

Sentencias DDL y DML

Sitio: Agencia de Habilidades para el Futuro

Curso: Administración de Base de Datos 1° D

Libro: Sentencias DDL y DML

Imprimido por: RODRIGO PINTO

Día: miércoles, 27 de noviembre de 2024, 09:02

Tabla de contenidos

1. Sentencias DDL

1.1. Creando la base de datos

1.2. Base de datos con FK

2. Sentencias DML

2.1. Funciones de agregación

2.2. Funciones de agrupación

2.3. Operaciones de pertenencia a conjuntos

2.4. Operaciones con valores nulos

2.5. Operaciones de insertar, borrar y modificar

Sentencias DDL (lenguaje de definición de datos)



¿Qué son las sentencias DDL?

Las sentencias DDL **son las sentencias de definición de datos**. Estas sentencias nos permiten **crear la base de datos y las tablas**.

Veamos un ejemplo práctico para entender como funciona.

El entorno de trabajo es una " **Biblioteca**". Esta biblioteca cuenta con muchos libros y de cada uno de ellos hay varios ejemplares. Los libros están escritos **por uno o varios autores** y a su vez el cada autor puede escribir más de un libro. Existen personas que son **socios de la biblioteca** y se llevan en calidad de préstamo uno o varios ejemplares de libros en cada operación. Los libros presentan la **fecha de escritura y la fecha de edición**. En los prestamos se observan las **fechas del préstamo, la fecha de devolución y la fecha tope** que tiene el socio para entregar los ejemplares.

Realizada la **normalización** se obtuvieron las siguientes estructuras.

SOCIO

| | | | | | |
|--------------------|--------------------|-----------------------|-------------------------|--------------------------|--------------------|
| CodSocio entero | DNI Varchar(10) | Nombre Varchar(60) | Apellido Varchar(60) | Direccion Varchar(50) | Tel Varchar(15) |
|--------------------|--------------------|-----------------------|-------------------------|--------------------------|--------------------|

AUTOR

| | | | | | |
|--------------------|-----------------------|-------------------------|--------------------|---------------------|--------------------|
| CodAutor entero | Nombre Varchar(60) | Apellido Varchar(60) | DNI Varchar(10) | Mail Varchar(20) | Tel Varchar(15) |
|--------------------|-----------------------|-------------------------|--------------------|---------------------|--------------------|

LIBRO

| | | | | | |
|--------------------|-----------------------|---------------------|--------------------------|---------------------|------------------|
| CodLibro entero | Titulo Varchar(90) | ISBN Varchar(15) | Editorial Varchar(30) | FEscriturat Date | FEdicion Date |
|--------------------|-----------------------|---------------------|--------------------------|---------------------|------------------|

EJEMPLAR

| | | | | |
|------------------|--------------------|---------------------|-------------------------|----------------------|
| idEjem entero | CodLibro entero | NEjemplar entero | Deteriorado booleano | Prestado booleano |
|------------------|--------------------|---------------------|-------------------------|----------------------|

PRESTAMO

| | | | | |
|---------------------|--------------------|-------------------|---------------------|---------------|
| NPrestamo entero | CodSocio entero | FPrestamo Date | FDevolucion Date | FTope Date |
|---------------------|--------------------|-------------------|---------------------|---------------|

DetallePRE

| | |
|---------------------|------------------|
| NPrestamo entero | idEjem entero |
|---------------------|------------------|

LibroAUTOR

| | |
|--------------------|--------------------|
| CodLibro entero | CodAutor entero |
|--------------------|--------------------|



Después de la normalización tenemos el modelo. **En el próximo capítulo vamos a crear la base de datos con sentencias SQL para que quede almacenada en el sistema gestor.**

Crear una base de datos: paso a paso



¿Cómo crear una base de datos?

Usamos el Panel de control para esta muestra. Para conservar las instrucciones y poder incluir comentarios que nos permitan luego consultar es que lo documentamos en un editor de texto.

Te sugerimos que uses alguno que te permita **identificar el lenguaje (SQL)**. La razón es que todo lenguaje de programación cuenta con sus palabras reservadas y estas figuran de un color particular.

Por ejemplo **"Notepad"** Las palabras en azul son reservadas de **SQL**.

```
create database Biblioteca;  
use Biblioteca;
```

Comenzamos

1. Las instrucciones indicadas en la imagen anterior muestran:

- Creación de la base Biblioteca.
- Indica que base debe usar para manipular los datos de esa base.

