

# Queries XL

## Repasemos lo visto

**DigitalHouse** >  
Coding School



**Certified Tech  
Developer**  
The Ultimate Degree

# Funciones de agregación

**DigitalHouse** >  
Coding School



**Certified Tech  
Developer**  
The Ultimate Degree

“

Las funciones de agregación **realizan cálculos** sobre un conjunto de datos y **devuelven un único resultado**.

Excepto **COUNT**, las funciones de agregación **ignorarán** los valores **NULL**.

”



# COUNT **sintaxis**

Devolverá la cantidad de **filas/registros** que cumplen con el criterio.

```
SQL  SELECT COUNT(*) FROM movies;
```

*Devolverá la cantidad de registros de la tabla movies*

```
SQL  SELECT COUNT(id) AS total FROM movies WHERE genre_id=3;
```

*Devolverá la cantidad de películas de la tabla movies con el `genre_id` 3 en una columna nombrada total*

# AVG, SUM **sintaxis**

**AVG** (*average*) devolverá el promedio de una columna con valores numéricos.

**SUM** (*suma*) devolverá la suma de una columna con valores numéricos.

```
SQL  SELECT AVG(rating) FROM movies;
```

*Devolverá el promedio del rating de las películas de la tabla movies.*

```
SQL  SELECT SUM(length) FROM movies;
```

*Devolverá la suma de las duraciones de las películas de la tabla movies*

# MIN, MAX **sintaxis**

**MIN** devolverá el valor mínimo de una columna con valores numéricos.

**MAX** devolverá el valor máximo de una columna.

```
SQL  SELECT MIN(rating) FROM movies;
```

*Devolverá el rating de la película menos ranqueada.*

```
SQL  SELECT MAX(length) FROM movies;
```

*Devolverá el rating de la película mejor ranqueada.*

# GROUP BY

**DigitalHouse** >  
Coding School



**Certified  
Developer**  
The Ultimate Tech Degree

# Group By **sintaxis**

**GROUP BY** se usa para **agrupar los registros** de la tabla resultante de una consulta por una o más columnas.

SQL

```
SELECT columna_1  
FROM nombre_tabla  
WHERE condition  
GROUP BY columna_1;
```



# Group By ejemplo

SQL

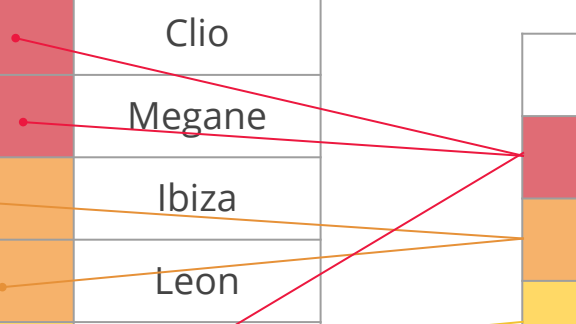
```
SELECT marca,  
FROM autos  
GROUP BY marca;
```

# Group By ejemplo

id	marca	modelo
1	Renault	Clio
2	Renault	Megane
3	Seat	Ibiza
4	Seat	Leon
5	Opel	Corsa
6	Renault	Clio

marca
Renault
Seat
Opel



# Group By **sintaxis**

Dado que **GROUP BY** agrupa la información, perdemos el detalle de cada una de las filas. Es decir, ya no nos interesa el valor de cada fila sino un resultado consolidado entre todas las filas.

La consulta:

SQL

```
SELECT id,marca  
FROM autos  
GROUP BY marca;
```

Nos daría un error. Si agrupamos los datos por **marca**, ya no podemos pedir el campo **id**

# Group By **sintaxis**

Por ende, al utilizar **GROUP BY**, en los campos que se muestran como resultado en el **SELECT** solamente podemos indicar:

- Datos agrupados
- Funciones de agregación

Veamos algunos ejemplos...

# Group By **sintaxis**

SQL

```
SELECT marca, MAX(precio)
FROM autos
GROUP BY marca;
```

SQL

```
SELECT genero.nombre, AVG(duracion)
FROM peliculas
INNER JOIN generos ON generos.id = genero_id
GROUP BY genero.nombre;
```

# HAVING

**DigitalHouse** >  
Coding School



**Certified  
Developer**  
The Ultimate Tech Degree

# HAVING sintaxis

Cumple la misma función que **WHERE**, a diferencia de que **HAVING** se va a poder usar en conjunto con las **funciones de agregación** para filtrar **datos agregados**.

