DDL (Data Definition Language)
Lenguaje de definición de datos

Se usan para crear, modificar o borrar objetos en una base de datos como tablas, vistas, activadores, y almacenar procedimientos.

Las instrucciones DDL más comunes son:
CREATE, ALTER y DROP.

Ejemplos:

- **CREATE TABLE** para crear una tabla.
- **ALTER TABLE** para modificar las características de una tabla.
- **DROP TABLE** para borrar la definición de la tabla de la base de datos.

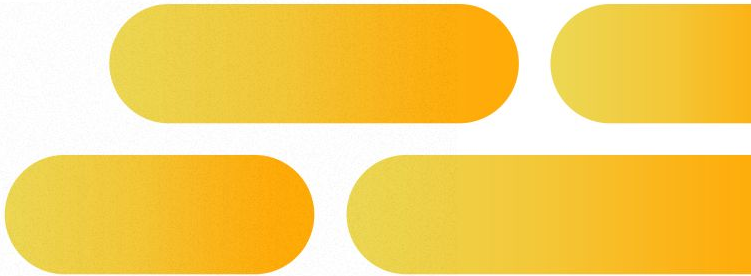
DCL (Data Control Language)
Lenguaje de control de datos

Permiten controlar quién o qué (un usuario en una base de datos puede ser una persona o un programa de aplicación) tiene acceso a objetos específicos en la base de datos.
Las instrucciones DCL también permiten controlar el tipo de acceso que cada usuario tiene a los objetos de una base de datos.

Las principales instrucciones DCL son:
GRANT y REVOKE.

Ejemplos:

- **GRANT** para dar permiso de lectura a una tabla.
- **REVOKE** para quitar todos los permisos a un usuario sobre una tabla.



TCL (Transaction Control Language)
Lenguaje de control de transacciones

Permiten manejar transacciones en una base de datos relacional.
Una transacción es una unidad lógica de trabajo que comprende una o más sentencias SQL, por lo general un grupo de Data Manipulation Language (DML).

Las principales instrucciones TCL son:
COMMIT y ROLLBACK.

Ejemplos:

- **COMMIT** para guardar el resultado de la transacción (grupo de sentencias DML).
- **ROLLBACK** para restaurar la base de datos a su estado original, hasta el último commit.

DML (Data Manipulation Language)

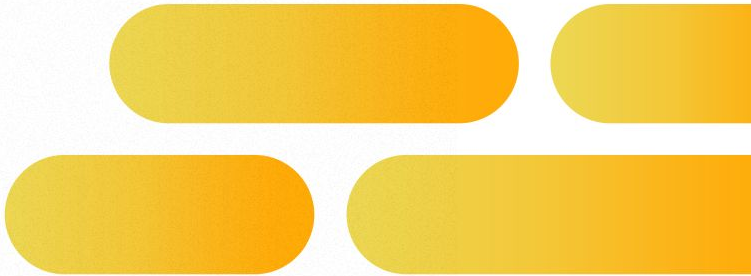
Lenguaje de manipulación de datos

Se usan para recuperar, agregar, modificar o borrar datos almacenados en los objetos de una base de datos. Son los tipos de instrucciones probablemente son más usadas.

Las palabras clave asociadas con las instrucciones DML son:
SELECT, INSERT, UPDATE y DELETE.

