Barry White Vol .2 | 12       |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

Les dues darreres instruccions donen el resultat esperat en cas que el producte cercat tingui temes introduïts, però si un producte no té temes no apareixeria en el resultat de les dues instruccions anteriors i en canvi sembla que la lògica ens diu que hauria d'aparèixer amb 0 com a número de temes. Recordem que per solucionar aquesta situació SQL ens facilita les operacions LEFT JOIN i RIGHT JOIN.

Suposem per un moment que també s'hagués fet la inserció:

```
insert into productes (codi_producte, titol, autor, suport, format, genere)
values ('XXX', 'Producte de prova', 664, 3, 2, 21);
```

Quina és la instrucció correcta per visualitzar tots els productes d'en Barry White acompanyats del número de temes de cada producte?

```
select pro.codi_producte "Producte", titol "Títol", count(*) "N. Temes"
from productes pro left join producte_titol_autor pta
on pro.codi_producte = pta.codi_producte
where autor = (select codi_autor from autors where upper(nom)="BARRY WHITE")
group by 1,2;
```

El resultat obtingut és:

```
+-----+-----+-----+
| Producte | Títol | N. Temes |
+-----+-----+-----+
| 8435108611742 | Greatest Hits Barry White Vol .1 | 12 |
| 8435108611759 | Greatest Hits Barry White Vol .2 | 12 |
| XXX | Producte de prova | 1 |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

Encara no és correcte, doncs el producte XXX hauria de tenir 0 temes. MySQL ens dona correctament 1 com a resultat doncs estem comptant registres i el resultat del `LEFT JOIN` és un registre que té els camps `codi_producte` i `títol` de la taula `productes` i valors nuls pels camps de la taula `producte_títol_autor`. Com que hem indicat `count(*)` i hi ha columnes amb valor no nul, el registre és comptabilitzat. En canvi, si fem:

```
select pro.codi_producte "Producte", titol "Títol",
       count(pta.codi_producte) "N. Temes"
from productes pro left join producte_títol_autor pta
    on pro.codi_producte = pta.codi_producte
where autor = (select codi_autor from autors where upper(nom)="BARRY WHITE")
group by 1,2;
```

el resultat obtingut és, per fi, l'esperat:

```
+-----+-----+-----+
| Producte | Títol | N. Temes |
+-----+-----+-----+
| 8435108611742 | Greatest Hits Barry White Vol .1 | 12 |
| 8435108611759 | Greatest Hits Barry White Vol .2 | 12 |
| XXX | Producte de prova | 0 |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

5.2. Sentència UPDATE

Aquesta és la sentència facilitada pel llenguatge SQL per a modificar files existents a les taules. La seva sintaxis és:

```
UPDATE <nom_taula>
SET col1=val1, col2=val2, col3=val3,...
[WHERE <condició>];
```

La clàusula `SET` indica les columnes a actualitzar i el valor amb què s'actualitzen. El valor d'actualització d'una columna pot ser el resultat obtingut per una sentència `SELECT` que recupera una única fila.

La clàusula optativa `WHERE` selecciona les files a actualitzar. En cas d'inexistència s'actualitzen totes les files de la taula.

Exemple: Ens hem assabentat del domicili de l'amiga Maria Teresa inserida més amunt: C/. Bona Sort, 13 amb codi postal 08700 d'Igualada i volem inserir aquesta informació a la base de dades.

Recordem que en inserir aquesta amiga només coneixíem la seva població. De fet, podem comprovar la informació existent amb la sentència:

```
select * from amics
where upper(nom_cognoms) LIKE "MARIA TERESA%";
```

Bé, una vegada tenim clar que el registre obtingut amb l'anterior sentència és el registre a modificar, executarem la sentència d'actualització:

```
update amics set adreca="C/. Bona Sort, 13", cp="08700"
where upper(nom_cognoms) LIKE "MARIA TERESA%";
```

Exemple: Modificar les descripcions dels gèneres introduïts a la taula `generes` de manera que quedin escrits amb la inicial en majúscula i la resta de lletres en minúscules.

```
update generes
set descripcio=concat(upper(substr(descripcio,1,1)),lower(substr(descripcio,2)));
```

Comentem una mica aquesta instrucció doncs pot costar d'entendre. Si MySQL incorporés una funció `X` que deixés la inicial d'una cadena en majúscula i la resta en minúscules, només ens caldria efectuar:

```
update generes
set descripcio=X(descripcio);
```

Però no és el cas i, per tant, ens hem de "buscar la vida" i la manera en que podem aconseguir el resultat esperat és:

– cerquem la cadena formada per la inicial:

```
substr(descripcio,1,1)
```

- la convertim a majúscula:

```
upper(substr(descripcio,1,1))
```

- cerquem la cadena formada per la resta de lletres:

```
substr(descripcio,2)
```

- la convertim a minúscules:

```
lower(substr(descripcio,2)
```

- concatenem les dues cadenes amb la funció `concat`.

