ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

CREACIÓN DE BASE DE DATOS Y LENGUAJE SQL

1 - Introducción

Objetivos

SQL es el lenguaje fundamental de los SGBD relacionales. Se trata de uno de los lenguajes más utilizados de la historia de la informática. Es sin duda el lenguaje fundamental para manejar una base de datos relacional.

SQL es un **lenguaje declarativo** en el que lo importante es definir **qué** se desea hacer, por encima de **cómo** hacerlo (qué 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

Historia

El nacimiento del lenguaje SQL data de 1970 cuando E. F. Codd publica su libro: "Un modelo de datos relacional para grandes bancos de datos compartidos". Ese libro dictaría las direcrices de las bases de datos relacionales. Apenas dos años después IBM (para quien trabajaba Codd) utiliza las directrices de Codd para crear el Standard English Query Language (Lenguaje Estándar Inglés para Consultas) al que se le llamó SEQUEL. Más adelante se le asignaron las siglas SQL (Standard Query Language, lenguaje estándar de consulta) aunque en inglés se siguen pronunciando secuel. En español se pronuncia esecuele.

En 1979 Oracle presenta la primera implementación comercial del lenguaje. Poco después se convertía en un estándar en el mundo de las bases de datos avalado por los organismos ISO y ANSI. En el año 1986 se toma como lenguaje estándar por ANSI de los SGBD relacionales. Un año después lo adopta ISO, lo que convierte a SQL en estándar mundial como lenguaje de bases de datos relacionales.

En 1989 aparece el estándar ISO (y ANSI) llamado SQL89 o SQL1. En 1992 aparece la nueva versión estándar de SQL (a día de hoy sigue siendo la más conocida) llamada SQL92. En 1999 se aprueba un nuevo SQL estándar que incorpora mejoras que incluyen triggers, procedimientos, funciones,… y otras características de las bases de datos objeto-relacionales; dicho estándar se conoce como SQL99.

El último estándar es el del año 2011 (SQL2011)

Funcionamiento

Componentes de un entorno de ejecución SQL

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
- 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.

Posibles agentes SQL. posibles modos de ejecución SQL

Ejecución directa. SQL interactivo

Las instrucciones SQL se introducen a través de un cliente que está directamente conectado al servidor SQL; por lo que las instrucciones se traducen sin intermediarios y los resultados se muestran en el cliente.
Normalmente es un modo de trabajo incómodo, pero permite tener acceso a todas las capacidades del lenguaje SQL de la base de datos a la que estamos conectados.

Ejecución incrustada o embebida

Las instrucciones SQL se colocan como parte del código de otro lenguaje que se considera anfitrión (C, Java, Pascal, Visual Basic,...). 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.

Ejecución a través de clientes gráficos

Se trata de software que permite conectar a la base de datos a través de un cliente. El software permite manejar de forma gráfica la base de datos y las acciones realizadas son traducidas a SQL y enviadas al servidor. Los resultados recibidos vuelven a ser traducidos de forma gráfica para un manejo más cómodo.

Ejecución dinámica

Se trata de SQL incrustado en módulos especiales que pueden ser invocados una y otra vez desde distintas aplicaciones.

Proceso de las instrucciones SQL

El proceso de una instrucción SQL es el siguiente:
(1) Se analiza la instrucción. Para comprobar la sintaxis de la misma

(2) Si es correcta se valora si los metadatos de la misma son correctos. Se comprueba esto en el diccionario de datos.
(3) Si es correcta, se optimiza, a fin de consumir los mínimos recursos posibles.
(4) Se ejecuta la sentencia y se muestra el resultado al emisor de la misma.

