Llenguatge SQL per a la manipulació i definició de les dades. Control de transaccions i concurrència

Cristina Obiols Llopart
Adaptació de continguts: Isidre Guixà Miranda i Cristina Obiols Llopart

Índex

Introducció

Sobre les bases de dades no solament hi hem d'aplicar instruccions per tal d'extreure'n informació, sinó que és necessari poder manipular la informació enregistrada (afegint-ne de nova, eliminant-ne i modificant-ne la ja introduïda). Per això en aquesta unitat aprendrem les instruccions SQL per a la manipulació de dades d'una base de dades.

Així mateix, cal disposar d'algun mecanisme per definir taules, vistes, índexs i altres objectes que conformen la base de dades, i també per modificar-ne l'estructura si és necessari. I, per descomptat, també és molt important poder controlar l'accés a la informació que hi ha en la base de dades. El llenguatge SQL ens proporciona sentències per assolir tots aquests objectius.

En un SGBD en explotació, s'acostuma a encomanar a l'administrador de l'SGBD la definició de les estructures de dades. Però això no treu que tot informàtic –tant si és especialitzat en desenvolupament d'aplicacions informàtiques o en administració de sistemes informàtics– ha de conèixer les principals sentències que proporciona el llenguatge SQL per a la definició de les estructures de dades. Penseu que una persona que desenvolupi aplicacions ha de ser capaç de crear l'estructura de la base de dades (taules, vistes, índexs...).

Així, doncs, començarem coneixent les diverses possibilitats per manipular la informació, per continuar definint les estructures que permeten emmagatzemar les dades i acabar controlant el concepte de *transacció* i també les eines per controlar-les i per controlar l'accés concurrent a les dades.

Concretament, en l'apartat d'"Instruccions per a la manipulació de dades", aprendreu les instruccions per a afegir i eliminar files, modificar dades, reemplaçar files i treballar amb XML en MySQL.

En l'apartat "DDL" començareu coneixent els diversos motors d'emmagatzematge que ofereix MySQL per a tot seguit començar a veure les diferents instruccions per a crear i eliminar taules de la base de dades, així com modificar-ne l'estructura. La definició de vistes, la creació d'índexs, el canvi de noms de taules o l'eliminació de totes les files d'una taula també són accions que es poden dur a terme a través d'instruccions en MySQL que aprendrem en aquest apartat.

En l'apartat "Control de transaccions i concurrències" es defineix el concepte de transacció i de bloqueig i es descriu el procediment per a realitzar de forma segura la concurrència d'operacions sobre les dades d'una base de dades.

Per assolir un bon coneixement del llenguatge SQL, cal que aneu reproduint al vostre ordinador tots els exemples incorporats en el text, i també les activitats i els exercicis d'autoavaluació. I, per poder-ho fer, continuarem utilitzant l'SGBD MySQL i les eines adequades seguint les instruccions del material web.

Així mateix, per aprendre a aplicar amb agilitat les tècniques de disseny en el model relacional, les quals són molt teòriques, és imprescindible efectuar totes les activitats proposades i els exercicis d'autoavaluació del material web.

Resultats d'aprenentatge

En finalitzar aquesta unitat l'alumne/a:

1. Consulta i modifica la informació emmagatzemada en una base de dades emprant assistents, eines gràfiques i el llenguatge de manipulació de dades.
    - Identifica eines i sentències per modificar el contingut de la base de dades.
    - Formula consultes per inserir, modificar i/o eliminar dades de la base de dades.
    - Insereix en una taula dades com a resultant de l'execució d'una consulta.
    - Identifica les transaccions i el seu funcionament.
    - Controla els canvis produïts per una transacció: parcialment o totalment.
    - Identifica els efectes de les diferents polítiques de bloqueig de registres.
    - Adopta mesures per mantenir la integritat i consistència de la informació.
    - Identifica les transaccions, concurrències i la recuperació d'errades.
2. Realitza el disseny físic de bases de dades utilitzant assistents, eines gràfiques i el llenguatge de definició de dades.
    - Identifica els tipus de llenguatges per definir i manipular dades sobre un SGBDR corporatiu de manera interactiva.
    - Identifica els elements de l'estructura d'una base de dades i els defineix emprant assistents, eines gràfiques i/o el llenguatge de definició de dades (DDL), a partir del disseny de la BBDD i dels requeriments d'usuari.
    - Empra assistents, eines gràfiques i el llenguatge de definició de dades per definir l'estructura d'una base de dades sobre un SGBDR corporatiu de manera interactiva i tenint en compte les regles sintàctiques.
    - Identifica les funcions, la sintaxi i les ordres bàsiques del llenguatge SQL per definir l'estructura d'una base de dades.
    - Defineix els índex en una bases de dades per tal de millorar el rendiment del sistema gestor de bases de dades.
    - Crea, modifica i elimina sinònims a taules i vistes de la BBDD.
    - Identifica i implanta les restriccions a les taules que estan reflectides en el disseny lògic.

1. Instruccions per a la manipulació de dades

Mitjançant la sentència `SELECT` podem consultar dades, però, com les podem manipular, definir i controlar?

El llenguatge SQL aporta un seguit d'instruccions amb les quals es poden realitzar les accions següents:

- La manipulació de les dades (instruccions LMD que ens han de permetre efectuar altes, baixes i modificacions).
- La definició de dades (instruccions LDD que ens han de permetre crear, modificar i eliminar les taules, els índexs i les vistes).
- El control de dades (instruccions LCD que ens han de permetre gestionar els usuaris i els seus privilegis).

Acrònims

Recordem els acrònims per als diferents apartats del llenguatge SQL: LC (llenguatge de consulta); LMD (llenguatge de manipulació de dades); LDD (llenguatge de definició de dades); LCD (llenguatge de control de dades).

El llenguatge SQL proporciona un conjunt d'instruccions, reduït però molt potent, per manipular les dades, dins el qual s'ha de distingir entre dos tipus d'instruccions:

- Les instruccions que permeten executar la manipulació de les dades, i que es redueixen a tres: `INSERT` per a la introducció de noves files, UPDATE per a la modificació de files, i `DELETE` per l'esborrament de files.
- Les instruccions per al control de transaccions, que han de permetre assegurar que un conjunt d'operacions de manipulació de dades s'executi amb èxit en la seva totalitat o, en cas de problema, s'avorti totalment o fins a un determinat punt en el temps.

Abans d'introduir-nos en l'estudi de les instruccions INSERT, UPDATE i DELETE, cal conèixer com l'SGBD gestiona les instruccions d'inserció, eliminació i modificació que hi puguem executar, ja que hi ha dues possibilitats de funcionament:

- Que quedin automàticament validades i no hi hagi possibilitat de tirar enrere. En aquest cas, els efectes de tota instrucció d'actualització de dades que tingui èxit són automàticament accessibles des de la resta de connexions de la base de dades.
- Que quedin en una cua d'instruccions, que permet tirar enrere. En aquest cas, es diu que les instruccions de la cua estan pendents de validació, i l'usuari ha d'executar, quan ho creu convenient, una instrucció per validar-les (anomenada *COMMIT*) o una instrucció per tirar enrere (anomenada *ROLLBACK*).

Aquest funcionament implica que els efectes de les instruccions pendents de validació no es veuen per la resta de connexions de la base de dades, però sí són accessibles des de la connexió on s'han efectuat. En executar la COMMIT, totes les connexions accedeixen als efectes de les instruccions validades. En cas d'executar ROLLBACK, les instruccions desapareixen de la cua i cap connexió (ni la pròpia ni la resta) no accedeix als efectes corresponents, és a dir, és com si mai haguessin existit.

Aquests possibles funcionaments formen part de la gestió de transaccions que proporciona l'SGBD i que cal estudiar amb més deteniment. A l'hora, però, d'executar instruccions INSERT, UPDATE i DELETE hem de conèixer el funcionament de l'SGBD per poder actuar en conseqüència.

Així, per exemple, un SGBD MySQL funciona amb validació automàtica després de cada instrucció d'actualització de dades llevat que s'indiqui el contrari i, en canvi, un SGBD Oracle funciona amb la cua d'instruccions pendents de confirmació o rebuig que ha d'indicar l'usuari.

En canvi, en MySQL, si es vol desactivar l'opció d'autocommit que hi ha per defecte, caldrà executar la instrucció següent:

```
SET autocommit=0;
```

1.1 Sentència INSERT

La sentència INSERT és la instrucció proporcionada pel llenguatge SQL per inserir noves files en les taules.

Admet dues sintaxis:

1. Els valors que s'han d'inserir s'expliciten en la mateixa instrucció en la clàusula `values`:

```
insert into <nom_taula> [(col1, col2...)]
values (val1, val2...);
```

2. Els valors que s'han d'inserir s'aconsegueixen per mitjà d'una sentència `SELECT`:

```
insert into <nom_taula> [(col1, col2...)]
select...;
```

En tot cas, es poden especificar les columnes de la taula què s'han d'emplenar i l'ordre en què se subministren els diferents valors. En cas que no s'especifiquin les columnes, l'SQL entén que els valors se subministren per a totes les columnes de la taula i, a més, en l'ordre en què estan definits en la taula.

La llista de valors de la clàusula `values` i la llista de resultats de la sentència SELECT han de coincidir en nombre, tipus i ordre amb la llista de columnes que s'han d'emplenar.

Exemple 1 de sentència INSERT

En l'esquema *empresa*, es demana inserir el departament 50 de nom 'INFORMÀTICA'.

La possible sentència per aconseguir l'objectiu és aquesta:

```
insert into dept (dept_no, dnom)
values (50, 'INFORMÀTICA');
```

Si executem una consulta per comprovar el contingut actual de la taula DEPT, trobarem la nova fila sense localitat assignada. L'SGBD ha permès deixar la localitat amb valor NULL perquè ho té permès així, com es pot veure en el descriptor de la taula DEPT:

```
SQL> desc dept;

Name                 Null     Type
-------------------- -------- ------------------------------
DEPT_NO              NOT NULL NUMBER(2)
DNOM                 NOT NULL VARCHAR2(14)
LOC                           VARCHAR2(14)

3 rows selected
```

Exemple 2 de sentència INSERT

En l'esquema *sanitat*, es demana donar d'alta el doctor de codi 100 i nom 'BARRUFET D.'.

La solució sembla que podria ser aquesta:

```
insert into doctor (doctor_no, cognom)
values (100, 'BARRUFET D.');
```

En executar aquesta sentència, l'SGBDR dóna un error.

El cert és que la taula DOCTOR no admet valors nuls en la columna hospital_cod, ja que aquesta columna forma part de la clau primària. Mirem el descriptor de la taula DOCTOR:

```
SQL> desc doctor;

Name                 Null     Type
-------------------- -------- ---------------
HOSPITAL_COD         NOT NULL NUMBER(2)
DOCTOR_NO            NOT NULL NUMBER(3)
COGNOM               NOT NULL VARCHAR2(13)
ESPECIALITAT         NOT NULL VARCHAR2(16)

4 rows selected
```

A banda de la columna hospital_cod, també hauríem de donar un valor en la columna especialitat, ja que tampoc no admet valors nuls.

Recordem que, en el nostre esquema *sanitat*, la columna especialitat és una cadena que no té cap tipus de restricció definida ni és clau forana de cap taula en la qual hi hagi totes les especialitats possibles. Per tant, si volem saber quines especialitats hi ha per tal d'escriure la del doctor que volem inserir, idènticament a les ja introduïdes en cas que hi hagués algun doctor amb la mateixa especialitat del que hi volem inserir, fem el següent:

```
SQL> select distinct especialitat from doctor;

ESPECIALITAT

Urologia
```

```
Pediatria
Cardiologia
Neurologia
Ginecologia
Psiquiatria

6 rows selected
```

Suposem que el doctor 'BARRUFET D.' és psiquiatre. Com que ja hi ha algun doctor amb l'especialitat 'Psiquiatria', correspondria fer la inserció utilitzant la mateixa grafia per a l'especialitat. A més, suposem que volem donar d'alta el doctor a l'hospital 66.

```
insert into doctor (doctor_no, cognom, hospital_cod, especialitat
    )
values (100, 'BARRUFET D.', 66, 'Psiquiatria');
```

Aquesta vegada, l'SGBD també se'ns queixa amb un altre tipus d'error: ha fallat la referència a la clau forana.

L'error ens informa que una restricció d'integritat definida en la taula ha intentat ser violada i, per tant, la instrucció no ha finalitzat amb èxit. L'SGBD ens passa dues informacions perquè tinguem pistes d'on hi ha el problema:

- Ens dóna una descripció breu del problema (no es pot afegir una fila filla -*child row*-), la qual ens dóna a entendre que es tracta d'un error de clau forana, és a dir, que no existeix el codi en la taula referenciada.
- Ens diu la restricció que ha fallat (`sanitat.doctor, CONSTRAINT ... FOREIGN KEY (HOSPITA_COD) ...`).

