3.3 LENGUAJE DE CONSULTA ESTRUCTURADO

- SQL usa los términos *tabla*, *fila* y *columna* para *relación*, *tupla* y *atributo*, respectivamente.

SQL significa Lenguaje Estructurado de Consultas (*Structured Query Languaje*) y se ha establecido Como el lenguaje estándar de bases de datos relacionales, esto significa que su uso esta generalizado a nivel internacional. El lenguaje SQL tiene varios componentes, los básicos son:

- El *lenguaje de definición de datos* para especificar el esquema de la base de datos.
- El *lenguaje de manipulación de datos* para expresar las consultas a la base de datos y las modificaciones.
- La definición de vistas,
- El control de transacciones.

- Integridad - órdenes para especificar las restricciones de integridad que deben satisfacer los datos almacenados en la base de datos. Las actualizaciones que violen las restricciones de integridad se rechazan.
- Autorización - órdenes para especificar derechos de acceso para las relaciones y las vistas.
- SQL incorporado y SQL dinámico - se pueden incorporar las instrucciones de SQL en lenguajes de programación de propósito general, tales como C, C++, Java, Cobol, Pascal y Fortran.

Existen tres maneras de utilizar SQL

1.- *Ejecución directa* - es el SQL interactivo, las instrucciones SQL se introducen a través de una herramienta que las traduce inmediatamente a la base de datos, por lo que se ejecutan al instante.

2.- *Ejecución dinámica* - El SQL se incrusta en módulos especiales que pueden ser invocados una y otra vez desde distintas aplicaciones.

3.- *Ejecución incrustada o embebida* - Las instrucciones SQL se colocan como parte del código de otro lenguaje anfitrión (C, Java, Pascal, Visual Basic). Estas instrucciones están separadas del resto del código de forma conveniente. Al compilar el código se utiliza un pre - compilador de la propia base de datos para traducir el SQL.

Algunas reglas sintácticas de SQL.

- En SQL no se distingue entre mayúsculas y minúsculas. Da igual como se escriba.
- El final de una instrucción lo determina el signo del punto y coma.
- Los comandos SQL (SELECT, INSERT) pueden ser partidos por espacios o saltos de línea antes de finalizar la instrucción.

Los tipos de datos de los campos.

Para poder definir el esquema de una tabla, es necesario especificar el tipo de dato de cada uno de sus campos (atributos). SQL define los siguientes tipos de datos:

- Números.
- Cadenas de caracteres.
- Fechas y horas.
- Cadenas de bits.

Tipos numéricos.

ZEROFILL se utiliza para rellenar con ceros a la izquierda la representación de un valor numérico hasta completar el ancho definido. Si se especifica ZEROFILL en un campo numérico, MariaDB añade automáticamente el atributo UNSIGNED, que significa “sin signo”, por lo que el campo solo podrá almacenar el cero y números positivos. Los corchetes cuadrados [ ] indican que el parámetro es opcional.

- **INT[(longitud)] [UNSIGNED] [ZEROFILL]**
- Número entero de tamaño normal.
- **longitud** indica el **ancho de visualización** (no limita el rango).
- Rango con signo: **-2,147,483,648 a 2,147,483,647**
- Rango sin signo: **0 a 4,294,967,295**

Ejemplo: id INT(8) ZEROFILL

BOOL

Tipo lógico.

- El valor **0** se considera **falso**
- Cualquier valor distinto de **0** se considera **verdadero**

Ejemplo: activo BOOL

DOUBLE[(longitud, decimales)] [UNSIGNED] [ZEROFILL]

- Número de **punto flotante**, permite valores decimales.
- **longitud**: ancho total del número (enteros + decimales)
- **decimales**: cantidad de dígitos a la derecha del punto decimal

Ejemplo: precio DOUBLE(8,2)

NUMERIC[(longitud, decimales)] [UNSIGNED] [ZEROFILL]

Número **exacto** con decimales fijos (sin errores de redondeo binario).

