Tema2. Elaboración del diseño físico

Contenido

1 El lenguaje SQL

En los temas anteriores hemos visto que mediante el **modelo Entidad-Relación** se pueden **modelar** situaciones del mundo real, de manera que disponemos de una herramienta gráfica para trasladar los elementos de un sistema de información **y sus relaciones**, a un esquema manejable que puede ser fácilmente interpretado por cualquiera que conozca las reglas por las que se rige. Luego hemos comprobado cómo se puede **trasladar ese modelo Entidad-Relación** a otro esquema de información más orientado a su tratamiento, como es el **modelo relacional**.

Para almacenar y tratar la información esquematizada en los modelos E-R y relacional por medio de un Sistema Gestor de Bases de Datos se utiliza **SQL**.

- **SQL** es un lenguaje que nos **permite interactuar con los SGBD Relacionales** para especificar las operaciones que deseamos realizar sobre los datos y su estructura.
- SQL son las siglas de **S**tructured **Q**uery **L**anguage (**Lenguaje de Consulta Estructurado).**
- Es un **lenguaje declarativo,** lo cual quiere decir que en él se especifica al Sistema Gestor de Base de Datos **qué queremos obtener** y no la manera de cómo conseguirlo.
- Es un **lenguaje no procedimental** porque no necesitamos especificar el procedimiento para conseguir el objetivo, sino el objetivo en sí. **No es un lenguaje de programación** como puede ser Java o C.

Hoy día **SQL es el lenguaje de consulta y manipulación de datos más extendido** y utilizado por todos los **desarrolladores y fabricantes de SGBD**. A lo largo de los años se han ido ampliando sus características conforme se sucedían avances en la tecnología informática.

SQL no es propiedad de ningún fabricante, sino que es una norma a seguir y los fabricantes de software no suelen implementar SQL puro en sus productos, sino que a menudo incorporan pequeñas variaciones para conseguir funcionalidades concretas en sus desarrollos. Esto hace que lo que debería ser un estándar no lo sea completamente en la realidad.

SQL es el lenguaje que usan los SGBDR en la actualidad.

1.1 ¿Cómo se usa SQL?

No hay que olvidar que estamos estudiando un lenguaje de interacción con bases de datos y que la herramienta para acceder y manipular esas bases de datos son los sistemas gestores de base de datos.

Según la normativa ANSI/ISO cuando se ejecuta SQL, existen los siguientes elementos a tener en cuenta en todo el entorno involucrado en la ejecución de instrucciones SQL:

- Un **agente SQL**. Entendido como cualquier elemento que cause la ejecución de instrucciones SQL que serán recibidas por un cliente SQL (**es un servicio** que ejecuta tareas administrativas programadas, denominadas trabajos en SQL, en el lado del servidor)
- Una **implementación SQL**. Se trata de un procesador software capaz de ejecutar las instrucciones pedidas por el agente SQL. Una implementación está compuesta por:
    - Un **cliente SQL**. Software conectado al agente que funciona como interfaz entre el agente SQL y el servidor SQL. Sirve para establecer conexiones entre sí mismo y el servidor SQL.
    - Un **servidor SQL** (puede haber varios). El software encargado de manejar los datos a los que la instrucción SQL lanzada por el agente hace referencia. Es el software que realmente realiza la instrucción, los datos los devuelve al cliente.

Los SGBDR permiten dos modos de acceso a las bases de datos:

- **Modo interactivo**, destinado principalmente a los usuarios finales, avanzados u ocasionales, en el que **las diversas sentencias SQL se introducen a través de un cliente** que está directamente **conectado al servidor SQL;** por lo que las instrucciones se traducen sin intermediarios utilizando **un intérprete de órdenes y los resultados se muestran en el cliente**.
- **Modo embebido**, destinado al uso por parte de los **programadores. En este caso las sentencias SQL se introducen en lenguajes de programación, llamados lenguajes anfitrión** (por ejemplo Java, lenguajes de la plataforma .NET de Microsoft, PHP, C++, etc.), de manera que el **resultado es una mezcla de ambos**. En este caso **el lenguaje anfitrión aporta lo que le falta a SQL, es decir la programación**. Al compilar el código se utiliza un precompilador de la propia base de datos para traducir el SQL y conectar la aplicación resultado con la base de datos a través de un software adaptador (**driver**) como **JDBC** u **ODBC** por ejemplo.

