**STATEMENTS**

CREATE TABLE: crea una nueva tabla en la base de datos

DROP TABLE: elimina una tabla en la base de datos

SELECT: te permite leer y mostrar datos. Esto es lo que se conoce como una “query” (consulta).

La estructura que siguen los statements siempre se hacen en el mismo orden y algunos de ellos son obligatorios mientras que otros son opcionales.

Hay que introducir el concepto de “CLAUSES” (cláusula), que son cada una de las posibles opciones que tenemos para hacer la consulta. Por ejemplo: SELECT, FROM.

**CLAUSULAS**

**El orden en el que vienen descritas las clausulas abajo es el que hay que respetar para las consultas.**

**SELECT:** te indica de qué columna o columnas queremos obtener la información

* Las columnas se separan por comas, sin necesidad de añadir coma al final de la última columna.
* Podemos usar \* para seleccionar todas las columnas.
* Es **OBLIGATORIA**
* Se pueden crear columnas derivadas que resulten de operaciones de otras columnas. Podemos: **sumar, restar, multiplicar y dividir**. Por ejemplo:

SELECT columnA + columnB **AS** sumaAB

SELECT columnA - columnB **AS** restaAB

SELECT columnA \* columnB **AS** multiAB

SELECT columnA / columnB **AS** divAB

* Se puede usar el **SELECT DISTINCT** para evitar algunas columnas con datos duplicados (si por ejemplo un ID en particular tiene distintos valores en otra columna, no saldrán repetidos).

SELECT DISTINCT a.name AS accountName, w.channel AS channel

FROM web\_events AS w

JOIN accounts AS a

ON w.account\_id = a.id

WHERE a.id = 1001;

**FROM:** especifica de qué tabla o tablas se coge la información

* Es **OBLIGATORIA**.

**WHERE:** te permite filtrar por condiciones

* Es **OPCIONAL**
* Se pueden usar condicionales como: >, <, >=, <=, =, !=
* Para comparar con columnas que no son numéricas se usan comillas simples: WHERE name = ‘Andres’
* **NO PERMITE** trabajar con **agregaciones**, es decir, no se puede usar WHERE SUM(columnA), para eso existe HAVING

**GROUP BY:** tiene sentido usarlo cuando usamos agregaciones. Lo que permite es agrupar ciertas operaciones teniendo en cuenta algún dato en particular (por ID, por región, por idioma, etc) al usar las agregaciones.

* Es **OPCIONAL.**
* **Cualquier columna en SELECT que no tenga una operación de agregación debe aparecer en la clausula GROUP BY**

**HAVING:** permite filtrar usando agregaciones, por ejemplo: HAVING SUM(columnA)

**ORDER BY:**

* Es **OPCIONAL**
* Ordena de forma predeterminada de forma ascendente (A-Z, menor-mayor, antes-después).
* Se puede añadir “DESC” para que se ordene justo al contrario tras el nombre de la columna de referencia.
* Se pueden añadir, separando con comas, más de una columna para ordenar.

**LIMIT:** te limita el número de filas que se muestra

* Es **OPCIONAL**.
* Ayuda para una primera toma de contacto con la tabla y ver qué datos nos pueden interesar de la misma.

**OPERACIONES LÓGICAS**

**LIKE:** permite realizar operaciones similares a las de WHERE y =, pero para casos en los que no se sabe exactamente lo que se busca.

* Por ejemplo si queremos buscar una URL usaríamos:

WHERE url LIKE ‘%google%’

El símbolo % significa que puede contener cualquier número o carácter antes o después que google.

**IN:** permite realizar operaciones similares a las de WHERE y =, pero para más de una condición.

* WHERE name IN (‘Amazon’, ‘Spotify’)
* WHERE value IN (1000, 2000)
* Ojo, según el workspace para usar el apostrofe podría usarse una doble comilla simple, por ejemplo para Macy’s pondríamos ‘Macy’’s’

**NOT:** se utiliza con IN y LIKE para seleccionar todas las filas que NO SEAN SIMILARES o NO ESTÉN EN una determinada condición.

