Módulo 5

Antonio Cano Galindo

Según la Wikipedia:

*Una **base de datos** es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En este sentido; una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta.*

Modelos de bases de datos

- Jerárquicas
- En red
- Relacionales
- Documentales
- Multidimensionales
- Orientadas a objetos

Bases de datos relacionales

- Una base de datos se compone de varias tablas, denominadas relaciones.
- No pueden existir dos tablas con el mismo nombre ni registro.
- Cada tabla es a su vez un conjunto de campos (columnas) y registros (filas).
- La relación entre una tabla padre y un hijo se lleva a cabo por medio de las llaves primarias y llaves foráneas (o ajenas).
- Las llaves primarias son la clave principal de un registro dentro de una tabla y estas deben cumplir con la integridad de datos.
- Las llaves ajenas se colocan en la tabla hija, contienen el mismo valor que la llave primaria del registro padre; por medio de estas se hacen las formas relacionales.

Normalización de bases de datos

Las bases de datos relacionales se normalizan para:

- Minimizar la redundancia de los datos.
- Disminuir problemas de actualización de los datos en las tablas.
- Proteger la integridad de datos.

En el modelo relacional es frecuente llamar «tabla» a una relación; para que una tabla sea considerada como una relación tiene que cumplir con algunas restricciones:

- Cada tabla debe tener su nombre único.
- No puede haber dos filas iguales. No se permiten los duplicados.
- Todos los datos en una columna deben ser del mismo tipo.

Claves de una tabla

- Clave candidata: es el conjunto mínimo de columnas que identifica unívocamente a cada fila
- Clave primaria: clave candidata designada como principal
- Clave compuesta: cuando una clave está formada por más de un campo
- Clave ajena: aquella columna que existiendo como dependiente en una tabla, es a su vez clave primaria en otra tabla

Formas normales: 1FN

- **Todos los atributos son «atómicos»**. Por ejemplo, en el campo teléfono no tenemos varios teléfonos.
- La tabla contiene una **clave primaria única**. Por ejemplo: el NIF para personas, la matrícula para vehículos o un simple id autoincremental. Si no tiene clave, no es 1FN.
- **La clave primaria no contiene atributos nulos**. No podemos tener filas para las que no haya clave (por ejemplo, personas sin NIF o vehículos sin matrícula).
- **No debe existir variación en el número de columnas**. Si algunas filas tienen 8 columnas y otras 3, pues no estamos en 1FN.
- **Los campos no clave deben identificarse por la clave**. Es decir, que los campos no clave dependen funcionalmente de la clave. Esto es prácticamente lo mismo que decir que existe clave primaria.
- **Debe Existir una independencia del orden tanto de las filas como de las columnas**, es decir, si los datos cambian de orden no deben cambiar sus significados.

Modelo relacional y formas normales - BI Geek Blog (bi-geek.com)