1.2 Componentes del lenguaje SQL

El lenguaje SQL está compuesto por **sentencias**. Esas sentencias se pueden clasificar en tres grupos:

- **Sentencias DDL (Lenguaje de Definición de Datos):** Sirven para **crear, modificar y borrar elementos estructurales en los SGBDR**, como:
  - bases de datos,
  - tablas,
  - índices,
  - restricciones, etc.

  Las definiciones de esos objetos quedan almacenadas en el diccionario de datos del sistema.

- **Sentencias DML (Lenguaje de Manipulación de Datos):** Nos permiten indicar al sistema las **operaciones que queremos realizar con los datos almacenados en las estructuras creadas por medio de las sentencias DDL**. Por ejemplo son las sentencias que permitirán:
  - generar consultas,
  - ordenar,
  - filtrar,
  - añadir,
  - modificar,
  - borrar,
  - etc.
- **Sentencias DCL (Lenguaje de Control):** Un conjunto de sentencias orientado a gestionar todo lo relativo a:
  - usuarios,
  - permisos,
  - seguridad,
  - etc.

Con el tiempo han surgido **nuevas necesidades** en los SGBDR que han obligado a incorporar **nuevas sentencias** que no se pueden clasificar en los tres grupos clásicos anteriores**, cómo son las Sentencias para gestión de transacciones y bloqueos, las sentencias para replicación, etc.**

Las sentencias SQL a su vez se construyen a partir de:

- **Cláusulas:** Que modifican el comportamiento de las sentencias. Constan de palabras reservadas y alguno de los siguientes elementos:

  - **Operadores lógicos y de comparación:** Sirven para ligar operandos y producir un resultado booleano (verdadero o falso).
  - **Funciones de agregación:** Para realizar operaciones sobre un grupo de filas de una tabla.
  - **Funciones:** Para realizar cálculos y operaciones de transformación sobre los datos.
  - **Expresiones:** Construidas a partir de la combinación de operadores, funciones, literales y nombres de columna.
- **Metadatos**. Obtenidos de la propia base de datos

Estudiaremos y practicaremos gran parte de las sentencias SQL, y por supuesto las más importantes y utilizadas. Aunque su estudio completo no es posible aquí, por su elevadísimo número y casuística de cada una.

1.3 Formato de las instrucciones en los apuntes

En este tema en muchos apartados se indica sintaxis de comandos. Esta sintaxis sirve para aprender a utilizar el comando, e indica la forma de escribirlo en el programa utilizado para escribir SQL. Ejemplo:

SELECT * | {[**DISTINCT**] ***columna*** | ***expresión*** [***alias***], ...}
FROM ***tabla*;**

Otras veces se describen códigos de ejemplo de un comando. Ejemplo:

SELECT nombre **FROM** cliente**;**

Los ejemplos sirven para escenificar una instrucción concreta, la sintaxis se utiliza para indicar las formas de utilizar un comando.

Para indicar la sintaxis de un comando se usan símbolos especiales. Los símbolos que se utilizan normalmente en cualquier documentación de este tipo son:

PALABRA Cuando en la sintaxis se utiliza una palabra en negrita, significa que es una palabra que hay que escribir literalmente (aunque sin importar si en mayúsculas o minúsculas).

texto. El texto que aparece en color normal sirve para indicar que no hay que escribirlo literalmente, sino que se refiere a un tipo de elemento que se puede utilizar en el comando. Ejemplo:

SELECT columna **FROM** tabla**;**

El texto *`columna`* hay que cambiarlo por un nombre concreto de columna (`nombre, apellidos`,...), al igual que *`tabla`* se refiere a un nombre de tabla concreto.

texto en negrita. Sirve para indicar texto o símbolos que hay que escribir de forma literal, pero que no son palabras reservadas del lenguaje.

[ ] (corchetes). Los corchetes sirven para encerrar texto que no es obligatorio en el comando, es decir para indicar una parte opcional.

| (barra vertical). Este símbolo (|), la barra vertical, indica opción. Las palabras separadas con este signo indican que se debe elegir una de entre todas las palabras.

... (puntos suspensivos) Indica que se puede repetir el texto anterior en el comando continuamente (significaría, **y así sucesivamente**)

{} (llaves) Las llaves sirven para indicar opciones mutuamente exclusivas pero obligatorias. Es decir, opciones de las que sólo se puede elegir una opción, pero de las que es obligado elegir una. Ejemplo:

```
SELECT { * | columna | expresión }
FROM tabla;
```

El ejemplo anterior indicaría que se debe elegir obligatoriamente el asterisco o un nombre de columna o una expresión. Si las llaves del ejemplo fueran corchetes, entonces indicarían que incluso podría no aparecer ninguna opción.

