**SQL** (Structured Query Language -> SGBDR)

[cheatsheet](https://dochub.com/%C3%81lvaro-s%C3%A1ez/ALzmZB7wMbp9557RX8J560/sql-full-cheatsheet-pdf?pg=2) 4552 2323 7287 5214

Un tipo de lenguaje de programación que permite **manipular** y **descargar** datos de una **base de datos**.

* Es un lenguaje **DECLARATIVO** (decimos el QUÉ queremos, pero no el cómo hacerlo).
* No es sensible a mayúsculas/minúsculas, excepto para los objetos creados.
* **DDL**: data **definition** language
* **DML**: data **manipulation** language
* **DCL**: data **control** language

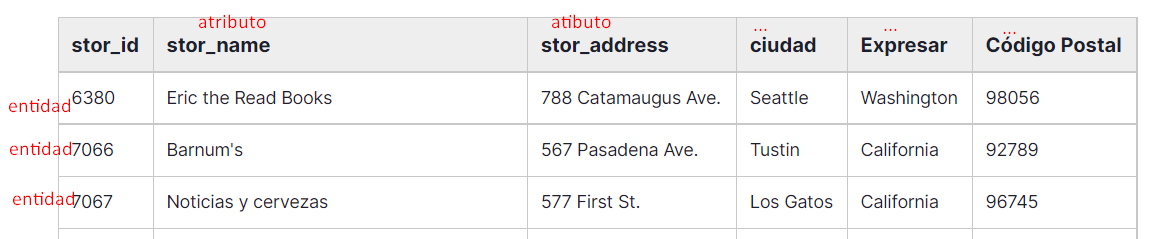
→ Nos permite **crear**, **modificar** y **eliminar** objetos dentro de la base de datos

→ Los nombres deben de ser únicos

**Servidor** > **BBDD** > **Tablas (**conjuntos de datos**) o Vistas (**con un alias**)** > **Columnas**

**ENTIDADES**: están representadas por **filas en una tabla** (suelen abarcar **varias columnas**, que representan **atributos** que contienen información sobre cada entidad).

En la siguiente tabla, las filas representan tiendas (entidades) y para cada tienda, tenemos un campo store\_id único , así como atributos (store\_name , store address , ciudad , estado y zip)



Las entidades en nuestros datos también pueden estar relacionadas con otras entidades (de **otras tablas**) a través de distintos tipos de **relaciones**:

* **1:1** → cuando una entidad puede tener o pertenecer a otra entidad únicamente (una entidad que solo aparece en una tabla)
* **1:M** → cuando una entidad puede tener o pertenecer a múltiples entidades (una entidad que aparece en muchas tablas y esa entidad es el único nexo entre ellas, es decir, solo hay un key-value)
* **M:M** → cuando una entidad puede tener o pertenecer a múltiples entidades y viceversa (una entidad que aparece en muchas tablas, pero que hay muchos nexos entre ellas) Ej.: tabla maestro de hoteles, tabla maestro de destino, ventas por hoteles y destinos (2 key-values diferentes que aparecen en distintas tablas)

→ Las **relaciones entre entidades** se mantienen mediante **valores clave (**llamada **clave externa)**, donde un campo de una tabla hace referencia al identificador de la tabla relacionada

**Modelado dimensional**

Divide los datos en dos categorías:

* HECHOS (Eventos): Representan eventos del negocio ventas , envíos , etc.) sobre los que se definen las medidas.

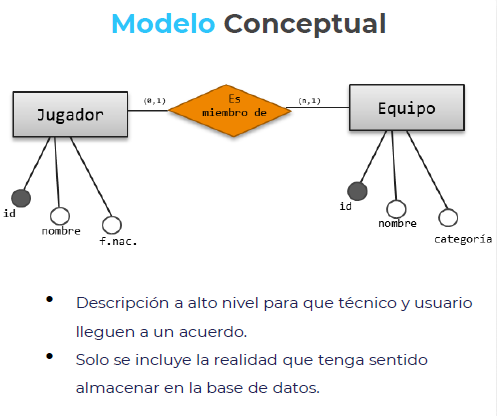
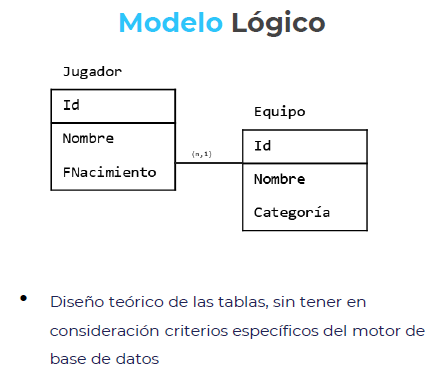
**→ Tablas de Hechos** (crecen rápidamente y no se modifican. Es decir, almacenan todos los eventos y su información entrante)

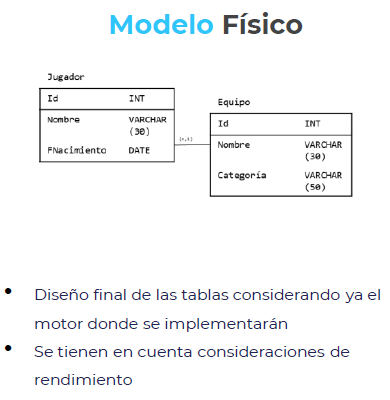
* DIMENSIONES: Proporcionan el contexto alrededor del evento : **el quién, qué, dónde y cuándo**

**→ Tablas de Dimensiones** (son los llamados master feed, que no contienen tanta información pero sí que se van modificando. Ej.: la tabla maestro de hoteles, cada vez que hay un nuevo hotel se añade info de este)

DIAGRAMAS ENTIDAD-RELACIÓN(E/R): Para diseñar una base de datos debemos de mostrarlo a través de:

* Modelo CONCEPTUAL: estructura de las tablas
* Modelo LÓGICO: diseño lógico de las tablas
* Modelo FÍSICO: diseño final de las tablas