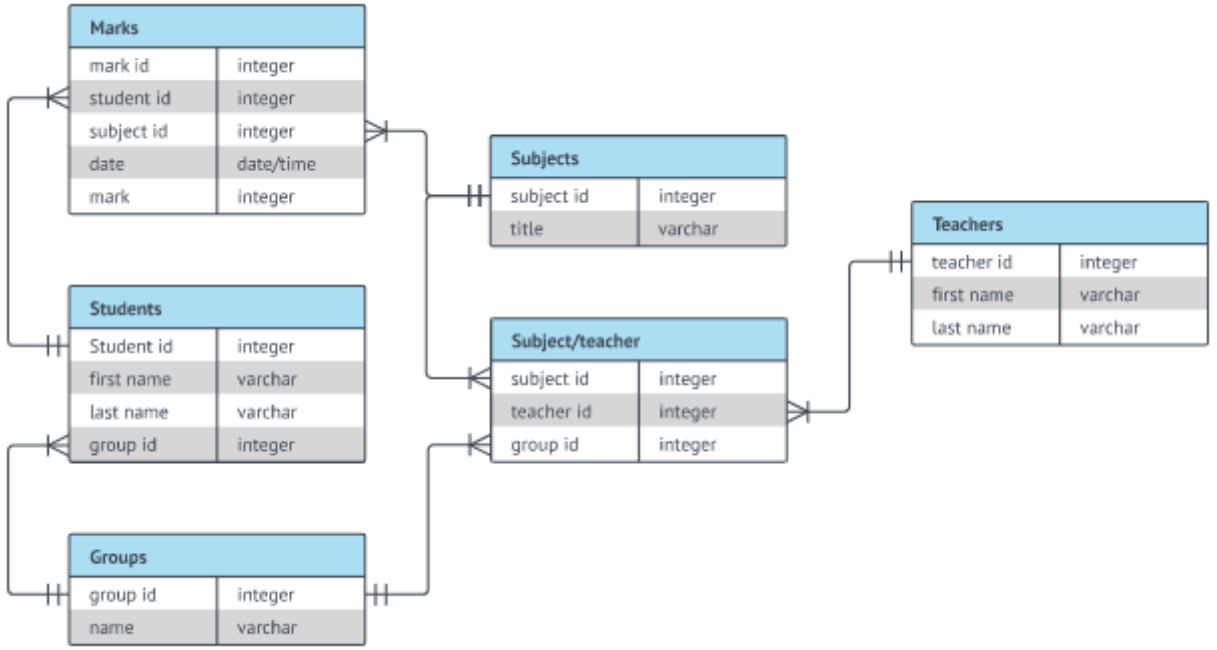
Formas normales: 2FN y 3FN

- **2FN: una tabla está en 2FN si además de estar en 1FN cumple que los atributos no clave depende de TODA la clave principal.**
- **3FN: Una tabla está en 3FN si además de estar en 2FN no existe ninguna dependencia transitiva entre los atributos que no son clave.**

- Resumen
- 1FN. No elementos repetidos o grupos de elementos
- 2FN. Sin dependencias parciales de llaves concatenadas
- 3FN. Sin dependencias de atributos que no son llaves

Diagramas Entidad-Relación

- Muestra relaciones entre entidades de información del sistema a modelar.
- Ayuda a conceptualizar elementos abstractos y entender la relación entre ellos.
- Está compuesto de entidades (cajas) y relaciones (conectores) con cardinalidad.



Antonio Cano Galindo

MariaDB es un fork de MySql.

Su creador es Michael (Monty) Widenius, creador también de MySQL.

El fork se produce a raíz de la venta de SUN Microsystem/MySQL a Oracle.

Se caracteriza por la alta compatibilidad con MySQL, asegurando que siempre será gratuito y open source.

Usaremos la instalación de MariaDB contenida en el paquete XAMPP.

PhpMyAdmin y MySQL Workbench

Son sistemas software de administración de bases de datos.

MySQL Workbench es un software de escritorio gratuito descargable desde la web de MySQL.

PhpMyAdmin es un portal web realizado en Php que permite la administración de sistemas basados en MySQL.

Tienen un grado de funcionalidad muy parecida.

Características de PhpMyAdmin

- Interfaz Web para la gestión gráfica.
- Administración y mantenimiento de base de datos MySQL y MariaDB.
- Explorar, eliminar bases de datos, tablas, vistas, campos e índices.
- Crear, copiar, eliminar, renombrar y modificar bases de datos, campos e índices.
- Mantenimiento de servidor, bases de datos y tablas, de cara a la configuración del servidor.
- Ejecutar, editar y marcar cualquier instrucción SQL, incluso peticiones por lotes.
- Administrar procesos almacenados.
- Importación de datos desde CSV y SQL
- Exporta datos a diferentes formatos: CSV, SQL, XML, etc.
- Administración de múltiples servidores.
- Crea gráficos PDF del diseño de la base de datos.
- Crea consultas complejas usando *Query-by-Example* (QBE).
- Búsqueda global en una base de datos o un subconjunto de esta.
- *Live charts* para monitorear las actividades del servidor MySQL tales como conexiones, procesos, uso de CPU y/o memoria, etc.

Conceptos de bases de datos

DDL: Data Definition Language

- Base de datos (database)
- Tabla y campo (table and field)
- Tipo de dato (data type)
- Clave primaria y ajena (primary and foreign key)
- Índice (index)
- Disparador (trigger)
- Procedimiento almacenado (stored procedure)
- Vistas (view)

DML: Data Manipulation Language

- Select
- Insert
- Update
- Delete
- Truncate
- Replace

Crear tablas

Comando CREATE TABLE

```
CREATE TABLE `cliente` (
  `idCliente` int(10) UNSIGNED NOT NULL,
  `nombre` varchar(30) NOT NULL,
  `direccion` varchar(30) NOT NULL,
  `poblacion` varchar(30) NOT NULL,
  `provincia` varchar(30) NOT NULL,
  `dni` char(9) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Una tabla está compuesta de campos, donde tenemos que especificar para cada uno de ellos el tipo de dato y algunas características más.

Nulos

Valor especial que indica la ausencia de valor en un campo.
Los campos que forman parte de una clave primaria no puede permitir nulos.