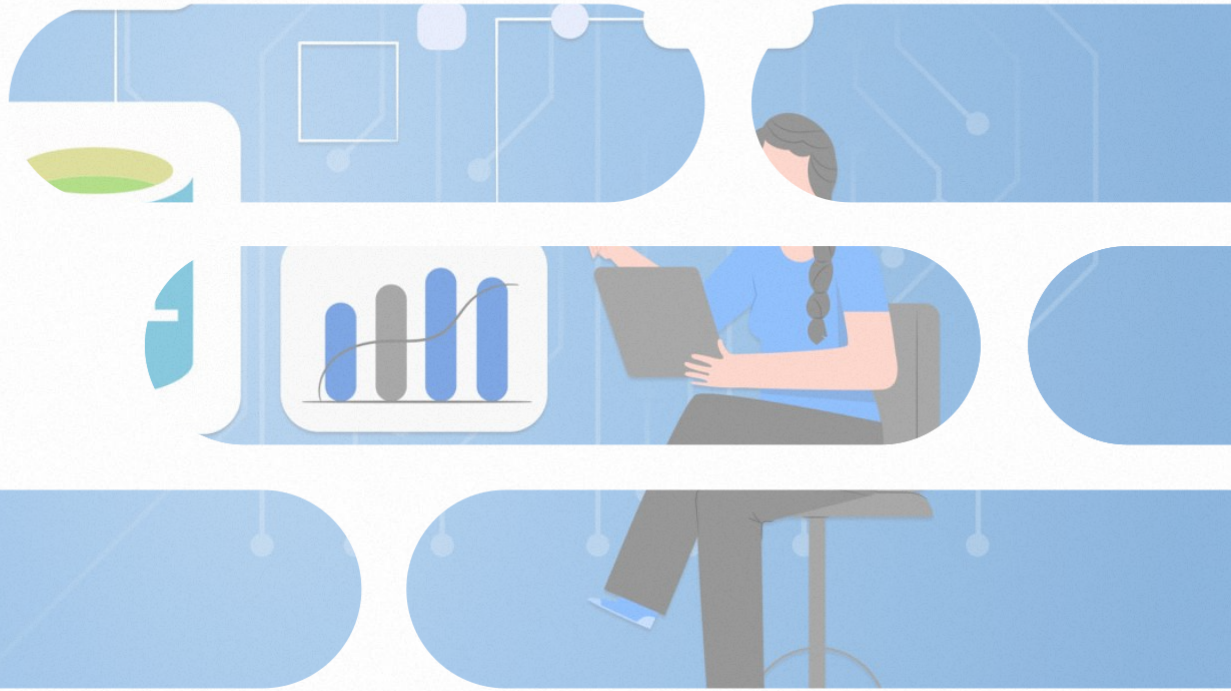
Ejemplos:

- **INSERT** para agregar datos a una tabla.
- **UPDATE** para modificar datos de una tabla.
- **DELETE** para eliminar datos de una tabla.
- **SELECT** para leer datos de una tabla.



Tipos de objetos

- Tablas
- Contraints
- Vistas
- Triggers
- Funciones
- Procedimientos



¿Que es una Tabla?

Son objetos de esquema que tienen los datos de SQL.

Tipos de Tablas:

Tablas base persistentes

Un objeto de esquema nombrado definido por la definición de una tabla en la instrucción CREATE TABLE.

Tablas temporales globales

Sólo existe cuando se hace referencia dentro del contexto de la sesión SQL en la cual se creó. Cuando la sesión termina, la tabla ya no existe.

Tablas temporales locales creadas

A diferencia de la tabla temporal global en una tabla temporal local sólo se podrá acceder dentro del módulo asociado.

Tablas temporales locales declaradas

Sólo se hace referencia a una tabla temporal local declarada dentro del contexto de la sesión SQL en la cual se creó.

Estructura o Esquema de una Tabla

- Cada tabla debe tener su nombre único.
- El nombre de cada columna (atributo o campo): es un elemento etiquetado de una tupla (como por ejemplo, el código de un empleado).
- El tipo de dato de cada columna.
- La fila (tupla o registro): es el conjunto de datos que representa un objeto simple.

