-

TEMA 5.
DISEÑO FÍSICO. DML

Base de Datos
CFGS DAW

Francisco Aldarias Raya
paco.aldarias@ceedcv.es
2019/2020
Fecha 08/12/19
Versión:191208.2321

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 Importante

 Atención

 Interesante

Revisiones

ÍNDICE DE CONTENIDO

TEMA 5. DISEÑO FÍSICO. DML

1. INSERCIÓN DE DATOS

Hasta ahora hemos conseguido crear tablas, el próximo paso será incluir registros en las tablas que hemos creado.

> **Importante**
>
> El DML (Data Manipulaton Language) lo forman las instrucciones capaces de modificar los datos de las tablas.

Al conjunto de instrucciones DML que se ejecutan consecutivamente, se las llama transacciones y se pueden anular todas ellas o aceptar, ya que una instrucción DML no es realmente efectuada hasta que no se acepta (COMMIT).

En todas las consultas el único dato que devuelve el SGBD es el número de registros modificados.

Para insertar un registro utliiaremos la instrucción INSERT cuya sintaxis (una de las varias formas que tene) es la siguiente:

```
INSERT INTO nombre_tabla [(nombre_col, ...)]

VALUES((expresión | DEFAULT),...)
```

En ella podemos distinguir las palabras reservadas INSERT INTO seguidas del nombre de la tabla en la que vamos a guardar los nuevos datos. Opcionalmente podemos poner entre paréntesis los nombres de los campos, si no los incluimos se deberán colocar los valores en el mismo orden en que fueron creados (es el orden de columnas según las devuelve el comando describe), pues de lo contrario se producirá un error si los tipos no coinciden o se almacenará la información de forma errónea en caso de que los tpos de datos de los campos sean compatibles.

La lista de campos a rellenar se indica si no queremos rellenar todos los campos. Los campos no rellenados explícitamente con la orden INSERT, se rellenan con su valor por defecto (DEFAULT) o bien con NULL si no se indicó valor alguno. Si algún campo tiene restricción de obligatoriedad (NOT NULL), ocurrirá un error si no rellenamos el campo con algún valor.

Veamos cómo funciona con un ejemplo. Recordemos las tablas, Departamentos y Empleados que creamos en el tema pasado:

Tabla departamentos.

CodDpto	Nombre	Ubicación
INF	Informática	Planta sótano U3
ADM	Administración	Planta quinta U2
COM	Comercial	Planta tercera U3
CONT	Contabilidad	Planta quinta U1
ALM	Almacén	Planta baja U1

Tabla empleados.

DNI	Nombre	Especialidad	FechaAlta	Dpto
12345678A	Alberto Gil	Contable	10/12/2010	CONT
23456789B	Mariano Sanz	Informática	04/10/2011	INF
34567890C	Iván Gómez	Ventas	20/07/2012	COM
45678901D	Ana Silván	Informática	25/11/2012	INF
56789012E	María Cuadrado	Ventas	02/04/2013	COM
67890123A	Roberto Milán	Logística	05/02/2010	ALM

La primera forma será indicando todos los campos y todos los valores (recuerda que si queremos colocar una cadena el valor tendrá que ir entre comillas (da igual simples o dobles).

Para insertar el primer registro de la tabla departamentos haremos:

1.1 MYSQL

```
mysql> INSERT INTO departamentos (CodDpto, Nombre, Ubicacion)
    -> VALUES ('INF','Informática','Planta sótano U3');
Query OK, 1 row affected (0.05 sec)
```

Para ver si ha funcionado y el registro se ha insertado en la tabla utilizaremos la instrucción SELECT.

Esta instrucción, que veremos muy a fondo en el próximo tema, sirve para hacer consultas a la base de datos y mostrar la información que contiene. En este ejemplo vamos a utilizar la consulta más simple, por ejemplo:

```
SELECT * FROM departamentos;
```

Donde SELECT es palabra reservada que indica seleccionar, el asterisco indica todos los campos de la tabla, la palabra reservada FROM indica de dónde se van a sacar los datos e irá seguida del nombre de la tabla de donde obtendremos la información.

Es decir, la instrucción selecciona todos los campos de la tabla departamentos y muestra su contenido. El resultado será:

```
mysql> select * from departamentos;
+---------+-------------+-------------------+
| CodDpto | Nombre      | Ubicacion         |
+---------+-------------+-------------------+
| INF     | Informática | Planta sótano U3  |
+---------+-------------+-------------------+
1 row in set (0.00 sec)
```

Podemos observar el resultado obtenido donde aparece el registro que hemos incluido con la instrucción INSERT.