```
CREATE TABLE `cliente` (
  `idCliente` int(10) UNSIGNED NOT NULL,
  `nombre` varchar(30) NULL,
  `dni` char(9) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

El campo nombre permite guardar null como valor.

En una sentencia INSERT los campos no especificados asumen el valor null, siempre y cuando está definidos como null. En caso contrario, INSERT fallará.

Tipos de datos

Numéricos

- Enteros: INT, TINYINT, SMALLINT, MEDIUMINT, BIGINT
- Reales: FLOAT, DOUBLE, DECIMAL

Fecha y hora

- DATE, DATETIME, TIMESTAMP, TIME, YEAR

Cadenas

- CHAR, VARCHAR, BLOB, TINYBLOB, MEDIUMBLOB, LONGBLOB, ENUM

Clave primaria

La clave primaria puede contener uno o varios campos
Si tiene más de un campo sería una clave primaria compuesta.
No permite que ningún campo pueda contener nulos

```
ALTER TABLE `facturadetalle`
  ADD PRIMARY KEY (`idFactura`,`idProducto`);
```

La creación de una primary key implica la existencia de un índice único con esos campos.

Clave ajena

Permite relacionar una tabla “padre” con una “hija” mediante la clave primaria de la tabla padre en relación a los mismos campos de la tabla hija.
Cuando borramos o actualizamos el registro de la tabla padre, se aplican las reglas de consistencia ON DELETE o ON UPDATE: CASCADE, SET NULL, RESTRICT o NO ACTION.

```
ALTER TABLE `facturadetalle`
  ADD CONSTRAINT `factura` FOREIGN KEY (`idFactura`) REFERENCES
`factura` (`idFactura`),
  ADD CONSTRAINT `producto` FOREIGN KEY (`idProducto`) REFERENCES
`producto` (`idProducto`) ON DELETE CASCADE ON UPDATE CASCADE;
```

La creación de una foreign key implica la existencia de un índice con esos campos.

Índice

Un índice es una estructura de datos que mejora la velocidad de acceso a los datos de una tabla.

Similar al índice de un libro. Guarda el valor a indexar y su posición en la tabla.

Pueden contener una o varias columnas.

Puede ser único o permitir duplicados.

Un índice FULLTEXT definido sobre campos de tipo cadenas de caracteres permite la búsqueda de palabras dentro de un campo.

```
ALTER TABLE `cliente`
  ADD KEY `nombre` (`nombre`);
```

SELECT

Devuelve un conjunto de datos de una o más tablas

Especifica los campos que contendrá la tabla así como la lista de tablas implicadas y las condiciones que deben cumplir los registros resultado.

```
SELECT `tabla 1`.`campo 1`, ..., `tabla n`.`campo n`, <expresión>
  FROM `tabla 1`, `tabla 2`, ..., `tabla n`
 WHERE <condiciones>
 ORDER BY `tabla 1`.`campo 1` [ASC|DESC]
 LIMIT <número de filas>
```

En <condiciones> especificamos las restricciones que deberá cumplir el resultado.
Los campos en ORDER BY son los usados para ordenar la tabla resultado.

SELECT (II)

La lista de campos, literales o expresiones separados por comas que van después de la palabra SELECT es lo que obtendremos de la ejecución de la query.

Cada valor puede ir precedido por la palabra reservada AS que permite renombrar al campo o expresión.

Para especificar un campo lo podemos realizar prefijando el nombre del campo con el nombre de la tabla. Esto es opcional si el nombre del campo es único entre todas las tablas implicadas en la query.

SELECT DISTINCT permite eliminar las filas duplicadas.

SELECT COUNT (DISTINCT <campo>) cuenta los valores diferentes que toma ese campo.

SELECT - FROM

Después de la palabra reservada **FROM** se relacionan las tablas implicadas en la query, pudiendo renombrar cualquier de ellas con la palabra reservada **AS**.

```
SELECT A.CAMPO11, B.CAMPO21
 FROM TABLA1 AS A, TABLA2 AS B
```

Si no especificamos ninguna condición en la cláusula **WHERE**, este query nos devolverá el producto cartesiano de las tablas **TABLA1** y **TABLA2**. Es decir, nos devolverá cada valor del **CAMPO11**, emparejado con **CAMPO21**. Si **TABLA1** tiene 100 registros y **TABLA2** tiene 50, la query devolverá 100x50=5000 registros.

