TEMA 6: SQL

IMPORTANTE:

Los campos que son **IDENTIFICADORES** se escriben tal cual (nombre tablas, bases de datos, etc.)

Los campos que son valores/texto se escriben entre → **'comillas simples'**

Las sentencias se finalizan usando → **;**

Los valores se separan con → **,**

Lo que sale en **AZUL** son palabras reservadas

Para seleccionar todo de una base de datos/tabla se usa → ***

Para acceder a una parte de algo se utiliza el operador → **.**

SENTENCIAS: → *Para usar una base en específico (crear/modificar tablas, etc.)* → `USE nombre_base;`

Crear base de datos

```
CREATE DATABASE IF NOT EXISTS Ejemplo DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

Ver bases de datos

```
SHOW DATABASES;
```

Crear usuarios

```
CREATE USER 'usuario'@'localhost' IDENTIFIED BY 'password';
```

Dar privilegios a usuarios

```
GRANT ALL PRIVILEGES ON Ejemplo.* TO 'usuario'@'localhost';
```

Crear tablas

```
CREATE TABLE ejemplo1(
Campo1 INT(7) PRIMARY KEY, /*Atributo Tipo(Longitud máxima) MODIFICADORES*/
Campo2 VARCHAR (10) UNIQUE NOT NULL /*El último no lleva ,(coma) al final!!!*/
...
);
```

```
CREATE TABLE IF NOT EXISTS ejemplo2 (
Campo1 INT(7) PRIMARY KEY,
Campo2 DECIMAL(20), /*(M,D)→ M dígitos , D de los M son decimales*/
...                 /*Lo de debajo crea una clave foránea para referenciar la otra tabla*/
CONSTRAINT FK_Campo1 FOREIGN KEY (Campo1) REFERENCES Ejemplo1(Campo1)
);
```

Modificadores: *(Los que no están en azul van antes que los otros)*

```
PRIMARY KEY ; UNIQUE ; NULL ; NOT NULL ; UNSIGNED ;ZEROFILL ; AUTO_INCREMENT
```

Tipos de datos:

```
BIT ; INT ; INTEGER ; BIGINT ; DOUBLE ; FLOAT ; DATE ; TIME ; DATETIME ; YEAR ; CHAR ;
VARCHAR ; BINARY
```

Crear índices (dentro de la creación de la tabla): *(Mejorar eficiencia consultas, actualmente muy similares)*

```
INDEX idx_id (id) /*Crear un índice general → nombre_indice(campo_indice)*/
```

```
KEY idx_nombre (nombre) /*Crear un índice para FOREIGN KEY→ nombre_indice(campo_indice)*/
```

Modificar tablas

```
ALTER TABLE ejemplo1 ADD edad INT(7) NOT NULL;
/*Atributo Tipo(Longitud máx.) MODIFICADORES ; CONSTRAINT ... ; PRIMARY KEY (Atributo)*/

ALTER TABLE ejemplo1 DROP edad;
/*Atributo ; FOREGIN KEY nombre_FK ; PRIMARY KEY */

ALTER TABLE ejemplo2 MODIFY Campo2 VARCHAR(100) UNIQUE;
/*Cambiar el tipo ; Agregar modificadores ; NO NOMBRE COLUMNA*/

ALTER TABLE ejemplo2 CHANGE Campo2 Campo2N INT(40) NOT NULL;
/*Lo mismo que MODIFY, pero permite cambiar el nombre de la tabla ; Campo2 → Campo2N*/

ALTER TABLE ejemplo1 RENAME TO ejemplo1N; /*Cambiar el nombre de la tabla*/
```

Eliminar tablas

```
DROP TABLE IF EXISTS ejemplo2;
```

Insertar datos *(Utilizo un ejemplo diferente para que sea más entendible)*

```
INSERT INTO departamentos(idDepar, nomDepar, idDeparResp, idSede, presup, idGerente)
VALUES ('DIR', 'Dirección', NULL, '1', '245000.87', '1');
```

Seleccionar Información *(Los datos retornados son almacenados en una tabla con formato de resultados)*