Vamos a probar insertando el segundo registro de la tabla departamentos sin incluir los nombres de los campos en la instrucción.

```
mysql> INSERT INTO departamentos
    -> VALUES ('ADM','Administración','Planta quinta U2');
Query OK, 1 row affected (0.27 sec)
```

Como puedes apreciar la instrucción ha funcionado correctamente y ahora utilizaremos de nuevo el SELECT para ver el contenido de la tabla.

```
mysql> select * from departamentos;
+---------+----------------+-------------------+
| CodDpto | Nombre         | Ubicacion         |
+---------+----------------+-------------------+
| ADM     | Administración | Planta quinta U2  |
| INF     | Informática    | Planta sótano U3  |
+---------+----------------+-------------------+
2 rows in set (0.00 sec)
```

Correcto, ya tenemos dos registros en nuestra tabla departamentos, pero ¿siempre hay que ir uno a uno?

El SQL de MySQL nos proporciona un INSERT extendido que nos permite realizar la inserción de varios registros en una sola instrucción. La sintaxis es la siguiente:

```
INSERT [INTO] nombre_tabla [(nombre_col, ...)]

VALUES ({expre | DEFAULT}, ... ), (...), ...
```

Como puedes observar la sintaxis es prácticamente igual a la anterior con la salvedad que ahora podemos colocar los datos de varios registros entre paréntesis separados por comas. También ahora los nombres de las columnas son opcionales.

Para probarlo vamos a insertar dos registros más en nuestra tabla de departamentos:

```
mysql> INSERT INTO departamentos (CodDpto, Nombre, Ubicacion)
    -> VALUES ('COM','Comercial','Planta tercera U3'),
    -> ('CONT','Contabilidad','Planta quinta U1');
Query OK, 2 rows affected (0.05 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

Podemos ver en el mensaje como indica que se han añadido dos registros.

```
mysql> select * from departamentos;
+---------+----------------+--------------------+
| CodDpto | Nombre         | Ubicacion          |
+---------+----------------+--------------------+
| ADM     | Administración | Planta quinta U2   |
| COM     | Comercial      | Planta tercera U3  |
| CONT    | Contabilidad   | Planta quinta U1   |
| INF     | Informática    | Planta sótano U3   |
+---------+----------------+--------------------+
4 rows in set (0.00 sec)
```

Ahora nuestra tabla ya tIene cuatro registros insertados.

1.2 ORACLE

```
SQL> INSERT INTO departamentos (CodDpto, Nombre, Ubicacion)
  2  VALUES ('INF', 'Informática', 'Planta sótano U3');

1 row created.
```

Hemos ejecutado la misma instrucción en Oracle y el resultado ha sido satisfactorio, indicando una fila creada.

Para ver si ha funcionado utilizaremos la misma instrucción SELECT comentada en la parte de MySQL.

```
SQL> select * from departamentos;

CODDPTO    NOMBRE                         UBICACION
---------- ------------------------------ ------------------------------
INF        Informática                    Planta sótano U3
```

Como podemos ver ha funcionado perfectamente y el departamento de Informática ya está en la tabla. Incluyamos ahora el segundo registro sin especificar los campos:

```
SQL> insert into departamentos
  2  values('ADM','Administración','Planta quinta U2');

1 row created.
```

El registro se creó satisfactoriamente y comprobamos el contenido de la tabla con SELECT.

```
SQL> select * from departamentos;

CODDPTO    NOMBRE                         UBICACION
---------- ------------------------------ ------------------------------
INF        Informática                    Planta sótano U3
ADM        Administración                 Planta quinta U2
```

La inserción extendida que hemos visto en MySQL no se puede aplicar en Oracle. Pero Oracle también tiene su forma de poder insertar varios registros seguidos. Sin embargo, desde mi punto de vista el ahorro de código a teclear es prácticamente nulo. Es casi lo mismo que hacer varios insert individuales en un script normal.

Para insertar los dos registros en Oracle podemos utilizar:

```
INSERT ALL
    INTO departamentos values('COM','Comercial','Planta tercera U3')
    INTO departamentos values('CONT','Contabilidad','Planta quinta U1')
SELECT * FROM DUAL;
```

La tabla DUAL es una tabla especial de una sola columna presente de manera predeterminada en todas las instalaciones de Oracle.

Lo guardamos en un archivo y ejecutamos el script correspondiente.