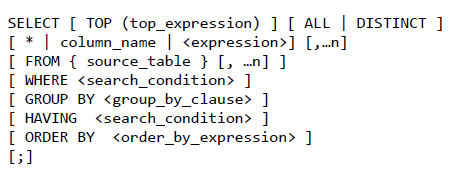


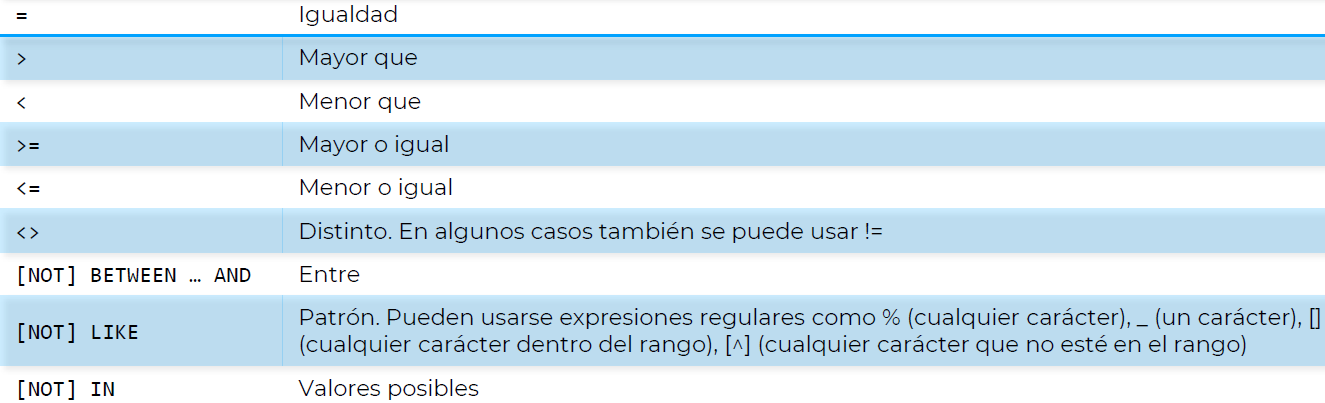
Comandos básicos de SQL: SALVO QUE CREEMOS UNA NUEVA TABLA, CON UNA CONSULTA ESTANDAR ESTARÍAMOS CREANDO UNA TABLA TEMPORAL.

* **SELECT**: **Devuelve registros** de la base de datos y permite la selección de uno o varios registros y columnas de una o varios conjuntos de datos de origen. Lo que pongamos en el select será lo que se muestre, por lo que si filtramos o agrupamos reduciremos el número de registros (salvo con el OVER que añadimos nuevas columnas a las ya existentes)

→ **\***, columnas, funciones agregadas (count, ..), OVER, expresiones, CASE …

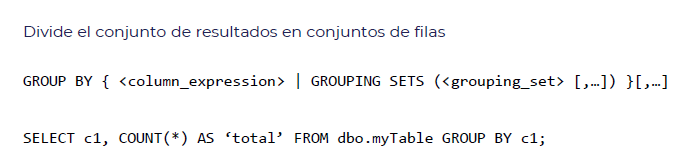
→ **AS** (cambia el nombre de la columna o tabla con un alias (para acortar el trabajo))



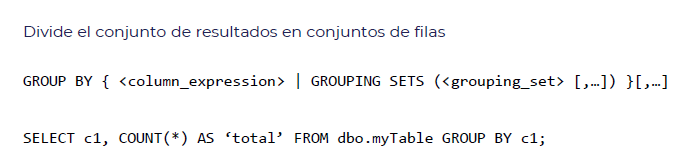
* **FROM**: Seleccionas **las TABLAS** de las que quieres descargarte los datos…**AS**
  + **JOINS … AS**
  + **WHERE**: Se usa para **filtrar registros** del conjunto de datos.
    - **operadores**:
    - **AND, OR, IN, CASE:** para continuar con más filtros
* **GROUP BY:** organiza filas en grupos por una o más columnas y muestra solo esos registros únicos (como si deduplicase).
  + **con funciones agregadas:** realiza un cálculo en un grupo y devuelve un valor único por grupo.

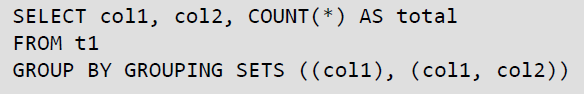
(un ejemplo sería ordenar por nombre de producto, y hacer un count, así en vez de mostrar todos los records, mostrará la info agrupada (deduplicando los nombre de los productos y sumando su count).

→ ¡IMPORTANTE! Va después del WHERE, por lo que si hay un filtro solo se agruparán esos datos filtrados

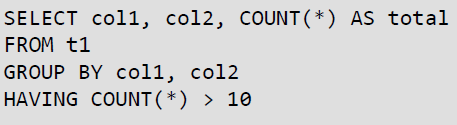


**GROUP BY GROUPING SETS** → CREA SUBGRUPOS (añadiendo filas nuevas) → ejemplo de un subgrupo ((pc.Name), (pc.Name, p.Name))



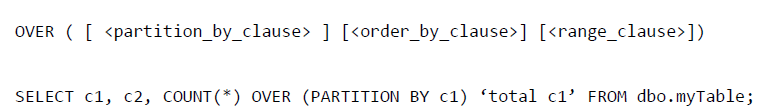


* + **HAVING *condition*:** Filtra los grupos de la fase anterior que cumplan las condiciones (es como el WHERE para el SELECT)



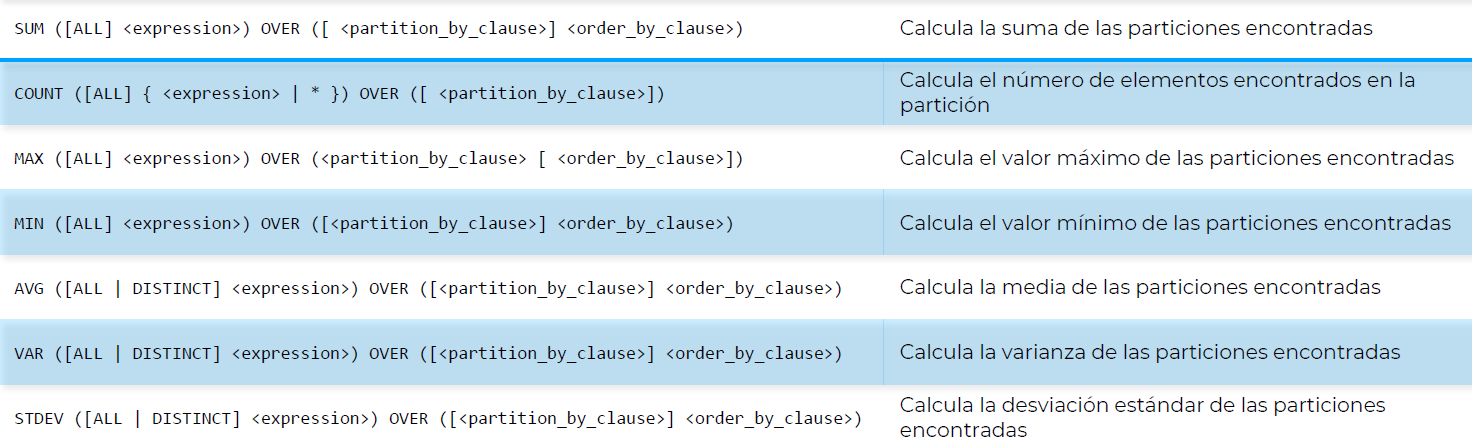
* **OVER: (***SE APLICA EN EL SELECT***)** Group by utiliza las funciones agregadas reduciendo el número de registros devueltos. Si queremos operar sobre los grupos de registros **sin reducir datos** (dejar de no mostrar los que no agrupaba group by) deberíamos de usar las **funciones ventana** → **no reducen el conjunto de resultados final PQ SE APLICA EN EL SELECT (es decir, hace una agrupación y un cálculo sobre esa agrupación y lo añade como una columna más)**.