L'SGBD té tota la raó. Recordem que la columna hospital_cod de la taula HOSPITAL és clau forana de la taula HOSPITAL. Això vol dir que qualsevol inserció en la taula DOCTOR ha de ser per a hospitals existents en la taula HOSPITAL, i això no passa amb l'hospital 66, com es pot veure en consultar els hospitals existents:

```
SQL> select * from hospital;

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

4 rows selected
```

Així, doncs, o ens hem equivocat d'hospital o hem de donar d'alta prèviament l'hospital 66. Suposem que és el segon cas i que, per tant, hem de donar d'alta l'hospital 66:

```
insert into hospital (hospital_cod, nom, adreca)
values (66,'General','De la font, 13');
```

L'SGBD ens accepta la instrucció. Fixem-nos que hem informat del codi d'hospital, del nom i de l'adreça. Mirem el descriptor de la taula HOSPITAL:

```
SQL> desc hospital;

Name               Null     Type
─────────────────── ──────── ──────────────────────────
HOSPITAL_COD       NOT NULL NUMBER(2)
NOM                NOT NULL VARCHAR2(10)
ADRECA        VARCHAR2(20)
TELEFON                     VARCHAR2(8)
QTAT_LLITS                  NUMBER(3)

5 rows selected
```

Hi veiem cinc camps, dels quals només els dos primers tenen marcada l'obligatorietat de valor. Per tant, no se'ns ha queixat perquè no hàgim indicat el telèfon de l'hospital ni la quantitat de llits que té l'hospital.

Comprovem la informació que ara hi ha en la taula `HOSPITAL`:

```
SQL> select * from hospital;

HOSPITAL_COD   NOM        ADREÇA               TELÈFON
    QTAT_LLITS
------------ ---------- -------------------- --------
    ----------
13             Provincial O Donell 50          964-4264 88
18             General    Atocha s/n           595-3111 63
22             La Paz     Castellana 1000      923-5411 162
45             San Carlos Ciudad Universitaria 597-1500 92
66             General    De la font, 13                0

5 rows selected
```

Sorpresa! Per al nou hospital, la columna telèfon no té valor (valor NULL), però la columna qtatllits té el valor 0. D'on ha sortit? Això es deu al fet que la columna qtatllits de la taula `HOSPITAL` té definit el valor per defecte (0) que l'SGBD utilitza per emplenar la columna qtat_llits quan es produeix una inserció en la taula sense indicar valor per a aquesta columna.

Ara sembla que ja hi podem inserir el nostre doctor 'BARRUFET D'.:

```
insert into doctor (doctor_no, cognom, hospital_cod, especialitat
    )
values (100, 'BARRUFET D.', 66, 'Psiquiatria');
```

No ens oblidem d'enregistrar els canvis amb la instrucció `COMMIT` o de fer `ROLLBACK`, si tenim l'autocommit desactivat.

Exemple 3 de sentència INSERT

Abans de començar, desactivarem l'autocommit que té configurat per defecte MySQL per poder practicar el `commit` i el `rollback`:

```
SET AUTOCOMMIT=0;
```

En l'esquema *empresa*, es vol inserir la instrucció identificada pel número 1.000, amb data d'ordre l'1 de setembre de 2000 i per al client 500.

Potser ens cal conèixer, en primer lloc, el descriptor de la taula `COMANDA`:

```
SQL> desc comanda;

Name                Null     Type
------------------- -------- ------------------------------
COM_NUM             NOT NULL NUMBER(4)
COM_DATA                     DATE
COM_TIPUS                    VARCHAR2(1)
CLIENT_COD          NOT NULL NUMBER(6)
DATA_TRAMESA                 DATE
TOTAL                        NUMBER(8,2)

6 rows selected
```

Fixem-nos que tenim la informació corresponent a tots els camps obligatoris. Per tant, podem executar el següent:

```
insert into comanda (com_num, com_data, client_cod)
values (1000,'2000/09/01',500);
```

L'SGBD ens reporta l'error de restricció d'integritat sobre la clau forana. I, per descomptat, l'SGBD torna a tenir raó, ja que en l'esquema *empresa* la taula COMANDA té una restricció de clau forana en la columna client_cod.

Si consultem el contingut de la taula client, veurem que no hi ha cap client amb codi 500. Per això, l'SGBD ha donat un error. Suposem que era un error nostre i l'ordre corresponia al client 109 (que sí existeix en la taula CLIENT). Aquesta vegada la instrucció següent no ens dóna cap problema.

```
insert into comanda (com_num, com_data, client_cod)
values (1000,'2000/09/01',109);
```

Podem comprovar com ha quedat inserida l'ordre:

```
SQL> select * from comanda where com_num=1000;

COM_NUM    COM_DATA      COM_TIPUS CLIENT_COD DATA_TRAMESA  TOTAL
---------- ------------- --------- ---------- ------------- -----
1000       01/09/2000               109
```

Fem rollback per tirar enrere la inserció efectuada i així poder comprovar que també la podríem fer de diferents maneres. Recordem que no és obligatori indicar les columnes per a les quals s'introdueixen els valors. En aquest cas, l'SGBD espera totes les columnes de la taula en l'ordre en què estan definides en la taula. Així, doncs, podem fer el següent:

```
insert into comanda
values (1000,'2000/09/01',NULL,109,NULL,NULL);
```

Fem rollback per provar una altra possibilitat. Fixem-nos que l'SGBD també ens deixa introduir un preu total d'ordre qualsevol:

```
rollback;

insert into comanda
values (1000,DATE '2000-09-01',NULL,109,NULL,9999);

commit;
```

Disparadors

Un disparador és un conjunt d'instruccions que s'executen automàticament davant un esdeveniment determinat. Així, podem controlar que, en inserir, esborrar o modificar files de detall d'una ordre, l'import total de l'ordre s'actualitzi automàticament.

L'SGBDR ha acceptat aquesta sentència i hi ha inserit la fila corresponent. Però, hem introduït un import total que no es correspon amb la realitat, ja que no hi ha cap línia de detall. És a dir, el valor 9999 no és vàlid! Els SGBD proporcionen mecanismes (disparadors) per controlar aquests tipus d'incoherències de les dades.

Exemple 4 de sentència INSERT

En primer lloc, tornem a activar l'opció d'autocommit per tal que sigui més còmoda la feina.

```
SET AUTOCOMMIT=1;
```

Com a detall de l'ordre 1000 inserida en l'exemple anterior, en l'esquema *empresa* es volen inserir les mateixes línies que conté l'ordre 620.

En aquest cas, executarem una instrucció INSERT prenent com a valors que cal inserir els que ens dóna el resultat d'una sentència SELECT:

```
insert into detall
select 1000, detall_num, prod_num, preu_venda, quantitat, import
from detall
where com_num=620;
```

En aquesta instrucció, hem seleccionat les files de detall de l'ordre 620 i les hem inserit com a files de detall de l'ordre 1000. Hem de ser conscients que l'import total de l'ordre 1000 continua sent, però, incorrecte.

Com ja hem comentat, hem utilitzat una sentència SELECT per inserir valors en una taula. És una coincidència que totes dues sentències actuïn sobre la mateixa taula DETALL.

En no indicar, en la sentència INSERT, les columnes en què s'han d'inserir els valors, ha calgut construir la sentència SELECT de manera que les columnes de la clàusula `select` coincidissin, en ordre, amb les columnes de la taula en què s'ha d'efectuar la inserció. A més, com que per a totes les files de l'ordre 620 calia indicar 1000 com a número d'ordre, la clàusula `SELECT` ha incorporat la constant 1000 com a valor per a la primera columna.

1.2 Sentència UPDATE

La sentència `UPDATE` és la instrucció proporcionada pel llenguatge SQL per modificar files que hi ha en les taules.

La seva sintaxi és aquesta:

```
update <nom_taula>
set col1=val1, col2=val2, col3=val3...
[where <condició>];
```

La clàusula optativa `where` selecciona les files que s'han d'actualitzar. En cas d'inexistència, s'actualitzen totes les files de la taula.

La clàusula `set` indica les columnes que s'han d'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:

```
update <nom_taula>
set col1=(select exp1 from ... ),
set col2=(select exp2 from ... ),
set col3=val3,
...
[where <condició>];
```

En tals situacions, la sentència `SELECT` és una subconsulta de la sentència `UPDATE` que pot utilitzar valors de les columnes de la fila que s'està modificant en la sentència `UPDATE`.

Com que de vegades és possible que calgui actualitzar els valors de més d'una columna a partir de diferents resultats d'una mateixa sentència `SELECT`, no seria gens eficient executar diverses vegades la mateixa sentència `SELECT` per actualitzar més d'una columna. Per tant, la sentència `UPDATE` també admet la sintaxi següent:

```
update <nom_taula>
set (col1, col2)=(select exp1, exp2 from ... ),
set col3=val3,
...
[where <condició>];
```

Exemple 1 de sentència UPDATE

En l'esquema *empresa*, es vol modificar la localitat dels departaments de manera que quedin tots els caràcters amb minúscules.

La instrucció per resoldre la sol·licitud pot ser aquesta:

```
update dept
set loc = lower(loc);
```

Ara es vol modificar la localitat dels departaments de manera que quedin amb la inicial en majúscula i la resta de lletres amb minúscules.

La instrucció per resoldre la sol·licitud pot ser aquesta:

```
update dept
set loc=concat(upper(left(loc,1)),right(lower(loc),length(loc)-1)
    );
```

Exemple 2 de sentència UPDATE

En l'esquema *empresa*, es vol actualitzar l'import total real de la comanda 1000 a partir dels imports de les diferents línies de detall que formen la comanda.

```
update comanda c
set total = (select sum(import) from detall
     where com_num=c.com_num)
where com_num=1000;
```

Ara podem comprovar la correcció de la informació que hi ha en la base de dades sobre la comanda 1000:

```
SQL> select * from detall where com_num=1000;

COM_NUM    DETALL_NUM PROD_NUM   PREU_VENDA QUANTITAT  IMPORT
---------- ---------- ---------- ---------- ---------- ----------
1000       1          100860     35         10         350
1000       2          200376     2,4        1000       2400
1000       3          102130     3,4        500        1700

3 rows selected

SQL> select * from comanda where com_num=1000;

COM_NUM    COM_DATA   COM_TIPUS CLIENT_COD DATA_TRAMESA  TOTAL
---------- ---------- --------- ---------- ------------- -----
1000       01/09/2000           109                      4450

1 rows selected
```

1.3 Sentència DELETE

La sentència `DELETE` és la instrucció proporcionada pel llenguatge SQL per esborrar files existents que hi ha en les taules.

La seva sintaxi és aquesta:

```
delete from <nom_taula>
[where <condició>];
```

La clàusula optativa `where` selecciona les files que s'han d'eliminar. Si no n'hi ha, s'eliminen totes les files de la taula.

Exemple de sentència DELETE

En l'esquema *empresa*, es vol eliminar la comanda 1000.

La instrucció sembla que podria ser aquesta:

```
delete from comanda
where com_num=1000;
```

En executar aquesta sentència, però, ens trobem amb un error que ens indica que no es pot eliminar una fila pare (*a parent row*).

El motiu és que la columna com_num de la taula DETALL és clau forana de la taula COMANDA, fet que impossibilita eliminar una capçalera de comanda si hi ha línies de detall corresponents. Aquestes s'eliminarien de manera automàtica si hi hagués definida l'eliminació en cascada, però no és el cas. Així, doncs, caldrà fer el següent:

```
delete from detall where com_num=1000;
delete from comanda where com_num=1000;
```

1.4 Sentència REPLACE

MySQL té una extensió del llenguatge SQL estàndard que permet inserir una nova fila que, en cas que la clau primària coincideixi amb una altra fila, prèviament sigui eliminada. Es tracta de la sentència REPLACE.

Hi ha tres possibles sintaxis per a la sentència **REPLACE**:

```
REPLACE [INTO] nom_taula [(columna1,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...

REPLACE [INTO] nom_taula
    SET columna1={expr | DEFAULT}, ...

REPLACE [INTO] nom_taula [(columna1,...)]
    SELECT ...
```

1.5 Sentència LOAD XML

LOAD XML permet llegir un fitxer en format xml i emmagatzemar les dades contingudes en una taula de la base de dades. La seva sintaxi és:

XML

XML és l'acrònim d'*extensible markup language* ('llenguatge d'etiquetatge extensible') un metallenguatge de marques que facilita l'organització de les dades en fitxers plans.

```
LOAD XML [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'nom_fitxer'
[REPLACE | IGNORE]
INTO TABLE [nom_base_dades.]nom_taula
[CHARACTER SET nom_charset]
[ROWS IDENTIFIED BY '<nom_tag>']
[IGNORE número [LINES | ROWS]]
[(columnes,...)]
[SET nom_columna = expressió,...]
```

2. DDL

A banda de les conegudes instruccions per consultar i modificar les dades, el llenguatge SQL aporta instruccions per definir les estructures en què s'emmagatzemen les dades. Així, per exemple, tenim instruccions per a la creació, eliminació i modificació de taules i índexs, i també instruccions per definir vistes.