```
SQL> @ c:\src\insertvarios.sql

2 rows created.
```

Después podemos observar el resultado.

```
SQL> SELECT * FROM DEPARTAMENTOS;

CODDPTO    NOMBRE                         UBICACION
---------- ------------------------------ ------------------------------
COM        Comercial                      Planta tercera U3
CONT       Contabilidad                   Planta quinta U1
INF        Informática                    Planta sótano U3
ADM        Administración                 Planta quinta U2
```

1.3 Errores en la inserción

A la hora de insertar datos se pueden producir errores por diversas causas, algunos errores que irán en contra de las reglas establecidas mostrarán el error y ya está, pero otros, los más difíciles de encontrar son los que producimos los usuarios y que para la base de datos pueden pasar como

correctos.

Un error típico puede ser cuando repetimos la inserción de un registro que ya existe. Como la clave primaria debe ser única debe mostrar un error al intentar insertar un registro con la misma clave.

1.3.1 MySQL

```
mysql> select * from departamentos;
+---------+----------------+--------------------+
| CodDpto | Nombre         | Ubicacion          |
+---------+----------------+--------------------+
| ADM     | Administración | Planta quinta U2   |
| COM     | Comercial      | Planta tercera U3  |
| CONT    | Contabilidad   | Planta quinta U1   |
| INF     | Informática    | Planta sótano U3   |
+---------+----------------+--------------------+
4 rows in set (0.00 sec)

mysql> insert into departamentos values('INF','Informática','Planta sótano U3');

ERROR 1062 (23000): Duplicate entry 'INF' for key 1
mysql>
```

Como podéis ver en la imagen aparece el error de entrada duplicada para la clave 1.

Puede ocurrir que al realizar un alta o inserción de un registro no rellenemos todos los campos de la tabla. Por ejemplo supongamos que vamos a tener un nuevo departamento de Marketng con el código MKT pero que no saber aún su ubicación. Podemos hacer su alta de la siguiente forma:

```
mysql> insert into departamentos (CodDpto, Nombre)
    -> values ('MKT','Marketing');
Query OK, 1 row affected (0.06 sec)

mysql> select * from departamentos;
+---------+----------------+--------------------+
| CodDpto | Nombre         | Ubicacion          |
+---------+----------------+--------------------+
| ADM     | Administración | Planta quinta U2   |
| COM     | Comercial      | Planta tercera U3  |
| CONT    | Contabilidad   | Planta quinta U1   |
| INF     | Informática    | Planta sótano U3   |
| MKT     | Marketing      | NULL               |
+---------+----------------+--------------------+
5 rows in set (0.03 sec)
```

Como puedes observar en el insert solamente hemos colocado el nombre de los dos campos que conocemos y el tercero, la ubicación se ha puesto a NULL.

Supongamos ahora que conocemos el código de otro nuevo departamento, AUD, pero no tenemos claro el nombre que va a tener, aunque su ubicación será en la Planta cuarta U1. Veamos

cómo hacer el INSERT.

Como habrás imaginado, la nueva instrucción será igual que la anterior, pero cambiando el campo Nombre por el campo Ubicación:

```
mysql> insert into departamentos (CodDpto, Ubicacion)
    -> values ('AUD','Planta cuarta U1');
ERROR 1364 (HY000): Field 'Nombre' doesn't have a default value
mysql>
```

Vaya, esto si que es un error. Dice que Nombre no tene un valor por defecto. ¿Qué es esto?

Pues bien, si miramos la estructura de nuestra tabla:

```
mysql> desc departamentos;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| CodDpto   | varchar(10) | NO   | PRI | NULL    |       |
| Nombre    | varchar(30) | NO   |     | NULL    |       |
| Ubicacion | varchar(30) | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
3 rows in set (0.03 sec)

mysql>
```

Podemos ver que el campo nombre no puede ser NULO ya que se creo con NOT NULL, por otro lado no tener establecido ningún valor por defecto (aparece NULL) luego no podemos insertar un registro que no tenga un valor para el campo Nombre, es decir el valor de ese campo es obligatorio incluirlo en el INSERT.

Pero, tal y como he comentado antes, el error más difícil de encontrar es el error que cometemos los usuarios sin que para la base de datos sea considerado un error. Por ejemplo, si hacemos el siguiente INSERT:

```
mysql> insert into departamentos
    -> values('ALM','Planta baja U1','Almacén');
Query OK, 1 row affected (0.02 sec)
```

Vemos que no hay ningún mensaje de error, la instrucción INSERT es correcta y se añade el nuevo registro indicado. Vamos a comprobar los datos para verificarlo.