SELECT – WHERE (I)

En la cláusula WHERE se establecen las restricciones que deben superar las filas y los campos de las tablas involucradas para ser devueltos por la sentencia.

Después de la palabra reservada WHERE se especificará una expresión lógica que deberá cumplir la fila de campos combinados del producto cartesiano de las tablas expresadas en el FROM para ser seleccionada.

La expresión lógica puede estar compuesta de subexpresiones lógicas encadendas por AND y OR.

Una expresión lógica “normal” consiste en comparar campos de una tabla contra valores literales o campos de otra tabla mediante los operadores lógicos habituales: >, <, <=, >=, <>, =

SELECT – WHERE (II)

Comparar un campo con NULL: <campo/expresión> IS [NOT] NULL

Comparar con un rango de valores: BETWEEN <valor inferior> AND <valor superior>

Buscar patrones en un campo string: LIKE 'patrón', donde patrón usa los caracteres % para expresar cualquier cantidad de caracteres o _ para expresar solo un carácter.

Testear la pertenencia a un conjunto de valores usando
[NOT] IN (<valor1>,<valor2>,...,<valorn>)

SELECT – Funciones internas

MAX : Máximo de un conjunto de valores.

MIN : Mínimo de un conjunto de valores.

AVG : Media de un conjunto de valores.

SUM : Suma total de un conjunto de valores.

COUNT : Cuenta el número de valores que hay en un conjunto.

```
SELECT MAX(population), COUNT(*)
  FROM city
 WHERE countrycode = “ESP”
```

Devuelve el máximo del campo población y el número de registros cuyo countrycode sea igual a “ESP”.

SELECT – GROUP BY

Permite agrupar las filas devueltas por una SELECT en función de los campos especificados después de GROUP BY.

Los campos de la cláusula deberán ser los que aparezcan en la SELECT exceptuando aquellos valores devueltos por funciones de agregación. La cláusula permite especificar un orden de agrupación para así poder aplicar las funciones de agregación. Los nombres de los campos se pueden cambiar por su posición

```
SELECT IndepYear, COUNT(*)
  FROM country
 GROUP BY IndepYear
```

```
SELECT IndepYear, COUNT(*)
  FROM country
 GROUP BY 1
```

Esta sentencia devuelve cuantos paises se han independizado en cada año.

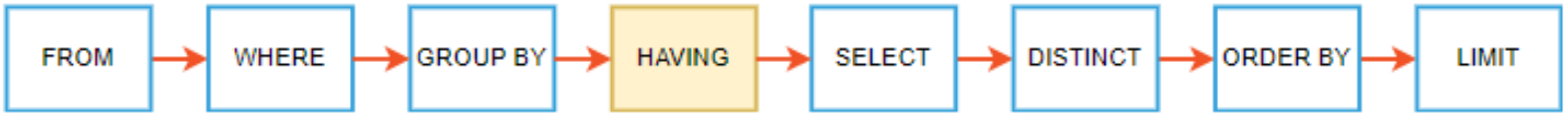
SELECT – HAVING

Es la cláusula equivalente al **WHERE** pero aplicando a los grupos creados por el **GROUP BY**.

```
SELECT IndepYear, COUNT(*)
  FROM country
 GROUP BY 1
 HAVING COUNT(*) > 10
```

Esta query solo devuelve los grupos donde el contador sea mayor que 10. Es decir, solo aquellos año en los que hubo más de 10 países que se independizaran.

Orden de evaluación de la cláusula **HAVING**:



SELECT – Otras funciones internas

La lista completa de funciones está en: https://mariadb.com/kb/en/function-and-operator-reference/

CURRENT_TIMESTAMP () Fecha y hora actuales.

LOWER (cadena) Convierte una cadena de tipo varchar a minúsculas.

SUBSTRING (fuente, n, lon) Extrae de la cadena fuente una subcadena , comenzando en el carácter n-ésimo, con una longitud lon

UPPER (cadena) Convierte una cadena de tipo varchar a mayúsculas.

LEFT (cadena, lon) Devuelve de la cadena una subcadena, comenzando por la izquierda y con una longitud lon.

LEN (cadena) Devuelve la longitud de la cadena.

LTRIM (cadena) Quita los blancos de la izquierda en la cadena.

RTRIM (cadena) Quita los blancos de la derecha en la cadena.