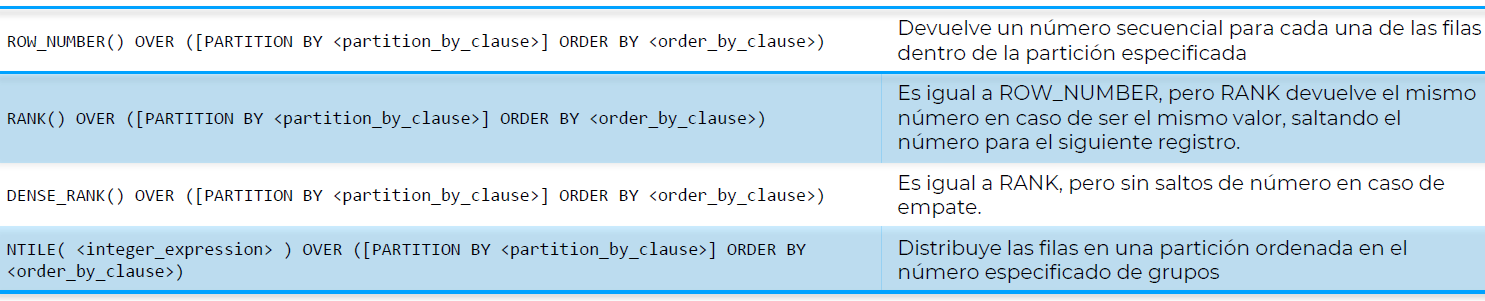
2 POSIBILIDADES:

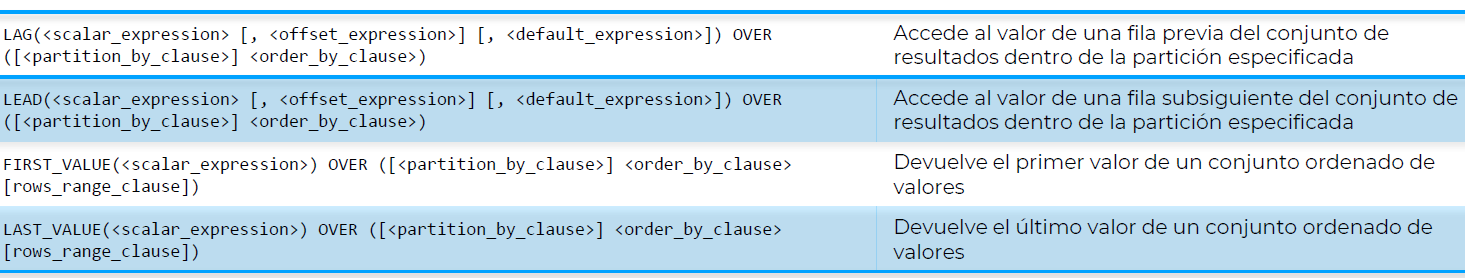
1. OVER() → haría un cálculo de todo
2. OVER (PARTITION BY x) → haría un cálculo solo de x

→ aquí, el “count()” se estaría aplicando sólo a C1 (esa sería la agrupación)







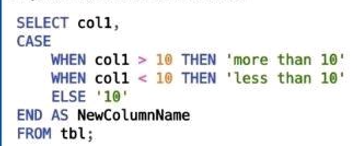


* **ORDER BY … ASC | DESC:** Se ordena el conjunto de datos de la fase anterior según la lista

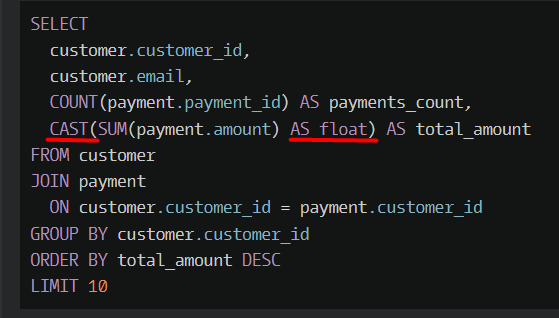
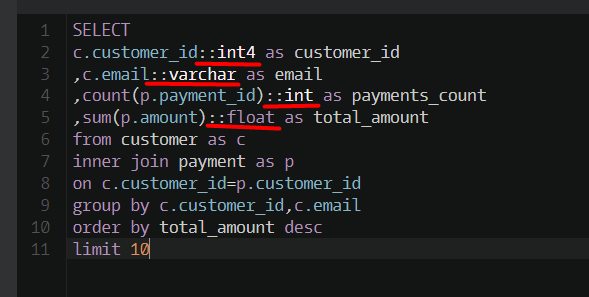




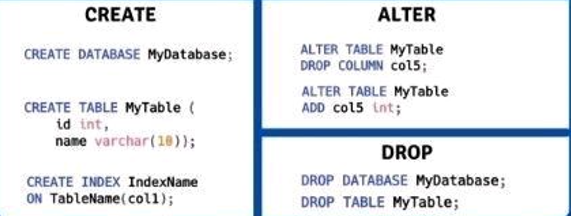
* **TOP … LIMIT:** Filtra registros del conjunto de datos
* **LIKE**
* **CASE:** CREA UNA NUEVA COLUMNA BAJO UNA CONDICIÓN
  + **CASE**
    - **WHEN** condicion **THEN** pinta lo siguiente
    - **ELSE** pinta lo siguiente
  + **END AS** nombre de la columna



**¡FORMATO! → Se cambia con “ :: {format}“ o “CAST(.......AS{format})”**

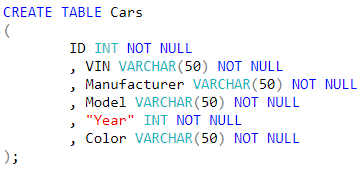
****

*DDL: data definition language*

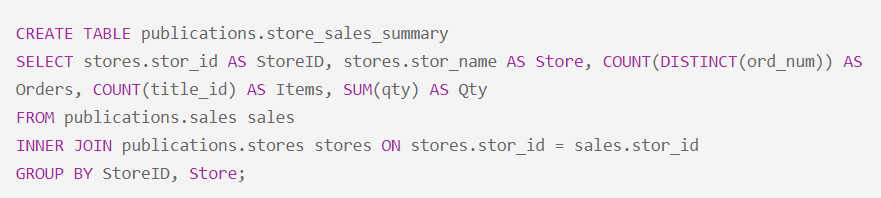


* **CREATE**:

1. **crea una tabla PERMANENTE**
   * **crear una tabla desde cero**



* + **crear una a partir de una query (forma1)**

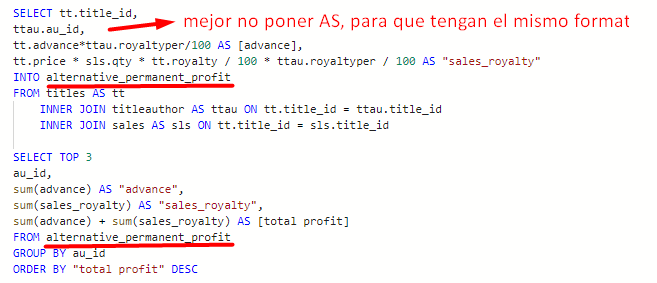


* + **crear una a partir de una query (forma 2)**

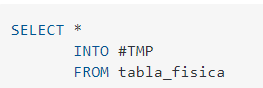
**SELECT** campo1, campo2 **INTO** **TablaPermanente FROM** Tabla;

→ PARA LLAMARLA LUEGO SERÍA:

**SELECT** campo1, campo2 **FROM** **TablaPermanente**



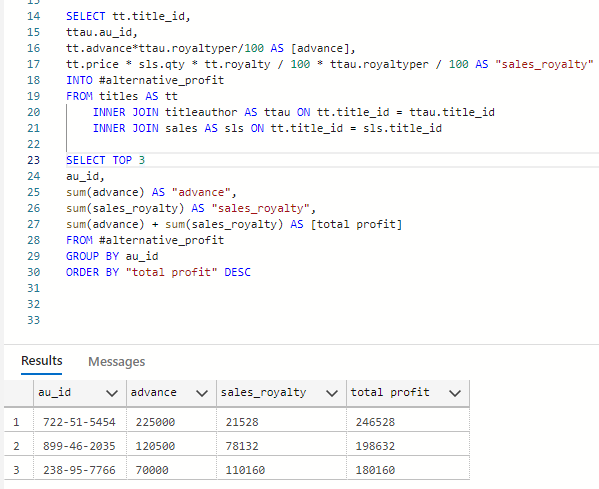
1. **crea una tabla TEMPORAL**



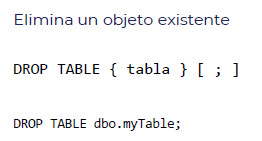
**SELECT** campo1, campo2 **INTO** **#TablaTemporal FROM** Tabla;

→ PARA LLAMARLA LUEGO SERÍA:

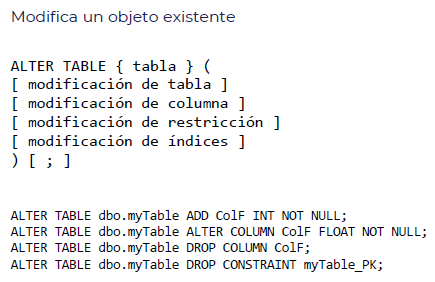
**SELECT** campo1, campo2 **FROM** **#TablaTemporal**



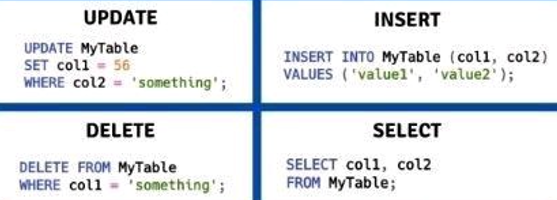
* **DROP**: elimina LA TABLA

****

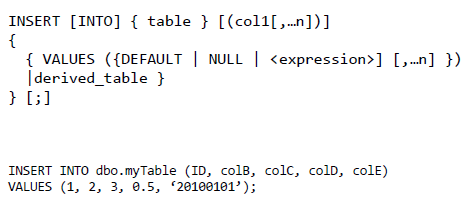
* **ALTER**:



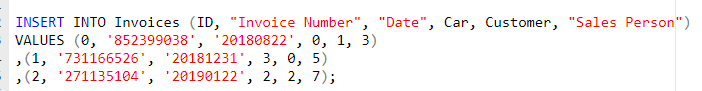
*DML: data manipulation language*



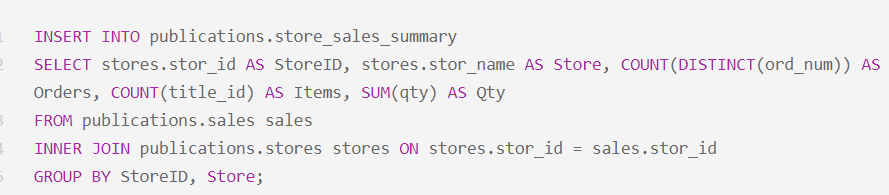
* **INSERT**: introduce registros en una tabla **INTO** table (columns) ***VALUES*** *(“x”, “y”);*

**

* + **insertar valores desde cero**

**

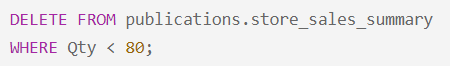
* + **insertar valores a partir de una query**

**

* **DELETE**: elimina registros → **FROM** table ***WHERE …***

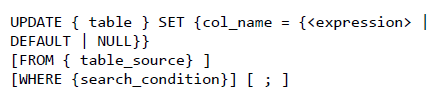
******

******

****

**→ Si no ponemos WHERE eliminamos TODOS los registros de la tabla, pero ésta y su estructura sigue existiendo**

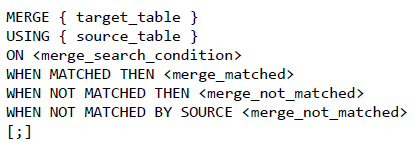
* **UPDATE**: modifica registros. **SET … *WHERE …***

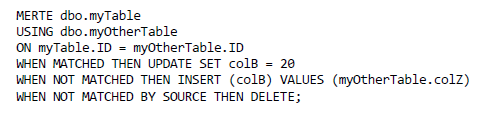
******





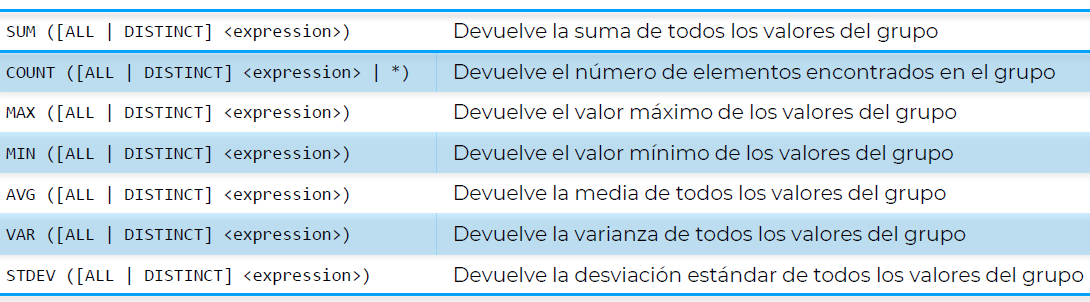
* **MERGE**: inserta, actualiza o ejecuta operaciones de borrado en la tabla de destino a partir del resultado de una combinación con una tabla de origen