2 - Elementos del lenguaje SQL

Código SQL

El código SQL consta de los siguientes elementos:

- ➔ Comandos. Las distintas instrucciones que se pueden realizar desde SQL
  - ◆ SELECT. Se trata del comando que permite realizar consultas sobre los datos de la base de datos. Obtiene datos de la base de datos. A ésta parte del lenguaje se la conoce como DQL (Data Query Language, Lenguaje de consulta de datos); pero es parte del DML del lenguaje.
  - ◆ DML, Data Manipulation Language (Lenguaje de manipulación de datos). Modifica filas (registros) de la base de datos. Lo forman las instrucciones INSERT, UPDATE, MERGE y DELETE.
  - ◆ DDL, Data Definition Language (Lenguaje de definición de datos). Permiten modificar la estructura de las tablas de la base de datos. Lo forman las instrucciones CREATE, ALTER, DROP, RENAME y TRUNCATE.
  - ◆ DCL, Data Control Language (Lenguaje de control de datos). Administran los derechos y restricciones de los usuarios. Lo forman las instrucciones GRANT y REVOKE.
  - ◆ Instrucciones de control de transacciones (DTL). Administran las modificaciones creadas por las instrucciones DML. Lo forman las instrucciones ROLLBACK y COMMIT. Se las considera parte del DML.
- ➔ Cláusulas. Son palabras especiales que permiten modificar el funcionamiento de un comando (WHERE, ORDER BY,...)
- ➔ Operadores. Permiten crear expresiones complejas. Pueden ser aritméticos (+,-,*,/,...) lógicos (>, <, !=,<>, AND, OR,...)
- ➔ Funciones. Para conseguir valores complejos (SUM(), DATE(),...)
- ➔ Literales. Valores concretos para las consultas: números, textos, caracteres,... Ejemplos: 2, 12.34, 'Avda Cardenal Cisneros'
- ➔ Metadatos. Obtenidos de la propia base de datos

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)

2 - DDL - Lenguaje de Definición de Datos

Introducción

El DDL es la parte del lenguaje SQL que realiza la función de definición de datos del SGBD.
Fundamentalmente se encarga de la creación, modificación y eliminación de los objetos de la base de datos (es decir de los metadatos). Por supuesto es el encargado de la creación de las tablas.
Cada usuario de una base de datos posee un esquema. El esquema suele tener el mismo nombre que el usuario y sirve para almacenar los objetos de esquema, es decir los objetos que posee el usuario.
Esos objetos pueden ser: tablas, vistas, índices y otras objetos relacionados con la definición de la base de datos. Los objetos son manipulados y creados por los usuarios. En principio sólo los administradores y los usuarios propietarios pueden acceder a cada objeto, salvo que se modifiquen los privilegios del objeto para permitir el acceso a otros usuarios.
Hay que tener en cuenta que ninguna instrucción DDL puede ser anulada por una instrucción ROLLBACK (la instrucción ROLLBACK está relacionada con el uso de transacciones que se comentarán más adelante) por lo que hay que tener mucha precaución a la hora de utilizarlas. Es decir, las instrucciones DDL generan acciones que no se pueden deshacer (salvo que dispongamos de alguna copia de seguridad).

Creación de base de datos

Esta es una tarea administrativa que se comentará más profundamente en otros temas. Por ahora sólo se comenta de forma simple. Crear la base de datos implica indicar los archivos y ubicaciones que se utilizarán para la misma, además de otras indicaciones técnicas y administrativas que no se comentarán en este tema.
Lógicamente sólo es posible crear una base de datos si se tienen privilegios DBA (DataBase Administrator) (SYSDBA en el caso de Oracle).
El comando SQL de creación de una base de datos es CREATE DATABASE. Este comando crea una base de datos con el nombre que se indique.
Ejemplo:

```
CREATE DATABASE prueba;
```

Pero normalmente se indican más parámetros. Ejemplo (parámetros de Oracle):

```
CREATE DATABASE prueba
        LOGFILE prueba.log
        MAXLOGFILES 25
        MAXINSTANCES 10
        ARCHIVELOG
        CHARACTER SET WIN1214
        NATIONAL CHARACTER SET UTF8
        DATAFILE prueba1.dbf AUTOEXTEND ON MAXSIZE 500MB;
```