```
SELECT * FROM ejemplo1; /*Selecciona todas las columnas → USA EL * */

SELECT Campo1, Campo2 , ... FROM ejemplo1; /*Selecciona las columnas que pones*/

SELECT DISTINCT Campo1, Campo2 , ...
FROM ejemplo1;
/*El uso de DISTINCT asegura que solo se retornen valores únicos, elimina duplicados*/
```

Funciones de agregado (para hacer sobre atributos): `COUNT /*Cuenta nº filas criterio*/ ; AVG /*Promedio*/; SUM /*Suma*/; MAX /*Máximo*/; MIN /*Mínima*/`

Partículas de construcción:

- *Comparar:* `= ; <> != ; > ; < ; >= ; <=`
- *Lógicos:* `AND /* … Y … */ ; OR /* … O … → Se cumple 1 y aparece */; NOT /* NO … */`
- *Valores:* `BETWEEN ... AND ... /*Entre esos valores*/; Columna IN (valor, valor2, ...)/*Los que tengan esos valores*/;`
- `IS NULL ; IS NOT NULL ; LIKE 'A%' ; NOT LIKE '%A'; /*LIKE busca un patrón*/ /*A%=Empieza por A ; %A%=Contiene ; %A=Acaba*/`
- `EXISTS ; ANY ; ALL ; SET ...`

WHERE: *(INTRODUCE A UN PREDICADO → SELECT * FROM … WHERE … ORDER BY… ;)*

```
SELECT Campo1, Campo2 , ...
FROM ejemplo1
WHERE Campo1 > 10;
/*El uso de WHERE permite aplicar una condición para filtrar los resultados*/
```

ORDER BY:

```
SELECT Campo1, Campo2 , ...
FROM ejemplo1
ORDER BY Campo2  DESC, ... ; /*También existe → ASC */
/*El uso de ORDER BY permite aplicar una condición para ordenar los resultados*/
```

INNER JOIN:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
/*JOIN se utiliza para combinar filas de dos o más tablas, basado en una columna
relacionada entre ellas*/
```

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID = Customers.CustomerID;
/*Esto seleccionará la "OrderID" de la tabla "Orders" y "CustomerName" de la tabla
"Customers", donde "CustomerID" es común entre ambas tablas*/
```

GROUP BY:

```
SELECT column1, COUNT(column2)
FROM table_name
GROUP BY column1;
/*GROUP BY se utiliza con funciones de agregado (COUNT, AVG, SUM, etc.)
para agrupar el resultado por una o más columnas */
```

```
SELECT Country, COUNT(CustomerID)
FROM Customers
GROUP BY Country;
/*Esto seleccionará la columna "Country" y el número de "CustomerID" para cada "Country"
de la tabla "Customers"*/
```

HAVING:

```
SELECT column1, COUNT(column2)
FROM table_name
GROUP BY column1
HAVING COUNT(column2) > value;
/*HAVING se utiliza para filtrar los resultados de una consulta que incluye una función
de agregado*/
```

```
SELECT Country, COUNT(CustomerID)
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
/*Esto seleccionará la columna "Country" y el número de "CustomerID" para cada "Country"
de la tabla "Customers", pero solo los países que tienen más de 5 clientes*/
```

Consultas anidadas → utilizar el resultado de una consulta como parte del predicado o criterio de selección de otra.

```
SELECT apellido1, apellido2, nombre
    FROM empleados
        WHERE idDepar IN (SELECT idDepar FROM departamentos
            WHERE idSede IN (SELECT idSede FROM sedes
                WHERE ciudadSede='Sevilla'));

/*IN indica que el valor del atributo que la precede se encuentre entre los devueltos
(una lista) por la consulta que la sigue (Cada color ≠ es un anidado) */

/* Además de IN/NOT IN, se pueden utilizar otros operadores como ANY o ALL*/
```

Funciones sobre atributos →Devuelve los cálculos de las funciones de agregado

```
SELECT AVG(salarioAnual) FROM empleados WHERE idDepar = 'DIR' ;
/* Muestra el salario anual promedio de los empleados del departamento 'DIR'*/
```

Producto cartesiano → Combina cada fila de una tabla con cada fila de otra tabla