En el MySQL, l'SQLServer i el PostgreSQL qualsevol instància de l'SGBD gestiona un conjunt de bases de dades o esquemes, anomenat *cluster database*, el qual pot tenir definit un conjunt d'usuaris amb els privilegis d'accés i gestió que corresponguin.

En el MySQL, l'SQLServer i el PostgreSQL, el llenguatge SQL proporciona una instrucció `CREATE DATABASE <nom_base_dades>` que permet crear, dins la instància, les diverses bases de dades. Aquesta instrucció CREATE DATABASE es pot considerar dins l'àmbit del llenguatge LDD.

Distinció entre àmbits LDD i LCD en el llenguatge SQL

Sovint, els àmbits LDD (llenguatge de definició de dades) i LCD (llenguatge per al control de les dades) es fonen en un únic àmbit i es parla únicament d'LDD.

La sentència CREATE SCHEMA en l'àmbit del llenguatge LDD està destinada a la creació d'un esquema en què es puguin definir taules, índexs, vistes, etc. En MySQL, CREATE SCHEMA i CREATE TABLE són sinònims.

Disposem, també, de la instrucció `USE <nom_base_dades>` per decidir la base de dades en què es treballarà (establiment de la base de dades de treball per defecte).

2.1 Regles i indicacions per anomenar objectes en MySQL

Dins de MySQL, a banda de taules, trobarem altres tipus d'objectes: índexs, columnes, àlies, vistes, procediments, etc.

Els noms dels objectes dins d'una base de dades, i les bases de dades mateixes, en MySQL actuen com a identificadors, i, com a tals no es podran repetir en un mateix àmbit. Per exemple, no podem tenir dues columnes d'una mateixa taula que s'anomenin igual, però sí en taules diferents.

> Els noms amb què anomenem els objectes dins d'un SGBD hauran de seguir unes regles sintàctiques que hem de conèixer.

En general, les taules i les bases de dades són *not case sensitive*, és a dir, que hi podem fer referència en majúscules o minúscules i no hi trobarem diferència, si el sistema operatiu sobre el qual estem treballant suporta *not case sensitive*.

Per exemple, en Windows podem executar indiferentment:

```
1 select * from emp;
```

O bé:

```
1 select * from EMP;
```

El que no acostumem a fer, però, és que dins d'una mateixa sentència ens referim a un mateix objecte en majúscules i en minúscules a la vegada:

```
1 select * from emp where EMP.EMP_NO=7499;
```

Els noms de columnes, índexs, procediments i disparadors (*triggers*), en canvi, sempre són *not case sensitive*.

Els noms dels objectes en MySQL admeten qualsevol tipus de caràcter, excepte / \ i .

De totes maneres, es recomana utilitzar caràcters alfabètics estrictament. Si el nom inclou caràcters especials és obligatori fer-hi referència entre cometes del tipus accent greu (`). Per exemple:

```
1 create table `ES UNA PROVA` (a int);
```

S'admeten també les cometes dobles (") si activem el mode ANSI_QUOTES:

```
1 SET sql_mode='ANSI_QUOTES';
2 create table "ES UNA ALTRA PROVA" (a int);
```

Qualsevol objecte pot ser referit utilitzant les cometes, encara que no calgui, com ara en l'exemple:

```
1 select * from `empresa`.`emp` where `emp`.`emp_no`=7499;
```

Tot i que no és recomanable, es poden anomenar objectes amb paraules reservades del mateix llenguatge com ara SELECT, INSERT, DATABASE, etc. Aquests noms, però, s'hauran de posar obligatòriament entre cometes.

La longitud màxima dels objectes de la base de dades és 64 caràcters, excepte per als àlies, que poden arribar a ser de 256.

Finalment, vegem algunes indicacions per anomenar objectes:

- **Utilitzar noms sencers, descriptius i pronunciables i, si no és factible, bones abreviatures**. En anomenar objectes, sospeseu l'objectiu d'aconseguir noms curts i fàcils d'utilitzar davant l'objectiu de tenir noms que siguin descriptius. En cas de dubte, escolliu el nom més descriptiu, ja que els objectes de la base de dades poden ser utilitzats per molta gent al llarg del temps.
- **Utilitzar regles d'assignació de noms que siguin coherents**. Així, per exemple, una regla podria consistir a començar amb gc_ tots els noms de les taules que formen part d'una gestió comercial.
- **Utilitzar el mateix nom per descriure la mateixa entitat o el mateix atribut en diferents taules**. Així, per exemple, quan un atribut d'una taula

és clau forana d'una altra taula, és molt convenient anomenar-lo amb el nom que té en la taula principal.

2.2 Comentaris en MySQL

El servidor MySQL suporta tres estils de comentaris:

- # fins al final de la línia.
- `-` `<espai en blanc>` fins al final de la línia.
- `/* fins a la propera seqüència */`. Aquests tipus de comentaris admeten diverses línies de comentari.

Exemples dels diferents tipus de comentaris són els següents:

```
SELECT 1+1; # Aquest és el primer tipus de comentari

SELECT 1+1; -- Aquest és el segon tipus de comentari

SELECT 1 /* Aquest és un tipus de comentari que es pot posar enmig de la línia
    */ + 1;

SELECT 1+
/*
Aquest és un
comentari
que es pot posar
en diverses línies*/
1;
```

2.3 Motors d'emmagatzematge en MySQL

MySQL suporta diferents tipus d'emmagatzematge de taules (motors d'emmagatzemament o *storage engines*, en anglès). I quan es crea una taula cal especificar en quin sistema dels possibles el volem crear.

Per defecte, MySQL a partir de la versió 5.5.5 crea les taules de tipus **InnoDB**, que és un sistema transaccional, és a dir, que suporta les característiques que fan que una base de dades pugui garantir que les dades es mantindran consistents.

Les propietats que garanteixen els sistemes transaccionals són les característiques anomenades `ACID`. *ACID* és l'acrònim anglès d'*atomicity, consistency, isolation, durability*:

- **Atomicitat**: es diu que un SGBD garanteix atomicitat si qualsevol transacció o bé finalitza correctament (*commit*), o bé no deixa cap rastre de la seva execució (*rollback*).

- **Consistència**: es parla de consistència quan la concurrència de diferents transaccions no pot produir resultats anòmals.
- **Aïllament (o isolament)**: cada transacció dins del sistema s'ha d'executar com si fos l'única que s'executa en aquell moment.
- **Definitivitat**: si es confirma una transacció, en un SGBD, el resultat d'aquesta ha de ser definitiu i no es pot perdre.

Només el motor **InnoDB** permet crear un sistema transaccional en MySQL. Els altres tipus d'emmagatzemament no són transaccionals i no ofereixen control d'integritat a les bases de dades creades.

Evidentment, aquest sistema (**InnoDB**) d'emmagatzematge és el que sovint interessarà utilitzar per a les bases de dades que creem, però hi pot haver casos en què sigui interessant considerar altres tipus de motors d'emmagatzematge. Per això, MySQL també ofereix altres sistemes com ara, per exemple:

- **MyISAM**: era el sistema per defecte abans de la versió 5.5.5 de MySQL. S'utilitza molt en aplicacions web i en aplicacions de magatzem de dades (*datawarehousing*).
- **Memory**: aquest sistema emmagatzema tot en memòria RAM i, per tant, s'utilitza per a sistemes que requereixin un accés molt ràpid a les dades.
- **Merge**: agrupa taules de tipus MyISAM per optimitzar llistes i cerques. Les taules que cal agrupar han de ser de semblants, és a dir, han de tenir el mateix nombre i tipus de columnes.

Per obtenir una llista dels motors d'emmagatzemament suportats per la versió MySQL que tingueu instal·lada, podeu executar l'ordre `SHOW ENGINES`.

2.4 Creació de taules

La sentència `CREATE TABLE` és la instrucció proporcionada pel llenguatge SQL per a la creació d'una taula.

Recordeu que els elements que es posen entre claudàtors (`[ ]`) són opcionals.

És una sentència que admet múltiples paràmetres, i la sintaxi completa es pot consultar en la documentació de l'SGBD que correspongui, però la sintaxi més simple i usual és aquesta:

```
1 create table [<nom_esquema>.]<nom_taula>
2 ( <nom_columna> <tipus_dada> [default <expressió>][<
      llista_restriccions_pera_a_la_columna>],
3 <nom_columna> <tipus_dada> [default <expressió>][<
      llista_restriccions_per_a_la_columna>],
4 ...
5 [<llista_restriccions_addicionals_per_a_una_o_vàries_columnes>]);
```

Fixem-nos que hi ha força elements que són optatius:

- Les parts obligatòries són el nom de la taula i, per cada columna, el nom i el tipus de dada.
- El nom de l'esquema en què es crea la taula és optatiu i, si no s'indica, la taula s'intenta crear dins l'esquema en què estem connectats.
- Cada columna té permès definir-hi un valor per defecte (opció default) a partir d'una expressió, el qual utilitzarà l'SGBD en les instruccions d'inserció quan no s'especifiqui un valor per a les columnes que tenen definit el valor per defecte. En MySQL el valor per defecte ha de ser constant, no pot ser, per exemple, una funció com ara `NOW()` ni una expressió com ara `CURRENT_DATE`.
- La definició de les restriccions per a una o més columnes també és optativa en el moment de procedir a la creació de la taula.

També és molt usual crear una taula a partir del resultat d'una consulta, amb la sintaxi següent:

```
1 create table [<nom_esquema>.]<nom_taula> [(<noms_dels_camps>]
2 as <sentència_select>;
```

En aquesta sentència, no es defineixen els tipus de camps que es corresponen amb els tipus de les columnes recuperades en la sentència `SELECT`. La definició dels noms dels camps és optativa; si no s'efectua, els noms de les columnes recuperades passen a ser els noms dels camps nous. Caldrà, però, afegir-hi les restriccions que corresponguin. La taula nova conté una còpia de les files resultants de la sentència `SELECT`.

A l'hora de definir taules, cal tenir en compte diversos conceptes:

- Els tipus de dades que l'SGBD possibilita.
- Les restriccions sobre els noms de taules i columnes.
- La integritat de les dades.

En l'apartat "Consultes de selecció simples" de la unitat "Llenguatge SQL. Consultes", es presenten àmpliament els tipus de dades més importants en l'SGBD MySQL.

L'SGBD MySQL proporciona diversos tipus de restriccions (*constraints* en la nomenclatura que s'ha d'utilitzar en els SGBD) o opcions de restricció per facilitar la integritat de les dades. En general, es poden definir en el moment de crear la taula, però també es poden alterar, afegir i eliminar amb posterioritat.

Cada restricció porta associat un nom (únic en tot l'esquema) que es pot especificar en el moment de crear la restricció. Si no s'especifica, l'SGBD n'assigna un per defecte.

Vegem, a continuació, els diferents tipus de restriccions:

Clau primària

Per definir la clau primària d'una taula, cal utilitzar la *constraint* `primary key`.

Si la clau primària és formada per una única columna, es pot especificar en la línia de definició de la columna corresponent, amb la sintaxi següent:

```
<columna> <tipus_dada> primary key
```

En canvi, si la clau primària és formada per més d'una columna, s'ha d'especificar obligatòriament en la zona final de restriccions sobre columnes de la taula, amb la sintaxi següent:

```
[constraint <nom_restricció>] primary key (col1,col2,...)
```

Les claus primàries que afecten una única columna també es poden especificar amb aquest segon procediment.

Obligatorietat de valor

Per definir l'obligatorietat de valor en una columna, cal utilitzar l'opció **not null**.

Aquesta restricció es pot indicar en la definició de la columna corresponent amb aquesta sintaxi:

```
<columna> <tipus_dada> [not null]
```

Per descomptat, no cal definir aquesta restricció sobre columnes que formen part de la clau primària, ja que formar part de la clau primària implica, automàticament, la impossibilitat de tenir valor nuls.

Unicitat de valor

Per definir la unicitat de valor en una columna, cal utilitzar la *constraint* **unique**.

Si la unicitat s'especifica per a una única columna, es pot assignar en la línia de definició de la columna corresponent, amb la sintaxi següent:

```
<columna> <tipus_dada> unique
```

En canvi, si la unicitat s'aplica sobre diverses columnes simultàniament, cal especificar-la obligatòriament en la zona final de restriccions sobre columnes de la taula, amb la sintaxi següent:

```
[constraint <nom_restricció>] unique (col1, col2...)
```

Aquest segon procediment també es pot emprar per aplicar la unicitat a una única columna.

Per descomptat, no cal definir aquesta restricció sobre un conjunt de columnes que formen part de la clau primària, ja que la clau primària implica, automàticament, la unicitat de valors.

Condicions de comprovació

Per definir condicions de comprovació en una columna, cal utilitzar l'opció `check (<condició>)`.