2 Normas de escritura

- En SQL no se distingue entre mayúsculas y minúsculas.
- Las instrucciones finalizan con el signo de punto y coma.
- Cualquier comando SQL (SELECT, INSERT,...) puede ser partidos por espacios o saltos de línea antes de finalizar la instrucción
- Se pueden tabular líneas para facilitar la lectura si fuera necesario.
- Los comentarios en el código SQL comienzan por /* y terminan por */ (excepto en algunos SGBD)

3 Lenguaje de descripción de datos. DDL

SQL es un lenguaje declarativo en el que lo importante es definir qué se desea hacer, por encima de cómo hacerlo (que es la forma de trabajar de los lenguajes de programación de aplicaciones como C o Java). Con este lenguaje se pretendía que las instrucciones se pudieran escribir como si fueran órdenes humanas; es decir, utilizar un lenguaje lo más natural posible. De ahí que se le considere un lenguaje de cuarta generación.

Se trata de un lenguaje que intenta agrupar todas las funciones que se le pueden pedir a una base de datos, por lo que es el lenguaje utilizado tanto por administradores como por programadores o incluso usuarios avanzados.

3.1 Creación de tablas

No puede haber dos tablas con el mismo nombre para el mismo esquema (pueden coincidir los nombres si están en distintos esquemas)

No puede coincidir con el nombre de una palabra reservada SQL (por ejemplo no se puede llamar SELECT a una tabla)

Es la orden SQL que permite crear una tabla. Por defecto será almacenada en el espacio y esquema del usuario que crea la tabla. Sintaxis:

```
CREATE TABLE [esquema.] nombreDeTabla
(nombreDeLaColumna1 tipoDeDatos [, ...]);
```

Ejemplo:

```
CREATE TABLE proveedores (nombre VARCHAR(25));
```

Crea una tabla con un solo campo de tipo VARCHAR.

Sólo se podrá crear la tabla si el usuario posee los permisos necesarios para ello. Si la tabla pertenece a otro esquema (suponiendo que el usuario tenga permiso para grabar tablas en ese otro esquema), se antepone al nombre de la tabla , el nombre del esquema:

```
CREATE TABLE otroUsuario.proveedores (nombre VARCHAR(25));
```

3.2 Tipos de datos

A la hora de crear tablas, hay que indicar el tipo de datos de cada campo. Necesitamos pues, conocer los distintos tipos de datos. Estos son:

Descripción	Tipos ANSI SQL	SQL Server	Oracle SQL	Tipo MySQL
Texto de anchura fija	CHARACTER(*n*) CHAR(*n*)	CHAR(*n*)	CHAR(*n*)	CHAR(*n*)
Texto de anchura variable	CHARACTER VARYING(*n*) CHAR VARYING(*n*)	VARCHAR(*n*)	VARCHAR2(*n*)	VARCHAR(*n*)
Texto de anchura fija para caracteres nacionales	NATIONAL CHARACTER(*n*) NATIONAL CHAR(*n*) NCHAR(*n*)	NCHAR(*n*)	NCHAR(*n*)	
Texto de anchura variable para caracteres nacionales	NATIONAL CHARACTER VARYING(*n*) NATIONAL CHAR VARYING(*n*) NCHAR VARYING(*n*)	NVARCHAR(*n*)	NVARCHAR2(*n*)	
Enteros	INTEGER INT SMALLINT	INT INT SMALL INT	NUMBER(*38*)	INT SMALL INT TINY INT
Decimal de coma variable	FLOAT(*b*) DOUBLE DOUBLE PRECISSION REAL	FLOAT	NUMBER	FLOAT(*m,d*) DOUBLE(*m,d*)
Decimal de coma fija	NUMERIC(*m,d*) DECIMAL(*m.d*)	NUMERIC(*m,d*) DECIMAL(*m.d*)	NUMBER(*m,d*)	DECIMAL(*m.d*)
Fechas	DATE		DATE	DATE
Fecha y hora	TIMESTAMP	TIMESTAMP DATETIME SMALLDATETIME	TIMESTAMP	TIMESTAMP DATETIME TIME
Intervalos	INTERVAL		INTERVAL	YEAR
Lógicos	BIT	BINARY		BIT BOOL
Texto gran longitud	CLOB	TEXT y NTEXT	LONG (en desuso) y CLOB	TEXT, MEDIUM TEXT y LONG TEXT
Binario de gran longitud	BLOB	IMAGE	RAW, LONG RAW BLOB	BLOB, MEDIUM BLOB y LONG BLOB

3.3 Claves primarias

La clave primaria de una tabla la forman las columnas que identifican a cada registro de la misma. La clave primaria hace que los campos que la forman sean NOT NULL (sin posibilidad de quedar vacíos) y que los valores de los campos sean de tipo UNIQUE (sin posibilidad de repetición).

