1. Instrucciones para la manipulación de datos

Mediante la sentencia `SELECT`podemos consultar datos, pero, como las podemos manipular, definir y controlar?

El lenguaje SQL aporta una serie de instrucciones con las que se pueden realizar las siguientes acciones:

- La manipulación de los datos (instrucciones LMD que nos deben permitir efectuar altas, bajas y modificaciones).
- La definición de datos (instrucciones LDD que nos deben permitir crear, modificar y eliminar las tablas, los índices y las vistas).
- El control de datos (instrucciones LCD que nos deben permitir gestionar los usuarios y sus privilegios).

> **acrónimos**
>
> Recordemos los acrónimos para los diferentes apartados del lenguaje SQL: LC (lenguaje de consulta); LMD (lenguaje de manipulación de datos); LDD (lenguaje de definición de datos); LCD (lenguaje de control de datos).

El lenguaje SQL proporciona un conjunto de instrucciones, reducido pero muy potente, para manipular los datos, dentro del cual se ha de distinguir entre dos tipos de instrucciones:

- Las instrucciones que permiten ejecutar la manipulación de los datos, y que se reducen a tres: `INSERT`para la introducción de nuevas filas, UPDATE para la modificación de filas, y `DELETE`por el borrado de filas.
- Las instrucciones para el control de transacciones, que deben permitir asegurar que un conjunto de operaciones de manipulación de datos se ejecute con éxito en su totalidad o, en caso de problema, se aborte total o hasta un determinado punto en el tiempo.

Antes de introducirnos en el estudio de las instrucciones INSERT, UPDATE y DELETE, hay que conocer como el SGBD gestiona las instrucciones de inserción, eliminación y modificación que podamos ejecutar, ya que hay dos posibilidades de funcionamiento:

- Que queden automáticamente validadas y no haya posibilidad de tirar atrás. En este caso, los efectos de toda instrucción de actualización de datos que tenga éxito son automáticamente accesibles desde el resto de conexiones de la base de datos.
- Que queden en una cola de instrucciones, que permite echar atrás. En este caso, se dice que las instrucciones de la cola están pendientes de validación, y el usuario debe ejecutar, cuando lo cree conveniente, una instrucción para validarlas (llamada *COMMIT*) o una instrucción para echar atrás (llamada *ROLLBACK*).

Este funcionamiento implica que los efectos de las instrucciones pendientes de validación no se ven por el resto de conexiones de la base de datos, pero sí son accesibles desde la

instrucciones desaparecen de la cola y ninguna conexión (ni la propia ni el resto) no accede a los efectos correspondientes, es decir, es como si nunca hubieran existido.

Estos posibles funcionamientos forman parte de la gestión de transacciones que proporciona el SGBD y que hay que estudiar con más detenimiento. A la hora, pero, de ejecutar instrucciones INSERT, UPDATE y DELETE debemos conocer el funcionamiento del SGBD para poder actuar en consecuencia.

Así, por ejemplo, un SGBD MySQL funciona con validación automática después de cada instrucción de actualización de datos no se indica lo contrario y, en cambio, un SGBD Oracle funciona con la cola de instrucciones pendientes de confirmación o rechazo que ha indicar al usuario.

En cambio, en MySQL, si se quiere desactivar la opción de autocommit que hay por defecto, habrá que ejecutar la siguiente instrucción:

```
SET autocommit = 0 ;
```

1.1. sentencia INSERT

> La sentencia INSERT es la instrucción proporcionada por el lenguaje SQL para insertar nuevas filas en las tablas.

Admite dos sintaxis:

1. Los valores que se han de insertar explicitan en la misma instrucción en la cláusula `values`:

```
INSERT  INTO  < nombre_tabla >  [ ( col1 , COL2 ... ) ]
VALUES  ( val1 , val2 ... ) ;
```

2. Los valores que se han de insertar consiguen por medio de una sentencia `SELECT`:

```
INSERT  INTO  < nombre_tabla >  [ ( col1 , COL2 ... ) ]
SELECT ... ;
```

En todo caso, se pueden especificar las columnas de la tabla que se han de rellenar y el orden en que se suministran los diferentes valores. En caso de que no se especifiquen las columnas, el SQL entiende que los valores se suministran para todas las columnas de la tabla y, además, en el orden en que están definidos en la tabla.