Aquesta restricció es pot indicar en la definició de la columna corresponent:

```
<columna> <tipus_dada> check (<condició>)
```

També es pot indicar en la zona final de restriccions sobre columnes de la taula, amb la sintaxi següent:

```
check (<condició>)
```

AUTO_INCREMENT

Podem definir les columnes numèriques amb l'opció `AUTO_INCREMENT`. Aquesta modificació permet que en inserir un valor null o no inserir valor explícitament en aquella columna definida d'aquesta manera, s'hi afegeixi un valor per defecte consistent en el més gran ja introduït incrementat en una unitat.

```
<columna> <tipus_dada> AUTO_INCREMENT
```

Comentaris de columnes

Podem afegir comentaris a les columnes de manera que puguin quedar guardats a la base de dades i ser consultats. La manera de fer-ho és afegir la paraula `COMMENT` seguida del text que calgui posar com a comentari entre cometes simples.

```
<columna> <tipus_dada> COMMENT 'comentari'
```

Integritat referencial

Per definir la integritat referencial, cal utilitzar la *constraint* `foreign key`.

Si la clau forana és formada per una única columna, es pot especificar en la línia de definició de la columna corresponent, amb la sintaxi següent:

```
<columna> <tipus_dada> [constraint <nom_restricció>] references <taula> [(
    columna)]
```

En canvi, si la clau forana és formada per més d'una columna, cal especificar-la obligatòriament en la zona final de restriccions sobre columnes de la taula, amb la sintaxi següent:

```
[constraint <nom_restricció>] foreign key (col1, col2...)
references <taula> [(col1, col2...)]
```

Les claus foranes que afecten una única columna també es poden especificar amb aquest segon procediment.

En qualsevol dels dos casos, es fa referència a la taula principal de la qual estem definint la clau forana, la qual cosa es fa amb l'opció `references <taula>`.

> En MySQL la integritat referencial només s'activa si es treballa sobre el motor **InnoDB** i s'utilitza la sintaxi de *constraint* de la zona de definició de restriccions del final i, a més, es defineix un índex per a les columnes implicades en la clau forana.

La sintaxi per a la integritat referencial activa en MySQL és la següent (si s'utilitzen altres sintaxis, l'SGBD les reconeix però no les valida):

```
index [<nom_index>] (col1, col2...)
[constraint <nom_restricció>] foreign key (col1, col2...) references <taula> (
    col1, col2...)
```

La sintaxi que hem presentat per tal de definir la integritat referencial no és completa. Ens falta tractar un tema fonamental: l'actuació que esperem de l'SGBD davant possibles eliminacions i actualitzacions de dades en la taula principal, quan hi ha files en altres taules que hi fan referència.

La *constraint* **foreign key** es pot definir acompanyada dels apartats següents:

- `on delete <acció>`, que defineix l'actuació automàtica de l'SGBD sobre les files de la nostra taula que es veuen afectades per una eliminació de les files a les quals fan referència.
- `on update <acció>`, que defineix l'actuació automàtica de l'SGBD sobre les files de la nostra taula que es veuen afectades per una actualització del valor al qual fan referència.

Per si no us ha quedat clar, pensem en les taules DEPT i EMP de l'esquema *empresa*. La taula EMP conté la columna dept_no, que és clau forana de la taula DEPT. Per tant, en la definició de la taula EMP hem de tenir definida una *constraint foreign key* en la columna dept_no fent referència a la taula DEPT. En definir

aquesta restricció de clau forana, el dissenyador de la base de dades va haver de prendre decisions respecte al següent:

- Com ha d'actuar l'SGBD davant l'intent d'eliminació d'un departament en la taula DEPT si hi ha files en la taula EMP que hi fan referència? Això es defineix en l'apartat `on delete <acció>`.
- Com ha d'actuar l'SGBD davant l'intent de modificació del codi d'un departament en la taula DEPT si hi ha files en la taula EMP que hi fan referència? Això es defineix en l'apartat `on update <acció>`.

En general, els SGBD ofereixen diverses possibilitats d'acció, però no sempre són les mateixes. Abans de conèixer aquestes possibilitats, també ens cal saber que alguns SGBD permeten diferir la comprovació de les restriccions de clau forana fins a la finalització de la transacció, en lloc d'efectuar la comprovació -i actuar en conseqüència- després de cada instrucció. Quan això és factible, la definició de la *constraint* va acompanyada del mot `deferrable` o `not deferrable`. L'actuació per defecte acostuma a ser no diferir la comprovació.

Per tant, la sintaxi de la restricció de clau forana es veu clarament ampliada. Si s'efectua en el moment de definir la columna, tenim el següent:

```
1 [constraint <nom_restricció>] foreign key (col1, col2, ...) references <taula>
      (col1,
2 col2, ...) [on delete <acció>] [on update <acció>]
```

Les opcions que ens podem arribar a trobar en referència a l'acció que acompanyi els apartats **on update** i **on delete** són aquestes: RESTRICT | CASCADE | SET NULL | NO ACTION

- `NO ACTION` o `RESTRICT`: són sinònims. És l'opció per defecte i no permet l'eliminació o actualització de dades en la taula principal.
- `CASCADE`: quan s'actualitza o elimina la fila pare, les files relacionades (filles) també s'actualitzen o eliminen automàticament.
- `SET NULL`: quan s'actualitza o elimina la fila pare, les files relacionades (filles) s'actualitzen a NULL. Cal haver-les definit de manera que admetin valors nuls, és clar.
- `SET DEFAULT`: quan s'actualitza o elimina la fila pare, les files relacionades (filles) s'actualitzen al valor per defecte. MySQL suporta la sintaxi, però no actua, davant d'aquesta opció.

L'opció `CASCADE` és molt perillosa en utilitzar-la amb `on delete`. Pensem què passaria, en l'esquema *empresa*, si algú decidís eliminar un departament de la taula DEPT i la clau forana en la taula EMP fos definida amb `on delete cascade`: tots els empleats del departament serien, immediatament, eliminats de la taula EMP.

De vegades, però, és molt útil acompanyant `on delete`. Pensem en la relació d'integritat entre les taules COMANDA i DETALL de l'esquema *empresa*. La

taula DETALL conté la columna com_num, que és clau forana de la taula COMANDA. En aquest cas, pot tenir molt de sentit tenir definida la clau forana amb `on delete cascade`, ja que l'eliminació d'una ordre provocarà l'eliminació automàtica de les seves línies de detall.

A diferència de la caució en la utilització de l'opció `cascade` per a les actuacions `on delete`, s'acostuma a utilitzar molt per a les actuacions `on update`.

La taula 2.1 mostra les opcions proporcionades per alguns SGBD actuals.

TAULA 2.1. Opcions de la restricció foreign key proporcionades per alguns SGBD actuals

SGBD	on update	on delete	Diferir actuació	no action	restrict	cascade	set null	set default
Oracle	No	Sí	No	Sí	No	Sí	Sí	No
MySQL	Sí	Sí	No	Sí	Sí	Sí	Sí	Sí
PostgreSQL	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
SQLServer 2005	Sí	Sí	No	Sí	No	Sí	Sí	Sí
MS-Access 2003	Sí	Sí	No	Sí	No	Sí	3	No

Exemple 1 de creació de taules. Taules de l'esquema empresa

```
CREATE TABLE IF NOT EXISTS empresa.DEPT (
 DEPT_NO  TINYINT (2) UNSIGNED,
 DNOM     VARCHAR(14) NOT NULL UNIQUE,
 LOC      VARCHAR(14),
 PRIMARY KEY (DEPT_NO) ) ;


CREATE TABLE IF NOT EXISTS empresa.EMP (
 EMP_NO    SMALLINT (4) UNSIGNED,
 COGNOM    VARCHAR (10) NOT NULL,
 OFICI     VARCHAR (10),
 CAP       SMALLINT (4) UNSIGNED,
 DATA_ALTA DATE,
 SALARI    INT UNSIGNED,
 COMISSIO  INT UNSIGNED,
 DEPT_NO   TINYINT (2) UNSIGNED NOT NULL,
 PRIMARY KEY (EMP_NO),
 INDEX IDX_EMP_CAP (CAP),
 INDEX IDX_EMP_DEPT_NO (DEPT_NO),
 FOREIGN KEY (DEPT_NO) REFERENCES empresa.DEPT(DEPT_NO)) ;
```

En primer lloc, cal destacar l'opció `IF NOT EXISTS`, opció molt utilitzada a l'hora de crear bases de dades per tal d'evitar errors en cas que la taula ja existeixi prèviament.

D'altra banda, cal destacar que la taula es defineix amb el nom de l'esquema `empresa`. Això no és necessari si prèviament es defineix aquest esquema com a esquema per defecte. Això es pot fer amb la sentència `USE`:

```
USE empresa;
```

En les dues definicions de taula, la clau primària s'ha definit en la part inferior de la sentència, no pas en la mateixa definició de la columna, malgrat que es tractava de claus primàries que només tenien una única columna.

Fixeu-vos com s'han definit dos índexs per a les columnes `CAP` i `DEPT_NO` per tal de crear *a posteriori* les claus foranes corresponents. En el moment de la creació es crea la clau

forana que fa referència de EMP a DEPT. No es crea, en canvi, la que fa referència de EMP a la mateixa taula i que serveix per referir-se al cap d'un empleat. El motiu és que si es definís aquesta clau forana en el moment de la creació de la taula, no hi podríem inserir valors fàcilment, ja que no tindria el valor de referència prèviament introduït a la taula. Per exemple, si hi volem inserir l'empleat número (EMP_NO) 7369 que té com a empleat cap el 7902 i aquest no és encara a la taula, no el podríem inserir perquè no existeix el 7902 encara a la taula. Una possible solució a aquest problema que s'anomena *interbloqueig* o *deadlock*, en anglès, és definir la clau forana *a posteriori* de les insercions. O bé, si l'SGBD ho permet, desactivar la **foreign key** abans d'inserir-la i tornar-la a activar en acabar. Així, doncs, després de les insercions caldrà modificar la taula i afegir-hi la restricció de clau forana.

Observeu, també, que s'ha utilitzat el modificador de tipus UNSIGNED per tal de definir els camps que necessàriament són considerats positius. De manera que si es fa una inserció del tipus següent, l'SGBD ens retornarà un error:

```
insert into EMP values (100, 'Rodríguez', 'Venedor', NULL,
     sysdate, -5000, NULL, 10)
```

Fixem-nos, també, en la importància de l'ordre en què es defineixen les taules, ja que no seria possible definir la integritat referencial en la columna dept_no de la taula EMP sobre la taula DEPT si aquesta encara no fos creada.

```
CREATE TABLE IF NOT EXISTS empresa.CLIENT (
 CLIENT_COD          INT(6) UNSIGNED PRIMARY KEY,
 NOM                 VARCHAR (45) NOT NULL,
 ADREÇA              VARCHAR (40) NOT NULL,
 CIUTAT              VARCHAR (30) NOT NULL,
 ESTAT               VARCHAR (2),
 CODI_POSTAL         VARCHAR (9) NOT NULL,
 AREA                SMALLINT(3),
 TELEFON             VARCHAR (9),
 REPR_COD            SMALLINT(4) UNSIGNED,
 LIMIT_CREDIT        DECIMAL(9,2) UNSIGNED,
 OBSERVACIONS        TEXT,
 INDEX IDX_CLIENT_REPR_COD (REPR_COD),
 FOREIGN KEY (REPR_COD) REFERENCES empresa.EMP(EMP_NO));
```

En aquesta taula, en canvi, la clau primària s'ha definit en la mateixa definició de la columna.

```
CREATE TABLE IF NOT EXISTS empresa.PRODUCTE (
 PROD_NUM     INT (6) UNSIGNED PRIMARY KEY,
 DESCRIPCIO   VARCHAR (30) NOT NULL  UNIQUE);

CREATE TABLE IF NOT EXISTS empresa.COMANDA(
 COM_NUM             SMALLINT(4) UNSIGNED PRIMARY KEY,
 COM_DATA            DATE,
 COM_TIPUS           CHAR (1) CHECK (COM_TIPUS IN ('A','B','C')),
 CLIENT_COD          INT (6) UNSIGNED NOT NULL,
 DATA_TRAMESA        DATE,
 TOTAL               DECIMAL(8,2) UNSIGNED,
 INDEX IDX_COMANDA_CLIENT_COD (CLIENT_COD),
 FOREIGN KEY (CLIENT_COD) REFERENCES empresa.CLIENT(CLIENT_COD) );


CREATE TABLE IF NOT EXISTS empresa.DETALL  (
 COM_NUM             SMALLINT(4) UNSIGNED,
 DETALL_NUM          SMALLINT(4) UNSIGNED,
 PROD_NUM            INT(6) UNSIGNED NOT NULL,
 PREU_VENDA          DECIMAL(8,2) UNSIGNED,
 QUANTITAT           INT (8),
 IMPORT              DECIMAL(8,2),
 CONSTRAINT DETALL_PK PRIMARY KEY (COM_NUM,DETALL_NUM),
 INDEX IDX_DETAL_COM_NUM (COM_NUM),
 INDEX IDX_PROD_NUM (PROD_NUM),
 FOREIGN KEY (COM_NUM) REFERENCES empresa.COMANDA(COM_NUM),
 FOREIGN KEY (PROD_NUM) REFERENCES empresa.PRODUCTE(PROD_NUM));
```