2. Creamos la tabla

La sintaxis a utilizar es la siguiente:

```
create table Socio(  
  CodSocio int,  
  DNI varchar (10),  
  Nombre varchar (60),  
  Apellido varchar (60),  
  Direccion varchar (50),  
  Tel varchar (15),  
  constraint pk_socio primary key (CodSocio)  
);
```

Como se ve en la imagen, vamos a utilizar las palabras reservadas => "create table" / "constraint" / "primary key"

La primera línea indica que la acción es una creación, el objeto que estamos creando, en este caso una tabla y el nombre de esa tabla (Socio).

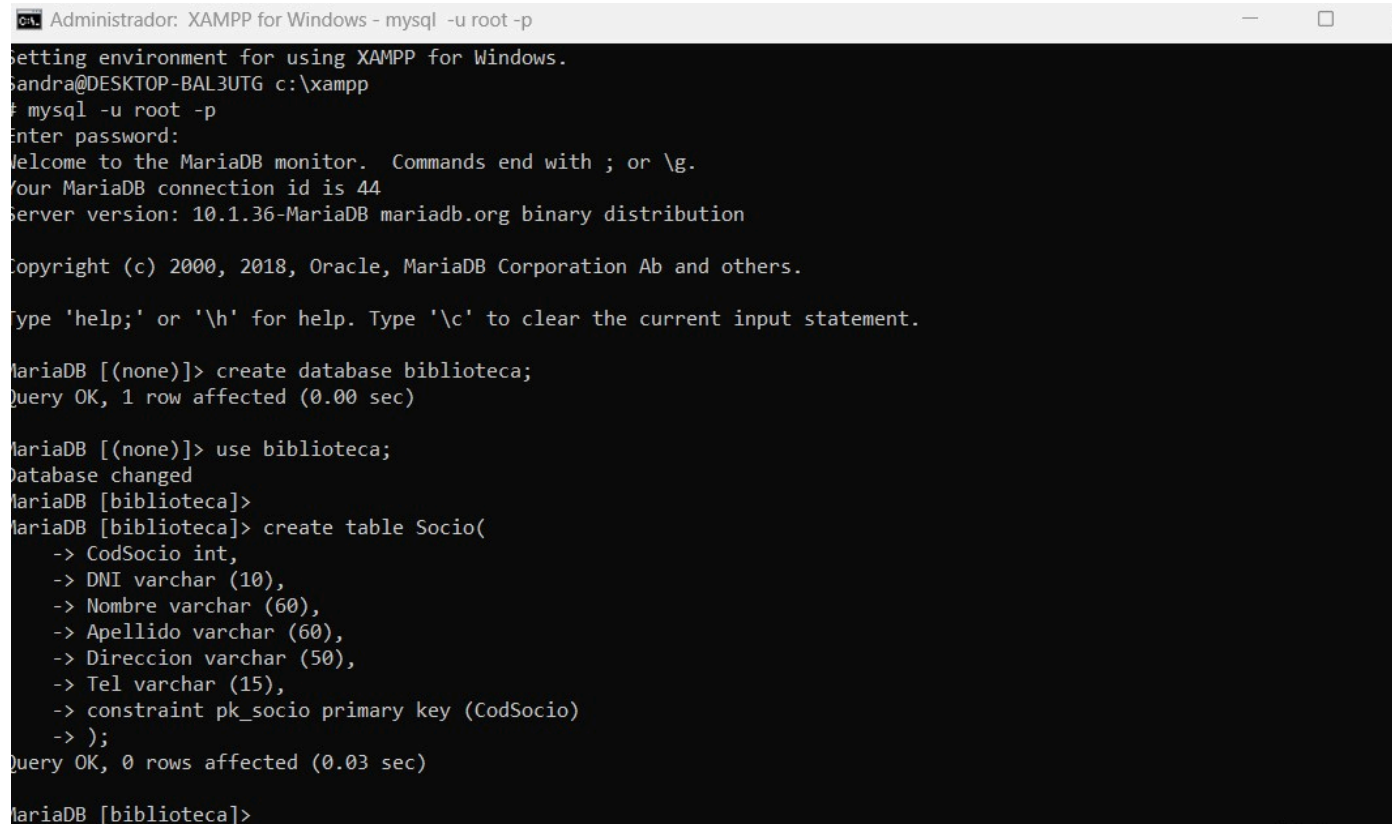
2.1 ¿Cómo escribimos la sintaxis?

- Para comenzar aparece un paréntesis que abre y al final ese paréntesis cierra para indicar que la tabla cuenta con todos sus atributos, claves y restricciones. El cierre va acompañado de un punto y coma.
- Luego, tenemos a cada uno de los atributos con el nombre que le asignamos y el tipo de dato.