STR (cadena) Devuelve la cadena alineada a la derecha.

RIGHT (cadena, num_elem) Devuelve el número de elementos de la cadena que están a la derecha. ASCII (cadena) Devuelve el nº ASCII correspondiente a la cadena

SELECT – Funciones de fecha

Existe una gran variedad de funciones que operan con campos de tipo fecha: DATE, DATETIME, TIME o TIMESTAMP.

La relación completa está en: https://mariadb.com/kb/en/date-time-functions/

Entre las más frecuentes encontramos:

- EXTRACT (<unit> FROM <fecha>): EXTRACT (YEAR FROM '2009-01-23') devuelve 2009.
- DATE_FORMAT (<fecha>, <string de formato>): DATE_FORMAT('2022-10-13','%Y%m') devuelve "202210".
- CURRENT_DATE: devuelve la fecha actual del sistema donde se ejecuta el motor. Equivalente a CURDATE.
- CURRENT_TIME: devuelve la hora actual. Equivalente a CURTIME.
- CURRENT_TIMESTAMP: devuelve la fecha y la hora actual. Equivalente a NOW.
- YEAR(): obtiene el año de una fecha.
- MONTH(): obtiene el mes de una fecha.
- DAY(): obtiene el año de una fecha.

SELECT – JOIN

La gran utilidad de las bases de datos relacionales está en poder relacionar unas tablas con otras. Para ello la sentencia SELECT permite especificar varias tablas en la cláusula FROM.

Si no se ajusta bien el contenido de la cláusula WHERE a la hora de relacionar las tablas se producirán resultados indeseados (productos cartesianos de tablas).

Es necesario igualar la claves ajena de una tabla con la clave primaria de la tabla "padre" correspondiente. Y esto para todas las tablas involucradas en la query.

Supongamos que "cruzamos" una tabla de clientes, con clave primaria idCliente, con una tabla de facturas, con clave primaria idFactura y clave ajena idCliente. Es decir, en la tabla de factura tenemos un campo idCliente que identifica al cliente propietario de esa factura. El encuentro, cruce o join entre ambas tablas sería así:

```
SELECT *
FROM clientes, facturas
WHERE clientes.idCliente = facturas.idCliente
```

INSERT (I)

Permite añadir nuevos registros a una tabla.

INSERT [INTO] <tabla> [(campo1, campo2,...,campoN)]

VALUES (valor1, valor2,...,valorN) [, (valor1,valor2,...,valorN)]*

```
INSERT INTO `city`(`ID`, `Name`, `CountryCode`, `District`,
`Population`)
 VALUES (4900,"Garrucha","ESP","Andalusía",8000)
```

Si se especifica ID y es un campo autoincremental, se usará el valor especificado en la sentencia y se ajustará el contador a este último valor usado. Es decir, el siguiente registro insertado tendrá valor 4901.

Los campos no especificados en INSERT asumirán valor NULL o su valor por defecto en caso de tenerlo.

INSERT (II)

INSERT [INTO] <tabla> [(campo1, campo2,...,campoN)]

SELECT campo1, valor2,...,campoN FROM <tabla2> WHERE <condición>

Este formato de insert permite insertar registros en una tabla con los registros obtenidos de una sentencia SELECT.

Deberá existir correspondencia entre los campos especificados entre paréntesis con los devueltos por la SELECT, respetando el orden en que se corresponden.

UPDATE

Permite la actualización de los datos de una tabla

UPDATE <tabla> SET <campo1> = <valor1>, <campo2> = <valor2>, ...

[WHERE <condición>]

Si no se especifica la cláusula WHERE la sentencia actualizará todos los registros de la tabla, informando el valor1 en el campo1 y el valor2 en el campo2

Se pueden especificar uno o más campos de la tabla.

DELETE

Permite el borrado de datos de una tabla

DELETE FROM <tabla> [WHERE <condición>]

Si no se especifica la cláusula WHERE la sentencia BORRARÁ TODOS LOS DATOS DE LA TABLA.

Especificando una condición solo se borrarán los registros que la cumplan.

Convierte una SELECT en una pseudo tabla, normalmente de solo lectura.

```
CREATE view CITY_WITH_COUNTRY AS
SELECT city.ID, city.NAME, city.CountryCode,
       country.Name as nameCountry
FROM city, country
WHERE city.CountryCode = country.Code
```

A partir de este momento podremos realizar la siguiente query:

```
SELECT * FROM CITY_WITH_COUNTRY WHERE CountryCode = 'ESP'
```

TRANSACCIONES

Una transacción es un grupo de sentencias ejecutadas como un todo. Es decir, o se ejecutan todas o no se ejecuta ninguna.