Código	*Nombre*	*Posición*	*Salario*
1	Edgardo Trujillo	Gerente	19000
2	Lidimarie Fonsi	Empleada	12000
3	Jean Piaget	Empleado	13500
4	Jerome Bruner	Empleado	14000

Ejemplo:

- **Nombre de la Tabla:** Usuario
- **Llave Primaria:** ID
  - **Tipo de Dato:** Integer
- **Campos o Atributos:**
  - Nombre
    - **Tipo de Dato:** Varchar
  - Apellido
    - **Tipo de Dato:** Varchar
  - Email
    - **Tipo de Dato:** Varchar
  - Fecha_Nacimiento:
    - **Tipo de Dato:** Datetime
  - Activo
    - **Tipo de Dato:** TinyInt

Usuario	
PK	**ID (INT)**
	Nombre (VARCHAR)
	Apellido (VARCHAR)
	Email (VARCHAR)
	Fecha_Nacimiento (DATETIME)
	Activo (TINYINT)

Tipos de Datos

- **Numérico**
  - 🟢**TINYINT** (números desde -128 hasta 127)
  - 🟢**SMALLINT** (números desde -32768 hasta 32767)
  - 🟢**MEDIUMINT** (números desde -8388608 hasta 8388607)
  - 🟢**INT** (números desde -2147483648 hasta 2147483647)
  - 🟢**BIGINT** (números desde -9223372036854775808 hasta 9223372036854775807)
  - 🟣**DECIMAL** (números decimales de punto fijo grandes)
  - 🟣**FLOAT** (números decimales de punto flotante pequeños)
  - 🟣**DOUBLE** (números decimales de punto flotante grandes)
- **Fecha y marca temporal**
  - **DATE** (formato YYYY-MM-DD)
  - **DATETIME** (formato YYYY-MM-DD HH:MM:SS, rango soportado de '1000-01-01 00:00:00' a '9999-12-31 23:59:59')
  - **TIMESTAMP** (formato YYYY-MM-DD HH:MM:SS, rango soportado de '1970-01-01 00:00:01' UTC a '2038-01-19 03:14:07' UTC)
  - **TIME** (formato HH:MM:SS)
  - **YEAR** (formato YYYY)

Tipos de Datos

- **Cadena**
  - 🟢**CHAR** (Siempre reservará espacio para la longitud definida aunque no se utilice. La longitud máxima es de 255.)
  - 🟢**VARCHAR** (No reserva el espacio de la longitud máxima definida. La longitud máxima es de 255.)
  - 🟣**TINYTEXT** (Almacena una cadena de datos (solo caracteres; no admite números ni caracteres especiales) de una longitud máxima de 255 caracteres.)
  - 🟣**TEXT** (longitud máxima de 65,535.)
  - 🟣**MEDIUMTEXT** (longitud máxima de 16,777,215 caracteres.)
  - 🟣**LONGTEXT** (longitud máxima de 4,294,967,295 caracteres.)
  - 🟠**BLOB** (Binary Large Object, Longitud máxima de 65,535 bytes de datos)
  - 🟠**MEDIUMBLOB**
  - 🟠**LONGBLOB**
- **Espacial**
- **JSON**

https://blog.hubspot.es/website/tipos-de-datos-mysql

Constraints

Con el fin de asegurar la integridad de los datos, SQL proporciona una serie de restricciones de integridad, reglas que se aplican a la base de datos para restringir los valores que se pueden colocar en esas tablas. Se pueden aplicar restricciones a columnas individuales, a tablas individuales o a múltiples tablas.

Usuario	
PK	ID (INT)
UK	Nombre (VARCHAR)
	Apellido (VARCHAR)
	Email (VARCHAR)
	Fecha_Nacimiento (DATETIME)
	Activo (TINYINT)

Constraints más comunes:

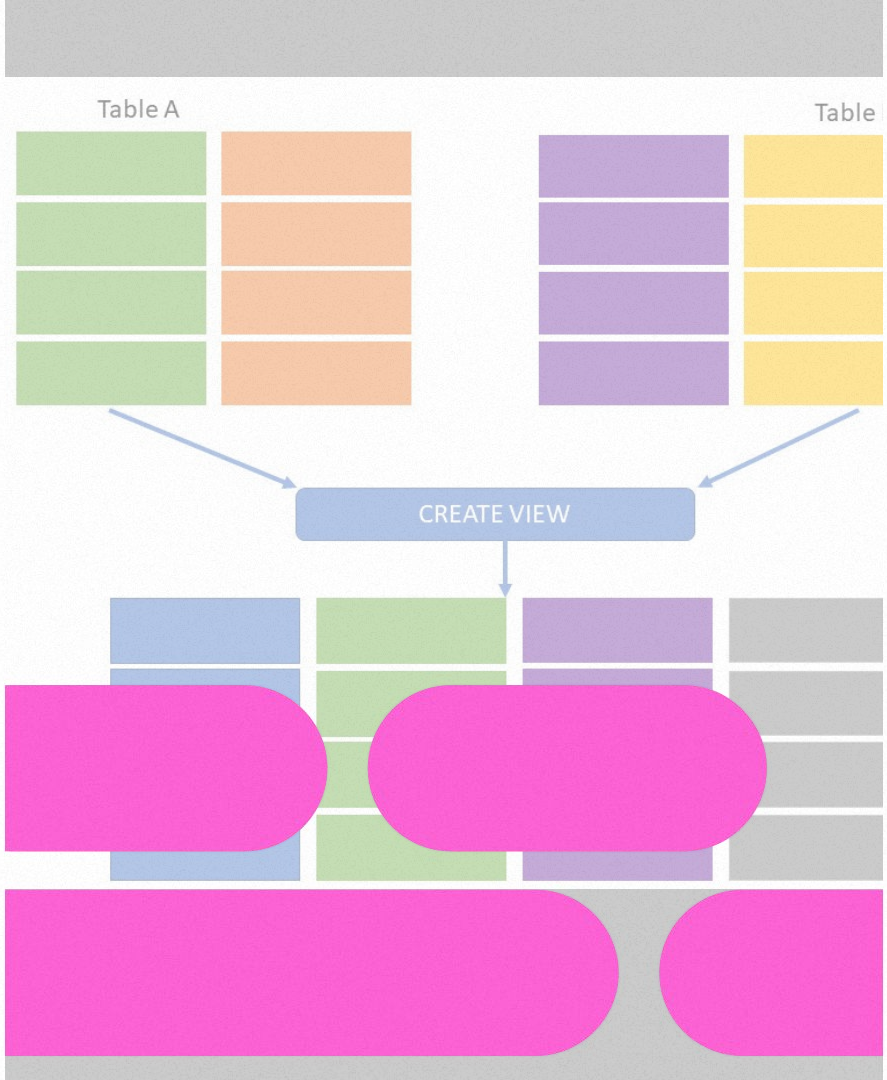
- **NOT NULL:** Asegura que en una columna no haya valores nulos.
- **UNIQUE:** Asegura que cada valor sea único.
- **PRIMARY KEY:** Una combinación de NOT NULL y UNIQUE. Identifica cada fila de una tabla.
- **FOREIGN KEY:** Evita que se destruya la relación entre tablas.
- **CHECK:** Garantiza que los valores de una columna cumplan una condición.
- **DEFAULT:** Asigna un valor por default cuando ningún valor es especificado.
- **CREATE INDEX:** Sirve para leer información más rápidamente.

https://www.w3schools.com/mysql/mysql_constraints.asp

¿Qué son las Vistas?

Es una tabla virtual que a diferencia de las tablas base persistentes, en la vista no hay datos almacenados.

Permiten seleccionar información específica de una o más tablas, basada en las instrucciones de consulta en esa definición. Una vez que se crea una vista, simplemente se invoca llamándola por su nombre en una consulta como en una tabla base. Los datos entonces se presentan como si se buscaran en una tabla base.