Observá que cada línea finaliza con una coma, esta coma separa a los elementos de la creación. La última línea antes del cierre no lleva la coma.

- El tipo de **dato varchar representa una cadena de caracteres** por eso lleva entre paréntesis un número que indica la cantidad de caracteres que tiene el dominio de ese atributo.
- El **“constraint” es una restricción**, en este caso, de clave primaria e indica cuál o cuáles son los atributos que tiene el privilegio de identificar de manera univoca a la fila. El nombre que tiene “pk_socio” es para identificarlo dentro del sistema.

3. Ejecutemos



```
Administrator: XAMPP for Windows - mysql -u root -p
Setting environment for using XAMPP for Windows.
Sandra@DESKTOP-BAL3UTG c:\xampp
# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 44
Server version: 10.1.36-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create database biblioteca;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> use biblioteca;
Database changed
MariaDB [biblioteca]>
MariaDB [biblioteca]> create table Socio(
  -> CodSocio int,
  -> DNI varchar (10),
  -> Nombre varchar (60),
  -> Apellido varchar (60),
  -> Direccion varchar (50),
  -> Tel varchar (15),
  -> constraint pk_socio primary key (CodSocio)
  -> );
Query OK, 0 rows affected (0.03 sec)

MariaDB [biblioteca]>
```

Al ejecutar el **"create"** aparece un **"query ok ..."** esto quiere decir que nuestra query se ejecutó satisfactoriamente. Indicamos que vamos a usar esa base y creamos la tabla socio.



En el próximo capítulo veremos un ejemplo de **tabla con clave foránea**.

Crear una tabla con clave Foránea (Foreign key)

Veamos ahora como creamos una tabla que tenga además de su **clave primaria (PK)** una **clave foránea (FK)**.

Ejemplo

Observemos la siguiente imagen:

```
create table Ejemplar(  
  idEjem int auto_increment,  
  CodLibro int,  
  NEjemplar int,  
  Deteriorado boolean,  
  Prestado boolean,  
  constraint pk_Ejemplar primary key (idEjem),  
  constraint fk_Ejemplar foreign key (CodLibro) references Libro (CodLibro)  
);
```

La tabla “Ejemplar” **tiene como PK al idEjem**, pero contiene el **CodLibro**. Para mantener la consistencia de datos cada código debe ser alguno de los dominios de la **tabla “libro”** para ese mismo atributo. Tenemos aquí a una **FK**.

Esta creación tiene datos “**boolean**”. Estos datos tienen como dominio dos posibles valores, estos son, verdadero o falso (1 o 0). Los atributos en cuestión son “**Deteriorado**” y “**Prestado**”.

Si el valor del dominio es **verdadero** quiere decir que afirma el nombre de la columna; está deteriorado o está prestado.

Además vemos **dos constraint**, una define a la **PK** y la otra a la **FK**. El constraint de la FK muestra al atributo que tiene esa restricción especificando de cuál es clave foránea, vemos que detrás de la palabra reservada **"references"** muestra la tabla y el atributo de esta última tabla que es **PK**.



¿Cómo creamos la tabla cuando tiene una PK compuestas y a su vez esos atributos con FK?

Esta situación la vemos en la tabla "libroAutor", esta formada por la PK de la tabla libro y de la tabla autor.

```
create table LibroAutor(  
  CodLibro int,  
  CodAutor int,  
  constraint pk_LibroAutor primary key (CodLibro, CodAutor),  
  constraint fk_LibroAutor_L foreign key (CodLibro) references Libro (CodLibro),  
  constraint fk_LibroAutor_A foreign key (CodAutor) references Autor (CodAutor)  
);
```



¿Qué debemos tener en cuenta para crear una tabla cuando tiene una PK compuesta?

1. Cuando la **PK es compuesta**, en este caso de dos atributos, se colocan ambos separados por una coma. Vemos que cada uno de esos atributos deben contener un **dominio de la columna** en donde son **PK**. Esta condición respeta la **integridad referencial**. Es así que aparecen dos **constraint** uno para cada atributo **FK**.

2. Una vez que creamos todas las tablas podemos insertar datos para comenzar con las primeras consultas.

3. Para insertar datos se usa la palabra reservada **"Insert"** indicando en que tabla vamos a introducir datos y que dominios asignamos a sus atributos.