```
mysql> select * from departamentos;
+---------+----------------+--------------------+
| CodDpto | Nombre         | Ubicacion          |
+---------+----------------+--------------------+
| ADM     | Administración | Planta quinta U2   |
| ALM     | Planta baja U1 | Almacén            |
| COM     | Comercial      | Planta tercera U3  |
| CONT    | Contabilidad   | Planta quinta U1   |
| INF     | Informática    | Planta sótano U3   |
| MKT     | Marketing      | NULL               |
+---------+----------------+--------------------+
6 rows in set (0.00 sec)
```

Aparentemente todo parece correcto, pero ¿qué ha ocurrido con el segundo registro, el de código ALM? Fíjate, donde está el nombre aparece la ubicación y en ubicación aparece el nombre. Esto ha ocurrido porque espera los datos de los campos en el mismo orden en el que están en la tabla ya que no hemos incluido el nombre de los campos en la instrucción.

Por ello, aunque sea un poco más pesado, mi consejo es que siempre que realices un INSERT incluyas los nombres de los campos, de tal forma que cuando vayas escribiendo los valores de los campos veas que el orden se corresponde con los campos indicados.

1.3.2 Oracle

```
SQL> select * from departamentos;

CODDPTO    NOMBRE                         UBICACION
---------- ------------------------------ ------------------------------
INF        Informática                    Planta sótano U3
ADM        Administración                 Planta quinta U2
COM        Comercial                      Planta tercera U3
CONT       Contabilidad                   Planta quinta U1

SQL> insert into departamentos
  2  values('INF','Informática','Planta sótano U3');
insert into departamentos
*
ERROR at line 1:
ORA-00001: unique constraint (USUARIO_PRUEBA.SYS_C006997) violated


SQL> _
```

El error que muestra Oracle es diferente pero con el mismo significado, indica que la restricción de unicidad está siendo incumplida.

```
SQL> insert into departamentos (CodDpto, Nombre)
  2  values ('MKT', 'Marketing');

1 row created.

SQL> select * from departamentos;

CODDPTO    NOMBRE                         UBICACION
---------- ------------------------------ ------------------------------
INF        Informática                    Planta sótano U3
ADM        Administración                 Planta quinta U2
COM        Comercial                      Planta tercera U3
CONT       Contabilidad                   Planta quinta U1
MKT        Marketing
```

En Oracle, no aparece el valor nulo al hacer el SELECT pero deja el campo en blanco. Veamos ahora cómo reacciona Oracle al introducir sólo los valores para CODDPTO y UBICACION:

```
SQL> insert into departamentos (CodDpto,Ubicacion)
  2  values ('AUD','Planta cuarta U1');
insert into departamentos (CodDpto,Ubicacion)
*
ERROR at line 1:
ORA-01400: cannot insert NULL into ("USUARIO_PRUEBA"."DEPARTAMENTOS"."NOMBRE")
```

Pero fíjate en la información que aporta el mensaje de error, dice que no puede insertar NULO en el esquema USUARIO_PRUEBA, dentro de la tabla DEPARTAMENTOS, en el campo NOMBRE.

Si ahora ejecutamos el siguiente insert:

```
SQL> insert into departamentos
  2  values('ALM','Planta baja U1','Almacén');

1 row created.
```

2. MODIFICACIÓN DE DATOS

Bien, en el apartado anterior nos hemos equivocado al insertar los datos, ahora tendremos que modificarlos para dejarlos correctos. Esto implica actualizar con nuevos datos el registro erróneo y para ello SQL nos suministra la instrucción UPDATE. La sintaxis que utilizaremos será:

```
UPDATE Nombre_tabla
SET columna1=valor1 [, columna2=valor2]...
[WHERE condición]
```

La instrucción indica que queremos actualizar (UPDATE) una tabla (Nombre_tabla) estableciendo (SET) los campos que queremos modificar a una expresión o valor determinado. Podemos colocar tantos campos con sus nuevos valores separados por comas como necesitemos.

Por último, aunque es opcional, debemos incluir la cláusula WHERE condición que nos permitirá seleccionar sobre qué registro se debe realizar la actualización.

> CUIDADO: Si no ponemos el WHERE la actualización se realizará en todos los registros de la tabla

Veamos cómo podemos cambiar en nuestra tabla los datos que aún están pendientes de introducir. Para realizar la actualización del registro de Marketing haremos:

```
UPDATE departamentos
SET ubicacion = 'Planta cuarta U5'
WHERE CodDpto = 'MKT';
```

Aunque las condiciones las veremos con muchísima más profundidad en el próximo tema, tenemos que saber que WHERE CodDpto = 'MKT' indica que la actualización se aplicará a los registros cuyo código de departamento sea igual a MKT.

En la condición se puede utliiar cualquiera de los siguientes operadores de comparación:

Operador	Significado
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
=	Igual
!=	Distinto
<>	Distinto

Además se puede utilizar:

Operador	Significado
AND	Devuelve verdadero si las expresiones a su izquierda y derecha son ambas verdaderas.
OR	Devuelve verdadero si cualquiera de las dos expresiones a izquierda y derecha del OR, son verdaderas.
NOT	Invierte la lógica de la expresión que está a su derecha. Si era verdadera, mediante NOT pasa a ser falso.

2.1 MySQL

Probamos con MySQL.

```
mysql> update departamentos
    -> set ubicacion = 'Planta cuarta U5'
    -> where CodDpto = 'MKT';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0
```

Vemos el resultado de la actualización.

```
mysql> select * from departamentos;
+---------+----------------+--------------------+
| CodDpto | Nombre         | Ubicacion          |
+---------+----------------+--------------------+
| ADM     | Administración | Planta quinta U2   |
| ALM     | Planta baja U1 | Almacén            |
| COM     | Comercial      | Planta tercera U3  |
| CONT    | Contabilidad   | Planta quinta U1   |
| INF     | Informática    | Planta sótano U3   |
| MKT     | Marketing      | Planta cuarta U5   |
+---------+----------------+--------------------+
6 rows in set (0.00 sec)
```

Podemos apreciar cómo se ha modificado la ubicación del departamento de Marketing.

Bien, ahora vamos a realizar la modificación del registro del departamento de Almacén cambiando los dos campos en la misma instrucción.

Fíjate en la condición, el registro que queremos modificar es el del Almacén, por ello, la condición será CodDpto = 'ALM'.

```
mysql> UPDATE departamentos
    -> SET nombre = 'Almacén',
    -> ubicacion = 'Planta baja U1'
    -> WHERE CodDpto = 'ALM';
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Comprobamos el resultado.

```
mysql> select * from departamentos;
+---------+----------------+--------------------+
| CodDpto | Nombre         | Ubicacion          |
+---------+----------------+--------------------+
| ADM     | Administración | Planta quinta U2   |
| ALM     | Almacén        | Planta baja U1     |
| COM     | Comercial      | Planta tercera U3  |
| CONT    | Contabilidad   | Planta quinta U1   |
| INF     | Informática    | Planta sótano U3   |
| MKT     | Marketing      | Planta cuarta U5   |
+---------+----------------+--------------------+
6 rows in set (0.00 sec)
```

Podemos observar que el cambio se ha realiiado y ahora tenemos todos los datos correctos.

2.2 Oracle

Probemos ahora con Oracle.

```
SQL> update departamentos
  2  set ubicacion = 'Planta cuarta U5'
  3  where CodDpto = 'MKT';

1 row updated.
```

Oracle nos muestra un mensaje indicando que una de las filas ha sido actualizada.

Comprobamos el resultado.

```
SQL> select * from departamentos;

CODDPTO    NOMBRE                         UBICACION
---------- ------------------------------ ------------------------------
INF        Informática                    Planta sótano U3
ADM        Administración                 Planta quinta U2
COM        Comercial                      Planta tercera U3
CONT       Contabilidad                   Planta quinta U1
MKT        Marketing                      Planta cuarta U5
ALM        Planta baja U1                 Almacén

6 rows selected.
```

Si realizamos el siguiente cambio:

```
SQL> update departamentos
  2  set nombre = 'Almacén',
  3  ubicacion = 'Planta baja U1'
  4  where CodDpto = 'ALM';

1 row updated.
```

Comprobamos el resultado:

```
SQL> select * from departamentos;

CODDPTO    NOMBRE                         UBICACION
---------- ------------------------------ --------------------------
INF        Informática                    Planta sótano U3
ADM        Administración                 Planta quinta U2
COM        Comercial                      Planta tercera U3
CONT       Contabilidad                   Planta quinta U1
MKT        Marketing                      Planta cuarta U5
ALM        Almacén                        Planta baja U1

6 rows selected.
```

3. BORRADO DE DATOS

Para eliminar un registro utliiaremos la instrucción DELETE de la siguiente forma:

```
DELETE FROM nombre_tabla
[WHERE condición]
```

Recordad que si no ponemos la condición se eliminarán todos los registros de la tabla!!!