Cuando se crea una clave primaria en una tabla se crea un índice que facilita el acceso a la tabla.

Si la clave está formada por un solo campo basta con:

```
CREATE TABLE cliente(
dni VARCHAR(9) PRIMARY KEY,
nombre VARCHAR(50)) ;
```

O, poniendo un nombre a la restricción:

```
CREATE TABLE cliente(
dni VARCHAR(9) CONSTRAINT cliente_pk PRIMARY KEY,
nombre VARCHAR(50)) ;
```

Si la clave está formada por más de un campo:

```
CREATE TABLE alquiler(dni VARCHAR(9),
cod_pelicula NUMBER(5),
CONSTRAINT alquiler_pk
  PRIMARY KEY(dni,cod_pelicula)) ;
```

3.4 Claves ajenas

Una clave `secundaria, ajena o foránea`, es uno o más campos de una tabla que están relacionados con la clave principal de otra tabla.

La forma de indicar una clave foránea (aplicando una restricción de integridad referencial) es:

```
CREATE TABLE alquiler(
        dni VARCHAR2(9) CONSTRAINT dni_fk REFERENCES clientes(dni),
        cod_pelicula NUMBER(5) CONSTRAINT pelicula_fk REFERENCES peliculas(cod),
        CONSTRAINT alquiler_pk PRIMARY KEY(dni,cod_pelicula) );
```

Significa esta instrucción (en cuanto a claves foráneas) que el campo `dni` se relaciona con la columna `dni` de la tabla `clientes`.

En este caso se entiende que los campos hacen referencia a las claves principales de las tablas referenciadas (si la referencia la forma más de un campo, el orden de los campos debe de ser el mismo).

Esto forma una relación entre dichas tablas, que además obliga al cumplimiento de la integridad referencial. Esta integridad obliga a que cualquier dni incluido en la tabla alquiler tenga que estar obligatoriamente en la tabla de clientes. De no ser así el registro no será insertado en la tabla (ocurrirá un error).

Otra forma de crear claves foráneas (útil para claves formadas por más de un campo) es:

```
CREATE TABLE existencias(
```

```
tipo CHAR2(9),
modelo NUMBER(3),
n_almacen NUMBER(1)
cantidad NUMBER(7),
CONSTRAINT exi_t_m_fk FOREIGN KEY(tipo,modelo)
REFERENCES piezas,
CONSTRAINT exi_nal_fk FOREIGN KEY(n_almacen)
REFERENCES almacenes,
CONSTRAINT exi_pk PRIMARY KEY(tipo,modelo,n_almacen)
);
```

Si la definición de clave secundaria se pone al final, hace falta colocar el texto FOREIGN KEY para indicar en qué campos se coloca la restricción de clave ajena. En el ejemplo anterior es absolutamente necesario que la clave principal de la tabla piezas a la que hace referencia la clave la formen las columnas **tipo** y **modelo** y que estén en ese orden.

La integridad referencial es una herramienta imprescindible de las bases de datos relacionales. Pero provoca varios problemas. Por ejemplo, si borramos un registro en la tabla principal que está relacionado con uno o varios de la secundaria ocurrirá un error, ya que de permitírsenos borrar el registro ocurrirá fallo de integridad (habrá claves secundarias refiriéndose a una clave principal que ya no existe).

Por ello se nos pueden ofrecer soluciones a añadir tras la cláusula **REFERENCES**. Son:

ON DELETE SET NULL. Coloca nulos todas las claves secundarias relacionadas con la borrada.

ON DELETE CASCADE. Borra todos los registros cuya clave secundaria es igual que la clave del registro borrado.

ON DELETE SET DEFAULT. Coloca en el registro relacionado el valor por defecto en la columna relacionada.

ON DELETE NO ACTION. Impide la operación de borrado o actualización.