Objetos de la base de datos

Según los estándares actuales, una base de datos es un conjunto de objetos pensados para gestionar datos. Estos objetos están contenidos en esquemas, los esquemas suelen estar asociados al perfil de un usuario en particular.
En el estándar SQL existe el concepto de catálogo que sirve para almacenar esquemas. Así el nombre completo de un objeto vendría dado por:

```
catálogo.esquema.objeto
```

Si no se indica el catálogo se toma el catálogo por defecto. Si no se indica el esquema se entiende que el objeto está en el esquema actual. En Oracle, cuando se crea un usuario, se crea un esquema cuyo nombre es el del usuario.

Creación de Tablas

Nombre de las tablas

Deben cumplir las siguientes reglas (reglas de Oracle, en otros SGBD podrían cambiar):

- Deben comenzar con una letra
- No deben tener más de 30 caracteres
- Sólo se permiten utilizar letras del alfabeto (inglés), números o el signo de subrayado (también el signo $ y #, pero esos se utilizan de manera especial por lo que no son recomendados)
- 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)
- En el caso de que el nombre tenga espacios en blanco o caracteres nacionales (permitido sólo en algunas bases de datos), entonces se suele entrecomillar con comillas dobles. En el estándar SQL 99 (respetado por Oracle) se pueden utilizar comillas dobles al poner el nombre de la tabla a fin de hacerla sensible a las mayúsculas (se diferenciará entre “FACTURAS” y “Facturas”)

orden CREATE TABLE

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 [DEFAULT valor] [restricciones] [, ...]);
```

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));
```

Se puede indicar un valor por defecto para el atributo mediante la cláusula DEFAULT:

```
CREATE TABLE Proveedores (
                    nombre VARCHAR(25),
                    localidad VARCHAR(30) DEFAULT ‘Palencia’);
```

De este modo si añadimos un proveedor y no indicamos localidad, se tomará Palencia como localidad de dicho Proveedor.

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. Para el caso de MySQL de Oracle pueden encontrar más información en: https://dev.mysql.com/doc/refman/8.0/en/data-types.html

Tipos de dato numéricos

Listado de cada uno de los tipos de datos numéricos en MySQL, su ocupación en disco y valores.

- INT (INTEGER): Ocupación de 4 bytes con valores entre -2147483648 y 2147483647 o entre 0 y 4294967295.
- SMALLINT: Ocupación de 2 bytes con valores entre -32768 y 32767 o entre 0 y 65535.
- TINYINT: Ocupación de 1 byte con valores entre -128 y 127 o entre 0 y 255.
- MEDIUMINT: Ocupación de 3 bytes con valores entre -8388608 y 8388607 o entre 0 y 16777215.
- BIGINT: Ocupación de 8 bytes con valores entre -263y 263-1o entre 0 y 16777215.
- DECIMAL (NUMERIC): Almacena los números de coma fija como cadenas o string.
- FLOAT (m,d): Almacena números de coma flotante, donde 'm' es el número de dígitos de la parte entera y 'd' el número de decimales.
- DOUBLE (REAL): Almacena un número de coma flotante con precisión doble. Igual que FLOAT, la diferencia es el rango de valores posibles.
- BIT (BOOL, BOOLEAN): Número entero con valor 0 o 1.

Tipos de dato con formato fecha

Listado de cada uno de los tipos de datos con formato fecha en MySQL, su ocupación en disco y valores.

- DATE: Válido para almacenar una fecha con año, mes y día, su rango oscila entre '1000-01-01′ y '9999-12-31′.
- DATETIME: Almacena una fecha (año-mes-día) y una hora (horas-minutos-segundos), su rango oscila entre '1000-01-01 00:00:00′ y '9999-12-31 23:59:59′.
- TIME: Válido para almacenar una hora (horas-minutos-segundos). Su rango de horas oscila entre -838-59-59 y 838-59-59. El formato almacenado es 'HH:MM:SS'.
- TIMESTAMP: Almacena una fecha y hora UTC. El rango de valores oscila entre '1970-01-01 00:00:01′ y '2038-01-19 03:14:07′.
- YEAR: Almacena un año dado con 2 o 4 dígitos de longitud, por defecto son 4. El rango de valores oscila entre 1901 y 2155 con 4 dígitos. Mientras que con 2 dígitos el rango es desde 1970 a 2069 (70-69).