Imaginemos que queremos eliminar el registro correspondiente al departamento de Marketing,ejecutaríamos la siguiente instrucción:

```
DELETE FROM departamentos
WHERE CodDpto = 'MKT';
```

3.1 MySQL

```
mysql> delete from departamentos
    -> where CodDpto = 'MKT';
Query OK, 1 row affected (0.03 sec)
```

Comprobamos el resultado, el departamento de Marketing ha sido eliminado.

```
mysql> select * from departamentos;
+---------+----------------+--------------------+
| CodDpto | Nombre         | Ubicacion          |
+---------+----------------+--------------------+
| ADM     | Administración | Planta quinta U2   |
| ALM     | Almacén        | Planta baja U1     |
| COM     | Comercial      | Planta tercera U3  |
| CONT    | Contabilidad   | Planta quinta U1   |
| INF     | Informática    | Planta sótano U3   |
+---------+----------------+--------------------+
5 rows in set (0.00 sec)
```

3.2 Oracle

```
SQL> delete from departamentos
  2  where CodDpto = 'MKT';

1 row deleted.
```

Comprobamos el resultado, el departamento de Marketing ha sido eliminado.

```
SQL> select * from departamentos;

CODDPTO    NOMBRE                         UBICACION
---------- ------------------------------ --------------------
INF        Informática                    Planta sótano U3
ADM        Administración                 Planta quinta U2
COM        Comercial                      Planta tercera U3
CONT       Contabilidad                   Planta quinta U1
ALM        Almacén                        Planta baja U1
```

4. TRANSACCIONES

Como se ha comentado anteriormente, una transacción está formada por una serie de instrucciones DML. Una transacción comienza con la primera instrucción DML que se ejecute y finaliza con alguna de estas circunstancias:

- Una operación COMMIT o ROLLBACK
- Una instrucción DDL (como ALTER TABLE por ejemplo)
- Una instrucción DCL (como GRANT)
- El usuario abandona la sesión
- Caída del sistema

Hay que tener en cuenta que cualquier instrucción DDL o DCL da lugar a un COMMIT implícito, es decir todas las instrucciones DML ejecutadas hasta ese instante pasan a ser defnitvas.

4.1 COMMIT

La instrucción COMMIT hace que los cambios realizados por la transacción sean definitivos, irrevocables. Sólo se debe utilizar si estamos de acuerdo con los cambios, conviene asegurarse mucho antes de realizar el COMMIT ya que las instrucciones ejecutadas pueden afectar a miles de registros.

Además el cierre correcto de la sesión da lugar a un COMMIT, aunque siempre conviene ejecutar explícitamente esta instrucción a fin de asegurarnos de lo que hacemos.

4.2 ROLLBACK

Esta instrucción regresa a la instrucción anterior al inicio de la transacción, normalmente el último COMMIT, la última instrucción DDL o DCL o al inicio de sesión.

Anula definitivamente los cambios, por lo que conviene también asegurarse de esta operación.

Un abandono de sesión incorrecto o un problema de comunicación o una caída del sistema dan lugar a un ROLLBACK implícito.

4.3 Estado de los datos durante la transacción

Si se inicia una transacción usando comandos DML hay que tener en cuenta que:

- Se puede volver a la instrucción anterior a la transacción cuando se desee.
- Las instrucciones de consulta SELECT realizadas por el usuario que inició la transacción muestran los datos ya modificados por las instrucciones DML.
- El resto de usuarios ven los datos tal cual estaban antes de la transacción, de hecho los registros afectados por la transacción aparecen bloqueados hasta que la transacción finalice. Esos usuarios no podrán modificar los valores de dichos registros.
- Tras la transacción todos los usuarios ven los datos tal cual quedan tras el fin de transacción. Los bloqueos son liberados y los puntos de ruptura borrados.

4.4 COMMIT y ROLLBACK en MySQL

Por defecto, la conexión con el servidor MySQL comienza con el modo de autocommit habilitado, el cual automáticamente confirma cada sentencia SQL mientras lo ejecuta. Este modo de operación puede ser desconcertante si se tiene experiencia con otros sistemas de bases de datos, donde es una práctica estándar emitir una secuencia de instrucciones DML y confirmarlas (COMMIT) o deshacerlas (ROLLBACK) todas juntas.