La lista de valores de la cláusula `valuesy` la lista de resultados de la sentencia SELECT deben coincidir en número, tipo y orden con la lista de columnas que se deben rellenar.

> **Ejemplo 1 de sentencia INSERT**

La posible sentencia para conseguir el objetivo es ésta:

```
INSERT  INTO dept ( dept_no , dnom )
VALUES  ( 50 ,  INFORMÁTICA ' ) ;
```

Si ejecutamos una consulta para comprobar el contenido actual de la tabla DEPT, encontraremos la nueva fila sin localidad asignada. El SGBD ha permitido dejar la localidad con valor NULLporque lo tiene permitido así, como se puede ver en el descriptor de la tabla DEPT:

```
SQL> desc dept;

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

3 rows selected
```

Ejemplo 2 de sentencia INSERT

En el esquema *sanidad* , se pide dar de alta el doctor de código 100 y nombre 'BARRUFET D.'.

La solución parece que podría ser esta:

```
INSERT  INTO doctor ( doctor_no , apellido )
VALUES  ( 100 ,  'BARRUFET D.' ) ;
```

Al ejecutar esta sentencia, el SGBDR da un error.

Lo cierto es que la tabla DOCTORno admite valores nulos en la columna hospital_cod, ya que esta columna forma parte de la clave primaria. Miremos el descriptor de la tabla DOCTOR:

```
SQL> desc doctor;

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

4 rows selected
```

Aparte de la columna hospital_cod, también deberíamos dar un valor en la columna especialidad, ya que tampoco admite valores nulos.

Recordemos que, en nuestro esquema *sanidad* , la columna especialidad es una cadena que no tiene ningún tipo de restricción definida ni es clave foránea de ninguna mesa en la que estén todas las especialidades posibles. Por lo tanto, si queremos saber qué especialidades hay para escribir la del doctor que queremos insertar, idénticamente a las ya introducidas en caso de que hubiera algún doctor con la misma especialidad del que queremos insertar, hacemos lo siguiente:

```
SQL> select distinct especialidad from doctor;

ESPECIALIDAD

Urología
pediatría
Cardiología
Neurología
```

```
6 rows selected
```

Supongamos que el doctor 'BARRUFET D.' es psiquiatra. Como ya hay algún doctor con la especialidad 'Psiquiatría', correspondería hacer la inserción utilizando la misma grafía para la especialidad. Además, supongamos que queremos dar de alta el doctor en el hospital 66.

```
INSERT  INTO doctor ( doctor_no , apellido , hospital_cod , especialidad )
VALUES  ( 100 ,  'BARRUFET D.' ,  66 ,  'Psiquiatría' ) ;
```

Esta vez, el SGBD también se nos queja con otro tipo de error: ha fallado la referencia a la clave foránea.

El error nos informa que una restricción de integridad definida en la tabla ha intentado ser violada y, por tanto, la instrucción no ha finalizado con éxito. El SGBD nos pasa dos informaciones que tengamos pistas de dónde está el problema:

- Nos da una descripción breve del problema (no se puede añadir una fila hija - *child row* -), la cual nos da a entender que se trata de un error de clave foránea, es decir, que no existe el código en la tabla referenciada.
- Nos dice la restricción que ha fallado (`sanitat.doctor, CONSTRAINT … FOREIGN KEY (HOSPITA_COD) …`).

El SGBD tiene toda la razón. Recordemos que la columna hospital_cod de la tabla HOSPITAL es clave foránea de la tabla HOSPITAL. Esto significa que cualquier inserción en la tabla DOCTOR debe ser para hospitales existentes en la tabla HOSPITAL, y esto no ocurre con el hospital 66, como se puede ver en consultar los hospitales existentes:

```
SQL >  SELECT  *  FROM hospital;

HOSPITAL_COD NOMBRE DIRECCIÓN TELÉFONO QTAT_LLITS
------------ ---------- -------------------- -------- -----------
13           Provincial O Donella 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
```