SQL

```
SELECT columna_1  
FROM nombre_tabla  
WHERE condition  
GROUP BY columna_1  
HAVING condition_Group  
ORDER BY columna_1;
```

# HAVING sintaxis

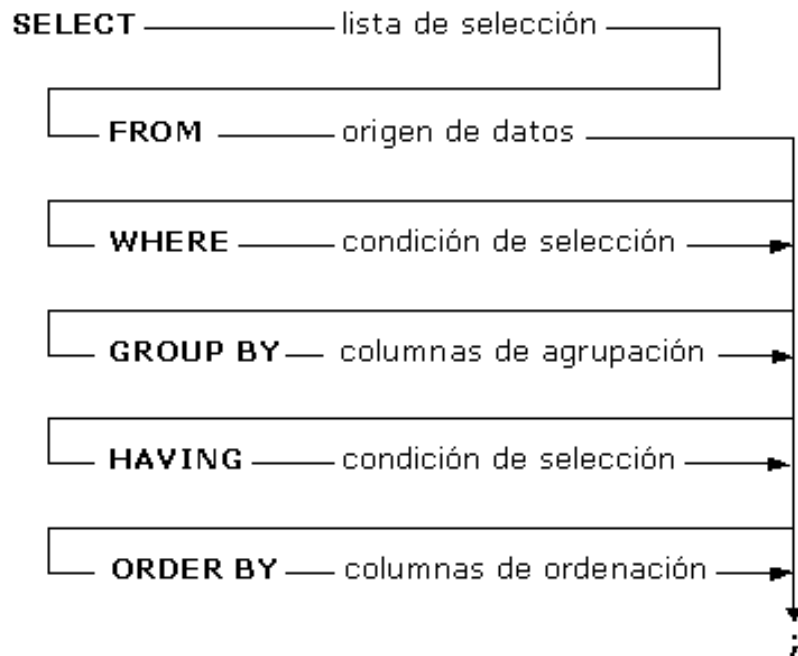
Esta consulta devolverá la cantidad de clientes por país (*agrupados por país*). Solamente se incluirán en el resultado aquellos países que tengan **al menos 3** clientes.

SQL

```
SELECT pais, COUNT(clienteId)
FROM clientes
GROUP BY pais
HAVING COUNT(clienteId)>3;
```



## Estructura de una QUERY



# JOINS

**DigitalHouse** >  
Coding School



**Certified  
Developer**  
The Ultimate Tech Degree

# ¿Por qué usar **JOINS**?

Además de hacer consultas dentro de una tabla o hacia muchas tablas a través de **table reference**, también es posible y necesario hacer consultas a **distintas tablas** y unir esos resultados con **JOINS**.

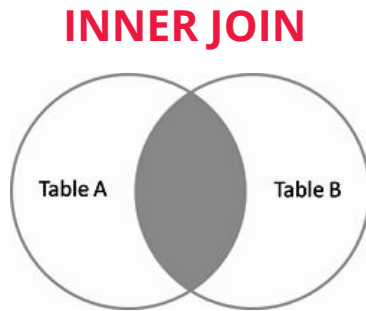
Si bien cumplen la misma función que **table reference**, los **JOINS**:

- Proveen ciertas flexibilidades adicionales.
- Su sintaxis es mucho más utiliza.
- Presentan una mejor performance.

# INNER JOIN

El **INNER JOIN** hará una **cruza** entre dos tablas. Si cruzáramos las tablas de **clientes** y **ventas** y hubiese algún cliente **sin ventas**, el INNER JOIN **no traería** a ese cliente como resultado.

CLIENTES		
id	nombre	apellido
1	Juan	Perez
2	Clara	Sanchez
3	Marta	García



VENTAS		
id	cliente_id	fecha
1	2	12/03/2019
2	2	22/08/2019
3	1	04/09/2019

# Creando un INNER JOIN

**Antes** *escribíamos:*

```
SQL  SELECT clientes.id AS id, clientes.nombre, ventas.fecha  
      FROM clientes, ventas
```

**Ahora** *escribiremos:*

```
SQL  SELECT clientes.id AS id, clientes.nombre, ventas.fecha  
      FROM clientes  
      INNER JOIN ventas
```

“

Si bien ya dimos el primer paso que es **cruzar** ambas tablas, aún nos falta aclarar **dónde** está ese cruce.

Es decir, qué **clave primaria (PK)** se cruzará con qué **clave foránea (FK)**.

”



## Creando un INNER JOIN *(cont.)*

La sintaxis del JOIN **no utiliza** el **WHERE** si no que **requiere** la palabra **ON**. Es ahí en donde indicaremos el **filtro** a tener en cuenta para realizar el cruce.

Es decir, que lo que antes escribíamos en el **WHERE** ahora lo escribiremos en el **ON**.

SQL

```
SELECT clientes.id AS id, clientes.nombre, ventas.fecha  
FROM clientes  
INNER JOIN ventas  
ON clientes.id = ventas.cliente_id
```

DigitalHouse>  
Coding School