Para trabajar en el modo en que lo hace Oracle, es decir, para utilizar transacciones de sentencias múltiples, desactivaremos autocommit con la instrucción SQL SET autocommit = 0 y finalizaremos cada transacción con COMMIT o ROLLBACK según corresponda.

Cuando trabajemos con autocommit activado, comenzaremos cada transacción con START

TRANSACTION y la finaliza con COMMIT o ROLLBACK. El siguiente ejemplo muestra dos transacciones, la primera finaliza con COMMIT y la segunda con ROLLBACK.

```
mysql> CREATE TABLE customer (a INT, b CHAR (20), INDEX (a));
Query OK, 0 rows affected (0.00 sec)
mysql> -- Do a transaction with autocommit turned on.
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer VALUES (10, 'Heikki');
Query OK, 1 row affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql> -- Do another transaction with autocommit turned off.
mysql> SET autocommit=0;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO customer VALUES (20, 'Paul');
Query OK, 1 row affected (0.00 sec)
mysql> DELETE FROM customer WHERE b = 'Heikki';
Query OK, 1 row affected (0.00 sec)
mysql> -- Now we undo those last 2 inserts and the delete.
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM customer;
+------+--------+
| a | b
|
+------+--------+
| 10 | Heikki |
+------+--------+
1 row in set (0.00 sec)
mysql>
```

4.5 COMMIT y ROLLBACK en ORACLE

Debido a que la causa más frecuente de error durante la ejecución de una transacción es la violación de restricciones de integridad definidas en el esquema de la base de datos, es importante conocer la estrategia seguida por el sistema para realizar esta comprobación.

La comprobación de una restricción de integridad se puede realizar después de la ejecución de cada instrucción SQL relevante para la restricción, modo inmediato (IMMEDIATE) o después de la finalización de cualquier transacción que incluya una instrucción SQL relevante para la restricción, modo diferido (DEFERRED).

El modo de una restricción se puede cambiar localmente para una transacción si la restricción se ha definido como diferible (DEFERRABLE) . El modo de una restricción se define en el esquema de la base de datos con la cláusula [cuando comprobar] :

```
cuando comprobar:= [ [NOT] DEFERRABLE] [INITIALLY {IMMEDIATE | DEFERRED} ]
```

Si no se utiliza esta cláusula el valor por defecto para una restricción es el modo IMMEDIATE y la restricción no es DEFERRABLE. Es decir, si hay un error se deshacen todos los cambios y se comprueban las restricciones una a una. sin esperar al commit.

Si durante la ejecución de una transacción se viola una restricción con modo IMMEDIATE inmediato el sistema deshace el efecto de la operación SQL que ha causado la violación de la restricción (statement rollback) y la transacción puede continuar; si al finalizar una transacción se viola una restricción con modo DEFERRED diferido el sistema anula la transacción y deshace su efecto global (transaction rollback)

Ejemplo de uso de transacciones con oracle. Este ejemplo permite crear un empleado e indicar con quien esta casado.

Creación de tabla con el fichero empleado.sql

```
create table empleado (
dni char (3) ,
nombre varchar (25) ,
edad number (2) ,
casadocon char (3)  ,
constraint pk_empleado primary key ( dni ) ,
constraint fk_empemp foreign key ( casadocon ) references empleado ( dni )
);
```

Inserción de filas con el fichero empleado1.sql

```
insert into empleado ( dni , nombre , edad , casadocon ) \
values ( '1' , ' Pepe ' ,35 , '2' );
insert into empleado ( dni , nombre , edad , casadocon ) \
values ( '2' , ' Juana ' ,26 , '1' );
commit ;
```

Como hace la comprobación de clave ajena la cual no existe, sale este error:

```
insert into empleado ( dni , nombre , edad , casadocon ) \
values ( '1' , ' Pepe ' ,35 , '2 ')
*
ERROR at line 1:
ORA -02291: integrity constraint ( PRUEBA . FK_EMPEMP ) \
violated - parent key not found
insert into empleado ( dni , nombre , edad , casadocon ) \
values ( '2' , ' Juana ' ,26 , '1 ')
*
ERROR at line 1:
ORA -02291: integrity constraint ( PRUEBA . FK_EMPEMP ) \
violated - parent key not found
```