- Se comporta igual que DECIMAL
- Recomendado para **cantidades monetarias**

Ejemplo: monto NUMERIC(8,2)

- Número con 8 dígitos totales, de los cuales 3 son decimales:
- NUMERIC(8,3)

Tipos de cadenas de caracteres

CHAR(longitud) [BINARY]

- Cadena de caracteres de **longitud fija**.
- Almacena siempre el número de caracteres especificado en *longitud*, rellenando con **espacios a la derecha** cuando el valor almacenado es más corto.
- **longitud**: número de caracteres que puede almacenar la columna.
  - Rango: **0 a 255 caracteres**.
- La opción **BINARY** indica que las comparaciones se realizan de forma **binaria**, es decir, distinguen entre mayúsculas y minúsculas.
- Ejemplo: nombre CHAR(20)
- Ejemplo sensible a mayúsculas: codigo CHAR(5) BINARY

VARCHAR(longitud)

Cadena de caracteres de **longitud variable**.

- **longitud** representa la longitud máxima de la columna.
- En MariaDB, el tamaño máximo permitido es de **hasta 65,535 bytes**, dependiendo del conjunto de caracteres utilizado.
- Almacena únicamente los caracteres necesarios, más:
  - **1 byte adicional** para la longitud (hasta 255 caracteres)
  - **2 bytes** si la longitud declarada es mayor a 255
- Los valores VARCHAR:
  - **No se recortan** al almacenarse.
  - **Conservan los espacios finales**, conforme al estándar SQL.

Ejemplo: correo VARCHAR(100)

ENUM('valor1','valor2',)

Una **enumeración**.

Permite almacenar **un solo valor** de una lista predefinida.

- Puede almacenar:
  - Uno de los valores definidos
  - NULL
  - El valor especial vacío ''
- Puede tener hasta **65,535 valores distintos**.
- Internamente, los valores ENUM se representan como **enteros**.

Ejemplo: estado ENUM('ACTIVO','INACTIVO','SUSPENDIDO')

SET('valor1','valor2',)

Un **conjunto de valores**.

- Permite almacenar **cero o más valores** de una lista definida.
- Una columna SET puede tener hasta **64 valores distintos**.
- Los valores se almacenan internamente como **enteros**.
- Los valores se separan por comas.

Ejemplo: permisos SET('LECTURA','ESCRITURA','ELIMINAR')

Tipos de fechas y hora.

MariaDB proporciona varios tipos de datos para almacenar fechas y horas, así como funciones para operar con ellas.

DATE

Almacena una **fecha** (año, mes y día).

- Rango soportado: **'1000-01-01' a '9999-12-31'**
- Formato de visualización: **YYYY-MM-DD**
- Acepta valores como cadenas de caracteres, números o funciones.

Ejemplo: fecha_nacimiento DATE

DATETIME

Almacena una **combinación de fecha y hora**.

- Rango soportado: **'1000-01-01 00:00:00' a '9999-12-31 23:59:59'**
- Formato: **YYYY-MM-DD HH:MM:SS**
- No depende de la zona horaria.

Ejemplo: fecha_registro DATETIME

TIMESTAMP[(fsp)]

Almacena una **marca temporal**.

- Rango soportado (según plataforma):
- **'1970-01-01 00:00:01' a '2038-01-19 03:14:07'**
- Formato: **YYYY-MM-DD HH:MM:SS**
- **Depende de la zona horaria** del servidor.
- Puede actualizarse automáticamente en operaciones INSERT o UPDATE.

Ejemplo:

fecha_modificacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP
ON UPDATE CURRENT_TIMESTAMP

El parámetro (fsp) indica la precisión de fracciones de segundo (0 a 6).

TIME

Almacena una **hora** o un **intervalo de tiempo**.

- Rango soportado: **'-838:59:59' a '838:59:59'**
- Formato: **HH:MM:SS** (o HHH:MM:SS para valores grandes)

Ejemplo: duracion TIME

YEAR

Almacena un **año**.