5.3. Sentència DELETE

Aquesta és la sentència facilitada pel llenguatge SQL per a esborrar files existents a les taules. La seva sintaxis és:

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

La clàusula optativa WHERE selecciona les files a eliminar. En cas d'inexistència s'eliminen totes les files de la taula.

Exemple: Esborrar l'amiga de nom "Maria Teresa".

```
delete from amics  
where upper(nom_cognoms) like "MARIA TERESA%";
```

Exemple: Esborrar l'autor "Barry White"

```
delete from autors  
where upper(nom) = "BARRY WHITE";
```

En executar aquesta sentència, el SGBDR comunica el següent error:

```
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails  
(`musica/producte_titol_autor`, CONSTRAINT `FK_protitaut_autors` FOREIGN KEY  
(`Codi_Autor`) REFERENCES `autors` (`Codi_Autor`) ON UPDATE CASCADE)
```

El motiu radica en la integritat referencial definida entre les taules autors i producte_títol_autor, que impossibilita eliminar un autor si hi ha informació corresponent introduïda a la taula producte_títol_autor. Aquestes s'eliminarien si hi hagués definida l'eliminació en cascada, però no és el cas.

Per tant, per aconseguir l'objectiu d'eliminar l'autor "Barry White" cal fer, en primer lloc:

```
delete from producte_títol_autor
where codi_autor=(select codi_autor from autors where upper(nom)like "BARRY WHITE");
```

per tal d'eliminar totes les files de la taula `producte_títol_autor` corresponents a "Barry White". Fixem-nos que aquesta instrucció funcionarà si només hi ha un autor que tingui per nom "Barry White" (o qualsevol combinació possible utilitzant majúscules i minúscules) i donaria un error si tinguéssim varis "Barry White". Si és el cas de poder tenir varis "Barry White" i voler-los eliminar tots, faríem:

```
delete from producte_títol_autor
where codi_autor IN
      (select codi_autor from autors where upper(nom)like "BARRY WHITE");
```

Bé, una vegada eliminades les files de la taula `producte_títol_autor`, podem procedir a eliminar l'autor (o autors), fent:

```
delete from autors
where upper(nom) = "BARRY WHITE";
```

5.4. Transaccions

Una transacció és una seqüència d'instruccions SQL que el SGBD gestiona com una unitat.

Pensem, per exemple, en el darrer exemple presentat, en el que ens hem vist obligats a eliminar primer unes files de la taula `producte_títol_autor` i posteriorment la fila de la taula `autors`. És obvi que ens agradaria assegurar la unitat de les dues instruccions (o s'executen les dues en conjunt o no se n'executa cap). Per aconseguir això ens interessa que les dues instruccions formin part d'una transacció.

La majoria de SGBD faciliten unes instruccions per treballar amb transaccions. El funcionament és similar però no idèntic en tots els SGBD. MySQL facilita suport de transaccions (i també suport d'integritat referencial) per les seves taules segons la tipologia d'aquestes. En altres SGBD sempre es facilita suport de transaccions i d'integritat referencial. MySQL suporta diferents tipus de taules, tals com `MyISAM`, `InnoDB`, `Memory`, `MERGE`, `NDB`, `BDB` i `ISAM`, fet que s'ha d'indicar en el moment de creació de la taula. Alguns d'aquests tipus (no tots) faciliten control de transaccions i d'integritat referencial. Els comportaments que veurem a

continuació tindran lloc si les taules involucrades són d'un tipus que faciliti control transaccional, com és el InnoDB.

Tipus de taules en la base de dades música

Les taules inicials de la base de dades música que utilitzem en tots els exemples són del tipus InnoDB i, per tant, les instruccions transaccionals s'hi executaran.