Veamos la forma de solucionarlo usando transacciones. Crear fichero: ed empleado2.sql

```
DROP TABLE EMPLEADO ;
create table empleado (
dni char (3) ,
nombre varchar (25) ,
edad number (2) ,
casadocon char (3) ,
constraint pk_empleado primary key ( dni ) ,
constraint fk_empemp foreign key ( casadocon ) references \
empleado ( dni ) DEFERRABLE
);
set constraint all deferred ;
```

```
insert into empleado ( dni , nombre , edad , casadocon ) \
values ( '1' , ' Pepe ' ,35 , '2' );
insert into empleado ( dni , nombre , edad , casadocon ) \
values ( '2' , ' Juana ' ,26 , '1' );
commit ;
```

Se puede ver que con el comando set constraint all deferred ; podemos hacer que se comprueben las restricciones al final.

Ejecutarlo: @empleado2

```
Commit complete .
Constraint set .
1 row created .
1 row created .
Commit complete .
```

Resultado:

```
select * from empleado;

DNI NOMBRE                          EDAD CAS
--- ------------------------ ---------- ---
1   Pepe                             35 2
2   Juana                            26 1
```

5. OTRAS INSTRUCCIONES DDL

5.1 Índices

Los índices son objetos que forman parte del esquema que hacen que las bases de datos aceleren las operaciones de consulta y ordenación sobre los campos a los que el índice hace referencia sin necesidad de revisar todos los datos disponibles para devolver el resultado.

Se almacenan aparte de la tabla a la que hace referencia, lo que permite crearlos y borrarlos en cualquier momento.

Lo que realizan es una lista ordenada por la que el SGBD puede acceder para facilitar la búsqueda de los datos. Cada vez que se añade un nuevo registro, los índices involucrados se actualizan a fin de que su información esté al día. De ahí que cuantos más índices haya, más le cuesta al SGBD añadir registros, pero más rápidas se realizan las instrucciones de consulta.

La mayoría de los índices se crean de manera implícita, como consecuencia de las restricciones PRIMARY KEY, UNIQUE y FOREIGN KEY. Estos son índices obligatorios, por los que los crea el propio SGBD.

5.1.1 Creación de índices

Aparte de los índices obligatorios comentados anteriormente, se pueden crear índices de forma explícita. Éstos se crean para aquellos campos sobre los cuales se realizarán búsquedas e instrucciones de ordenación frecuente.

Veamos la sintaxis completa:

```
CREATE [UNIQUE|BITMAP] INDEX [esquema.]index_name
ON [esquema.]table_name [tbl_alias]
(col [ASC | DESC])
```

Y la sintaxis sencilla:

```
CREATE INDEX nombre
ON tabla (columna1 [,columna2...])
```

Ejemplo:

```
CREATE INDEX nombre_completo
ON clientes (apellido1, apellido2, nombre);
```

El ejemplo crea un índice para los campos apellido1, apellido2 y nombre. Esto no es lo mismo que crear un índice para cada campo, este índice es efectivo solo cuando se buscan u ordenan clientes usando los tres campos (apellido1, apellido2 y nombre) a la vez.

Se aconseja crear índices en campos que:

- Contengan una gran cantidad de valores
- Contengan una gran cantidad de nulos
- Sean parte habitual de cláusulas WHERE, GROUP BY u ORDER BY
- Sean parte de listados de consultas de grandes tablas sobre las que casi siempre se muestran como mucho un 4% de su contenido.

No se aconseja en campos que:

- Pertenezcan a tablas pequeñas
- No se usen a menudo en las consultas
- Pertenezcan a tablas cuyas consultas muestren menos de un 4% del total de registros
- Pertenezcan a tablas que se actualicen frecuentemente
- Se utilicen en expresiones

Los índices se pueden crear utilizando expresiones complejas:

```
CREATE INDEX nombre_complejo
ON clientes (UPPER(nombre));
```

Esos índices tienen sentido si en las consultas se utilizan exactamente esas expresiones.

5.1.2 Lista de índices

Para ver la lista de índices en Oracle se utliia la vista USER_INDEXES. Mientras que la vista USER_IND_COLUMNS muestra la lista de columnas que son utilizadas por índices.

En MySQL utliiamos el comando SHOW INDEX que nos devolverá la información de la tabla de índices. Su sintaxis es la siguiente:

```
SHOW {INDEX | INDEXES | KEYS}
{FROM | IN} tbl_name
[{FROM | IN} db_name]
[WHERE expr]
```

5.1.3 Borrar índices

La instrucción DROP INDEX seguida del nombre del índice permite eliminar el índice en cuestión.