- Formato: **4 dígitos**
- Rango permitido: **1901 a 2155**, y **0000**

Ejemplo: anio YEAR

Formatos válidos de entrada

- 'YYYY-MM-DD'
- 'YYYY-MM-DD HH:MM:SS'

Como **cadenas sin delimitadores**:

- 'YYYYMMDD'
- 'YYYYMMDDHHMMSS'

Como **números**:

- 20240115
- 20240115123045

Como **funciones**:

- NOW()
- CURRENT_DATE
- CURRENT_TIMESTAMP

Operadores relacionales y lógicos.

- Antes de iniciar, la instrucción **SELECT**, permite **consultar datos** de la base de datos. Esta parte del lenguaje se conoce como **DQL (Data Query Language)** y forma parte del **DML (Data Manipulation Language)**.

Operadores relacionales (comparación)

- Los operadores relacionales comparan valores y devuelven:
- **1** (TRUE / verdadero)
- **0** (FALSE / falso)
- **NULL** (desconocido)

Funcionan tanto con **números** como con **cadenas de caracteres**.

- **= (igual)**

Comprueba si dos valores son iguales.

Ejemplo: SELECT 1 = 0; Resultado: 0

- **<> , != (diferente)**

Comprueba si dos valores son distintos.

- Ejemplo: SELECT 'zapp' <> 'zappp'; Resultado = 1

- **< (menor que)**

Comprueba si un valor es menor que otro.

Ejemplo: SELECT 2 < 2; Resultado = 0

- **<= (menor o igual)**

Ejemplo SELECT 0.1 <= 2; Resultado = 1

- **> (mayor que)**

Ejemplo SELECT 2 > 2; Resultado = 0

- **>= (mayor o igual)**

Ejemplo: SELECT 2 >= 2; Resultado = 1

- **IS valor_booleano / IS NOT valor_booleano**

Permite comprobar si un valor es **TRUE**, **FALSE** o **UNKNOWN**.

Ejemplo: SELECT NULL IS UNKNOWN; Resultado: 1

- **IS NULL / IS NOT NULL**

Comprueba si un valor es o no NULL.

Ejemplo: SELECT NULL IS NULL; Resultado: 1

- **GREATEST(valor1, valor2, ...)**

Devuelve el **mayor valor** de la lista.

Ejemplo: SELECT GREATEST(34, 3, 5); Resultado: 34

- **LEAST(valor1, valor2, ...)**

Devuelve el **menor valor** de la lista.

Ejemplo: SELECT LEAST(34, 3, 5); Resultado: 3

Operadores lógicos

- Los operadores lógicos evalúan expresiones y devuelven:

➢**1** (TRUE)

➢**0** (FALSE)

➢**NULL** (desconocido)

- **NOT , !**

Negación lógica.

Ejemplo: SELECT NOT 0; Resultado: 1

- **AND , &&**

Devuelve **1** si todos los operandos son verdaderos.

Ejemplo: SELECT 1 AND 0; Resultado: 0

- **OR , ||**

Devuelve **1** si al menos un operando es verdadero.

Ejemplo: SELECT 1 OR 0; Resultado: 1

3.3.2 Creación de tablas

- Antes de comenzar a trabajar con una base de datos en **MariaDB 12.1**, es necesario crearla. Esto se realiza con el comando:

MariaDB> CREATE DATABASE nombre_base_datos;

- Como ejemplo, crearemos la base de datos llamada **escuela**:

MariaDB> CREATE DATABASE escuela;

- El comando CREATE DATABASE únicamente crea la base de datos, pero no la seleccionapara su uso. Para poder trabajar con ella, es necesario utilizar el comando USE, de la siguiente manera:

MariaDB> USE nombre_base_datos;

- En este caso:

MariaDB> USE escuela;

- Cuando se inicien sesiones posteriores en MariaDB, no será necesario crear nuevamente la base de datos; bastará con seleccionarla usando el comando USE.

