INDICE

1. Versión de SQL

Aunque estos apuntes sirven como guía de uso de SQL estándar, el gestor de base de datos que se utiliza como referencia fundamental es MySQL Server.

2. Formato de las instrucciones

En este manual en muchos apartados se indica sintaxis de comandos. Esta sintaxis sirve para aprender a utilizar el comando, e indica la forma de escribir dicho comando en el programa utilizado para escribir SQL. En el presente manual la sintaxis de los comandos se escribe con el reborde de color marrón anaranjado.

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

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

Los ejemplos sirven para escenificar una instrucción concreta, la sintaxis se utiliza para indicar las forma de utilizar un comando. Para indicar la sintaxis de un comando se usan símbolos especiales. Los símbolos que utilizan en estos apuntes 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 negro sirve para indicar que no hay que escribirle 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.

3. Introducción

a. 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 lo 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**.

b. Historia del lenguaje SQL

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 directrices 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 2012 (**SQL2012**).

c. 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.

4. Elementos del lenguaje SQL

a. 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 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.**
  - **TCL,** Transaction Control Language **(Lenguaje de Control de Transacciones).** 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.

b. 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).

5. Lenguaje DDL

a. 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 otros 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).

En este enlace https://dev.mysql.com/doc/refman/8.0/en/sql-data-definition-statements.html podéis ver la sintaxis completa de los comandos que veremos a continuación.

b. Creación y Utilización de Bases de Datos

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. Necesita el permiso **CREATE** en la base de datos.

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/create-database.html

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

```
CREATE DATABASE pruebas;
```

`USE db_name;`	Indica al servidor la base de datos a usar

```
USE pruebas;
```

`SHOW DATABASES;`	Muestra las bases de datos alojadas en el servidor

c. Creación de tablas

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/create-table.html

Nombre de las tablas. Reglas

- Deben comenzar con una letra.
- No deben tener más de 30 caracteres.
- Sólo se permiten utilizar letras del alfabeto, números o el signo de subrayado.
- 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).

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tableName
   [(create_definition,...)]
   [table_options] [select_statement]
```

```
CREATE TABLE empleados (
        codigo  INT AUTO_INCREMENT PRIMARY KEY,
        nombre VARCHAR (40),
        comision INTEGER
);
```

- **CREATE TABLE** crea una tabla con el nombre dado.
- Debe tener el permiso **CREATE** para la tabla.
- En MySQL 5.0, el nombre de tabla puede especificarse como **databaseName.tableName** para crear la tabla en la base de datos específica. Esto funciona haya una base de datos actual o no. Si usa identificadores entre comillas, entrecomille el nombre de base de datos y de tabla por separado. Por ejemplo, **`mydb`.`mytbl`** es legal, pero **`mydb.mytbl`** no.

Listado de las tablas de una base de datos

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/show-tables.html

```
SHOW TABLES;
```

Mostrar la estructura de las tablas

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/describe.html

```
DESCRIBE tbl_name;
```

```
DESCRIBE empleados;
```

Clonar una tabla usando SELECT

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/create-table-select.html

La sentencia **CREATE TABLE (..) SELECT** nos permite crear la tabla con la estructura y los registros que devuelva la consulta de selección, pero tiene las siguientes limitaciones:

- No traspasa las restricciones de tipo PRIMARY KEY.
- No traspasa las definiciones de AUTO_INCREMENT.
- No traspasa las definiciones de DEFAULT CURRENT_TIMESTAMP.
- Solamente traspasa los registros afectados por la SELECT, que podría no devolver ninguno y crear la tabla vacía.

```
INSERT INTO empleados (nombre,comision) VALUES ('Pedro',20);
SELECT * FROM empleados;
```

```
CREATE TABLE tbl_nueva SELECT * FROM tbl_origen;
```

```
CREATE TABLE nuevosEmpleados SELECT * FROM empleados;

DESCRIBE nuevosEmpleados;
SELECT * FROM nuevosEmpleados;
```

Clonar una tabla usando LIKE

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/create-table-like.html

La sentencia **CREATE TABLE (..) LIKE**, creará una tabla vacía que conserva la estructura de la original, pero no los registros.