En el caso explicado se aplicarían las cláusulas cuando se eliminen filas de la clave principal relacionada con la clave secundaria. En esas cuatro cláusulas se podría sustituir la palabra `DELETE` por la palabra `UPDATE`, haciendo que el funcionamiento se refiera a cuando se modifica un registro de la tabla principal; en muchas bases de datos se admite el uso tanto de `ON DELETE` como de `ON UPDATE`. Pero Oracle no implementa directamente las opciones `ON UPDATE CASCADE|SET NULL O SET DEFAULT`, de forma que no permite modificar el valor de una clave primaria si existen filas en otra o la misma tabla que la referencian

La sintaxis completa para añadir claves foráneas es:

```
CREATE TABLE tabla(lista_de_campos,
CONSTRAINT nombreRestriccion FOREIGN KEY (listaCampos) REFERENCES tabla(clavePrincipalRelacionada)
[ON DELETE | ON UPDATE
[SET NULL | CASCADE | DEFAULT | NO ACTION]
);
```

Si es de un solo campo existe esta alternativa:

```
CREATE TABLE tabla(lista_de_campos tipos propiedades, nombreCampoClaveSecundaria
CONSTRAINT nombreRestriccion
REFERENCES tabla(clavePrincipalRelacionada) [ON DELETE | ON UPDATE
[SET NULL | CASCADE | DEFAULT] );
```

Ejemplo:

```
CREATE TABLE alquiler(
        dni VARCHAR(9),
        cod_pelicula NUMBER(5),
        CONSTRAINT alquiler_pk PRIMARY KEY(dni,cod_pelicula),
        CONSTRAINT dni_fk FOREIGN KEY (dni)
          REFERENCES clientes(dni)
          ON DELETE SET NULL,
        CONSTRAINT pelicula_fk FOREIGN KEY (cod_pelicula)
          REFERENCES peliculas(cod)
          ON DELETE CASCADE
        );
```

3.5 Restricciones de integridad

Una restricción es una condición de obligado cumplimiento para una o más columnas de la tabla. A cada restricción se le pone un nombre.

Su sintaxis general es:

```
{CREATE TABLE nombreTabla |
ALTER TABLE nombreTabla {ADD | MODIFY}}
(campo tipoDeDatos [propiedades]
[[CONSTRAINT nombreRestricción ]] tipoRestricción (columnas)
[,siguienteCampo…]
[,CONSTRAINT nombreRestricción tipoRestricción (columnas) …)
```

Las restricciones tienen un nombre, se puede hacer que sea la base de datos la que les ponga nombre, pero entonces sería críptico. Por eso es mejor ponerle un nombre nosotros para que sea más fácil de recordar.

Los nombres de restricción no se pueden repetir para el mismo esquema, debemos buscar nombres únicos. Es buena idea incluir de algún modo el nombre de la tabla, los campos involucrados y el tipo de restricción en el nombre de la misma. Por ejemplo **`pieza_id_pk`** podría indicar que el campo id de la tabla pieza tiene una clave principal (PRIMARY KEY).

Por ejemplo para hacer que la clave principal de la tabla Alumnos sea el código del alumno, el nombre de la restricción podría ser: **`alu_cod_pk`**

3.5.1 Prohibir nulos

La restricción NOT NULL permite prohibir los nulos en una determinada tabla. Eso obliga a que la columna tenga que tener obligatoriamente un valor para que sea almacenado el registro.

Se puede colocar durante la creación (o modificación) del campo añadiendo la palabra NOT NULL tras el tipo:

```
CREATE TABLE cliente(dni VARCHAR(9) NOT NULL);
```

En ese caso el nombre lo coloca la propia base de datos. Aunque es recomendable poner nombre a las restricciones para controlarlas mejor, cuando se incumple una restricción not null, aunque se dé nombre a la restricción, éste no aparece en los mensajes de error.