Por ejemplo para los casos anteriores:

* WHERE url NOT LIKE ‘%google%’
* WHERE name NOT IN (‘Amazon’, ‘Spotify’)
* WHERE value NOT IN (1000, 2000)

**AND & BETWEEN:** permite combinar operaciones en las que todas las condiciones combinadas deben ser verdaderas.

* **BETWEEN es inclusivo con los valores límite con valores numéricos PERO NO CON FECHAS.**

**OR:** permite combinar operaciones en las que al menos una de las condiciones combinadas debe ser verdadera.

**JOIN**

Esto sirve para unir información de distintas tablas. Al hacerlo complica un poco la lógica, ahora para usar el SELECT lo ideal es poner nombreTabla.nombreColumna

El JOIN sigue la siguiente estructura:

SELECT \*

FROM accounts

JOIN orders

ON accounts.id =orders.account\_id;

Para usar los JOIN también conviene usar el concepto de claves:

**Clave primaria (PK; Primary Key)**

Una clave **primaria** es una columna única en una tabla determinada. **Es común que la clave primaria sea la primera columna en nuestras tablas en la mayoría de las bases de datos.**

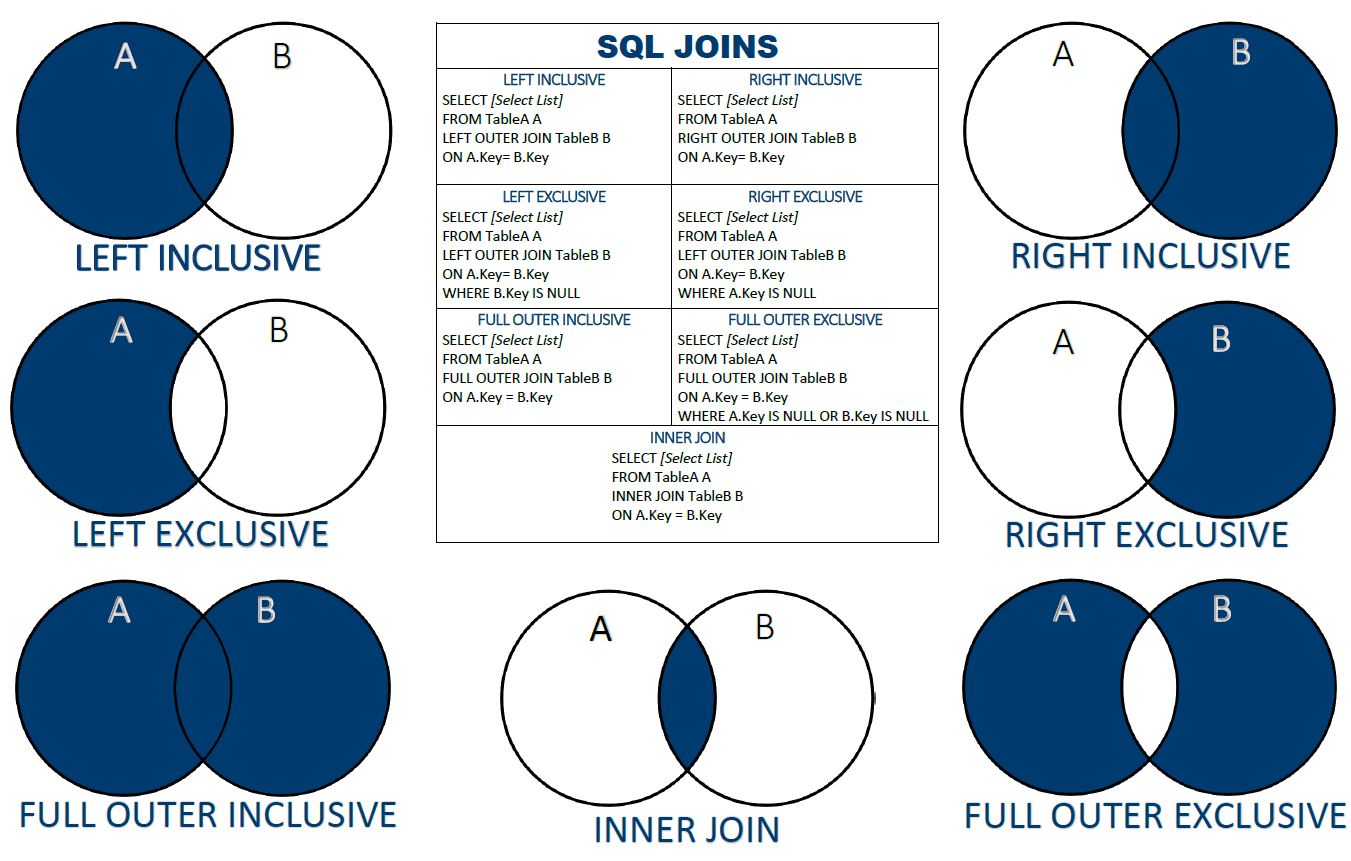
**Clave “extranjera” (FK; Foreign Key)**

Una clave **“extranjera”** es una columna en una tabla que es clave primaria en otra tabla diferente.

Generalmente el ON conecta el PK de una tabla con el FK de otra, es decir:

ON tableA.PK = tableB.FK.