Por ejemplo, si queremos insertar datos en la tabla socio la instrucción es:

```
insert into socio values  
(20145, "11452452", "Maria Josefun", "Luro", "Av Pacheco 21478 CABA", "47857855");
```

4. Como insertamos dominios en todos los atributos no colocamos la lista con sus nombres. Los datos que son cadena de caracteres van entre comillas. Los dominios van separados por comas y deben ir en el orden de la declaración de los atributos.

5. Si queremos insertar varias tuplas a la tabla:

```
insert into socio values  
(20154, "19785452", "Marcos", "Nevarez", "Trelles 1234 CABA", "47852154"),  
(21489, "20145874", "Juliana", "Laprida", "Bacacay 10789 Haedo", "49061236"),  
(21474, "22145986", "Karina", "Quirno", "Bolivia 52345 CABA", "47851414"),  
(21523, "20333564", "Viviana", "Martinez", "Mendoza 123 Martinez", "45038796");
```


Sentencias DML (lenguaje de manipulación de datos)



Ya sabemos cómo es el **formato básico** de una consulta SQL:

```
SELECT    lista_de_Atributos
FROM      lista_de_Tablas
WHERE     predicado ;
```

En el siguiente cuadro se muestran las cláusulas básicas para manipular los datos almacenados:

| Operador | Definición | Sentencia |
|----------|---|--|
| DISTINCT | Elimina tuplas repetidas en el resultado. | SELECT DISTINCT oficio FROM empleado; Muestra una sola vez cada uno de los oficios de los empleados. |

| | | |
|------------------|--|---|
| BETWEEN | Cuando el dominio del predicado pertenece a un rango de valores. | <p>SELECT codigo_c, nombre FROM empleado WHERE salario BETWEEN 10000 AND 1600;</p> <p>Muestra el código y el nombre de los empleados con salario entre 10000 y 16000 pesos inclusive.</p> |
| Cambio de nombre | Cuando se desea cambiar el nombre de las tablas en los productos cartesianos se emplea AS o se deja un espacio entre el nombre de la tabla y su alias. | <p>SELECT * FROM alumno A, materia M WHERE A.legajo = M.legajo;</p> <p>Renombra a la tabla alumno con A y a la tabla materia con M.</p> |
| ORDER BY | Permite ordenar el resultado de la consulta. | <p>SELECT * FROM empleado WHERE salario = 15000 ORDER BY nombre;</p> <p>Muestra los datos de los empleados que cumplen la condición ordenadas de manera ascendente por su nombre. Para ordenarlos de manera descendente se debe agregar DESC. Se puede indicar más de un criterio de ordenación, estos van separados por coma.</p> |

| | | |
|------|---|---|
| LIKE | <p>Cuando necesitamos buscar tuplas que "contengan" determinada información, sin necesidad de coincidir exactamente. Utiliza un comodín => %</p> | <pre>SELECT * FROM empleado WHERE nombre LIKE "M%";</pre> <p>Muestra los datos de los empleados cuyo nombre comienza con M.</p> <p>Formatos:</p> <p>"Ma%" => comienza con Ma "%Ma%" => contiene Ma "%Ma" => termina con Ma</p> |
|------|---|---|

¿Qué son las Funciones de agregación?

Las funciones de agregación son funciones que operan sobre un conjunto de tuplas de entrada y producen un único valor de salida. En la siguiente tabla se describen las funciones de agregación definidas por SQL.

| | | |
|-------|---|---|
| AVG | Retorna como resultado el promedio del atributo indicado para todas las tuplas del conjunto. El atributo debe ser de tipo numérico. | SELECT AVG (salario) Muestra el salario promedio de los empleados. |
| COUNT | Retorna como resultado la cantidad de tuplas involucradas para el atributo indicado. | SELECT COUNT (*) FROM empleado WHERE dpto_nro = 24; Muestra la cantidad de empleados que tiene el departamento número 24. |
| MAX | Retorna como resultado el valor más grande dentro del conjunto de tuplas para el atributo indicado. | SELECT MAX(salario) FROM empleado; Muestra al empleado que tiene el salario más alto. |
| MIN | Retorna como resultado el valor más pequeño dentro del conjunto de tuplas para el atributo indicado. | SELECT MIN (comision) FROM empleado; Muestra la comisión más pequeña entre todos los empleados. |

| | | |
|-----|---|---|
| SUM | Retorna como resultado la suma del valor del atributo indicado para todas las tuplas del conjunto. El atributo debe ser de tipo numérico. | SELECT SUM (salario) FROM empleado; Muestra la suma de todos los salarios de los empleados. |
|-----|---|---|

¿Qué son las Funciones de agrupación?