Para poner el nombre se usa:

```
CREATE TABLE cliente(dni VARCHAR(9)
CONSTRAINT cli_dni_nn NOT NULL);
```

3.5.2 Valores únicos

Las restricciones de tipo UNIQUE obligan a que el contenido de una o más columnas no puedan repetir valores. Nuevamente hay dos formas de colocar esta restricción:

```
CREATE TABLE cliente(dni VARCHAR(9) UNIQUE);
```

En ese caso el nombre de la restricción lo pone el sistema. Otra forma es:

```
CREATE TABLE cliente(dni VARCHAR(9)
CONSTRAINT dni_u UNIQUE);
```

Esta forma permite poner un nombre a la restricción. Si la repetición de valores se refiere a varios campos, la forma sería:

```
CREATE TABLE alquiler(dni VARCHAR(9), cod_pelicula NUMBER(5),
CONSTRAINT alquiler_uk UNIQUE(dni,cod_pelicula) ;
```

La coma tras la definición del campo **`cod_pelicula`** hace que la restricción sea independiente de ese campo. Eso obliga a que, tras UNIQUE se indique la lista de campos. Incluso para un solo campo se puede colocar la restricción al final de la lista en lugar de definirlo a continuación del nombre y tipo de la columna.

Las claves candidatas deben llevar restricciones `UNIQUE` y `NOT NULL`.

3.6 Restricciones de validación

Son restricciones que dictan una condición que deben cumplir los contenidos de una columna. Una misma columna puede tener múltiples `CHECKS` en su definición (se pondrían varios `CONSTRAINT` seguidos, sin comas). La restricción `CHECK` puede hacer referencia a una o más columnas, pero no a valores de otras filas. En una cláusula `CHECK` no cabe incluir subconsultas.

Ejemplo:

```
CREATE TABLE ingresos(cod NUMBER(5) PRIMARY KEY,
concepto VARCHAR(40) NOT NULL,
importe NUMBER(11,2) CONSTRAINT importe_min
CHECK (importe>0)
CONSTRAINT importe_max CHECK (importe<8000)        );
```

En este caso la CHECK prohíbe añadir datos cuyo importe no esté entre 0 y 8000.

Para poder hacer referencia a otras columnas hay que construir la restricción de forma independiente a la columna (es decir al final de la tabla):

```
CREATE TABLE ingresos(cod NUMBER(5) PRIMARY KEY,
concepto VARCHAR(40) NOT NULL,
importe_max NUMBER(11,2),
importe NUMBER(11,2),
CONSTRAINT importe_maximo CHECK (importe<importe_max)
);
```

3.7 Modificación y eliminación de tablas de la base de datos

3.7.1 Eliminación de tablas

La orden **DROP TABLE** seguida del nombre de una tabla, permite eliminar la tabla en cuestión.

Al borrar una tabla:

- Desaparecen todos los datos
- Cualquier vista y sinónimo referente a la tabla seguirá existiendo, pero ya no funcionará (conviene eliminarlos)
- Las transacciones pendientes son aceptadas (`COMMIT`), en aquellas bases de datos que tengan la posibilidad de utilizar transacciones.
- Lógicamente, sólo se pueden eliminar las tablas sobre las que tenemos permiso de borrado.

Normalmente, **el borrado de una tabla es irreversible**, y no hay ninguna petición de confirmación, por lo que conviene ser muy cuidadoso con esta operación.

3.7.2 Modificación de tablas

Cambiar de nombre a una tabla

De forma estándar (SQL estándar) se hace:

```
ALTER TABLE nombreViejo RENAME TO nombreNuevo;
```

En Oracle y MySQL se realiza mediante la orden RENAME (que permite el cambio de nombre de cualquier objeto). Sintaxis:

```
RENAME nombreViejo TO nombreNuevo;
```

Borrar contenido de tablas

Oracle y MySQL disponen de una orden no estándar para eliminar definitivamente los datos de una tabla; es la orden **TRUNCATE TABLE** seguida del nombre de la tabla a borrar. Hace que se elimine el contenido de la tabla, pero no la estructura de la tabla en sí. Incluso borra del archivo de datos el espacio ocupado por la tabla.

Añadir columnas

ALTER TABLE nombreTabla **ADD**(nombreColumna TipoDatos [Propiedades] [,columnaSiguiente tipoDatos [propiedades]…)

Permite añadir nuevas columnas a la tabla. Se deben indicar su tipo de datos y sus propiedades si es necesario (al estilo de `CREATE TABLE`). Si la columna **no** está definida como `NOT NULL`, se puede añadir en cualquier momento

Las nuevas columnas se añaden al final, no se puede indicar otra posición (hay que recordar que el orden de las columnas no importa). Ejemplo:

ALTER TABLE facturas **ADD** (fecha **DATE**);

Muchas bases de datos (pero no Oracle) requieren escribir la palabra COLUMN tras la palabra ADD. Normalmente suele ser opcional.

Borrar columnas

ALTER TABLE nombreTabla **DROP**(columna [,columnaSiguiente,…]);

Elimina la columna indicada de manera irreversible e incluyendo los datos que contenía.