Exemple 2 de creació de taules. Taules de l'esquema sanitat

```
CREATE TABLE IF NOT EXISTS sanitat.HOSPITAL (
 HOSPITAL_COD   TINYINT (2) PRIMARY KEY,
 NOM            VARCHAR(10) NOT NULL,
 ADREÇA         VARCHAR(20),
 TELEFON        VARCHAR(8),
 QTAT_LLITS     SMALLINT(3) UNSIGNED DEFAULT 0 );



CREATE TABLE IF NOT EXISTS sanitat.SALA (
 HOSPITAL_COD  TINYINT (2),
 SALA_COD      TINYINT (2),
 NOM           VARCHAR(20) NOT NULL,
 QTAT_LLITS    SMALLINT(3) UNSIGNED DEFAULT 0,
 CONSTRAINT SALA_PK PRIMARY KEY (HOSPITAL_COD, SALA_COD),
 INDEX IDX_SALA_HOSPITAL_COD (HOSPITAL_COD),
 FOREIGN KEY (HOSPITAL_COD) REFERENCES sanitat.HOSPITAL(
     HOSPITAL_COD)) ;
```

Recordem que la taula es defineix amb el nom de l'esquema `sanitat`, però que això no és necessari si prèviament es defineix aquest esquema com a esquema per defecte:

```
USE sanitat;
```

La definició de la taula `SALA` necessita declarar la constraint `PRIMARY KEY` al final de la definició de la taula, ja que és formada per més d'un camp. En casos com aquest, aquesta és l'única opció i no és factible definir la constraint `PRIMARY KEY` al costat de cada columna, ja que una taula només admet una definició de constraint `PRIMARY KEY`.

```
CREATE TABLE IF NOT EXISTS sanitat.PLANTILLA (
 HOSPITAL_COD  TINYINT (2),
 SALA_COD      TINYINT (2),
 EMPLEAT_NO    SMALLINT(4) NOT NULL,
 COGNOM        VARCHAR (15) NOT NULL,
 FUNCIO        VARCHAR (10),
 TORN          VARCHAR (1) CHECK (TORN IN ('M','T','N')),
 SALARI        INT (10),
 CONSTRAINT PLANTILLA_PK PRIMARY KEY (HOSPITAL_COD, SALA_COD,
     EMPLEAT_NO),
 INDEX IDX_PLANTILLA_HOSP_SALA (HOSPITAL_COD, SALA_COD),
 FOREIGN KEY (HOSPITAL_COD, SALA_COD) REFERENCES sanitat.SALA (
     HOSPITAL_COD, SALA_COD)) ;
```

La definició de la taula `PLANTILLA` necessita declarar les restriccions `PRIMARY KEY` i `FOREIGN KEY` al final de la definició de la taula perquè ambdues fan referència a una combinació de columnes.

```
CREATE TABLE IF NOT EXISTS sanitat.MALALT (
 INSCRIPCIO    INT (5) PRIMARY KEY,
 COGNOM        VARCHAR (15) NOT NULL,
 ADREÇA        VARCHAR (20),
 DATA_NAIX     DATE,
 SEXE          CHAR (1) NOT NULL  CHECK (SEXE = 'H' OR SEXE = 'D'),
 NSS           CHAR(9)) ;

CREATE TABLE IF NOT EXISTS sanitat.INGRESSOS (
 INSCRIPCIO    INT (5) PRIMARY KEY,
 HOSPITAL_COD  TINYINT (2) NOT NULL,
 SALA_COD      TINYINT (2) NOT NULL,
 LLIT          SMALLINT(4) UNSIGNED,
 INDEX IDX_INGRESSOS_INSCRIPCIO (INSCRIPCIO),
 INDEX IDX_INGRESSOS_HOSP_SALA (HOSPITAL_COD, SALA_COD),
 FOREIGN KEY (INSCRIPCIO) REFERENCES sanitat.MALALT(INSCRIPCIO),
 FOREIGN KEY (HOSPITAL_COD, SALA_COD) REFERENCES sanitat.SALA (HOSPITAL_COD,
     SALA_COD));

```

```
19
20 CREATE TABLE IF NOT EXISTS sanitat.DOCTOR (
21  HOSPITAL_COD  TINYINT (2),
22  DOCTOR_NO     SMALLINT(3),
23  COGNOM        VARCHAR(13) NOT NULL,
24  ESPECIALITAT  VARCHAR(16) NOT NULL,
25  CONSTRAINT DOCTOR_PK PRIMARY KEY (HOSPITAL_COD, DOCTOR_NO),
26  INDEX IDX_DOCTOR_HOSP (HOSPITAL_COD),
27  FOREIGN KEY (HOSPITAL_COD) REFERENCES sanitat.HOSPITAL(HOSPITAL_COD)) ;
```

2.5 Eliminació de taules

La sentència `DROP TABLE` és la instrucció proporcionada pel llenguatge SQL per a l'eliminació (dades i definició) d'una taula.

La sintaxi de la sentència `DROP TABLE` és aquesta:

```
1 drop table [<nom_esquema>.]<nom_taula> [if exists];
```

L'opció `if exists` es pot especificar per tal d'evitar un error en cas que la taula no existeixi.

També es poden afegir les opcions `cascade` o `restrict` que en alguns SGBD fan que s'eliminin totes les definicions de restriccions d'altres taules que fan referència a la taula que es vol eliminar abans de fer-ho, o que s'impedeixi l'eliminació, respectivament. Sense l'opció `cascade` la taula que és referenciada per altres taules (a nivell de definició, independentment que hi hagi o no, en un moment determinat, files referenciades), l'SGBDR no l'elimina.

En MySQL, però, no es tenen efecte les opcions `cascade` o `restrict` i sempre cal eliminar les taules referides per tal de poder eliminar la taula referenciada.

Exemple d'eliminació de taules

Suposem que volem eliminar la taula DEPT de l'esquema *empresa*.

L'execució de la sentència següent és errònia:

```
1 drop table dept;
```

L'SGBD informa que hi ha taules que hi fan referència i que, per tant, no es pot eliminar. I és lògic, ja que la taula `DEPT` està referenciada per la taula `EMP`.

Si de veritat es vol aconseguir eliminar la taula `DEPT` i provocar que totes les taules que hi fan referència eliminin la definició corresponent de clau forana, caldrà eliminar EMP prèviament. I, abans que aquesta, les altres taules que fan referència a aquesta altra. De forma que l'ordre per eliminar les taules sol ser l'ordre invers en què les hem creades.

```
1 use empresa;
2 drop table detall;
```

```
drop table comanda;
drop table producte;
drop table client;
drop table emp;
drop table dept;
```

2.6 Modificació de l'estructura de les taules

De vegades, cal fer modificacions en l'estructura de les taules (afegir-hi o eliminar-ne columnes, afegir-hi o eliminar-ne restriccions, modificar els tipus de dades...).

> La sentència `ALTER TABLE` és la instrucció proporcionada pel llenguatge SQL per modificar l'estructura d'una taula.

La seva sintaxi és aquesta:

```
alter [IGNORE] table [<nom_esquema>.]<nom_taula>
<clàusules_de_modificació_de_taula>;
```

És a dir, una sentència `alter table` pot contenir diferents clàusules (com a mínim una) que modifiquin l'estructura de la taula. Hi ha clàusules de modificació de taula que poden anar acompanyades, en una mateixa sentència `alter table`, per altres clàusules de modificació, mentre que n'hi ha que han d'anar soles.

Cal tenir present que, per efectuar una modificació, l'SGBD no hauria de trobar cap incongruència entre la modificació que s'ha d'efectuar i les dades que ja hi ha en la taula. No tots els SGBD actuen de la mateixa manera davant aquestes situacions.

Així, l'SGBD MySQL, per defecte, no permet especificar la restricció d'obligatorietat (not null) a una columna que ja conté valors nuls (cosa lògica, no?) ni tampoc disminuir l'amplada d'una columna de tipus `varchar` a una amplada inferior a l'amplada màxima dels valors continguts en la columna.

En canvi, però, si s'activa l'opció `ignore`, la modificació especificada s'intenta fer, encara que calgui truncar o modificar dades de la taula ja existent. Per exemple, si intentem modificar la característica `not null` de la columna telèfon de la taula hospital, de l'esquema sanitat, atès que hi ha un valor nul, la sentència següent no s'executarà:

```
alter table hospital
  modify telefon varchar(8) not null;
```

I el resultat de l'execució d'aquesta modificació serà un missatge tipus `Error: Data truncated for column telefon at row 5`.

En canvi, si utilitzem l'opció `ignore` podem executar la sentència següent que ens permetrà modificar l'opció `not null` de telèfon de manera que posarà un *string*

buit en lloc de valor nul en les columnes que no compleixin la condició:

```
alter ignore table hospital
  modify telefon varchar(8) not null;
```

De manera similar, si volem disminuir la mida de la columna adreça de la taula hospital:

```
alter table hospital
  modiy adreca varchar(7);
```

Aquest codi mostrarà un error similar a `Error: Data truncated for column adreca at row 1` i no s'executarà la sentència. En canvi, si hi afegim l'opció `ignore`, el resultat serà la modificació de l'estructura de la taula i el truncament dels valors de les columnes afectades.

```
alter ignore table hospital
  modiy adreca varchar(7);
```

Vegem, a continuació, les diferents possibilitats d'alteració de taula, tenint en compte que en MySQL s'admeten diversos tipus d'alteracions en una mateixa clàusula d'`alter table`, separades per coma:

1. Per afegir una columna

```
ADD [COLUMN] nom_columna definició_columna [FIRST | ALFTER nom_columna ]
```

O bé, si cal definir-ne unes quantes de noves:

```
ADD [COLUMN] (nom_columna definició_columna,...)
```

2. Per eliminar una columna

```
DROP [COLUMN] <nom_columna>
```

3. Per modificar l'estructura d'una columna

```
MODIFY [COLUMN] nom_columna definició_columna [FIRST | AFTER col_name]
```

O bé:

```
CHANGE [COLUMN] nom_columna_antic nom_columna_nou definició_columna
        [FIRST|AFTER nom_columna]
```

4. Per afegir restriccions

```
ADD [CONSTRAINT <nom_restricció>] <restricció>
```

Concretament, les restriccions que es poden afegir en MySQL són les següents:

```
ADD [CONSTRAINT [símbol]] PRIMARY KEY [tipus_index] (nom_columna_index,...) [
    opcions_index] ...

```

```
ADD [CONSTRAINT [símbol]] UNIQUE [INDEX|KEY] [nom_index] [tipus_index] (
    nom_columna_index,...) [opcions_index] ...

ADD [CONSTRAINT [símbol]] FOREIGN KEY [nom] (nom_columna1,...) REFERENCES taula
     (columna1, ....)
```

5. Per eliminar restriccions

```
DROP PRIMARY KEY

DROP {INDEX|KEY} nom_index

DROP FOREIGN KEY nom
```

6. Per afegir índexs

```
ADD {INDEX|KEY} [nom_index]
        [tipus_index] (nom_columna,...) [opcions_index] ...
```

7. Per habilitar o deshabilitar els índexs

```
DISABLE KEYS

ENABLE KEYS
```

8. Per reanomenar una taula

```
RENAME [TO] nom_nou_taula
```

9. Per reordenar les files d'una taula

```
ORDER BY nom_columna1 [, nom_columna2] ...
```

10. Per canviar o eliminar el valor per defecte d'una columna

```
ALTER [COLUMN] nom_columna {SET DEFAULT literal | DROP DEFAULT}
```

Exemple 1 de modificació de l'estructura d'una taula

Recordem l'estructura de la taula DEPT de l'esquema *empresa*:

```
SQL> desc DEPT;
Name          Null     Type
_____________ ________ _____________
DEPT_NO       NOT NULL TINYINT(2)
DNOM          NOT NULL VARCHAR(14)
LOC                    VARCHAR(14)
```

Es vol modificar l'estructura de la taula DEPT de l'esquema empresa de manera que passi el següent:

- La columna loc passi a ser obligatòria.
- Afegim una columna numèrica de nom numEmps destinada a contenir el nombre d'empleats del departament.
- Eliminem l'obligatorietat de la columna nom.
- Ampliem l'amplada de la columna dnom a vint caràcters.