- Es importante notar que, si se crea la base de datos con CREATE DATABASE y no se ejecuta posteriormente el comando USE, no será posible comenzar a trabajar con ella, ya que no se selecciona en qué base de datos deben ejecutarse las operaciones.

- **Consultar la base de datos activa**

- Para saber en qué base de datos se está trabajando actualmente, MariaDB proporciona la función DATABASE():

MariaDB> SELECT DATABASE();

- Una vez que nuestra base de datos está lista para trabajar, podemos crear una tabla dentro de esta base de datos utilizando el comando CREATE TABLE, especificando los campos correspondientes y sus tipos de datos:

```
MariaDB> CREATE TABLE alumnos (
nombre VARCHAR(20),
FechaIngreso DATE,
sexo CHAR(1),
promedio DOUBLE(4,2),
edad INT(2)
);
```

- Una vez creada la tabla, podemos consultar todas las tablas existentes en la base de datos activa utilizando el comando SHOW TABLES, como se muestra a continuación:

MariaDB> SHOW TABLES;

Instrucciones básicas del Lenguaje de Manipulación de Datos (LMD)

- Otro nivel del SQL es el **Lenguaje de Manipulación de Datos (LMD)**, conocido en inglés como **DML (Data Manipulation Language)**. En este nivel es donde principalmente trabajan los usuarios finales de la base de datos.

- El LMD está formado por un conjunto de instrucciones que permiten al usuario **consultar, insertar, modificar y eliminar datos**, así como llenar formularios y generar reportes, de acuerdo con los permisos o autorizaciones que tenga asignados.

El LMD es un **lenguaje declarativo**, lo que significa que el usuario indica **qué información desea obtener o modificar**, pero no especifica **cómo** el sistema gestor de bases de datos debe localizarla o procesarla internamente.

- El LMD se utiliza para manipular **tuplas (registros)** de la base de datos. Las principales instrucciones del LMD son:
- SELECT (consulta de datos – DQL)
- INSERT (inserción de registros)
- UPDATE (actualización de registros)
- DELETE (eliminación de registros)

Borrar una tupla (registro): DELETE

- Para borrar una tupla o registro de una tabla se utiliza el comando

DELETE, generalmente acompañado de la cláusula WHERE.

- La sintaxis general es la siguiente:

DELETE FROM nombre_tabla [WHERE condición];

- La instrucción DELETE elimina todos los registros de la tabla nombre_tabla que cumplen con la condición especificada y devuelve el número de filas eliminadas.

- Supongamos que la tabla **alumnos** contiene varios registros. Si se desea borrar el **segundo registro**, se debe utilizar una condición en la cláusula WHERE que identifique de manera única a dicho registro. Por ejemplo:

MariaDB> DELETE FROM alumnos WHERE nombre = 'Atenogenes Lopez';

- Esta instrucción elimina la segunda tupla de la tabla **alumnos**.

- Si se solicita borrar todos los registros cuyo campo sexo tenga el valor 'M':

MariaDB> DELETE FROM alumnos WHERE sexo = 'M';

- Entonces se eliminarán todos los registros que cumplan con dicha condición, por ejemplo, los dos primeros registros de la tabla.

- Finalmente, para borrar **todos los registros de una tabla**, se utiliza la instrucción DELETE sin la cláusula WHERE:

MariaDB> DELETE FROM alumnos;

- **Advertencia:** Esta operación elimina todos los registros de la tabla y no puede deshacerse si no se cuenta con un respaldo previo.

Haciendo consultas sencillas: SELECT

- **DQL** es la abreviatura de *Data Query Language* (Lenguaje de Consulta de Datos) y forma parte del **Lenguaje de Manipulación de Datos (LMD)**. El **único comando** que pertenece al DQL es SELECT.
- El comando SELECT permite:

➢Obtener datos de ciertas columnas de una tabla (**proyección**).