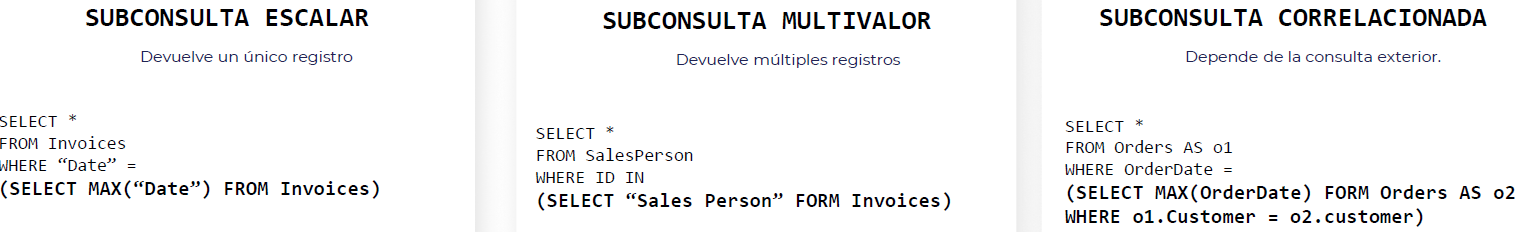


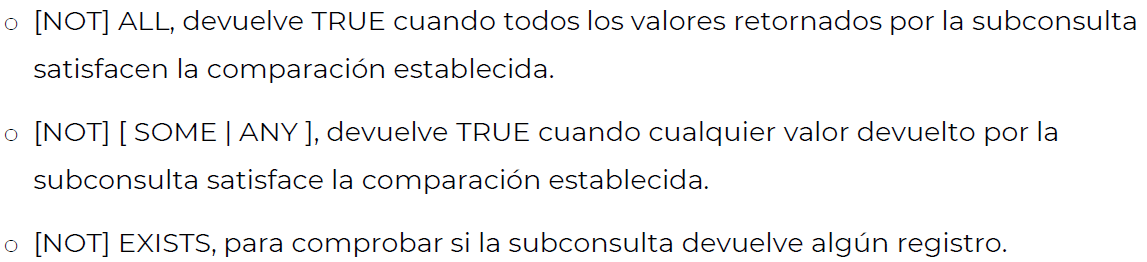
*OPERACIONES NUMÉRICAS*

* **COUNT (DISTINCT** nombre columna**)**
* **SUM ()**
* **AVG ()**
* **MIN ()**
* **MAX ()**
* **VAR ()**
* **STDEV()**



**SELECT** (como subconsulta): Son consultas dentro de consultas CRAN TABLAS TEMPORALES LLAMADAS [,**DERIVATED TABLE**](https://dev.mysql.com/doc/refman/8.0/en/derived-tables.html). Pueden ser **escalares** (solo devuelven un único valor), **multivalor** o como **tabla** (múltiples campos, múltiples registros).





Una tabla derivada es una expresión que genera una tabla dentro del ámbito de una FROMcláusula de consulta . Por ejemplo, una subconsulta en una cláusula de [SELECT](https://dev.mysql.com/doc/refman/8.0/en/select.html) declaración FROMes una tabla derivada:

SELECT ... FROM (*subquery*) [AS] *tbl\_name* ...

La [JSON\_TABLE()](https://dev.mysql.com/doc/refman/8.0/en/json-table-functions.html#function_json-table)función genera una tabla y proporciona otra forma de crear una tabla derivada:

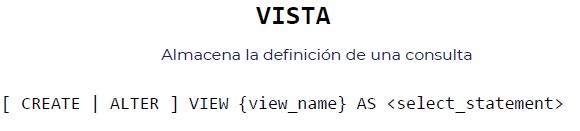
SELECT \* FROM JSON\_TABLE(*arg\_list*) [AS] *tbl\_name* ...

La cláusula es obligatoria porque cada tabla en una cláusula debe tener un nombre. Todas las columnas de la tabla derivada deben tener nombres exclusivos. Alternativamente, puede ir seguido de una lista entre paréntesis de nombres para las columnas de la tabla derivada: [AS] ***tbl\_name***FROM***tbl\_name***

SELECT ... FROM (*subquery*) [AS] *tbl\_name* (*col\_list*) ...

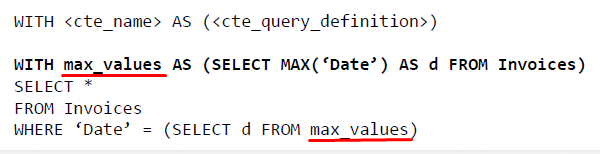
VISTAS:

Una forma de **reutilizar una consulta** (no correlacionada) es creando una vista. → Una vista no es más que **un alias dado a una consulta** cuya **definición** (no los datos) **persiste**.

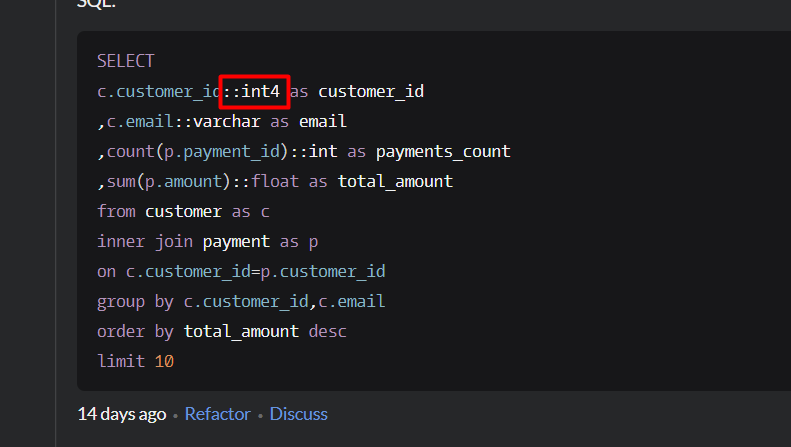


CTE (Common Table Expression):

Es un nombre temporal dado a un resultset → Realmente **crea una tabla temporal** la cual podremos utilizar.