Una transacción comienza con START TRANSACTION o BEGIN [WORK]. Son equivalentes.

Al terminar el grupo de sentencias que deseemos ejecutar como un todo, ejecutaremos la sentencia COMMIT, encargada de confirmar todos los cambios.

Si en la ejecución del grupo de sentencias encontramos algún error o simplemente, no queremos que los cambios realizados sean definitivos, podemos ejecutar la sentencia ROLLBACK, la cual dejará los datos como estaban al emitir BEGIN.

Los joins entre tablas y/o vistas permiten relacionar la información de éstas en una única query.

Una sentencia select con join permite mostrar campos de las diferentes tablas involucradas.

Hay cinco tipos de JOIN: INNER, LEFT, RIGHT, CROSS Y FULL.

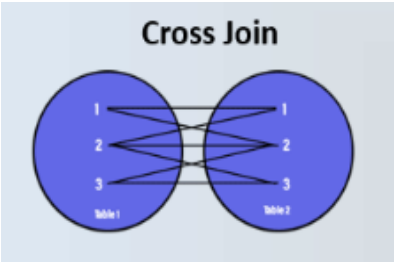
El tipo FULL no está soportado en MariaDB/MySql, pero se puede obtener mediante combinación de LEFT y RIGHT.

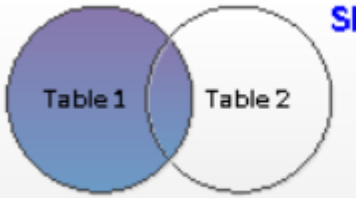
El tipo CROSS produce un producto cartesiano entre tablas y no admite el operador ON.

JOINS (II)

INNER JOIN

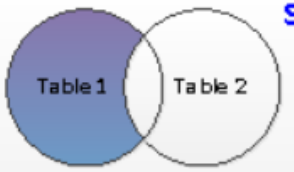
```
SELECT *
  FROM Table1 t1
 INNER JOIN Table2 t2
    ON t1.fk = t2.id;
```





LEFT OUTER JOIN

```
SELECT *
  FROM Table1 t1
 LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id;
```

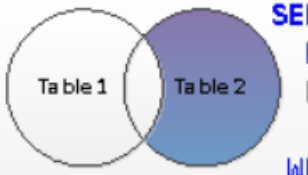


LEFT OUTER JOIN with exclusion

```
SELECT *
  FROM Table1 t1
 LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t2.id is null;
```

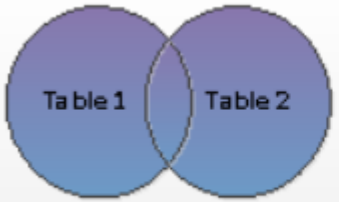
RIGHT OUTER JOIN

```
SELECT *
  FROM Table1 t1
 RIGHT OUTER JOIN Table2 t2
    ON t1.fk = t2.id;
```



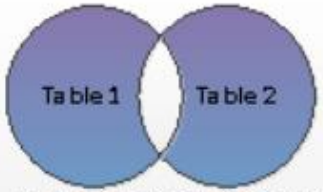
RIGHT OUTER JOIN with exclusion

```
SELECT *
  FROM Table1 t1
 RIGHT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t1.fk is null;
```