Así pues, o nos hemos equivocado de hospital o debemos dar de alta previamente el hospital 66. Supongamos que es el segundo caso y que, por tanto, tenemos que dar de alta del hospital 66:

```
insert into hospital (hospital_cod, nombre, direccion)
values (66, 'General', 'De la fuente, 13');
```

El SGBD nos acepta la instrucción. Fijémonos que hemos informado del código de hospital, del nombre y de la dirección. Miremos el descriptor de la tabla HOSPITAL:

```
SQL> desc hospital;

Name Null Type
----------------- -------- ------------------------ ---
HOSPITAL_COD NOT NULL NUMBER (2)
NOMBRE NOT NULL VARCHAR2 (10)
Dirección VARCHAR2 (20)
TELEFONO VARCHAR2 (8)
QTAT_LLITS NUMBER (3)

5 rows selected
```

Vemos cinco campos, de los cuales sólo los dos primeros tienen marcada la obligatoriedad de valor. Por tanto, no se nos ha quejado porque no hayamos indicado el teléfono del hospital ni la cantidad de camas que tiene el hospital.

Comprobamos la información que ahora hay en la tabla `HOSPITAL`:

```
HOSPITAL_COD NOMBRE DIRECCION TELEFONO QTAT_LLITS
------------ ---------- -------------------- ------ - -----------
13           Provincial O Donella 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 fuente ,  13               0

5  rows selected
```

Sorpresa! Para el nuevo hospital, la columna teléfono no tiene valor (valor NULL), pero la columna qtatllits tiene el valor 0. De dónde ha salido? Esto se debe a que la columna qtatllits de la tabla `HOSPITAL`tiene definido el valor por defecto (0) que el SGBD utiliza para rellenar la columna qtat_llits cuando se produce una inserción en la tabla sin indicar valor para esta columna.

Ahora parece que ya podemos insertar nuestro doctor 'BARRUFET De .:

```
INSERT  INTO doctor ( doctor_no , apellido , hospital_cod , especialidad )
VALUES  ( 100 ,  'BARRUFET D.' ,  66 ,  'Psiquiatría' ) ;
```

No nos olvidemos de registrar los cambios con la instrucción `COMMIT`o de hacer `ROLLBACK`, si tenemos la autocommit desactivado.

Ejemplo 3 de sentencia INSERT

Antes de empezar, desactivaremos el autocommit que tiene configurado por defecto MySQL para poder practicar el `commit`y el `rollback`:

```
SET AUTOCOMMIT = 0 ;
```

En el esquema *empresa* , se quiere insertar la instrucción identificada por número 1.000, con fecha de orden del 1 de septiembre de 2000 y para el cliente 500.

Quizás necesitamos conocer, en primer lugar, el descriptor de la tabla `COMANDA`:

```
SQL >  DESC pedido;

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
```

Fijémonos que tenemos la información correspondiente a todos los campos obligatorios. Por lo tanto, podemos ejecutar el siguiente:

```
INSERT  INTO pedido ( com_num , com_data , client_cod )
VALUES  ( 1000 , '01/09/2000' , 500 ) ;
```

El SGBD nos reporta el error de restricción de integridad sobre la clave foránea. Y, por supuesto, el SGBD vuelve a tener razón, ya que en el esquema *empresa* la mesa `COMANDA`tiene una restricción de clave foránea en la columna client_cod.

Si consultamos el contenido de la tabla `client`, veremos que no hay ningún cliente con código 500. Por ello, el SGBD ha dado un error. Supongamos que era un error nuestro y el orden correspondía al cliente 109 (que sí existe en la tabla CLIENTE). Esta vez la instrucción siguiente no nos da ningún problema.

```
INSERT  INTO pedido ( com_num , com_data , client_cod )
```

```
SQL >  SELECT  *  FROM pedido WHERE com_num = 1,000 ;

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

Hacemos rollback para echar atrás la inserción efectuada y así poder comprobar que también la podríamos hacer de diferentes maneras. Recordemos que no es obligatorio indicar las columnas para las que se introducen los valores. En este caso, el SGBD espera todas las columnas de la tabla en el orden en que están definidas en la tabla. Así pues, podemos hacer lo siguiente:

```
INSERT  INTO pedido
VALUES  ( 1000 , '01/09/2000' , NULL , 109 , NULL , NULL ) ;
```

Hacemos rollback para probar otra posibilidad. Fijémonos que el SGBD también nos deja introducir un precio total de orden cualquiera:

```
ROLLBACK ;

INSERT  INTO pedido
VALUES  ( 1000 , DATE  '2000-09-01' , NULL , 109 , NULL , 9999 ) ;

commit;
```

El SGBDR ha aceptado esta sentencia y hay insertado la fila correspondiente. Pero, hemos introducido un importe total que no se corresponde con la realidad, ya que no hay ninguna línea de detalle. Es decir, el valor 9999 no es válido! Los SGBD proporcionan mecanismos (disparadores) para controlar este tipo de incoherencias de los datos.

disparadores

Un disparador es un conjunto de instrucciones que se ejecutan automáticamente ante un evento determinado. Así, podemos controlar que, al insertar, borrar o modificar filas de detalle de una orden, el importe total de la orden se actualice automáticamente.

Ejemplo 4 de sentencia INSERT

En primer lugar, volvemos a activar la opción de autocommit para que sea más cómoda el trabajo.

```
SET AUTOCOMMIT = 1 ;
```

Como detalle de la orden 1000 insertada en el ejemplo anterior, en el esquema *empresa* se quieren insertar las mismas líneas que contiene la orden 620.

En este caso, ejecutaremos una instrucción `INSERT`tomando como valores que se insertarán los que nos da el resultado de una sentencia `SELECT`:

```
FROM detalle
WHERE com_num = 620 ;
```

En esta instrucción, hemos seleccionado las filas de detalle de la orden 620 y las hemos insertado como filas de detalle del orden 1000. Tenemos que ser conscientes de que el importe total de la orden 1000 sigue siendo, sin embargo, incorrecto .

Como ya hemos comentado, hemos utilizado una sentencia `SELECT`para insertar valores en una tabla. Es una coincidencia que las dos sentencias actúen sobre la misma mesa `DETALL`.

Al no indicar, en la sentencia INSERT, las columnas en que se han de insertar los valores, ha sido necesario construir la sentencia SELECT de manera que las columnas de la cláusula `select`coincidieran, en orden, con las columnas de la tabla en la que se ha de efectuar la inserción. Además, como para todas las filas de la orden 620 había que indicar 1000 como número de orden, la cláusula `SELECT`ha incorporado la constante 1000 como valor para la primera columna.

1.2. sentencia UPDATE

La sentencia `UPDATE`es la instrucción proporcionada por el lenguaje SQL para modificar filas que hay en las tablas.

Su sintaxis es la siguiente:

```
UPDATE  < nombre_tabla >
SET col1 = val1 , COL2 = val2 , col3 = val3 ...
[ WHERE  < condición > ] ;
```

La cláusula optativa `where`selecciona las filas que deben actualizarse. En caso de inexistencia, se actualizan todas las filas de la tabla.

La cláusula `set`indica las columnas que deben actualizarse y el valor con que se actualizan.

El valor de actualización de una columna puede ser el resultado obtenido por una sentencia SELECT que recupera una única fila:

```
UPDATE  < nombre_tabla >
SET col1 = ( SELECT exp1 FROM  ...  ) ,
SET COL2 = ( SELECT exp2 FROM  ...  ) ,
SET col3 = val3 ,
...
[ WHERE  < condición > ] ;
```

En tales situaciones, la sentencia `SELECT`es una subconsulta de la sentencia `UPDATE`que puede utilizar valores de las columnas de la fila que se está modificando en la sentencia `UPDATE`.

ejecutar varias veces la misma sentencia `SELECT`para actualizar más de una columna. Por tanto, la sentencia `UPDATE`también admite la siguiente sintaxis:

```
UPDATE  < nombre_tabla >
SET  ( col1 , COL2 ) = ( SELECT exp1 , exp2 FROM  ...  ) ,
SET col3 = val3 ,
...
[ WHERE  < condición > ] ;
```

Ejemplo 1 de sentencia UPDATE

En el esquema *empresa* , se quiere modificar la localidad de los departamentos de manera que queden todos los caracteres en minúsculas.

La instrucción para resolver la solicitud puede ser esta:

```
UPDATE dept
SET loc =  LOWER ( loc ) ;
```

Ahora se quiere modificar la localidad de los departamentos de forma que queden con la inicial en mayúscula y el resto de letras en minúsculas.

La instrucción para resolver la solicitud puede ser esta:

```
UPDATE dept
SET loc = concat ( UPPER ( LEFT ( loc , 1 ) ) , RIGHT ( LOWER ( loc ) , LENGTH
```

Ejemplo 2 de sentencia UPDATE

En el esquema *empresa* , se quiere actualizar el importe total real del pedido 1000 a partir de los importes de las diferentes líneas de detalle que forman el pedido.

```
UPDATE pedido c
SET total =  ( SELECT  SUM ( importe )  FROM detalle
     WHERE com_num = c . com_num )
WHERE com_num = 1,000 ;
```

Ahora podemos comprobar la corrección de la información que hay en la base de datos sobre el pedido 1000:

```
SQL >  SELECT  *  FROM detalle WHERE com_num = 1,000 ;

COM_NUM DETALL_NUM PROD_NUM PREU_VENDA CANTIDAD IMPORTE
---------- ---------- ---------- ---------- ---------- ------------
1000        1          100,860      35          10          350
1000        2          200,376a      2 , 4         1000        2400
1000        3          102,130o      3 , 4         500         1700

3  rows selected

SQL >  SELECT  *  FROM pedido WHERE com_num = 1,000 ;

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

1  rows selected
```

1.3. sentencia DELETE

> La sentencia `DELETE`es la instrucción proporcionada por el lenguaje SQL para borrar filas existentes que hay en las tablas.

Su sintaxis es la siguiente:

```
DELETE  FROM  < nombre_tabla >
[ WHERE  < condición > ] ;
```

La cláusula optativa `where`selecciona las filas que se deben eliminar. En su defecto, se eliminan todas las filas de la tabla.

> **Ejemplo de sentencia DELETE**
>
> En el esquema *empresa* , se quiere eliminar el pedido 1000.
>
> La instrucción parece que podría ser esta:
>
> ```
> DELETE  FROM pedido
> WHERE com_num = 1,000 ;
> ```
>
> Al ejecutar esta sentencia, sin embargo, nos encontramos con un error que nos indica que no se puede eliminar una fila padre (*a pariente row*).
>
> El motivo es que la columna com_num de la tabla DETALLE es clave foránea de la tabla `COMANDA`, lo que imposibilita eliminar una cabecera de pedido si hay líneas de detalle correspondientes. Estas se eliminarían de manera automática si hubiera definida la eliminación en cascada, pero no es el caso. Así pues, habrá que hacer lo siguiente:
>
> ```
> DELETE  FROM detalle WHERE com_num = 1,000 ;
> DELETE  FROM pedido WHERE com_num = 1,000 ;
> ```

1.4. sentencia REPLACE

> MySQL tiene una extensión del lenguaje SQL estándar que permite insertar una nueva fila que, en caso de que la clave primaria coincida con otra fila, previamente sea eliminada. Se trata de la sentencia `REPLACE`.

Hay tres posibles sintaxis para la sentencia **REPLACE** :

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

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

1.5. Sentencia LOAD XML

XML

XML es el acrónimo de *extensible markup language* ('lenguaje de etiquetado extensible') un metalenguaje de marcas que facilita la organización de los datos en ficheros planos.

`LOAD XML`permite leer un fichero en formato xml y almacenar los datos contenidos en una tabla de la base de datos. Su sintaxis es:

```
LOAD XML [ LOW_PRIORITY  | CONCURRENTE ]  [ LOCAL ]  INFILE  'nombrearchivo'
[ REPLACE  |  IGNORE ]
INTO  TABLE  [ nom_base_dades . ] nombre_tabla
[ CHARACTER  SET nom_charset ]
[ Rows  Identified  BY  <nom_tag> ' ]
[ IGNORE número [ LINES  |  Rows ] ]
[ ( Columnas , ... ) ]
[ SET nom_columna = expresión , ... ]
```