Ho podem aconseguir fent el següent:

```
alter table dept
modify loc varchar(14) not null,
add numEmps number(2) unsigned,
modify dnom varchar(20);
```

Exemple 2 de modificació de l'estructura d'una taula, per problemes de deadlock

Per tal de crear l'estructura de les taules `DEPT` i `EMP` de l'empresa es creen les taules següents i s'hi afegeixen les files amb els valors introduint les sentències següents:

```
CREATE TABLE IF NOT EXISTS empresa.DEPT (
 DEPT_NO  TINYINT (2) UNSIGNED,
 DNOM     VARCHAR(14) NOT NULL UNIQUE,
 LOC      VARCHAR(14),
 PRIMARY KEY (DEPT_NO) ) ;


INSERT INTO empresa.DEPT VALUES (10, 'COMPTABILITAT', 'SEVILLA');
INSERT INTO empresa.DEPT VALUES (20, 'INVESTIGACIÓ', 'MADRID');
INSERT INTO empresa.DEPT VALUES (30, 'VENDES', 'BARCELONA');
INSERT INTO empresa.DEPT VALUES (40, 'PRODUCCIÓ', 'BILBAO');


CREATE TABLE IF NOT EXISTS empresa.EMP (
 EMP_NO    SMALLINT (4) UNSIGNED,
 COGNOM    VARCHAR (10) NOT NULL,
 OFICI     VARCHAR (10),
 CAP       SMALLINT (4) UNSIGNED,
 DATA_ALTA DATE,
 SALARI    INT UNSIGNED,
 COMISSIO  INT UNSIGNED,
 DEPT_NO   TINYINT (2) UNSIGNED NOT NULL,
 PRIMARY KEY (EMP_NO),
 INDEX IDX_EMP_CAP (CAP),
 INDEX IDX_EMP_DEPT_NO (DEPT_NO),
 FOREIGN KEY (DEPT_NO) REFERENCES empresa.DEPT(DEPT_NO)) ;


INSERT INTO empresa.EMP VALUES (7369,'SÁNCHEZ','EMPLEAT',7902, '1980-12-17',
    104000,NULL,20);
INSERT INTO empresa.EMP VALUES (7499,'ARROYO','VENEDOR',7698, '1980-02-20',
    208000,39000,30);
INSERT INTO empresa.EMP VALUES (7521,'SALA','VENEDOR',7698, '1981-02-22',
    162500,65000,30);
INSERT INTO empresa.EMP VALUES (7566,'JIMÉNEZ','DIRECTOR',7839, '1981-04-02',
    386750,NULL,20);
INSERT INTO empresa.EMP VALUES (7654,'MARTÍN','VENEDOR',7698, '1981-09-29',
    162500,182000,30);
INSERT INTO empresa.EMP VALUES (7698,'NEGRO','DIRECTOR',7839, '1981-05-01',
    370500,NULL,30);
INSERT INTO empresa.EMP VALUES (7782,'CEREZO','DIRECTOR',7839, '1981-06-09',
    318500,NULL,10);
INSERT INTO empresa.EMP VALUES (7788,'GIL','ANALISTA',7566, '1981-11-09',
    390000,NULL,20);
INSERT INTO empresa.EMP VALUES (7839,'REY','PRESIDENT',NULL, '1981-11-17',
    650000,NULL,10);
INSERT INTO empresa.EMP VALUES (7844,'TOVAR','VENEDOR',7698, '1981-09-08',
    195000,0,30);
INSERT INTO empresa.EMP VALUES (7876,'ALONSO','EMPLEAT',7788, '1981-09-23',
    143000,NULL,20);
INSERT INTO empresa.EMP VALUES (7900,'JIMENO','EMPLEAT',7698, '1981-12-03',
    123500,NULL,30);
INSERT INTO empresa.EMP VALUES (7902,'FERNÁNDEZ','ANALISTA',7566, '1981-12-03',
     390000,NULL,20);
INSERT INTO empresa.EMP VALUES (7934,'MUÑOZ','EMPLEAT',7782, '1982-01-23',
    169000,NULL,10);
```

Per tal d'afegir la restricció que la columna cap fa referència a un empleat, de la mateixa taula, cal afegir-hi la restricció següent:

```
ALTER TABLE empresa.EMP
ADD FOREIGN KEY (CAP) REFERENCES EMP(EMP_NO);
```

Fixeu-vos que no es podria haver definit aquesta restricció abans d'inserir els valors en les files perquè ja la primera fila inserida ja no compliria la restricció que el codi del seu cap fos prèviament inserit en la taula.

2.7 Índexs per a taules

Índex tipus B-tree i hash

Els índexs B-tree són una organització de les dades en forma d'arbre, de manera que buscar un valor d'una dada resulti més ràpid que buscar-la dins d'una estructura lineal en què s'hagi de buscar des de l'inici fins al final passant per tots els valors.

Els índexs tipus hash tenen com a objectiu accedir directament a un valor concret mitjançant una funció anomenada funció de hash. Per tant, buscar un valor és molt ràpid.

Els SGBD utilitzen índexs per accedir de manera més ràpida a les dades. Quan cal accedir a un valor d'una columna en què no hi ha definit cap índex, l'SGBD ha de consultar tots els valors de totes les columnes des de la primera fins a l'última. Això resulta molt costós en temps i, com més files té la taula en qüestió, més lenta és l'operació. En canvi, si tenim definit un índex en la columna de cerca, l'operació d'accedir a un valor concret resulta molt més ràpid, perquè no cal accedir a tots els valors de totes les files per trobar el que es busca.

MySQL utilitza índexs per facilitar l'accés a columnes que són `PRIMARY KEY` o `UNIQUE` i sol emmagatzemar els índexs utilitzant el tipus d'índex B-tree. Per a les taules emmagatzemades en MEMORY s'utilitzen, però, índexs de tipus HASH.

És lògic crear índexs per facilitar l'accés per a les columnes que necessitin accessos ràpids o molt freqüents. L'administrador de l'SGBD té, entre les seves tasques, avaluar els accessos que s'efectuen a la base de dades i decidir, si escau, l'establiment d'índexs nous. Però també és tasca de l'analista i/o dissenyador de la base de dades dissenyar els índexs adequats per a les diferents taules, ja que és la persona que ha ideat la taula pensant en les necessitats de gestió que tindran els usuaris.

La sentència CREATE INDEX és la instrucció proporcionada pel llenguatge SQL per a la creació d'índexs.

La seva sintaxi simple és aquesta:

```
create index [<nom_esquema>.]<nom_índex>
on <nom_taula> (col1 [asc|desc], col2 [asc|desc], ...);
```

Tot i que la creació d'un índex té associades moltes opcions, que en MySQL poden ser les següents:

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX nom_index
  [USING {BTREE | HASH}
  ON nom_taula (columna1 [longitud][asc|desc], ....)
  [opcions_administració];
```

En MySQL podem definir índexs que mantinguin valors no repetits, especificant la clàusula `UNIQUE`. També podem indicar que indexin tenint en compte el camp sencer d'una columna de tipus `TEXT`, si utilitzem un motor d'emmagatzemament de tipus MyISAM. MySQL suporta índexs sobre els tipus de dades geomètriques que suporta (`SPATIAL`).

Les opcions `USING BTREE` i `USING HASH` permeten forçar la creació d'un índex d'un tipus o un altre (índex tipus B-tree o tipus hash).

La modificació `ASC` o `DESC` sobre cada columna, però, és suportada sintàcticament donant suport a l'estàndard SQL però no té efecte: tots els índexs en MySQL són ascendents.

La sentència `DROP INDEX` és la instrucció proporcionada pel llenguatge SQL per a l'eliminació d'índexs.

La seva sintaxi és aquesta:

```
drop index [<nom_esquema>.]<nom_índex> on <nom_taula>;
```

Exemple 1 de creació d'índexs. Taules de l'esquema empresa

El dissenyador de les taules de l'esquema *empresa* va considerar oportú crear els índexs següents:

```
-- Per tenir els empleats indexats pel cognom:

create index EMP_COGNOM on EMP (COGNOM);

-- Per tenir els empleats indexats pel departament al qual estan assignats:

create index EMP_DEPT_NO_EMP on EMP (DEPT_NO,EMP_NO);

-- Per tenir els clients indexats pel nom:

create index CLIENT_NOM on CLIENT (NOM);

-- Per tenir els clients indexats pel representant (+ codi de client):

create index CLIENT_REPR_CLI on CLIENT (REPR_COD, CLIENT_COD);

-- Per tenir les comandes indexades per la seva data (+ número de comanda):

create index COMANDA_DATA_NUM on COMANDA (COM_DATA, COM_NUM);

-- Per tenir les comandes indexades per la data de tramesa (+ número de comanda
    ):

create index COMANDA_DATA_TRAMESA on COMANDA (DATA_TRAMESA);

-- Per tenir les línies de detall indexades per producte (+ comanda + número de
     línia):

create index DETALL_PROD_COM_DET on DETALL (PROD_NUM,COM_NUM,DETALL_NUM);
```

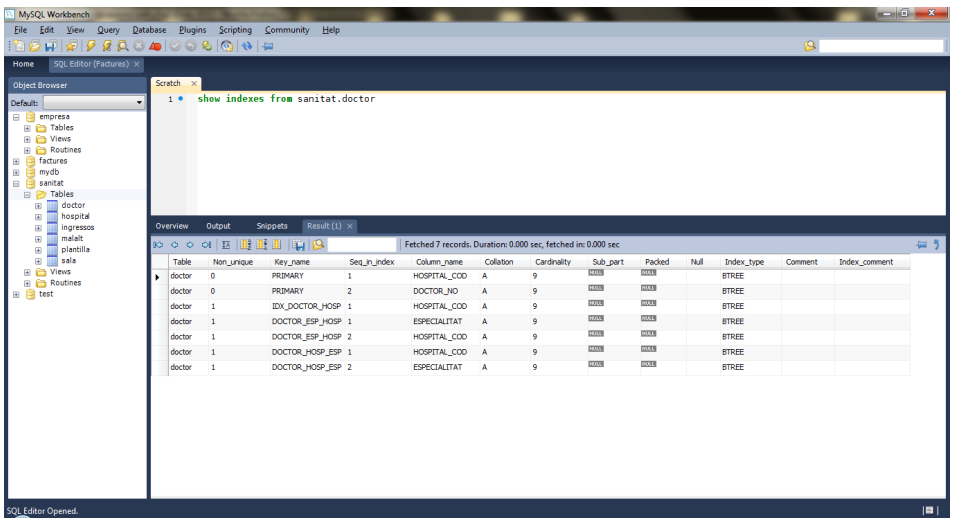
Tots aquests índexs s'afegeixen als existents a causa de les restriccions de clau primària i d'unicitat.

Exemple 2 de creació d'índexs. Taules de l'esquema sanitat

El dissenyador de les taules de l'esquema *sanitat* va considerar oportú crear els índexs següents:

```
-- Per tenir els hospitals indexats pel nom:

CREATE INDEX HOSPITAL_NOM ON HOSPITAL (NOM);

-- Per tenir les sales indexades pel nom dins cada hospital:

CREATE INDEX SALA_HOSP_NOM ON SALA (HOSPITAL_COD, NOM);

-- Per tenir la plantilla indexada per cognom dins cada hospital:

CREATE INDEX PLANTILLA_HOSP_COGNOM ON PLANTILLA (HOSPITAL_COD, COGNOM);

-- Per tenir la plantilla indexada per la funció dins cada hospital:

CREATE INDEX PLANTILLA_HOSP_FUNCIO ON PLANTILLA (HOSPITAL_COD, FUNCIO);

-- Per tenir la plantilla indexada per la funció (entre tots els hospitals
    sales):

CREATE INDEX PLANTILLA_FUNCIO_HOSP_SALA ON PLANTILLA (FUNCIO, HOSPITAL_COD,
    SALA_COD);

-- Per tenir els malalts indexats per data de naixement i cognom:

CREATE INDEX MALALT_NAIX_COGNOM ON MALALT (DATA_NAIX, COGNOM);

-- Per tenir els malalts indexats per cognom i data de naixement:

CREATE INDEX MALALT_COGNOM_NAIX ON MALALT (COGNOM, DATA_NAIX);

-- Per tenir els ingressats indexats per hospital sala:

CREATE INDEX INGRESSOS_HOSP_SALA ON INGRESSOS (HOSPITAL_COD, SALA_COD);

-- Per tenir els doctors indexats per la seva especialitat (entre tots els
    hospitals):

CREATE INDEX DOCTOR_ESP_HOSP ON DOCTOR (ESPECIALITAT, HOSPITAL_COD);

-- Per tenir els doctors indexats per la seva especialitat dins cada hospital:

CREATE INDEX DOCTOR_HOSP_ESP ON DOCTOR (HOSPITAL_COD, ESPECIALITAT);
```

FIGURA 2.1. Índexos de la taula 'doctor' mostrats a través de Workbench de MySQL



Tots aquests índexs s'afegeixen als existents a causa de les restriccions de clau primària i d'unicitat. Per visualitzar tots els índexs existents sobre una taula concreta podem utilitzar l'ordre:

```
show indexes from [nom_esquema.]nom_taula
```

En el cas de la taula doctor de l'esquema sanitat, podem observar el resultat visualitzat en l'eina Workbench de MySQL, en la figura 2.1.

2.8 Definició de vistes

Les vistes es corresponen amb els diferents tipus de consultes que proporciona l'SGBDR MS-Access.

Una **vista** és una taula virtual per mitjà de la qual es pot veure i, en alguns casos canviar, informació d'una o més taules.

Una vista té una estructura semblant a una taula: files i columnes. Mai no conté dades, sinó una sentència `SELECT` que permet accedir a les dades que es volen presentar per mitjà de la vista. La gestió de vistes és semblant a la gestió de taules.

La sentència `CREATE VIEW` és la instrucció proporcionada pel llenguatge SQL per a la creació de vistes.

La seva sintaxi és aquesta:

```
create [or replace] view [<nom_esquema>.]<nom_vista> [(col1, col2...)]
as <sentència_select>
[with [cascaded| local] check option];
```

Com observareu, aquesta sentència és similar a la sentència per crear una taula a partir del resultat d'una consulta. La definició dels noms dels camps és optativa; si no s'efectua, els noms de les columnes recuperades passen a ser els noms dels camps nous. La sentència SELECT es pot basar en altres taules i/o vistes.

L'opció `with check option` indica a l'SGBD que les sentències INSERT i UPDATE que es puguin executar sobre la vista han de verificar les condicions de la clàusula `where` de la vista.

L'opció `or replace` en la creació de la vista permet modificar una vista existent amb una nova definició. Cal tenir en compte que aquesta és l'única via per modificar una vista sense eliminar-la i tornar-la a crear.

La sentència `DROP VIEW` és la instrucció proporcionada pel llenguatge SQL per a l'eliminació de vistes.

La seva sintaxi és aquesta, que permet eliminar una o diverses vistes:

```
drop view [<nom_esquema>.]<nom_vista> [,[<nom_esquema>.]<nom_vista>] ;
```

La sentència `ALTER VIEW` és la instrucció proporcionada per modificar vistes. La seva sintaxi és aquesta:

```
alter view <nom_vista> [(columna1, ....)]
  as <sentència_select>;
```

Exemple 1 de creació de vistes

En l'esquema *empresa*, es demana una vista que mostri totes les dades dels empleats acompanyades del nom del departament al qual pertanyen.

La sentència pot ser la següent:

```
create view EMPD
as select emp_no, cognom, ofici, cap, data_alta, salari, comissio
    , e.dept_no, dnom
  from emp e, dept d
  where e.dept_no = d.dept_no;
```

Una vegada creada la vista, es pot utilitzar com si fos una taula, com a mínim per executar-hi sentències `SELECT`:

```
SQL> select * from empd;

EMP_NO  COGNOM     OFICI      CAP  DATA_ALTA   SALARI  COMISSIÓ   DEPT_NO  DNOM
_______ _________ __________ ____ ___________ _______ __________ ________
    ______________
7369    SÁNCHEZ    EMPLEAT    7902 17/12/1980  104000             20
    INVESTIGACIÓ
7499    ARROYO     VENEDOR    7698 20/02/1980  208000  39000      30
    VENDES
7521    SALA       VENEDOR    7698 22/02/1981  162500  65000      30
    VENDES
7566    JIMÉNEZ    DIRECTOR   7839 02/04/1981  386750             20
    INVESTIGACIÓ
7654    MARTÍN     VENEDOR    7698 29/09/1981  162500  182000     30
    VENDES
7698    NEGRO      DIRECTOR   7839 01/05/1981  370500             30
    VENDES
7782    CEREZO     DIRECTOR   7839 09/06/1981  318500             10
    COMPTABILITAT
7788    GIL        ANALISTA   7566 09/11/1981  390000             20
    INVESTIGACIÓ
7839    REY        PRESIDENT       17/11/1981  650000             10
    COMPTABILITAT
7844    TOVAR      VENEDOR    7698 08/09/1981  195000  0          30
    VENDES
7876    ALONSO     EMPLEAT    7788 23/09/1981  143000             20
    INVESTIGACIÓ
7900    JIMENO     EMPLEAT    7698 03/12/1981  123500             30
    VENDES
7902    FERNÁNDEZ  ANALISTA   7566 03/12/1981  390000             20
    INVESTIGACIÓ
7934    MUÑOZ      EMPLEAT    7782 23/01/1982  169000             10
    COMPTABILITAT
```

Exemple 2 de creació de vistes

En l'esquema *empresa*, es demana una vista per visualitzar els departaments de codi parell.

La sentència pot ser aquesta:

```
create view DEPT_PARELL
as select * from DEPT where mod(dept_no,2) = 0;
```

2.8.1 Operacions d'actualització sobre vistes en MySQL

Les operacions d'actualització (`INSERT`, `DELETE` i `UPDATE`) són, per als diversos SGBD, un tema conflictiu, ja que les vistes es basen en sentències `SELECT` en què poden intervenir moltes o poques taules i, fins i tot, altres vistes, i per tant cal decidir a quina d'aquestes taules i/o vistes correspon l'operació d'actualització sol·licitada.

Per a cada SGBD, caldrà conèixer molt bé les operacions d'actualització que permet sobre les vistes.

Cal destacar que les vistes en MySQL poden ser actualitzables o no actualitzables: les vistes en MySQL són actualitzables, és a dir, admeten operacions `UPDATE`, `DELETE` o `INSERT` com si es tractés d'una taula. Altrament són vistes no actualitzables.

Les vistes actualitzables han de tenir relacions un a un entre les files de la vista i les files de les taules a què fan referència. Així, doncs, hi ha clàusules i expressions que fan que les vistes en MySQL siguin no actualitzables, per exemple:

- Funcions d'agregació (SUM( ), MIN( ), MAX( ), COUNT( ), etc.)
- `DISTINCT`
- `GROUP BY`
- `HAVING`
- `UNION`
- Subconsultes en la sentència `select`
- Alguns tipus de `join`
- Altres vistes no actualitzables en la clàusula `from`
- Subconsultes en la sentència `where` que facin referència a taules de la clàusula `FROM`

Una vista que tingui diverses columnes calculades no és inserible, però sí que es poden actualitzar les columnes que contenen dades no calculades.

Exemple d'actualització en una vista

Recordem una de les vistes creades sobre l'esquema *empresa*:

```
create view EMPD
as select emp_no, cognom, ofici, cap, data_alta, salari, comissio
    , e.dept_no, dnom
  from emp e, dept d
  where e.dept_no = d.dept_no;
```

Si volem modificar la comissió d'un empleat concret (`emp_no=7782`) mitjançant la vista `EMPD` ho podem fer tal com segueix:

```
update empd set comissio=10000 where emp_no=7782;
```

Amb el resultat esperat, que s'haurà canviat la comissió de l'empleat, també, en la taula `EMP`.

Si volem canviar, però, el nom de departament d'aquest empleat (emp_no=7782) i ho fem amb la instrucció següent:

```
update empd set dnom='ASSESSORIA COMPTABLE' where emp_no=7782;
```

El resultat serà que també s'hauran canviat els noms dels departament de comptabilitat dels companys del mateix departament, ja que, efectivament, s'ha canviat el nom del departament dins de la taula de `DEPT`, i potser aquest no és el resultat que esperàvem.

Exemple d'eliminació i inserció en una vista

Recordem la vista EMPD creada sobre l'esquema *empresa*:

```
create view EMPD
as select emp_no, cognom, ofici, cap, data_alta, salari, comissio
    , e.dept_no, dnom
  from emp e, dept d
  where e.dept_no = d.dept_no;
```

Si intentem executar una sentència `DELETE` sobre la vista `EMPD`, el sistema no ho permetrà, en tractar-se d'una vista que conté una join i, per tant, dades de dues taules diferents.

```
delete from empd where emp_no=7782;
```

Podem executar `INSERT` sobre la vista EMPD i tampoc no ho podrem fer.

```
insert into empd values(7777,'PLAZA','VENEDOR',7698,'1984-05-01'
    ,200000,NULL,10,NULL);
```

Exemple d'operacions d'actualització sobre vistes

Recordem la vista `DEPT_PARELL` creada sobre l'esquema *empresa*:

```
create view DEPT_PARELL
as select * from DEPT where mod(dept_no,2) = 0;
```

Ara efectuarem algunes insercions, alguns esborraments i algunes modificacions de departaments parells per mitjà de la vista `DEPT_PARELL`.

```
insert into DEPT_PARELL values (60,'INFORMÀTICA', 'BARCELONA');
```

Aquesta instrucció provoca la inserció d'una fila sense cap problema en la taula `DEPT`.

```
insert into DEPT_PARELL values (55,'MAGATZEM','LLEIDA');
```

Aquesta instrucció provoca la inserció d'una fila sense cap problema, però aquesta inserció no es produiria si la vista hagués estat creada amb l'opció with check option, ja que en aquesta situació els departaments inserits en la taula `DEPT` per mitjà de la vista `DEPTPARELL` haurien de verificar la clàusula `where*` de la definició de la vista.

Podem comprovar que si fem aquesta modificació, ens dóna un error en la inserció d'un codi no parell:

```
create or replace view DEPT_PARELL
  as select * from DEPT where mod(dept_no,2) = 0 with check
       option;

insert into DEPT_PARELL values (65,'MAGATZEM2','GIRONA');
```

La instrucció següent provoca error perquè s'ha definit l'opció `with check` option, ja que en aquesta situació el departament 50 ha estat seleccionat però no ha pogut canviar a 51 perquè no compleix la clàusula where de la vista. Si tornem a evitar l'opció with check option en la definició de la vista, l'actualització del departament 50 (seleccionable per la vista, en ser parell) cap a departament 51 no donarà cap error i farà el canvi de codi volgut:

```
create or replace view DEPT_PARELL
  as select * from DEPT where mod(dept_no,2) = 0;

update DEPT_PARELL set dept_no = dept_no+1 where dept_no = 50;
```

```
delete from DEPT_PARELL where dept_no IN (50, 55);
```

Aquesta instrucció no esborra cap departament, ja que el 50 no existeix (l'hem canviat a 51), i el 55 existeix però no és seleccionable per mitjà de la vista ja que no és parell.

2.9 Sentència RENAME

La sentència `RENAME` és la instrucció proporcionada pel llenguatge SQL per modificar el nom d'una o diverses taules del sistema.

La seva sintaxi és aquesta:

```
rename <nom_actual> to <nou_nom> [, <nom_actual2> to <nou_nom2>, ....];
```

2.10 Sentència TRUNCATE

La sentència `TRUNCATE` és la instrucció proporcionada pel llenguatge SQL per eliminar totes les files d'una taula.

La seva sintaxi és aquesta:

```
truncate [table] <nom_taula>;
```

`TRUNCATE` és similar a `delete` de totes les files (sense clàusula `where`). Funciona, però, eliminant la taula (`DROP TABLE`) i tornant-la a crear (`CREATE TABLE`).

2.11 Creació, actualització i eliminació d'esquemes o bases de dades en MySQL

> Recordem que en MySQL un `SCHEMA` és sinònim de `DATABASE` i que consisteix en una agrupació lògica d'objectes de base de dades (taules, vistes, procediments, etc.).

Per crear una base de dades o un esquema es pot utilitzar la sintaxi bàsica següent:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] nom_bd;
```

Per modificar una base de dades o un esquema es pot utilitzar la sintaxi següent, que permet canviar el nom del directori en què està mapada la base de dades o el conjunt de caràcters:

```
ALTER {DATABASE | SCHEMA} nom_bd
  { UPGRADE DATA DIRECTORY NAME
  | [DEFAULT] CHARACTER SET [=] nom_charset
  | [DEFAULT] COLLATE [=] nom_collation } ;
```

Per eliminar una base de dades o un esquema es pot utilitzar la sintaxi bàsica següent:

```
DROP {DATABASE | SCHEMA} [IF NOT EXISTS] nom_bd;
```

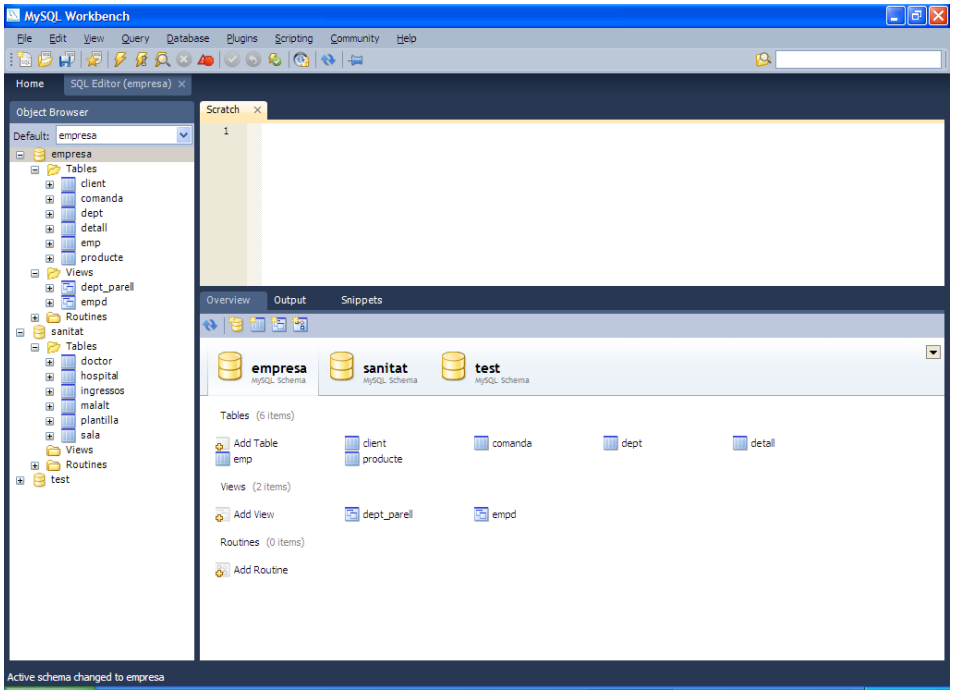
2.12 Com es poden conèixer els objectes definits en un esquema de MySQL

Una vegada que sabem definir taules, vistes, índexs o esquemes, i com modificar, en alguns casos, les definicions existents ens sorgeix un problema: com podem accedir de manera ràpida als objectes existents?