➢Obtener registros (tuplas) de una tabla de acuerdo con ciertos criterios (**selección**).

➢Combinar datos de diferentes tablas (**producto cartesiano** y **JOIN**).

➢Realizar cálculos sobre los datos.

➢Agrupar datos.

Sintaxis general del comando SELECT

- Para visualizar el contenido de una tabla en **MariaDB 12.1**, se utiliza el comando SELECT, cuya sintaxis general es la siguiente:

SELECT <lista_de_atributos>

FROM <lista_de_tablas>

[WHERE <condiciones>];

- **Lista de atributos**: indica la información que se desea mostrar. Puede ser una lista de columnas específicas o el carácter *, que indica que se deben mostrar **todas las columnas** de la tabla.
- **Lista de tablas**: especifica la tabla o tablas donde se encuentran los datos que se desean consultar.
- **WHERE**: es una cláusula opcional. Si se incluye, define las condiciones que deben cumplir los registros para formar parte del resultado de la consulta.

Ejemplo de consulta básica

- Para consultar todos los registros y todas las columnas de la tabla **alumnos**, se utiliza la siguiente instrucción:

MariaDB> SELECT * FROM alumnos;

- La cláusula WHERE se emplea cuando se requiere filtrar los registros. Las condiciones son expresiones lógicas (booleanas) que permiten identificar las tuplas que serán recuperadas por la consulta.

- Cuando una tabla no contiene registros, al ejecutar una instrucción SELECT, MariaDB mostrará un resultado vacío, indicando que no hay filas almacenadas.

- Esto significa que **antes de realizar consultas sobre una tabla, es necesario haber insertado datos previamente** mediante la instrucción INSERT.

Inserción de datos en una tabla: **INSERT INTO**

- Para introducir datos en una tabla, se utiliza la instrucción INSERT INTO. Su sintaxis general es la siguiente:

```
MariaDB> INSERT INTO nombreTabla
    VALUES ('valor_del_campo1', 'valor_del_campo2', ..., 'valor_del_campoN');
```

- Cada valor debe corresponder, en orden, a los campos definidos en la estructura de la tabla.

Por ejemplo, para insertar datos en la tabla **alumnos**:

```
MariaDB> INSERT INTO alumnos
     VALUES ('Atenogenes Lopez', '2024-01-15', 'M', 8.5, 20);
```

- Una vez que se han insertado registros en la tabla, ya es posible observar el contenido completo utilizando el comando SELECT:

```
MariaDB> SELECT * FROM alumnos;
```

- Esta instrucción muestra **todas las columnas y todos los registros** almacenados en la tabla **alumnos**.

- **Sintaxis general de UPDATE**

UPDATE nombre_tabla

SET nombre_columna = nuevo_valor

[WHERE condición];

- nombre_tabla: tabla que contiene los registros a modificar.
- SET: indica el campo o campos que se actualizarán.
- WHERE: cláusula opcional que define qué registros serán afectados.

- **Ejemplo de actualización con condición**
-
- Si se desea cambiar el departamento del profesor **Nicodemo Sanchez** en la tabla **profesores**, se puede utilizar la siguiente instrucción:

MariaDB> UPDATE profesores

SET depto = 'Quimica'

WHERE num_empl = 3172;

- Si la condición especificada en la cláusula WHERE se cumple para **varias tuplas**, entonces **todas ellas serán modificadas**.

Actualizaciones con operaciones aritméticas

- También es posible realizar operaciones aritméticas dentro de una instrucción UPDATE. Por ejemplo, si se desea multiplicar por 10 todas las calificaciones de la tabla **alumnos**, se puede utilizar:

MariaDB> UPDATE alumnos

SET promedio = promedio * 10;

- **Nota:** Si no se incluye la cláusula WHERE, la actualización se aplicará a **todos los registros** de la tabla.