```
CREATE TABLE tbl_nueva LIKE tbl_origen;
```

```
CREATE TABLE nuevosEmpleados1 LIKE empleado;
DESCRIBE nuevosEmpleados1;
SELECT * FROM nuevosEmpleados1;
```

d. Modificación de la Base de Datos

Tutorial: http://dev.mysql.com/doc/refman/5.6/en/alter-database.html

```
ALTER {DATABASE | SCHEMA} [db_name]
```

Permite cambiar las características globales de una base de datos.

- Estas características se almacenan en el fichero **db.opt** en el directorio de la base de datos.
- Para usar **ALTER DATABASE**, necesita el permiso **ALTER** en la base de datos.
- La cláusula **CHARACTER SET** y **COLLATE** función igual que para CREATE.

e. Modificación de tablas

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/alter-database.html

ALTER TABLE le permite cambiar la estructura de una tabla existente:

- Añadir o borrar columnas o índices.
- Cambiar el tipo y nombre de las columnas existentes.

```
ALTER [IGNORE] TABLE tbl_name
alter_specification...
```

```
ALTER TABLE empleados ADD numhijos TINYINT;
DESCRIBE empleados;
```

Puede renombrar una columna usando **CHANGE** old_col_name column_definition. Para ello, especifique el nombre de columna viejo y nuevo y el tipo de la columna actual. Por ejemplo, para renombrar una columna INTEGER de a a b, puede hacer:

```
ALTER TABLE tbl_name CHANGE old_name new_name data_type;
```

```
ALTER TABLE empleados CHANGE codigo identificador TINYINT;
DESCRIBE empleados;
```

Si quiere cambiar el tipo de una columna pero no el nombre, la sintaxis **CHANGE** necesita un nombre viejo y nuevo de columna, incluso si son iguales. Por ejemplo:

```
ALTER TABLE tbl_name CHANGE column_name column_name new_data_type;
```

```
ALTER TABLE empleados CHANGE identificador identificador INT;
DESCRIBE empleados;

ALTER TABLE empleados CHANGE identificador cod TINYINT;
DESCRIBE empleados;
```

Puede usar **MODIFY** para cambiar el tipo de una columna sin renombrarla:

```
ALTER TABLE tbl_name MODIFY column_name new_data_type;
```

```
ALTER TABLE empleados MODIFY cod INT;
DESCRIBE empleados;
```

f. Eliminación de Bases de Datos

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/drop-database.html

- **DROP DATABASE** borrar todas las tablas en la base de datos y borrar la base de datos. ¡CUIDADO!.
- Para usar **DROP DATABASE**, necesita el permiso **DROP** en la base de datos.
- **IF EXISTS** se usa para evitar un error si la base de datos no existe.

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name;
```

```
DROP DATABASE [IF EXISTS] pruebas;
```

```
DROP DATABASE pruebas;
```

g. Eliminación de Tablas

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/drop-table.html

- **DROP TABLE** borra una o más tablas. Debe tener el permiso **DROP** para cada tabla. Todos los datos de la definición de tabla son *borrados*, así que *tenga cuidado* con este comando!
- Use **IF EXISTS** para evitar un error para tablas que no existan.

```
DROP [TEMPORARY] TABLE [IF EXISTS] tbl_name [, tbl_name] ...
```

```
DROP TABLE IF EXISTS empleados;
```

h. Renombrar Tablas

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/rename-table.html

Este comando renombra una o más tablas.

```
RENAME TABLE tbl_name TO new_tbl_name,  ...
```

```
RENAME TABLE  nuevosEmpleados TO newEmpleados;
```

```
ALTER TABLE nuevosEmpleados1 RENAME TO oldEmpleados;
```