Diferentes tipos de dato con formato string

Listado de cada uno de los tipos de datos con formato string en MySQL, su ocupación en disco y valores.

- CHAR: Ocupación fija cuya longitud comprende de 1 a 255 caracteres.
- VARCHAR: Ocupación variable cuya longitud comprende de 1 a 65.535 caracteres (se cuenta entre todos los campos de la tabla).

- TINYBLOB: Una longitud máxima de 255 caracteres. Válido para objetos binarios como son un fichero de texto, imágenes, ficheros de audio o vídeo. No distingue entre minúsculas y mayúsculas.
- BLOB: Una longitud máxima de 65.535 caracteres. Válido para objetos binarios como son un fichero de texto, imágenes, ficheros de audio o vídeo. No distingue entre minúsculas y mayúsculas.
- MEDIUMBLOB: Una longitud máxima de 16.777.215 caracteres. Válido para objetos binarios como son un fichero de texto, imágenes, ficheros de audio o vídeo. No distingue entre minúsculas y mayúsculas.
- LONGBLOB: Una longitud máxima de 4.294.967.298 caracteres. Válido para objetos binarios como son un fichero de texto, imágenes, ficheros de audio o vídeo. No distingue entre minúsculas y mayúsculas.
- SET: Almacena 0, uno o varios valores una lista con un máximo de 64 posibles valores.
- ENUM: Igual que SET pero solo puede almacenar un valor.
- TINYTEXT: Una longitud máxima de 255 caracteres. Sirve para almacenar texto plano sin formato. Distingue entre minúsculas y mayúsculas.
- TEXT:Una longitud máxima de 65.535 caracteres. Sirve para almacenar texto plano sin formato. Distingue entre minúsculas y mayúsculas.
- MEDIUMTEXT:Una longitud máxima de 16.777.215 caracteres. Sirve para almacenar texto plano sin formato. Distingue entre minúsculas y mayúsculas.
- LONGTEXT: Una longitud máxima de 4.294.967.298 caracteres. Sirve para almacenar texto plano sin formato. Distingue entre minúsculas y mayúsculas.

Además de estos tipos MySQL incorpora JSON y datos espaciales.

Consultar las tablas del usuario

Consultar el diccionario de datos

Todas las bases de datos disponen de posibilidades para consultar el diccionario de datos. Siguiendo las reglas de Codd, la forma de consultar los metadatos es la misma que en el resto de tablas. Es decir existen tablas (en realidad **vistas**) que en lugar de contener datos, contienen los metadatos. En el caso de SQL estándar, el diccionario de datos es accesible mediante el esquema de información (**INFORMATION_SCHEMA**), un esquema especial que contiene el conjunto de vistas con el que se pueden consultar ñps metadatos de la base de datos. En concreto la vista **INFORMATION_SCHEMA.TABLES** obtiene una vista de las tablas creadas. Es decir:

```
SELECT * FROM INFORMATION_SCHEMA.TABLES
```

Esa instrucción muestra una tabla con diversas columnas, entre ellas la columna **TABLE_CATALOG** indica el catálogo en el que está la tabla, **TABLE_SCHEMA** el esquema en el que está la tabla y **TABLE_NAME** el nombre de la tabla.

Muchos SGBD respetan el estándar, pero en el caso de **Oracle** no. Oracle utiliza diversas vistas para mostrar las tablas de la base de datos. En concreto **USER_TABLES** y que contiene una lista de las tablas del usuario actual (o del esquema actual). Así para sacar la lista de tablas del usuario actual, se haría:

```
SELECT * FROM USER_TABLES;
```

Esta vista obtiene numerosas columnas, en concreto la columna **TABLES_NAME** muestra el nombre de cada tabla.