Uso de CASE en instrucciones UPDATE

- SQL, y específicamente **MariaDB 12.1**, ofrece la construcción CASE, la cual permite formular **múltiples condiciones de actualización dentro de una sola instrucción**.

- La sintaxis general es la siguiente:

```
UPDATE nombre_tabla
SET nombre_columna = CASE
    WHEN condición1 THEN valor1
    ELSE valor2
```

- END;

- **Ejemplo de UPDATE con CASE**

- Por ejemplo, para asignar un valor distinto al campo promedio dependiendo de su contenido actual:

```
MariaDB> UPDATE alumnos
SET promedio = CASE
    WHEN promedio >= 6 THEN promedio
    ELSE 0
END;
```

- Esta instrucción conserva el promedio de los alumnos aprobados y asigna **0** a los alumnos no aprobados.

Creación de índices y restricciones

- Los **índices** y las **restricciones** se utilizan para **mejorar el rendimiento de las consultas** y **garantizar la integridad de los datos** almacenados en las tablas.

- **Creación de índices**
- Un **índice** permite acelerar la búsqueda de registros en una tabla, especialmente cuando se realizan consultas frecuentes sobre una o varias columnas.

- **Sintaxis general para crear un índice:**

CREATE INDEX nombre_indice
ON nombre_tabla (nombre_columna);

Ejemplo:

- Para crear un índice sobre el campo nombre de la tabla alumnos:

MariaDB> CREATE INDEX idx_nombre
ON alumnos (nombre);

- Este índice mejora el desempeño de las consultas que filtran o buscan por el campo nombre.

- También es posible crear **índices únicos**, los cuales aseguran que no existan valores duplicados en una columna:

MariaDB> CREATE UNIQUE INDEX idx_nombre_unico
ON alumnos (nombre);

Creación de restricciones (constraints)

- Las **restricciones** permiten imponer reglas sobre los datos para garantizar su validez e integridad.

- Las restricciones más comunes en MariaDB son:

➢PRIMARY KEY (clave primaria)

➢FOREIGN KEY (clave foránea)

➢UNIQUE (valores únicos)

➢NOT NULL (campo obligatorio)

➢CHECK (condición lógica)

Ejemplo de restricciones al crear una tabla:

```
CREATE TABLE alumnos
   id_alumno INT PRIMARY KEY,
   nombre VARCHAR(50) NOT NULL,
   promedio DOUBLE(4,2) CHECK (promedio >= 0 AND promedio <= 10)
• );
```

En este ejemplo:

- PRIMARY KEY identifica de forma única cada registro.
- NOT NULL evita valores nulos.
- CHECK valida que el promedio esté en un rango permitido.

Gestión de usuarios

- La **gestión de usuarios** permite controlar quién puede acceder a la base de datos y qué operaciones puede realizar cada usuario, garantizando la **seguridad del sistema**.

Creación de usuarios

- Para crear un nuevo usuario se utiliza la instrucción CREATE USER:

MariaDB> CREATE USER 'usuario1'@'localhost'

IDENTIFIED BY 'password123';

Asignación de privilegios

- Los privilegios determinan qué acciones puede realizar un usuario sobre una base de datos o tabla.

- **Ejemplo: otorgar permisos de consulta e inserción**

MariaDB> GRANT SELECT, INSERT

ON escuela.*

TO 'usuario1'@'localhost’;

- Este comando permite al usuario realizar consultas e insertar datos en todas las tablas de la base de datos **escuela**.

Revocación de privilegios

- Para retirar permisos previamente otorgados se utiliza REVOKE:

MariaDB> REVOKE INSERT

ON escuela.*

FROM 'usuario1'@'localhost';

Eliminación de usuarios

- Si un usuario ya no debe tener acceso al sistema, puede eliminarse con:

MariaDB> DROP USER 'usuario1'@'localhost';

Verificación de privilegios

- Para consultar los privilegios asignados a un usuario:

MariaDB> SHOW GRANTS FOR 'usuario1'@'localhost';