FULL OUTER JOIN

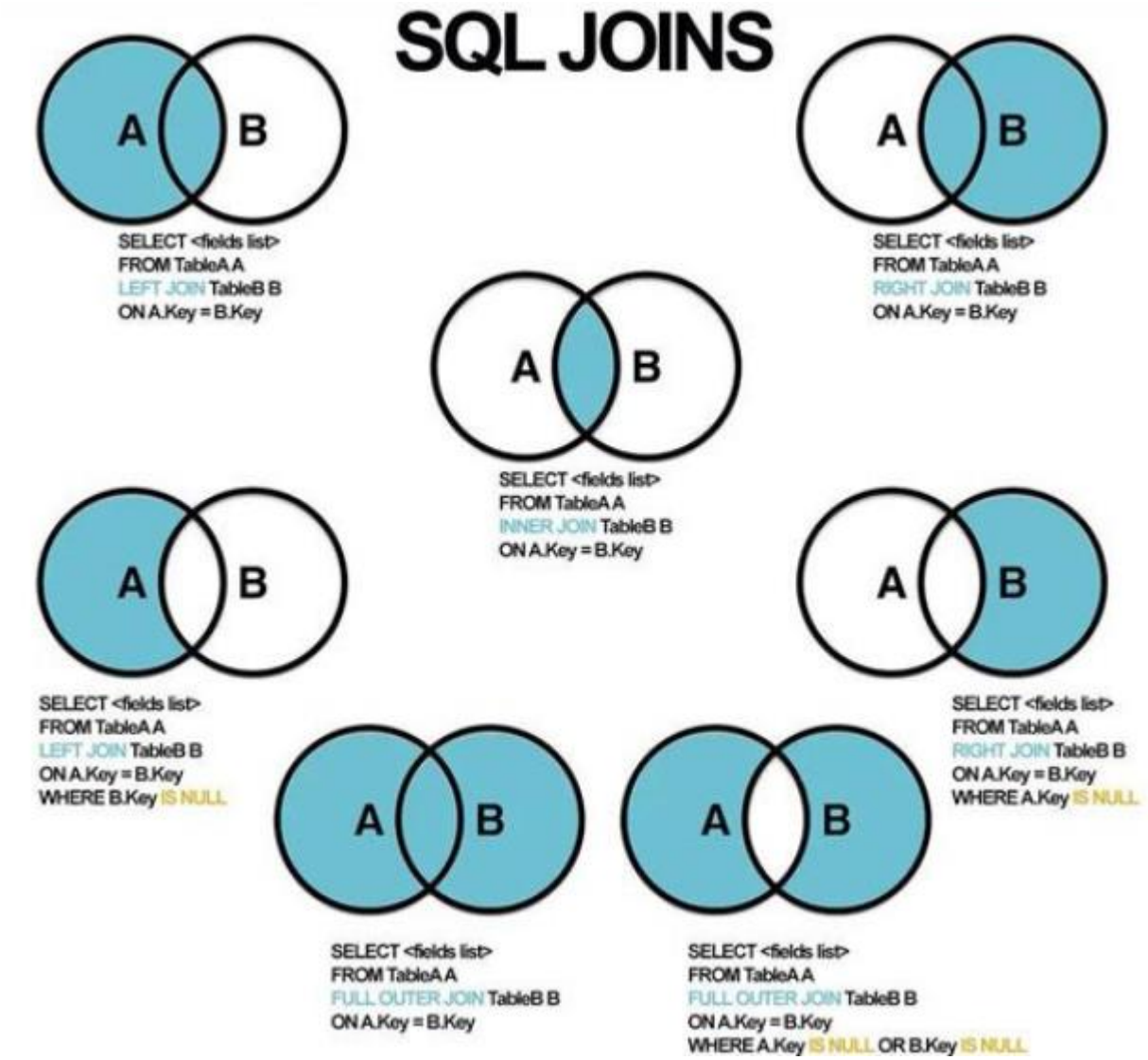
```
SELECT * FROM Table1 t1
  LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
UNION
SELECT * FROM Table1 t1
 RIGHT OUTER JOIN Table2 t2
    ON t1.fk = t2.id;
```



FULL OUTER JOIN with exclusion

```
SELECT * FROM Table1 t1
  LEFT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t2.id IS NOT NULL
UNION
SELECT * FROM Table1 t1
 RIGHT OUTER JOIN Table2 t2
    ON t1.fk = t2.id
 WHERE t1.ID IS NOT NULL;
```

JOINS (III)


SQL JOINS
A
B
SELECT <fields list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
SELECT <fields list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
SELECT <fields list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
SELECT <fields list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
SELECT <fields list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
SELECT <fields list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
SELECT <fields list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL OR B.Key IS NULL

JOINS (IV)


```
Select tabla1.campo1, tabla2.campo2
From tabla1 LEFT JOIN tabla2
ON tabla1.campo3 = tabla2.campo4 and tabla1.campo5 = tabla2.campo1
```

La cláusula ON permite especificar como se realizará el encuentro entre las dos tablas mediante una expresión lógica que las relacione. Aquellas filas que cumplan la condición lógica serán incluidas en la salida de la sentencia SELECT.

Normalmente igualaremos la clave primaria de una tabla con la correspondiente clave ajena de la tabla relacionada.