Otra vista es **ALL_TABLES** mostrará una lista de todas las tablas de la base de datos (no solo del usuario actual), aunque oculta las que el usuario no tiene derecho a ver. Finalmente **DBA_TABLES** es una tabla que contiene absolutamente todas las tablas del sistema; esto es accesible sólo por el usuario administrador (***DBA***). En el caso de ALL_TABLES y de DBA_TABLES, la columna **OWNER** indica el nombre del propietario de la tabla.

orden DESCRIBE

El comando DESCRIBE, permite obtener la estructura de una tabla. Ejemplo:

```
DESCRIBE existencias;
```

Y aparecerán los campos de la tabla proveedores. Esta instrucción no es parte del SQL estándar, pero casi es considerada así ya que casi todos los SGBD la utilizan. Un ejemplo del resultado de la orden anterior (en Oracle) sería:

Nombre	¿Nulo?	Tipo
N_ALMACEN	NOT NULL	NUMBER(2)
TIPO	NOT NULL	VARCHAR2(2)
MODELO	NOT NULL	NUMBER(2)
CANTIDAD		NUMBER(7)

obtener la lista de las columnas de las tablas

Otra posibilidad para poder consultar los datos referentes a las columnas de una tabla, es utilizar el diccionario de datos.

Oracle posee una vista llamada **USER_TAB_COLUMNS** que permite consultar todas las columnas de las tablas del esquema actual. Las vistas **ALL_TAB_COLUMNS** y **DBA_TAB_COLUMNS** muestran los datos del resto de tablas (la primera sólo de las tablas accesibles por el usuario).

En el caso de SQL estándar las columnas son accesibles mediante la vista

INFORMATION_SCHEMA.COLUMNS

Borrar 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.

Modificar tablas

Cambiar de nombre a una tabla

De forma estándar (SQL estándar) se hace:

ALTER TABLE nombreViejo **RENAME TO** nombreNuevo**;**

En Oracle, además de con la orden anterior, se realiza mediante la orden **RENAME** (que permite el cambio de nombre de cualquier objeto). Sintaxis:

RENAME nombreViejo **TO** nombreNuevo**;**

Pero por coherencia es mejor hacerlo de la primera forma (la del estándar).

Borrar contenido de tablas

Oracle dispone 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**).
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

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).

```
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 columna

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

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

Los cambios que se permiten son (en Oracle):

- Incrementar precisión o anchura de los tipos de datos
- Sólo se puede reducir la anchura si la anchura máxima de un campo si esa columna posee nulos en todos los registros, o todos los valores son tan pequeños como la nueva anchura o no hay registros
- Se puede pasar de **CHAR** a **VARCHAR2** y viceversa (si no se modifica la anchura)
- Se puede pasar de **DATE** a **TIMESTAMP** y viceversa
- Cualquier otro cambio sólo es posible si la tabla está vacía

Ejemplo:

```
ALTER TABLE facturas MODIFY(fecha TIMESTAMP);
```

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 TIMESTAMP;
```

Renombrar columna

Esto permite cambiar el nombre de una columna. Sintaxis

```
ALTER TABLE nombreTabla RENAME COLUMN nombreAntiguo TO 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 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**)

Restricciones

Una restricción es una condición de obligado cumplimiento para una o más columnas de la tabla. A cada restricción se le pone un nombre, en el caso de no poner un nombre (algo poco recomendable) entonces el propio Oracle le coloca el nombre que es un mnemotécnico con el nombre de tabla, columna y tipo de restricción.

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 de 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 (**P**RIMARY **K**EY).

Desde la empresa Oracle se aconseja la siguiente regla a la hora de poner nombre a las restricciones:

- Tres letras para el nombre de la tabla Carácter de subrayado
- Tres letras con la columna afectada por la restricción Carácter de subrayado
- Dos letras con la abreviatura del tipo de restricción. La abreviatura puede ser:
  - **NN**. NOT NULL.
  - **PK**. PRIMARY KEY
  - **UK**. UNIQUE
  - **FK**. FOREIGN KEY
  - **CK.** CHECK (validación)

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
```