Presentem, a continuació, les instruccions per al control de transaccions que facilita MySQL:

Instrucció	Funcionament
<code>start transaction</code>	Instrucció que serveix per indicar l'inici d'una transacció
<code>commit</code>	Instrucció per finalitzar la transacció activa deixant com a permanents els canvis efectuats a les taules durant la transacció
<code>rollback</code>	Instrucció per finalitzar la transacció activa desfent tots els canvis efectuats a les taules durant la transacció

Així, el darrer exemple hagués hagut d'estar resolt de la següent forma:

```
start transaction;

delete from producte_titol_autor
where codi_autor IN
      (select codi_autor from autors where upper(nom) like "BARRY WHITE");

delete from autors
where upper(nom) = "BARRY WHITE";

commit;
```

Els canvis realitzats a la base de dades en el transcurs d'una transacció només són visibles per la connexió des d'on s'executen. En executar un COMMIT, els canvis realitzats a la base de dades passen a ser permanents i, per tant, visibles per tots els usuaris.

Una transacció finalitza amb la sentència COMMIT, amb la sentència ROLLBACK o amb la desconnexió (intencionada o no) de la base de dades.

Hi ha SGBD, com Oracle, en els que el funcionament per defecte consisteix en iniciar una transacció (sense haver-ho d'indicar) en qualsevol de les situacions següents:

- en el moment d'establir connexió amb la base de dades
- després d'una sentència COMMIT
- després d'una sentència ROLLBACK

MySQL, però, té un funcionament per defecte diferent a l'esmentat: a no ser que el servidor MySQL es configuri explícitament d'altra manera (tasca administrativa que sobrepassa els objectius), les connexions que s'estableixen amb una base de dades MySQL funcionen de manera que

tota instrucció efectuada queda automàticament enregistrada a la base de dades i, per tant, no hi ha possibilitat d'efectuar `ROLLBACK` (o si es fa, no aconseguirem desfer cap canvi, ni tant sols el darrer) a no ser que s'hagi iniciat una transacció amb la sentència `START TRANSACTION`. MySQL anomena `AUTOCOMMIT` a aquest funcionament.

Per millor entendre el funcionament, exemplifiquem algunes situacions en les que apareixen unes instruccions A, B i C de modificació de dades (`INSERT`, `UPDATE`, `DELETE`).

Situació A:

```
mysql> use <nom_base_dades>
mysql> instrucció_A;
mysql> instrucció_B;
mysql> instrucció_C;
mysql> commit; o rollback;
```

En aquest cas els canvis que puguin provocar les instruccions A, B i C queden immediatament permanents per a totes les connexions i ni es poden desfer (`ROLLBACK`) ni cal validar-los (`COMMIT`). La darrera sentència `COMMIT` o `ROLLBACK` no té cap sentit.

Situació B:

```
mysql> use <nom_base_dades>
mysql> start transaction;
mysql> instrucció_A;
mysql> instrucció_B;
mysql> instrucció_consulta; (*)
mysql> rollback; (**)
mysql> instrucció_consulta; (***)
mysql> instrucció_C;
```

En aquest cas, quan s'arriba a la instrucció de consulta (*), aquesta visualitzarà els canvis efectuats per les instruccions A i B com si els canvis fossin permanents. En canvi, si en el mateix moment des d'una altra connexió s'executa la mateixa instrucció de consulta (*), els canvis efectuats per les instruccions A i B no són visibles, doncs encara no han estat validats.

L'execució de la instrucció `ROLLBACK (**)` provoca que els canvis efectuats per les instruccions A i B es desfacin, de manera que la instrucció de consulta (***) visualitzarà les taules sense els canvis efectuats per les instruccions A i B. La instrucció `ROLLBACK` provoca que la transacció activa finalitzi.

Els canvis efectuats per la instrucció C queden automàticament enregistrats doncs no s'ha iniciat cap altra transacció.

Situació C:

```
mysql> use <nom_base_dades>
mysql> start transaction;
mysql> instrucció_A;
mysql> instrucció_B;
mysql> instrucció_consulta; (*)
mysql> commit; (**)
mysql> instrucció_consulta; (***)
mysql> instrucció_C;
```

Aquí plantegem altra vegada la situació anterior però validant els canvis efectuats per les instruccions A i B amb el `COMMIT (**)` . El comportament fins la instrucció anterior al `COMMIT` és idèntic a la situació B.

L'execució de la instrucció `COMMIT (**)` provoca que els canvis efectuats per les instruccions A i B es validin, de manera que la instrucció de consulta `(***)` visualitzarà les taules amb els canvis efectuats per les instruccions A i B. Amb posterioritat al `COMMIT`, qualsevol altra connexió que executi la consulta `(***)` visualitzarà els canvis efectuats per les instruccions A i B (mentre que abans del `COMMIT` no eren visibles). La instrucció `COMMIT` provoca que la transacció activa finalitzi.