**CAMBIAR FORMATO:**



**JOIN**: para combinar distintos conjuntos de datos **+ ON** (defines los KEY VALUES de ambas tablas)

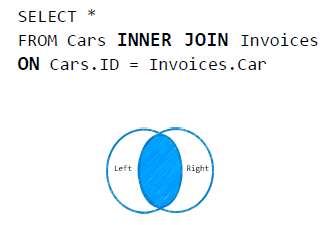


El primero que se le declara se le considera la parte izquierda de la combinación

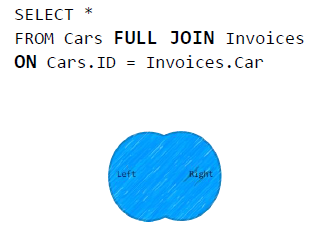
Es posible usar más de un join a la vez y de diferentes tipos de join

Es posible combinar una tabla consigo misma

* **inner join** → solo los registros key values que coincidan entre tablas

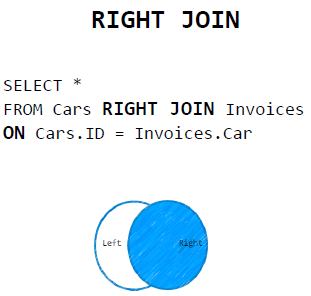
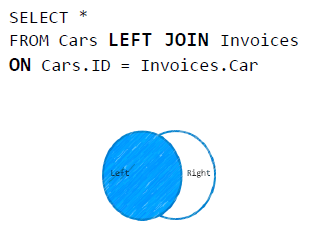


* **full join** → todos los registros, tengan key value o no (si no tienen que value aparecerán como null los de la otra tabla)

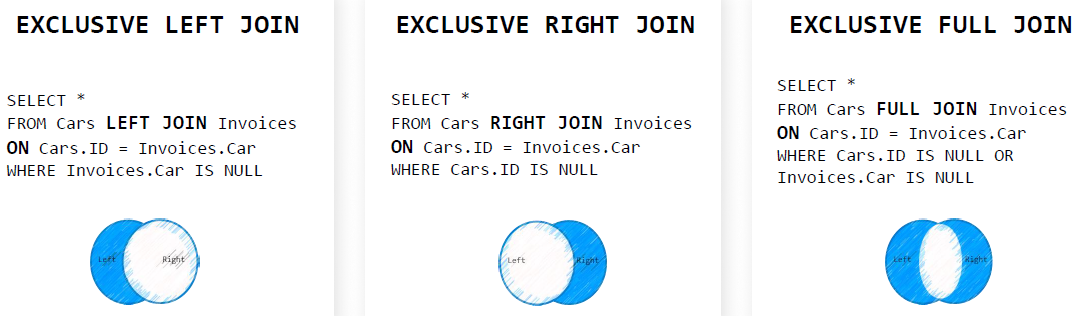


* **left join** → todos los registros de la izquierda + los registros de la derecha que estén presentes en el key value de la izq (sean null o no)

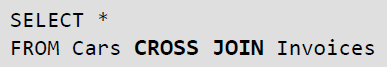
**right join** → todos los registros de la derecha + los registros de la izquierda que estén presentes en el key value de la der (sean null o no)



* **exclusive left join**
* **exclusive right join**
* **exclusive full join**



* **cross join:** saca productos cartesianos. Cada registro de la tabla izquierda se combina con un registro de la tabla derecha, obteniendo un conjunto que es el resultado de n x m filas de los conjuntos originales.

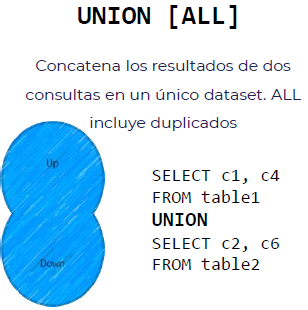


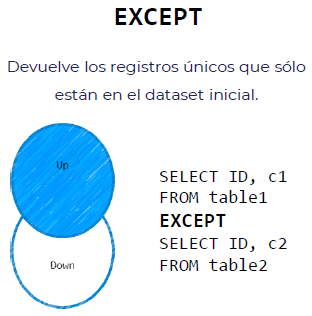
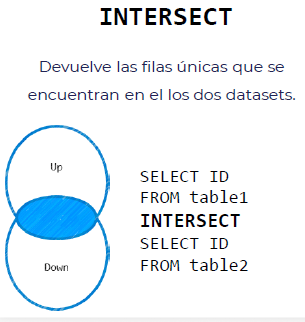
*Set Operators*

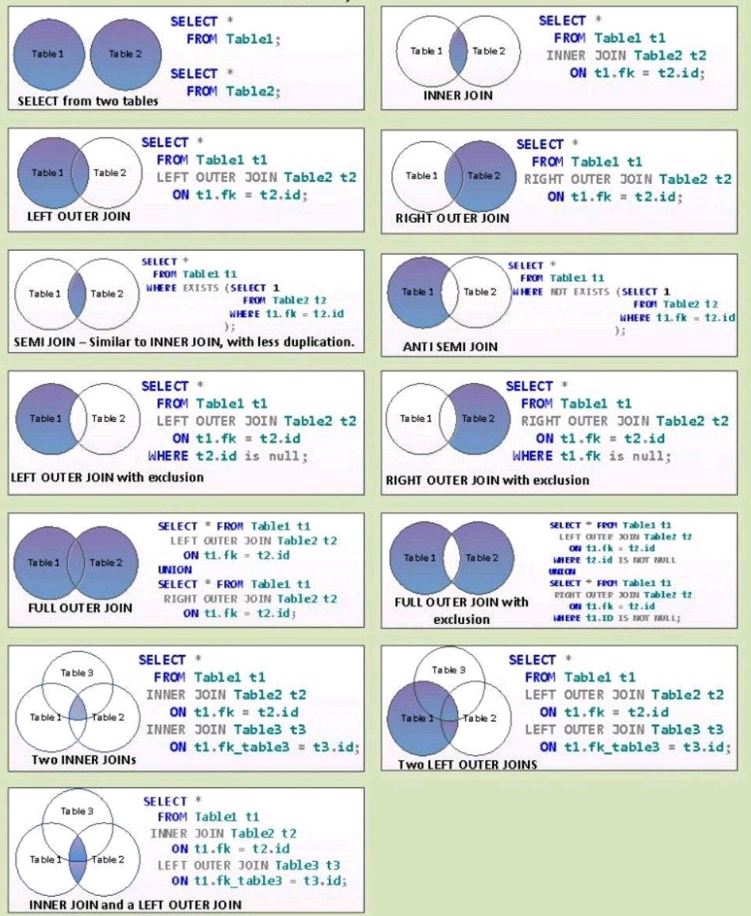
En el caso de los JOINs lo ensanchamos (añadimos columnas)