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 VARCHAR2(9) NOT NULL);
```

En ese caso el nombre le coloca la propia base de datos (en el caso de Oracle el nombre sería algo como ***SY002341*** por ejemplo). No es recomendable no poner nombre a las restricciones para controlarlas mejor.

Para poner el nombre se usa:

```
CREATE TABLE cliente(dni VARCHAR2(9) CONSTRAINT cli_dni_nn NOT NULL);
```

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 VARCHAR2(9) UNIQUE);
```

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

```
CREATE TABLE cliente(dni VARCHAR2(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 VARCHAR2(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**

clave primaria

La clave primaria de una tabla la forman las columnas que indican 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).

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)) ;
```

Clave secundaria o foránea

Una clave secundaria o foránea, es uno o más campos de una tabla que están relacionados con la clave principal (o incluso con una clave candidata) 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***.

Si el campo al que se hace referencia es la clave principal, se puede obviar el nombre del campo:

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

En este caso se entiende que los campos hacen referencia a las claves principales de las tablas referenciadas (si la relación la forma más 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 foránea. 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 en 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 secundarios 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 NOTHING.** No hace nada.

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.

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]
                        );
```

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
                    );
```

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).

Ejemplo:

```
CREATE TABLE ingresos(cod NUMBER(5) PRIMARY KEY, concepto
   VARCHAR2(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 VARCHAR2(40) NOT NULL,
                              importe_max NUMBER(11,2),
                              importe NUMBER(11,2),
                              CONSTRAINT importe_maximo
                              CHECK (importe<importe_max)
                    );
```

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.

Borrar 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 VARCHAR2(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
```

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 instrucción. La solución es utilizar CASCADE CONSTRAINT 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.

Desactivar restricciones

A veces conviene temporalmente desactivar una restricción para saltarse las reglas que impone. La sintaxis es (en Oracle):