No es importante el orden, también podría ponerse ON tableB.FK = tableA.PK



**ALIAS**

Podemos poner un alias a una tabla simplemente dejando un espacio o poniendo AS al usar el FROM o el JOIN.

**AGGREGATION**

**COUNT**: cuenta el número de filas en una columnas. Se puede aplicar a todos los tipos de datos, al contrario que otros como SUM, AVERAGE que solo aplican a valores numéricos.

**SUM**: suma el valor de todos los elementos de una columna. Trata a NULL como 0.

**MIN** y **MAX**: devuelve el mínimo y máximo valor de una columnas. Se puede aplicar a textos (MIN-MAX = A-Z)

**AVG**: devuelve la media de una columna. Ignora los NULL, si se quieren tener en cuenta como 0 hay que usar el SUM y dividir entre el número total de filas a tener en cuenta (usando COUNT por ej).

Estas operaciones actúan sobre las **columnas**, no sobre las filas.

Hay otros statements que podemos añadir en el SELECT, como por ejemplo:

**CASE** WHEN channel = ‘facebook’ THEN ‘yes’ END AS is\_facebook

Nota: si el canal no es facebook la columna resultante se rellena con NULL

**CASE** WHEN channel = ‘facebook’ THEN ‘yes’ ELSE ‘no’END AS is\_facebook

Se pueden añadir condicionales u otro tipo de operador como el WHERE en el WHEN:

WHEN channel = ‘facebook’ **OR** channel = ‘direct’ THEN ‘yes’ ELSE ‘no’END AS is\_facebook

**FECHAS**

Trabajar con fechas (ordenar, hacer agregaciones) no suele ser muy útil. Hay una serie de clausulas que nos sirven para trabajar con fechas.

Antes de eso hay que saber que en las bases de datos las fechas siguen el formato:

YYYY-MM-DD HH-MM-SS

**DATE\_TRUNC**

SELECT SUM(columnA) AS suma,

DATE\_TRUNC(‘day’, occurred\_at) AS day

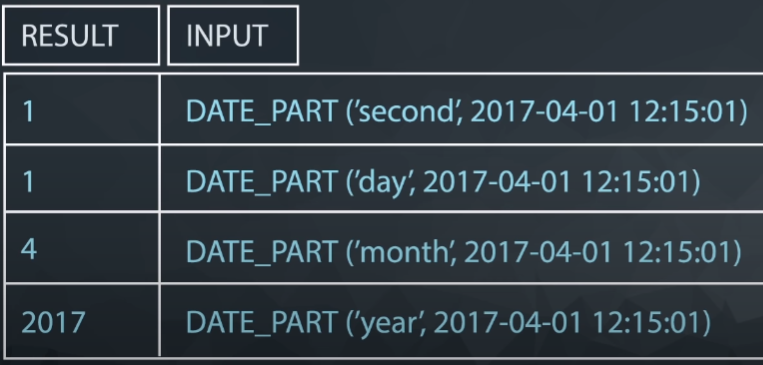
GROUP BY DATE\_TRUNC(‘day’, occurred\_at) AS day

ODER BY DATE\_TRUNC(‘day’, occurred\_at) AS day

Lo que hacer el DATE\_TRUNC básicamente es truncar el dato de la fecha hasta donde digamos (year, quarter, month, week, day, hour, minute, second) y con el GROUP BY, en este ejemplo, se agrupa el resultado del SUM por cada día.

**DATE\_PART**

Te devuelve el valor concreto del dato que le pidas de una fecha. Ejemplo:



**‘dow’** te devuelve el día de la semana de 0-6 (domingo-sabado)

**NULL**

Null significa que no hay datos (es distinto a que sea cero o un espacio en blanco). Suele ser frecuente que aparezca NULL cuando hacemos LEFT o RIGHT JOIN (también si hay datos que faltan en la base de datos).

WHERE x IS NULL

WHERE x IS NOT NULL

**BUENAS PRACTICAS**

1. Aunque SQL no es “case sensitive”, los comandos se suelen escribir en mayúsculas y las tablas y columnas en minúsculas. Ejemplo:

SELECT account\_id

FROM orders

select account\_id

from orders

SeLeCt AcCoUnt\_id

FrOm oRdErS

Estas tres consultas funcionarían exactamente igual, aunque la buena práctica es la primera forma.

1. Evitar espacios en los nombres de las tablas y columnas, poner en su lugar “\_”. Se puede llegar a ver en otros entornos: FROM [Table Name].
2. Puntos y comas: aunque no todos los entornos los exigen, es una buena práctica añadirlos al final de la consulta, además esto también permite en algunos entornos ejecutar más de una consulta a la vez.

SELECT account\_id FROM orders;