De manera idèntica a la situació B, els canvis efectuats per la instrucció C queden automàticament enregistrats doncs no s'ha iniciat cap altra transacció.

Observacions:

- Més amunt hem dit que les connexions MySQL estan configurades en mode `AUTOCOMMIT` per defecte, però si es vol canviar aquest funcionament es pot utilitzar el comandament `SET AUTOCOMMIT` tal i com es veu en l'exemple. La seva sintaxis és: 

```
SET AUTOCOMMIT=mode;
```

on mode pot valer 0 per desactivar-lo o 1 per activar-lo.

Per conèixer l'estat d'`AUTOCOMMIT` d'una connexió es pot executar:

```
select @@autocommit;
```

Així doncs, en la situació següent,

```
mysql> use <nom_base_dades>
mysql> set autocommit=0;
mysql> instrucció_A;
mysql> instrucció_B;
mysql> instrucció_consulta;
mysql> commit; o rollback;
mysql> instrucció_C;
```

una vegada desactivat l'AUTOCOMMIT, totes les instruccions que s'executen formen automàticament part d'una transacció sense necessitat d'utilitzar la sentència `START TRANSACTION` i ens veiem obligats a utilitzar les sentències `COMMIT` per validar els canvis o `ROLLBACK` per a desfer-los.

En aquest cas, una vegada executades les 2 instruccions A i B, si l'usuari de la connexió executa una sentència de consulta, visualitzarà els canvis efectuats, però des d'una altra connexió, en el mateix moment, els canvis no són visibles, doncs no s'han validat.

La sentència `COMMIT` o `ROLLBACK` valida o desfà els canvis de la transacció activa i automàticament s'inicia una altra transacció sense necessitat d'utilitzar la sentència `START TRANSACTION` doncs l'AUTOCOMMIT és desactivat. Per tant, els canvis de la instrucció C només seran visibles des de la connexió actual i passaran a ser visibles per altres connexions quan s'executi un `COMMIT` o no ho seran mai si s'executa un `ROLLBACK` (el que primer succeeixi).

- En totes les situacions plantejades hem suposat el funcionament per defecte de MySQL consistent en que totes les connexions s'inicien amb AUTOCOMMIT activat. Cal saber, però, que hi ha la possibilitat (tasca administrativa que sobrepassa el nivell d'aquest crèdit) de configurar el servidor de manera que les connexions dels usuaris no administradors s'estableixin amb l'AUTOCOMMIT desactivat. 
- El SGBD MySQL realitza un COMMIT implícit abans d'executar qualsevol sentència LDD. Altres SGBD tenen comportaments similars. 

És a dir, si ens trobem un seguit d'instruccions pendents de validar i executem una sentència LDD (creació/modificació/eliminació de taules, índexs, vistes,...) s'efectua un COMMIT de totes les instruccions pendents de validar.

- Hi ha la possibilitat de marcar punts de salvaguarda (SAVEPOINTS)) enmig d'una transacció, de manera que si s'efectua ROLLBACK aquest

pugui ser total (tota la transacció) o fins un dels punts de salvaguarda de la transacció. 

La sintaxis de les instruccions involucrades és:

```
SAVEPOINT <nom_punt_salvaguarda>;
```

per a marcar un punt de salvaguarda i

```
ROLLBACK TO <nom_punt_salvaguarda>;
```

per a desfer tots els canvis efectuats des del punt de salvaguarda.

Si en una transacció es crea un punt de salvaguarda amb el mateix nom que un punt de salvaguarda existent, l'existent queda substituït pel nou.

Així, en la situació següent:

```
mysql> use <nom_base_dades>
mysql> start transaction;
mysql> instrucció_A;
mysql> savepoint PB;
mysql> instrucció_B;
mysql> savepoint PC;
mysql> instrucció C;
mysql> instrucció_consulta_1;
mysql> rollback to PC;
mysql> instrucció_consulta_2;
mysql> rollback; o commit;
```

la instrucció de consulta 1 veu els canvis efectuats per les instruccions A, B i C, però el ROLLBACK TO PC desfà els canvis produïts des del punt de salvaguarda PC, per la qual cosa la instrucció de consulta 2 només veu els canvis efectuats per les instruccions A i B (els canvis per C han desaparegut) i el darrer ROLLBACK desfà tots els canvis efectuats per A i B mentre que el darrer COMMIT els deixaria com a permanents.