```
ALTER TABLE tabla DISABLE CONSTRAINT nombre [CASCADE]
```

La opción CASCADE hace que se desactiven también las restricciones dependientes de la que se desactivó.

Activar restricciones

Anula la desactivación. Formato (Oracle):

ALTER TABLE tabla **ENABLE CONSTRAINT** nombre **[CASCADE]**

Sólo se permite volver a activar si los valores de la tabla cumplen la restricción que se activa. Si hubo desactivado en cascada, habrá que activar cada restricción individualmente.

Cambiar de nombre a las restricciones

Para hacerlo se utiliza este comando (Oracle):

ALTER TABLE table **RENAME CONSTRAINT** nombreViejo **TO** nombreNuevo;

mostrar restricciones

SQL estándar

En SQL estándar hay dos vistas del diccionario de datos que permiten visualizar la información sobre las restricciones aplicadas en la base de datos. Son:

- **INFORMATION_SCHEMA.TABLE_CONSTRAINTS**
- **INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE**
- **INFORMATION_SCHEMA.CONSTRAINT_TABLE_USAGE.**

La primera permite analizar las restricciones colocadas. Devuelve una tabla con la siguiente estructura:

Columna	Tipo de datos	Descripción
TABLE_CATALOG	***texto***	Muestra el nombre del catálogo al que pertenece la tabla a la que se puso la restricción
TABLE_SCHEMA	***texto***	Muestra el nombre del esquema al que pertenece la tabla a la que se puso la restricción
TABLE_NAME	***texto***	Muestra el nombre de la tabla a la que se puso la restricción
CONSTRAINT_CATALOG	***texto***	Catálogo en el que está almacenada la restricción

CONSTRAINT_CATALOG	***texto***	Esquema al que pertenece la restricción
CONSTRAINT_NAME	***texto***	Nombre de la restricción
CONSTRAINT_TYPE	***carácter***	Indica el tipo de restricción, puede ser: **CHECK** (**C**), **FOREIGN KEY** (**F**), **PRIMARY KEY** (**P**) o **UNIQUE** (**U**)

Por su parte **INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE** obtiene información sobre las columnas a las que afecta la restricción. La tabla que obtiene es:

Columna	Tipo de datos	Descripción
TABLE_CATALOG	***texto***	Muestra el nombre del catálogo al que pertenece la tabla a la que se puso la restricción
TABLE_SCHEMA	***texto***	Muestra el nombre del esquema al que pertenece la tabla a la que se puso la restricción
TABLE_NAME	***texto***	Muestra el nombre de la tabla a la que se puso la restricción
CONSTRAINT_CATALOG	***texto***	Catálogo en el que está almacenada la restricción
CONSTRAINT_CATALOG	***texto***	Esquema al que pertenece la restricción
CONSTRAINT_NAME	***texto***	Nombre de la restricción
COLUMN_NAME	***texto***	Nombre de cada columna a la que afecta la restricción.

En el caso de **INFORMATION_SCHEMA.CONSTRAINT_TABLE_USAGE** simplemente nos dice el nombre de las restricciones y de las tablas a las que afecta.

Oracle

En el caso de Oracle, se puede utilizar la vista del diccionario de datos USER_CONSTRAINTS.

Esta vista permite identificar las restricciones colocadas por el usuario (**ALL_CONSTRAINTS** permite mostrar las restricciones de todos los usuarios, pero sólo está permitida a los administradores). En esa vista aparece toda la información que el diccionario de datos posee sobre las restricciones. En ella tenemos las siguientes columnas interesantes:

Columna	Tipo de datos	Descripción
OWNER	**VARCHAR2(20)**	Indica el nombre del usuario propietario de la tabla
CONSTRAINT_NAME	**VARCHAR2(30)**	Nombre de la restricción
CONSTRAINT_TYPE	**VARCHAR2(1)**	Tipo de restricción: **C.** De tipo CHECK o NOT NULL **P.** PRIMARY KEY **R.** FOREIGN KEY **U.** UNIQUE
TABLE_NAME	**VARCHAR2(30)**	Nombre de la tabla en la que se encuentra la restricción

En el diccionario de datos hay otra vista que proporciona información sobre restricciones, se trata de **USER_CONS_COLUMNS**, en dicha tabla se muestra información sobre las columnas que participan en una restricción. Así si hemos definido una clave primaria formada por los campos ***uno*** y ***dos***, en la tabla USER_CONS_COLUMNS aparecerán dos entradas, una para el primer campo del índice y otra para el segundo. Se indicará además el orden de aparición en la restricción.

Ejemplo (resultado de la instrucción **SELECT * FROM USER_CONS_COLUMNS**):

OWNER	CONSTRAINT_NAME	TABLE_NAME	COLUMN_NAME	POSITION
JORGE	EXIS_PK	EXISTENCIAS	TIPO	1
JORGE	EXIS_PK	EXISTENCIAS	MODELO	2
JORGE	EXIS_PK	EXISTENCIAS	N_ALMACEN	3

JORGE	PIEZA_FK	EXISTENCIAS	TIPO	1
JORGE	PIEZA_FK	EXISTENCIAS	MODELO	2
JORGE	PIEZA_PK	PIEZA	TIPO	1
JORGE	PIEZA_PK	PIEZA	MODELO	2

En esta tabla USER_CONS_COLUMNS aparece una restricción de clave primaria sobre la tabla ***existencias***, esta clave está formada por las columnas (***tipo, modelo*** y ***n_almacen***) y en ese orden. Una segunda restricción llamada ***pieza_fk*** está compuesta por ***tipo*** y ***modelo*** de la tabla existencias. Finalmente la restricción ***pieza_pk*** está formada por ***tipo*** y ***modelo***, columnas de la tabla pieza.

Para saber de qué tipo son esas restricciones, habría que acudir a la vista

USER_COL_CONSTRAINTS.