No se puede eliminar la única columna de una tabla que sólo tiene esa columna (habrá que usar **DROP TABLE**), ni se pueden eliminar las claves primarias referenciadas por claves ajenas.

ALTER TABLE facturas **DROP** (fecha);

Al igual que en el caso anterior, en SQL estándar se puede escribir el texto COLUMN tras la palabra DROP.

Modificar columnas

Permite cambiar el tipo de datos y propiedades de una determinada columna. Sintaxis:

ALTER TABLE nombreTabla **MODIFY**(columna **tipo** [**propiedades**] [columnaSiguiente **tipo** [**propiedades**] …]

Hay que tener en cuenta los datos ya insertados en las tablas a la hora de modificar las columnas, ya que no pueden entrar en contradicción con los datos ya insertados.

Ejemplo:

ALTER TABLE facturas **MODIFY**(fecha **DATE**);

En el caso de SQL estándar en lugar de MODIFY se emplea ALTER (que además opcionalmente puede ir seguida de COLUMN). Por ejemplo:

ALTER TABLE facturas **ALTER COLUMN** fecha **DATE**;

Renombrar columnas

Esto permite cambiar el nombre de una columna. Sintaxis **en Oracle**:

```
ALTER TABLE nombreTabla
RENAME COLUMN nombreAntiguo TO nombreNuevo;
```

Sintaxis **en MySQL**:

```
ALTER TABLE nombreTabla
CHANGE nombreAntiguo nombreNuevo;
```

Ejemplo:

ALTER TABLE facturas **RENAME COLUMN** fecha **TO** fechaYhora;

Valor por defecto

A cada columna se le puede asignar un valor por defecto durante su creación mediante la propiedad **DEFAULT**. Se puede poner esta propiedad durante la creación o modificación de la tabla, añadiendo la palabra **DEFAULT** tras el tipo de datos del campo y colocando detrás el valor que se desea por defecto.

Ejemplo:

CREATE TABLE articulo (cod **NUMBER**(7), nombre **VARCHAR2**(25), precio **NUMBER**(11,2) **DEFAULT** 3.5);

La palabra DEFAULT se puede añadir durante la creación o la modificación de la tabla (comando ALTER TABLE)

3.8 Añadir restricciones

Es posible querer añadir restricciones tras haber creado la tabla. En ese caso se utiliza la siguiente sintaxis:

```
ALTER TABLE tabla
ADD [CONSTRAINT nombre] tipoDeRestricción(columnas);
```

tipoRestricción es el texto **CHECK**, **PRIMARY KEY** o **FOREIGN KEY**. Las restricciones NOT NULL deben indicarse mediante ALTER TABLE .. MODIFY colocando NOT NULL en el campo que se modifica.

3.9 Eliminar restricciones

Sintaxis:

```
ALTER TABLE tabla
DROP {PRIMARY KEY | UNIQUE(campos) |
CONSTRAINT nombreRestricción [CASCADE]}
```

La opción **PRIMARY KEY** elimina una clave principal (también quitará el índice **UNIQUE** sobre las campos que formaban la clave. **UNIQUE** elimina índices únicos. La opción **CONSTRAINT** elimina la restricción indicada.

La opción **CASCADE** hace que se eliminen en cascada las restricciones de integridad que dependen de la restricción eliminada.

Por ejemplo en:

CREATE TABLE curso(
cod_curso **CHAR**(7) **PRIMARY KEY**, fecha_inicio **DATE**,

fecha_fin **DATE**, titulo **VARCHAR**(60), cod_siguientecurso **CHAR**(7),

CONSTRAINT fecha_ck **CHECK**(fecha_fin>fecha_inicio),

CONSTRAINT cod_ste_fk **FOREIGN KEY**(cod_siguientecurso)

REFERENCES curso **ON DELETE SET NULL**);

Tras esa definición de tabla, esta instrucción:

ALTER TABLE curso **DROP PRIMARY KEY;** Produce este error (en Oracle):

ORA-02273: a esta clave única/primaria hacen referencia algunas claves ajenas

Para ello habría que utilizar esta instrucción:

ALTER TABLE curso **DROP PRIMARY KEY CASCADE**;

Esa instrucción elimina la restricción de clave secundaria antes de eliminar la principal.