```
SELECT nomDepar, ciudadSede
FROM departamentos d, sedes s
WHERE d.idSede = s.idSede AND s.ciudadSede IN ('Madrid' , 'Sevilla' );
/*Muestra los nombres de los departamentos ubicados en Madrid y Sevilla*/

/* Se puede poner un "MOTE" usando … AS … o como arriba FROM departamentos d */
```

Replicar tablas

```
CREATE TABLE empleados_aux SELECT * FROM Empleados; /* Será de 'Read only'→ No tiene PK*/
/*Da lugar a una tabla de nombre empleados_aux con el mismo esquema y datos que la tabla
empleados*/
```

Agrupación de registros→ Agrupan registros de una o varias tablas según el conjunto de sus atributos

```
SELECT idDepar, COUNT(idEmpleado) AS numEmpleados,AVG (salarioAnual)
FROM empleados
GROUP BY idDepar
HAVING numEmpleados >= 5;
/*Muestra el número de empleados y el promedio del salario anual para los departamentos
con 5 o más empleados*/
```

Actualización de registros → Modificar el valor de los atributos de una determinada tabla

```
UPDATE empleados_aux
SET comision=comision*1.1
WHERE idDepar ='MRK' ;
/*Incrementa en un 10% la comisión de los empleados del departamento 'MRK'*/
```

Borrado de registros → Borrar el valor de los atributos de una determinada tabla

```
DELETE FROM empleados_aux WHERE idDepar ='MRK' ;
/*Elimina de la tabla empleados a los del departamento 'MRK'*/
```

Crear vistas → Tabla virtual cuyo contenido está definido por una consulta

Una vista no existe como conjunto de valores de datos almacenados en una base de datos.

Una vista actúa como filtro de las tablas subyacentes a las que se hace referencia en ella.

```
CREATE VIEW Nombre_de_vista AS
SELECT …
```

Para usar una vista: *(Igual que para seleccionar una tabla entera)*

```
SELECT *
FROM Nombre_de_vista;
```

Ejemplo:

```
CREATE VIEW vista_futbolistas AS
SELECT futbolistas.id, nombre, apellidos FROM futbolistas
INNER JOIN tarjetas_amarillas
ON futbolistas.id = tarjetas_amarillas.id_futbolista;
```

Modificar vistas

```
ALTER VIEW
vista_usuarios AS
SELECT * FROM usuarios WHERE edad < 30;
```

Borrar vistas

```
DROP VIEW [IF EXISTS] Nombre_de_vista
```

Cambiar el delimitador → `delimiter //`

Triggers → Se ejecutan cuando sucede algún evento sobre las tablas a las que se encuentra asociado

- TIPO: BEFORE → ANTES DE AFTER → DESPUÉS DE
- OPERACIONES: INSERT UPDATE DELETE
- MODIFICADORES: OLD (Lo que había antes) NEW

Trigger Event	OLD	NEW
INSERT	No	Yes
UPDATE	Yes	Yes
DELETE	Yes	No

Creación de Triggers

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | UPDATE| DELETE }
ON table_name FOR EACH ROW
trigger_body;

/*Por lo general cambiar el delimitador antes del trigger($$) y reestablecerlo al final*/
```

Mostrar Triggers

```
SHOW TRIGGERS
FROM classicmodels
LIKE 'employees';
/*Muestra todos los triggers asociados con la tabla employees de la bd classicmodels*/
```

Borrar Triggers

```
DROP TRIGGER IF EXISTS insercion_datos;
/*Borra el trigger insercion_datos*/
```

Transacciones → Sentencias SQL que son agrupadas como una sola

1. Iniciar una transacción con el uso de la sentencia BEGIN.

2. Actualizar, insertar o eliminar registros en la base de datos.

3. Si se quieren los cambios a la base de datos, completar la transacción con el uso de la sentencia COMMIT. Únicamente cuando se procesa un COMMIT los cambios hechos por las consultas serán permanentes.

4. Si sucede algún problema, podemos hacer uso de la sentencia ROLLBACK para cancelar los cambios que han sido realizados por las consultas que han sido ejecutadas hasta el momento.