JOINS (V)


```
Select cliente.nombre, sum(albaran.total)
From cliente left join albaran
On cliente.id = albaran.idCliente and albaran.fecha > '2022-01-01'
Group by cliente.nombre;
```

Esta query devolverá todos los nombres de clientes de la tabla de clientes y la suma de las ventas del 2022, incluyendo los clientes que no hayan tenido ventas en dicho periodo. El campo suma tendrá valor null en estos casos.

JOINS (VI)

La utilización de joins provoca la necesidad de usar el operador COALESCE, la función IFNULL o la sentencia CASE.

COALESCE(valor1,valor2,...,valorN) devuelve el primer valor no nulo

IFNULL(valor1,valor2) devuelve valor2 si valor1 es nulo.

```
SELECT  id, title,
    (CASE
        WHEN excerpt IS NULL THEN LEFT(body, 150)
        ELSE excerpt
    END) AS excerpt,
    published_at
FROM
    articles;
```

SUBQUERIES - Subconsultas

Se pueden usar en la cláusula SELECT, FROM o WHERE.

```
Select max(cliente.id), (select max(albaran.id) from albaran)
From cliente;

Select max(cliente.id), maxalbaran.max
From cliente, (select max(albaran.id) as max from albaran) as maxalbaran;

Select cliente.nombre
From cliente
Where cliente.idProvincia in (select id from provincia where comunidad='Andalucia');
```

SUBQUERIES WHERE

En la cláusula where aparecen los operadores ANY, SOME, ALL, EXISTS, IN para operar con Subqueries.

Operando operador ANY/SOME (subquery)
Operando IN (subquery)
Operando operador ALL (subquery)
[NOT] EXISTS (subquery)

ANY/SOME se cumple si el operando cumple el operador para algún registro devuelto por la subquery.
IN es equivalente a Operando = ANY (subquery).
ALL se cumple si el operando cumple el operador para todos los registros devueltos por la subquery.
EXISTS se cumple si la subquery devuelve algún registro.

SUBQUERIES CORRELADAS

Una subquery correlada o correlacionada es aquella que se evalua por cada fila devuelta por la query principal.

Suelen tener un rendimiento pobre comparada con otras alternativas.

Evaluar cuidadosamente su uso y rendimiento.

```
SELECT * FROM t1
  WHERE column1 >= ANY (SELECT column1 FROM t2
                        WHERE t2.column2 = t1.column2);
```

```
SELECT productname, buyprice FROM products p1
WHERE buyprice > (SELECT AVG(buyprice)  FROM products
                              WHERE productline = p1.productline);
```

STORED PROCEDURES

Permiten “empaquetar” un conjunto de sentencias SQL y otras escritas en lenguaje SQL/PSM (específico de cada motor de base de datos) y ejecutarlas como un todo.

El procedimiento almacenado es un programa que se almacena dentro de la propia base de datos y que permite una ejecución optimizada del código almacenado.

El lenguaje SQL/PSM permite definir parámetros de entrada/salida del procedimiento, así como definir variables, uso de IF y bucles, etc.

Se ejecutan mediante la sentencia CALL <stored procedure> (parámetros)

STORED PROCEDURES (II)


```
DELIMITER //

CREATE PROCEDURE GetOfficeByCountry(
        IN countryName VARCHAR(255)
)
BEGIN
        SELECT *
        FROM offices
        WHERE country = countryName;
END //

DELIMITER ;

CALL GetOfficeByCountry('SPAIN');
```

TRIGGERS

Es un subprograma escrito en lenguaje PSQL que se ejecuta cuando se inserta, actualiza o borra una fila de una tabla.

CREATE TRIGGER <nombre del trigger>
[BEFORE I AFTER] [INSERT I UPDATE I DELETE]
ON tablal FOR EACH ROW
<cuerpo del trigger>;

El cuerpo del trigger se ejecutará antes (BEFORE) o después (AFTER) de cada inserción (INSERT), actualización (UPDATE) o borrado (DELETE) realizados en la tablal

El lenguaje usado en el cuerpo del trigger es el usado en los procedimientos almacenados (SQL/PSM).

TRIGGERS (II)

Los prefijos NEW y OLD usados antes del nombre de un campo representan el valor del nuevo o viejo del campo, dependiendo de la operación.

```
CREATE TRIGGER upd_check BEFORE UPDATE ON account
       FOR EACH ROW
       BEGIN
           IF NEW.amount < 0 THEN
               SET NEW.amount = 0;
           ELSEIF NEW.amount > 100 THEN
               SET NEW.amount = 100;
           END IF;
       END;
```