También produce error esta instrucción:

ALTER TABLE curso **DROP**(fecha_inicio)**;**

ERROR en línea 1:

ORA-12991: se hace referencia a la columna en una restricción de multicolumna

El error se debe a que no es posible borrar una columna que forma parte de la definición de una restricción. La solución es utilizar **CASCADE CONSTRAINTS** elimina las restricciones en las que la columna a borrar estaba implicada:

ALTER TABLE curso **DROP**(fecha_inicio) **CASCADE CONSTRAINTS**;

Esta instrucción elimina la restricción de tipo **CHECK** en la que aparecía la fecha_inicio y así se puede eliminar la columna. En **SQL estándar** sólo se pone

CASCADE y no **CASCADE CONSTRAINTS**.

Por defecto las restricciones se activan al crearlas, se pueden desactivar de las siguiente forma:

Para desactivar una restricción sin borrarla se usa DISABLE

```
ALTER TABLE table
DISABLE CONSTRAINT nombre_constraint
```

Para activar una restricción de integridad actualmente desactivada se usa ENABLE

```
ALTER TABLE table
ENABLE CONSTRAINT nombreconstraint;
```

4 LENGUAJE DE MANIPULACIÓN DE DATOS. DML

4.1 Inserción de datos

La adición de datos a una tabla se realiza mediante la instrucción **INSERT.** Su sintaxis fundamental es:

```
INSERT INTO tabla [(listaDeCampos)]
VALUES (valor1 [,valor2 ...])
```

La **tabla** representa la tabla a la que queremos añadir el registro y los valores que siguen a VALUES son los valores que damos a los distintos campos del registro. Si no se especifica la lista de campos, la lista de valores debe seguir el orden de las columnas según fueron creados (es el orden de columnas según las devuelve el comando **DESCRIBE**).

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.

Por ejemplo, supongamos que tenemos una tabla de clientes cuyos campos son: ***dni, nombre, apellido1, apellido2, localidad y dirección;*** supongamos que ese es el orden de creación de los campos de esa tabla y que la localidad tiene como valor por defecto ***Palencia*** y la dirección no tiene valor por defecto. En ese caso estas dos instrucciones son equivalentes:

```
INSERT INTO clientes VALUES( '11111111','Pedro','Gutiérrez', 'Crespo',DEFAULT,NULL);
INSERT INTO clientes(dni,nombre,apellido1,apellido2)
VALUES('11111111','Pedro','Gutiérrez', 'Crespo');
```

Son equivalentes puesto que en la segunda instrucción los campos no indicados se rellenan con su valor por defecto y la dirección no tiene valor por defecto. La palabra **DEFAULT** fuerza a utilizar ese valor por defecto.

El uso de los distintos tipos de datos debe de cumplir los requisitos ya comentados en apartados anteriores.

4.2 Actualización de registros

La modificación de los datos de los registros lo implementa la instrucción **UPDATE**. Sintaxis:

```
UPDATE tabla
SET columna1=valor1 [,columna2=valor2...]
```

[WHERE condición**]**

Se modifican las columnas indicadas en el apartado SET con los valores indicados. La cláusula WHERE permite especificar qué registros serán modificados.

Ejemplos:

```
UPDATE clientes SET provincia='Ourense'
WHERE provincia='Orense';
UPDATE productos SET precio=precio*1.16;
```

El primer dato actualiza la provincia de los clientes de Orense para que aparezca como Ourense.

El segundo UPDATE incrementa los precios en un 16%. La expresión para el valor puede ser todo lo compleja que se desee (en el ejemplo se utilizan funciones de fecha para conseguir que los partidos que se juagaban hoy, pasen a jugarse el martes):

```
UPDATE partidos SET fecha= NEXT_DAY(SYSDATE,'Martes')
WHERE fecha=SYSDATE;
```

En la condición se pueden utilizar 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.

4.3 Borrado de registros

Se realiza mediante la instrucción DELETE:

```
DELETE [FROM] tabla
[WHERE condición]
```

Es más sencilla que las anteriores, elimina los registros de la tabla que cumplan la condición indicada. Ejemplo:

```
DELETE FROM empleados
WHERE seccion=23
```

Hay que tener en cuenta que el borrado de un registro no puede provocar fallos de integridad y que la opción de **integridad ON DELETE CASCADE** hace que no sólo se borren los registros indicados en el SELECT, sino todos los relacionados.

5 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 definitivas.

5.1 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.

5.2 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 de caída del sistema dan lugar a un ROLLBACK implícito.

5.3 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.