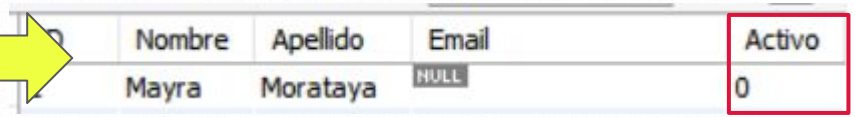
Triggers

Cuando se realiza una modificación específica, el activador (disparador o trigger) es invocado automáticamente, causando que ocurra una acción adicional.
Nunca se invoca directamente el trigger.

Ejemplo:

Cada vez que se inserta un registro en la tabla usuario sin email, se va a guardar como inactivo (Activo = 0).

```
INSERT INTO Usuario (Nombre, Apellido)
VALUES ('Mayra', 'Morataya')
```



	Nombre	Apellido	Email	Activo
	Mayra	Morataya	NULL	0

Trigger Creado

- **Nombre del Trigger:** BI_Usuario
- **Tipo:** Before Insert.
  - Significa que el trigger se va a ejecutar antes de insertar el registro.
- **NEW**: hace referencia a la fila que se está insertado.
  - *NEW + punto + Nombre Campo*, proporciona acceso al valor del campo que se especifique
- **Descripción:** Si el campo Email de la fila que se está insertado es null, el campo Activo se le asigna valor 0, sino se le asigna valor 1.
- **DELIMITER**: Sirve para indicarle a MySQL donde finaliza el bloque de código.

```
DELIMITER $$

CREATE TRIGGER BI_Usuario
    BEFORE INSERT ON Usuario
    FOR EACH ROW
BEGIN
    IF (NEW.Email IS NULL) THEN
        SET NEW.Activo = 0;
    ELSE
        SET NEW.Activo = 1;
    END IF;
END$$
```

Propiedades de los Triggers

BEFORE: Se ejecuta antes de la operación, si el trigger falla puede ocasionar que la operación (insert, update, delete) no se ejecute.
AFTER: Se ejecuta luego de que la operación se realizó exitosamente.

Los triggers solo se ejecutarán luego una operación INSERT, UPDATE o DELETE, según se haya creado, *nunca sobre una operación SELECT*.

Dentro del BEGIN se debe de indicar la lógica a ejecutar.

Dentro de un trigger se pueden hacer otros queries, como por ejemplo insertar datos a otra tabla.

Sintaxis general:

```
CREATE
    [DEFINER = usuario]
    TRIGGER [IF NOT EXISTS] nombre_trigger
    { BEFORE | AFTER }
    { INSERT | UPDATE | DELETE }
    ON nombre_tabla
    FOR EACH ROW
    { FOLLOWS | PRECEDES }
    BEGIN
        //Body
    END;
```

Funciones

Soportan solamente parámetros de entrada, y si no se proporciona ninguno, deberán utilizarse los paréntesis de todas formas. Si se proporciona más de un parámetro de entrada, deberán separarse utilizando comas. Siguiendo a las declaraciones de parámetro está la cláusula RETURNS. Se debe proporcionar el tipo de datos para el valor que es arrojado por la función.

Crear Funciones

- Necesitan un nombre único.
- Pueden recibir parámetros de entrada.
- En el body, se escribe la lógica de lo que la función hará.
- Retornan un resultado.
- Pueden ser usadas dentro de cualquier consulta DML.

DETERMINISTIC
Una función determinística es la que siempre devuelve el mismo resultado para la misma entrada. Por default las funciones se crean como no determinista.

READS SQL DATA
Indica que la función leerá datos de la BD.

MODIFIES SQL DATA
Indica que la función modificará datos en la BD.

```
CREATE
    [DEFINER = user]
    FUNCTION [IF NOT EXISTS] sp_name
    ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

 func_parameter:
    param_name type

characteristic: {
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL |
    READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
}

routine_body:
    Valid SQL routine statement
```

Procedimientos (Stored Procedure)

Soportan parámetros de entrada y de salida, y si no se proporciona ninguno, deberán utilizarse los paréntesis de todas formas. No retornan un valor aunque pueden tener parámetros de salida con valores generados.