Ejemplo:

```
/* Ahora veamos cómo usar transacciones */
/* Empezamos la transacción */
BEGIN;
INSERT INTO innotest VALUES(4);
SELECT * FROM innotest;

/* Si en este momento ejecutamos un ROLLBACK, la transacción no será
completada, y los cambios realizados sobre la tabla no tendrán efecto. */
ROLLBACK;
SELECT * FROM innotest;
```

Por defecto, las tablas InnoDB ejecutan un lectura consistente (consistent read). Esto significa que cuando una sentencia SELECT es ejecutada, MySQL regresa los valores presentes en la base de datos hasta la transacción más reciente que ha sido completada. Si alguna transacción está en progreso, los cambios hechos por alguna sentencia INSERT o UPDATE no serán reflejados. Sin embargo, existe una excepción: las transacciones abiertas si pueden ver sus propios cambios.

CONCURRENCIA → MIRAR EN EL DOCUMENTO RESUMEN T6 (NO ES CÓDIGO)

Ejemplos de TRIGGERS

BEFORE INSERT → Antes de cada sentencia INSERT para la tabla people y en el caso de que introduzcamos una persona en la tabla people con una edad negativa, cambiará la edad a 0

```
DELIMITER //
/* Creamos el trigger */
CREATE TRIGGER agecheck
BEFORE INSERT
ON people FOR EACH ROW
IF NEW.age < 0
THEN SET NEW.age = 0; END IF;//
/* Volvemos a especificar que el delimitador por defecto sea ; */
DELIMITER;
```

AFTER INSERT → Inserta un recordatorio en la tabla reminders si la fecha de nacimiento del miembro en la tabla members es NULL. El identificador insertado en la tabla reminders corresponde al campo id de members.

```
DELIMITER $$
CREATE TRIGGER after_members_insert
AFTER INSERT
ON members FOR EACH ROW
BEGIN
IF NEW.birthDate IS NULL THEN
INSERT INTO reminders(memberId, message)
VALUES(new.id,CONCAT('Hi ', NEW.name, ', please update your date of birth.'));
END IF;
END$$
DELIMITER ;
```

BEFORE UPDATE → Si actualiza el valor en la columna de quantity a un nuevo valor que es 3 veces mayor que el valor actual, el activador genera un error y detiene la actualización.

```
DELIMITER $$
CREATE TRIGGER before_sales_update
BEFORE UPDATE
ON sales FOR EACH ROW
BEGIN
DECLARE errorMessage VARCHAR(255);
SET errorMessage = CONCAT('The new quantity ',
NEW.quantity,
' cannot be 3 times greater than the current quantity ',
OLD.quantity);
IF NEW.quantity > OLD.quantity * 3 THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = errorMessage;
END IF;
END $$
DELIMITER ;
```

AFTER UPDATE → Si actualiza el valor en la columna de cantidad a un nuevo valor, el disparador inserta una nueva fila para registrar los cambios en la tabla SalesChanges.

```
DELIMITER $$
CREATE TRIGGER after_sales_update
AFTER UPDATE
ON sales FOR EACH ROW
BEGIN
IF OLD.quantity <> NEW.quantity THEN
INSERT INTO SalesChanges(salesId,beforeQuantity, afterQuantity)
VALUES(old.id, OLD.quantity, NEW.quantity);
END IF;
END$$
DELIMITER ;
```

BEFORE DELETE → Inserta una nueva fila en la tabla SalaryArchives antes de que se elimine una fila de la tabla Salaries.

```
DELIMITER $$
CREATE TRIGGER before_salaries_delete
BEFORE DELETE
ON salaries FOR EACH ROW
BEGIN
INSERT INTO SalaryArchives(employeeNumber,validFrom,amount)
VALUES(OLD.employeeNumber,OLD.validFrom,OLD.amount);
END$$
DELIMITER ;
```

AFTER DELETE → Actualiza el salario total en la tabla SalaryBudgets después de eliminar una fila de la tabla Salaries

```
CREATE TRIGGER after_salaries_delete
AFTER DELETE
ON Salaries FOR EACH ROW
UPDATE SalaryBudgets
SET total = total - old.salary;
Bases de datos [13]
/* Probarlo, borramos una fila de la tabla Salaries */
DELETE FROM Salaries
WHERE employeeNumber = 1002;
```