> L'eina **MySQL Workbench** és una eina gràfica que permet, entre altres coses, veure els objectes definits dins del SGBD MySQL i explorar les bases de dades que integra.

Es pot veure en la figura 2.2 l'eina MySQL Workbench en l'apartat SQL Development com es pot navegar pels diferents objectes de les bases de dades que gestiona MySQL.

Figura 2.2. MySQL Workbench: eina gràfica de MySQL. Navegació pels diferents objectes de la base de dades



És important saber, també, que l'SGBD MySQL ens proporciona un conjunt de taules (que formen el diccionari de dades de l'SGBD) que permeten accedir a les definicions existents. N'hi ha moltes, però ens interessa conèixer les de la taula 2.2. Totes incorporen una gran quantitat de columnes, la qual cosa fa necessari esbrinar-ne l'estructura, per mitjà de la instrucció `desc`, abans d'intentar trobar-hi una informació.

Taula 2.2. Vistes de l'SGBD MySQL que proporcionen informació sobre els objectes definits en l'esquema

Taula	Contingut	Exemple d'ús
information_schema.schemata	Informació sobre les bases de dades de l'SGBD.	select * from information_schema.schemata;
information_schema.tables	Informació sobre les taules de les diferents bases de dades de MySQL.	select * from information_schema.TABLES where table_schema='sanitat';
information_schema.columns	Informació sobre columnes de les taules de les diferents bases de dades de MySQL.	SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT FROM INFORMATION_SCHEMA.COLUMNS WHERE table_name = 'doctor' AND table_schema = 'sanitat';
information_schema.table_constraints	Informació sobre les restriccions de les taules de la base de dades.	SELECT * from information_schema.TABLE_CONSTRAINTS where table_schema='sanitat' ;
information_schema.views	Informació sobre les vistes de les diferents bases de dades de MySQL.	select * from information_schema.views where table_schema='empresa';
information_schema.referential_constraints	Informació sobre les claus foranes de les taules de la base de dades.	select * from information_schema.REFERENTIAL_CONSTRAINTS;

Hi ha formes abreujades, però, de mostrar la informació d'aquestes taules del diccionari; per exemple, per mostrar les taules o les columnes de les taules, podem utilitzar aquestes formes simplificades:

```
SHOW TABLES

SHOW COLUMNS
  FROM nom_taula
  [FROM nom_base_dades]
```

3. Control de transaccions i concurrències

Una **transacció** és una seqüència d'instruccions SQL que l'SGBD gestiona com una unitat. Les sentències `COMMIT` i `ROLLBACK` permeten indicar un fi de transacció.

Exemple de transacció: operació al caixer automàtic

Una transacció típica és una operació en un caixer automàtic, per exemple: si anem a un caixer automàtic a treure diners d'un compte bancari esperem que si l'operació acaba bé (i obtenim els diners extrets) es reflecteixi aquesta operació en el compte, i, en canvi, si hi ha hagut algun error i el sistema no ens ha pogut donar els diners, esperem que no es reflecteixi aquesta extracció en el nostre compte bancari. L'operació, doncs, cal que es consideri com una unitat i acabi bé (*commit*), però, que si acaba malament (*rollback*) tot quedi com estava inicialment abans de començar.

Transaccions en MySQL

Les transaccions en MySQL només tenen sentit sota el motor d'emmagatzemament InnoDB, que és l'únic motor transaccional de MySQL. Recordeu que els altres sistemes d'emmagatzemament són no transaccionals i, per tant, cada instrucció que s'executa és independent i funciona, sempre, de manera autocommitiva.

Una transacció habitualment comença en la primera sentència SQL que es produeix després d'establir connexió en la base de dades, després d'una sentència COMMIT o després d'una sentència ROLLBACK.

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

Els canvis realitzats en la base de dades en el transcurs d'una transacció només són visibles per a l'usuari que els executa. En executar una COMMIT, els canvis realitzats en la base de dades passen a ser permanents i, per tant, visibles per a tots els usuaris.

Si una transacció finalitza amb ROLLBACK, es desfan tots els canvis realitzats en la base de dades per les sentències de la transacció.

Recordem que MySQL té l'autocommit definit per defecte, de manera que s'efectua un **COMMIT** automàtic després de cada sentència SQL de manipulació de dades. Per desactivar-lo cal executar:

```
1 set autocommit=0;
```

Per tal de tornar a activar el sistema d'autocommit:

```
1 set autocommit=1;
```

Cal tenir en compte que una transacció només té sentit si no està definit l'autocommit.

El funcionament de transaccions no és el mateix en tots els SGBD i, per tant, caldrà esbrinar el tipus de gestió que proporciona abans de voler-hi treballar.

3.1 Sentència START TRANSACTION en MySQL

`START TRANSACTION` defineix explícitament l'inici d'una transacció. Per tant, el codi que hi hagi entre `start transaction` i `commit` o `rollback` formarà la transacció.

Iniciar una transacció implica un bloqueig de taules (`LOCK TABLES`), així com la finalització de la transacció provoca el desbloqueig de les taules (`UNLOCK TABLES`).

En MySQL `start transaction` és sinònim de `begin` i, també, de `begin work`. I la sintaxi per a `start transaction` és la següent:

```
1 START TRANSACTION [WITH CONSISTENT SNAPSHOT] | BEGIN [WORK]
```

L'opció `WITH CONSISTENT SNAPSHOT` inicia una transacció que permet lectures consistents de les dades.

3.2 Sentències COMMIT i ROLLBACK en MySQL

`COMMIT` defineix explícitament la finalització esperada d'una transacció. La sentència `ROLLBACK` defineix la finalització errònia d'una transacció.

La sintaxi de `commit` i `rollback` en MySQL és la següent:

```
1 COMMIT [WORK] [AND [NO] CHAIN | [NO] RELEASE]
2 ROLLBACK [WORK] [AND [NO] CHAIN | [NO] RELEASE]
```

L'opció `AND CHAIN` provoca l'inici d'una nova transacció que començarà tot just acabi l'actual.

L'opció `REALEASE` causarà la desconnexió de la sessió actual.

Quan es fa un `rollback` és possible que el sistema processi de manera lenta les operacions, ja que un `rollback` és una instrucció lenta. Si es vol visualitzar el conjunt de processos que s'executen es pot executar l'ordre `SHOW PROCESSLIST` i visualitzar els processos que s'estan *desfent* a causa del `rollback`.

L'SGBDR *MySQL* realitza una `COMMIT` implícita abans d'executar qualsevol sentència LDD (llenguatge de definició de dades) o `LCD` (llenguatge de control de dades), o en executar una desconnexió que no hagi estat precedida d'un error. Per tant, no té sentit incloure aquest tipus de sentències dintre de les transaccions.

3.3 Sentències SAVEPOINT i ROLLBACK TO SAVEPOINT en MySQL

Hi ha la possibilitat de marcar punts de control (`savepoints`) enmig d'una transacció, de manera que si s'efectua ROLLBACK aquest pugui ser total (tota la transacció) o fins a un dels punts de control de la transacció.

La instrucció `SAVEPOINT` permet crear punts de control. La seva sintaxi és aquesta:

```
savepoint <nom_punt_control>;
```

La sentència `ROLLBACK` per desfer els canvis fins a un determinat punt de control té aquesta sintaxi:

```
rollback [work] to  [savepoint] <nom_punt_control>;
```

Si en una transacció es crea un punt de control amb el mateix nom que un punt de control que ja existeix, aquest queda substituït pel nou.

Si es vol eliminar el punt de control sense executar un `commit` ni un `rollback` podem executar la instrucció següent:

```
release savepoint <nom_punt_control>
```

Exemple d'utilització de punts de control

Considerem la situació següent:

```
SQL> instrucció_A;
SQL> savepoint PB;
SQL> instrucció_B;
SQL> savepoint PC;
SQL> instrucció C;
SQL> instrucció_consulta_1;
SQL> rollback to PC;
SQL> instrucció_consulta_2;
SQL> 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 control `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.

3.4 Sentències LOCK TABLES i UNLOCK TABLES

Per tal de prevenir la modificació de certes taules i vistes en alguns moments, quan es requereix accés exclusiu a les mateixes, en sessions paral·leles (o concurrents), és possible bloquejar l'accés a les taules.

Concurrència

La concurrència en l'execució de processos implica l'execució simultània de diverses accions, cosa que habitualment té com a conseqüència l'accés simultani a unes dades comunes, que caldrà tenir en compte a l'hora de dissenyar els processos individuals a fi d'evitar inconsistència en les dades.

El bloqueig de taules (`LOCK TABLES`) protegeix contra accessos inapropiats de lectures o escriptures d'altres sessions.

La sintaxi per bloquejar algunes taules o vistes, i impedir que altres accessos puguin canviar simultàniament les dades, és la següent:

```
lock tables <nom_taula1> [[as] <alies1>] read | write
  [, <nom_taula1> [[as] <alies1>] read | write ] ...
```

L'opció `read` permet llegir sobre la taula, però no escriure-hi. L'opció `write` permet que la sessió que executa el bloqueig pugui escriure sobre la taula, però la resta de sessions només la puguin llegir, fins que acabi el bloqueig.

De vegades, els bloquejos s'utilitzen per simular transaccions (en motors d'emmagatzemament que no siguin transaccionals, per exemple) o bé per aconseguir accés més ràpid a l'hora d'actualitzar les taules.

Quan s'executa `lock tables` es fa un `commit` implícit, per tant, si hi havia alguna transacció oberta, aquesta acaba. Si acaba la connexió (normalment o anormalment) abans de desbloquejar les taules, automàticament es desbloquegen les taules.

És possible, també, en MySQL bloquejar totes les taules de totes les bases de dades de l'SGBD, per fer, per exemple, còpies de seguretat. La sentència que ho permet és `FLUSH TABLES WITH READ LOCK`.

Per desbloquejar les taules (totes les que estiguessin bloquejades) cal executar la sentència `UNLOCK TABLES`.

3.4.1 Funcionament dels bloquejos

Quan es crea un bloqueig per accedir a una taula, dins d'aquesta zona de bloqueig no es pot accedir a altres taules (a excepció de les taules del diccionari de l'SGBD -`information_schema`-) fins que no finalitzi el bloqueig. Per exemple:

```
mysql> LOCK TABLES t1 READ;
mysql> SELECT COUNT(*) FROM t1;
+-----------+
| COUNT(*) |
+-----------+
|         3 |
+-----------+
mysql> SELECT COUNT(*) FROM t2;
ERROR 1100 (HY000): Table 't2' was not locked with LOCK TABLES
```

No es pot accedir més d'una vegada a la taula bloquejada. Si es necessita accedir dos cops a la mateixa taula, cal definir un àlies per al segon accés a l'hora de fer el bloqueig.

```
mysql> LOCK TABLE t WRITE, t AS t1 READ;
mysql> INSERT INTO t SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> INSERT INTO t SELECT * FROM t AS t1;
```

Si es bloqueja una taula especificant-ne un àlies, cal fer-hi referència amb aquest àlies. Provoca un error accedir-hi directament amb el seu nom:

```
mysql> LOCK TABLE t AS myalias READ;
mysql> SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> SELECT * FROM t AS myalias;
```

Si es vol accedir a una taula (bloquejada) amb un àlies, cal definir l'àlies en el moment d'establir el bloqueig:

```
mysql> LOCK TABLE t READ;
mysql> SELECT * FROM t AS myalias;
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES
```

3.5 Sentència SET TRANSACTION

MySQL permet configurar el tipus de transacció amb la sentència `set transaction`.

La sintaxi per configurar les transaccions és:

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
  { READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }
```

Aquesta sentència permet definir determinats paràmetres per a la transacció en curs o per a la transacció següent que s'obrirà. Les característiques que es defineixen poden afectar globalment (`GLOBAL`) o bé afectar la sessió en curs (`SESSION`).

- `READ UNCOMMITTED`: es permet accedir a les dades de les taules, encara que no s'hagi fet un `commit`. Per tant, és possible accedir a dades no consistents (*dirty read*).
- `READ COMMITTED`: només es permet accedir a dades que s'hagin acceptat (**commit**).
- `REPEATABLE READ` (opció per defecte): permet accedir a les dades de manera consistent dins de les transaccions, de manera que totes les lectures de les dades, dins d'una transacció de tipus `REPEATABLE READ`, permetran obtenir les dades com a l'inici de la transacció, encara que ja haguessin canviat.
- `SERIALIZABLE`: permet accedir a les dades de manera consistent en qualsevol lectura de les dades, encara que no ens trobem dins d'una transacció.