Crear Procedimientos

- Necesitan un nombre único.
- Pueden recibir parámetros de entrada.
- Pueden tener parámetros de salida.
- En el body, se escribe la lógica de lo que el procedimiento hará..

DETERMINISTIC
Una función determinística es la que siempre devuelve el mismo resultado para la misma entrada. Por default las funciones se crean como no determinista.

READS SQL DATA
Indica que la función leerá datos de la BD.

MODIFIES SQL DATA
Indica que la función modificará datos en la BD.

```
CREATE
    [DEFINER = user]
    PROCEDURE [IF NOT EXISTS] sp_name
    ([proc_parameter[,...]])
    [characteristic ...] routine_body

 func_parameter:
    param_name type

characteristic: {
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL |
    READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
}

routine_body:
    Valid SQL routine statement
```

Sentencias

SELECT
INSERT
UPDATE
DELETE

SELECT: Consulta de Datos

Una vez que se crearon los objetos en una base de datos y las tablas base están pobladas de datos, se pueden presentar las consultas que permiten recuperar información específica de la base de datos.

- La instrucción SELECT para la recuperación de datos
- La cláusula WHERE para definir condiciones de búsqueda
- La cláusula GROUP BY para agrupar los resultados de una consulta
- La cláusula HAVING para especificar un grupo de condiciones de
- búsqueda
- La cláusula ORDER BY para ordenar los resultados de una consulta

```
SELECT [ DISTINCT | ALL ] { * | < selección de lista > }
FROM <tabla de referencia> [ {, <tabla de referencia> }... ]
[ WHERE <condición de búsqueda> ]
[ GROUP BY <especificación de agrupación> ]
[ HAVING <condición de búsqueda> ]
[ ORDER BY <condición de orden> ]
```

INSERT: Insertar datos

La instrucción INSERT permite agregar datos a las diferentes tablas en una base de datos.

- Si los nombres de columna no se especifican en la cláusula INSERT INTO, entonces deberá haber un valor por cada columna en la tabla y los valores deberán estar en el mismo orden en el que están definidos en la tabla.
- Se debe proporcionar un valor por cada columna en la tabla excepto para las columnas que permiten valores nulos o que tienen un valor definido por defecto.
- Se puede utilizar la palabra clave NULL (o null) como el valor de los datos en la cláusula VALUES para asignar un valor nulo a cualquier columna que permita valores nulos.

```
INSERT INTO <nombre de la tabla>
[ ( <nombre de la columna> ) [ { , <nombre de la columna> }... ] ) ]
VALUES ( <valor> [ { , <valor> }... ] )
```

UPDATE: Actualizar datos

Con la instrucción UPDATE se pueden modificar datos en una o más filas para una o más columnas.

- La cláusula UPDATE y la cláusula SET son obligatorias, mientras que la cláusula WHERE es opcional.
- La cláusula WHERE funciona aquí de una forma muy parecida a como lo hace en la instrucción SELECT.
- Solamente las filas que cumplen con estas condiciones son actualizadas.

```
UPDATE <nombre de la tabla>
SET <determinar expresión de la cláusula> [ {, <determinar expresión de la cláusula> }... ]
[ WHERE <condición de búsqueda> ]
```

DELETE: Eliminar datos

Contiene solamente dos cláusulas, y una sola de ellas es obligatoria.

- La cláusula WHERE, la cual es similar a la cláusula WHERE en una instrucción SELECT y en una instrucción UPDATE, requiere que se especifique una condición de búsqueda.
- *Si no se incluye una cláusula WHERE en la instrucción DELETE, todas las filas serán eliminadas de la tabla especificada.*
- Es importante comprender que la instrucción DELETE no elimina la tabla en sí, sino solamente filas en la tabla (la instrucción DROP TABLE, se utiliza para eliminar definiciones de tabla de la base de datos).

```
DELETE FROM <nombre de la tabla>
[ WHERE <condición de búsqueda> ]
```

¿Preguntas?