Las funciones de agrupación permiten generar subconjuntos de tuplas para obtener información precisa de cada grupo.

| | | |
|----------|---|--|
| GROUP BY | Permite agrupar las tuplas de una consulta por algún criterio para obtener un resultado. | <pre>SELECT A.nombre, COUNT (A.legajo) FROM alumno A, inscripcion I WHERE A.legajo = I.legajo GROUP BY A.nombre;</pre> <p>Muestra la cantidad de inscripciones que tiene cada uno de los alumnos.</p> |
| HAVING | Indica la condición de filtro que debe respetar un grupo. Dentro de esta clausula se pueden utilizar funciones de agregación. | <pre>SELECT A.nombre, nombre_materia FROM alumno A, examen E, materia M WHERE A.legajo = E.legajo AND E.idMateria = M.idMateria GROUP BY A.nombre materia, nombre_materia HAVING AVG (E.nota) > 6;</pre> <p>Muestra por alumno el nombre de las materias cuyo promedio sea mayor a 6.</p> |

¿Qué son las operaciones de pertenencia a conjuntos?

Las operaciones de pertenencia a conjuntos se emplean cuando se utilizan subconsultas, o sea la consulta SQL contiene otra consulta dentro de ella.

| | | |
|----|--|---|
| IN | <p>Permite comprobar si un elemento es parte de un conjunto. Para cada tupla del resulta se verifica que se encuentre en el resultado de la subconsulta.</p> | <pre>SELECT legajo FROM alumno WHERE idCarrera = IN (SELECT idCarrera FROM carrera WHERE duracion = 5);</pre> <p>La subconsulta retorna todos los id.Carrera correspondientes a las carreras que tengan 5 años de duración. Para cada tupla de la tabla alumno se verifica si el idCarrera del alumno está dentro de ese conjunto de idCarreras que arrojó la subconsulta. Si está el legajo del alumno es parte del resultado.</p> |
|----|--|---|

| | | |
|-----------|--|--|
| NOT IN | <p>Permite comprobar si un elemento no es parte de un conjunto. Para cada tupla del resultado se verifica que no se encuentre en el resultado de la subconsulta.</p> | <pre>SELECT legajo FROM alumno WHERE idcarrera NOT IN (SELECT idcarrera FROM carrera WHERE duracion = 5);</pre> <p>En este caso los legajos del resultado NO están dentro de los resultados de la subconsulta.</p> |
|-----------|--|--|

Tipos de operaciones con valores nulos

| Operaciones con valores nulos | | |
|-------------------------------|--|---|
| IS NUL/ NOT NULL | Cuando un dominio de una atributo puede tener valores nulos, incorpora el conjunto de valores posibles el valor NULL. Este valor se almacena por defecto. Entonces para preguntar en una consulta por este valor de dominio se utilizan esto operadores. | <pre>SELECT nombre FROM empleado WHERE idlocalidad IS NULL;</pre> |

¿Qué son las operaciones de insertar, borrar y modificar?

Son operaciones que nos permiten ingresar datos, borrar o modificar a una tabla.

INSERTAR

La cláusula utilizada para agregar tuplas a una tabla es **INSERT INTO**

INSERT INTO alumno (legajo, nombre, apellido) **VALUES**

(123, "María", "Lopez"),
(234, "Mirta", "Hilario");

INSERT INTO alumno **VALUES**

(123, "María", "Lopez"),
(234, "Mirta", "Hilario");

Ambas instrucciones son válidas, la primera contiene los atributos de la tabla alumno, y la segunda no. Se debe tener cuidado en el segundo formato de respetar el ingreso de los datos según el orden de los atributos dados en la creación de la tabla.

Si el dato es auto_increment y no se especifica la lista de atributos antes del Values, se debe dejar el lugar correspondiente a ese dato.

| | |
|-----------|---|
| BORRAR | <p>La cláusula utilizada es DELETE FROM</p> <pre>DELETE FROM alumno WHERE idLocalidad = (SELECT idLocalidad FROM localidad WHERE nombre = "Martinez");</pre> <p>Elimina de la tabla alumno todos los alumnos de la localidad de Martínez.</p> |
| MODIFICAR | <p>La cláusula utilizada para modificar es UPDATE...SET</p> <pre>UPDATE carrera SET duracion = 6 WHERE nombre LIKE "ingenieria%";</pre> <p>Modifica la duración de las carreras que comienzan con Ingeniería y las lleva a 6 años.</p> |