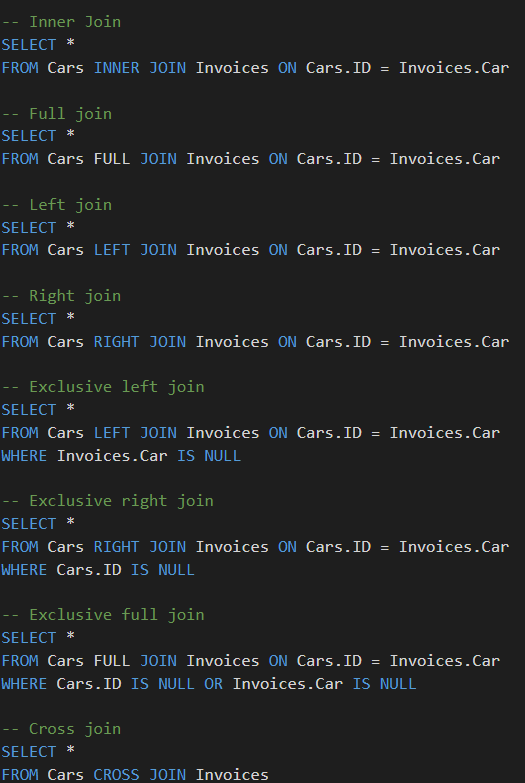
y en otro (Set Operators ) lo alargamos (añadimos registros)

* INTERSECT
* EXCEPT
* UNION o UNION [ALL] (el union ALL incluye duplicados)









## **SLQ en Python (jupyter)**

→ **pd.read\_sql\_query()** y **.to\_sql**

Hay diferentes librerías que nos permiten conectarnos con bases de datos MySQL. Con Pandas podemos hacerlo usando pymysql o pyodbc y create\_engine.

Pasos Previos que se realizan una única vez(en nuestro caso bdd de Azure):

* Configurar nuestra **DATABASE** de [Azure](https://portal.azure.com/#home). [DOC con los pasos explicados](https://docs.microsoft.com/en-us/azure/azure-sql/database/single-database-create-quickstart?tabs=azure-portal)
* Instalar el [**DRIVER**](https://docs.microsoft.com/en-us/sql/connect/odbc/download-odbc-driver-for-sql-server?view=sql-server-ver15)para SQL Server
* Configurar **pyodbc** → módulo Python que simplifica el acceso a las bases de datos ODBC. Es necesario instalarlo en la terminal para poder usar sqlalchemy y tirar queries a un database → **pip install pyodbc**
* **conda install -c anaconda sqlalchemy**

Pasos en Jupyter:

1º CONEXIÓN CON LA BBDD P@~~w0rd123

1. pymysql
2. importar librerías:

**import pymysql**

**from sqlalchemy import create\_engine**

**import** pandas **as** pd

1. **create\_engine function**: reemplazando username y password con el nombre de usuario y la contraseña para la base de datos MySQL en la máquina local. Asignaremos el resultado a una variable llamada engine.

→ **engine = create\_engine(**'mysql+pymysql://username:password@localhost/publications'**)**

1. pyodbc
2. importar librerías:

**import** os

**from** dotenv **import** load\_dotenv

**import** **pyodbc**

**import sqlalchemy**

**from** sqlalchemy **import** **create\_engine**

**from** sqlalchemy.engine **import** URL

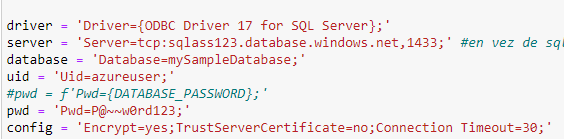
**import** pandas **as** pd

1. **archivo dot.env** para leer contraseñas (y que no aparezcan en el notebook) ¡OPCIONAL! (ya que se pueden poner contraseñas en el notebook)

load\_dotenv('.env')

DATABASE\_PASSWORD = os.environ.get("DATABASE\_PASSWORD")

1. **create\_engine function**: [link de Microsoft SQL Server](https://docs.sqlalchemy.org/en/14/dialects/mssql.html#pass-through-exact-pyodbc-string)
   1. connection\_string:





* 1. connection\_url:



**connection\_url = URL.create(**"mssql+pyodbc", query={"odbc\_connect":connection\_string**})**

*→ "mssql+pyodbc" = motor de búsqueda de la bbdd + driver para importar*

* 1. engine:



**engine = create\_engine(**connection\_url**)**

2º CREAR QUERIES:

1. IMPORTAR:

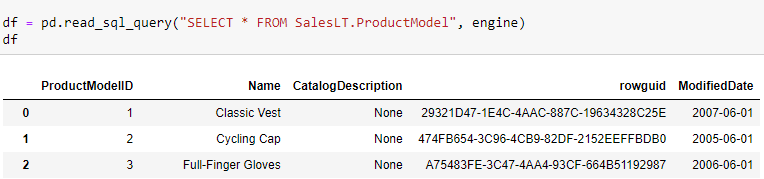
**data = pd.read\_sql\_query(**'SELECT \* FROM publications.employee', **engine)**

1. EXPORTAR:

**data.to\_sql(**“{nombre de la tabla}”, **engine**, if\_exists=**'replace'**, index=False**)**

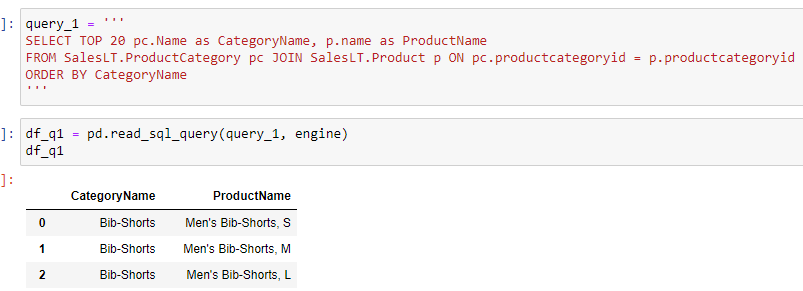
* **replace, append, fail, etc.**

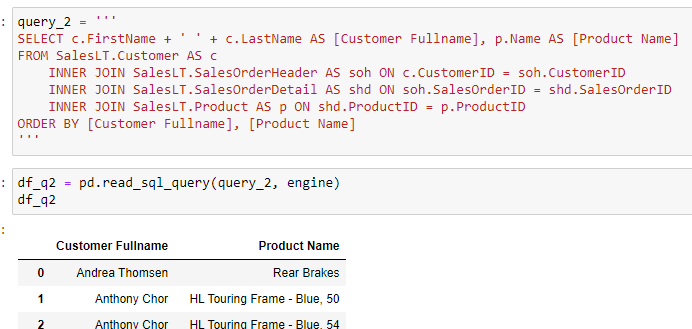
EJEMPLOS

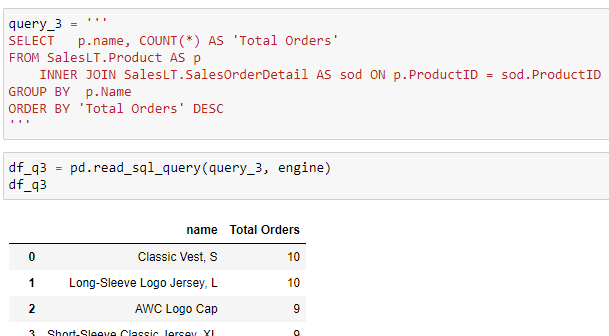


Lo mejor es crear:

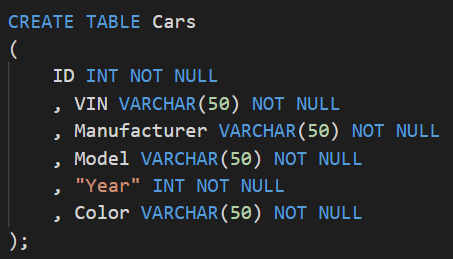
* una variable con la query
* un dataframe para importarla

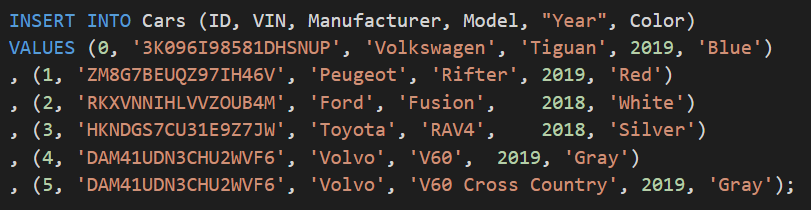


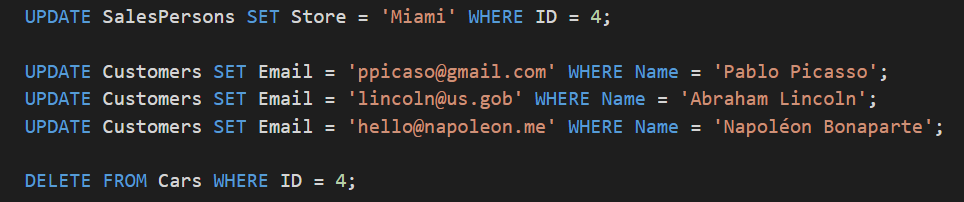




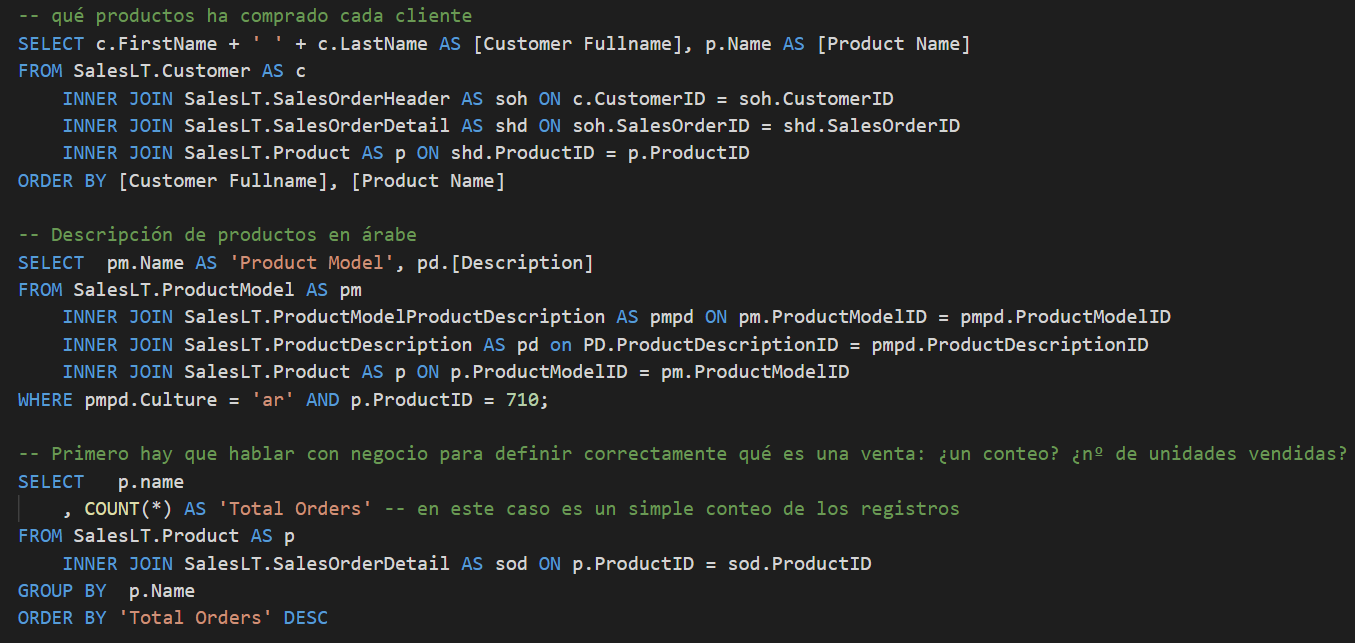
**EJEMPLOS**



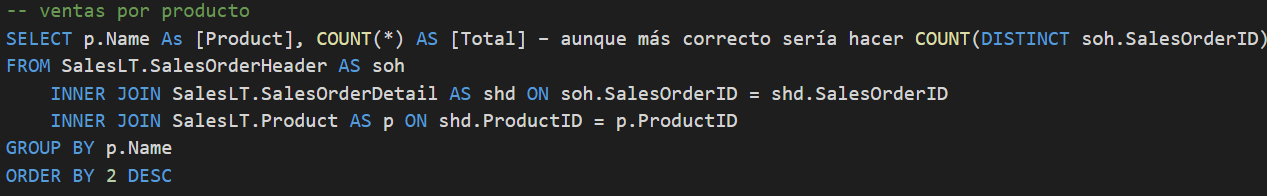


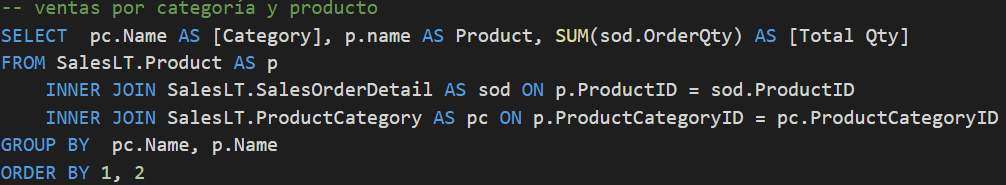


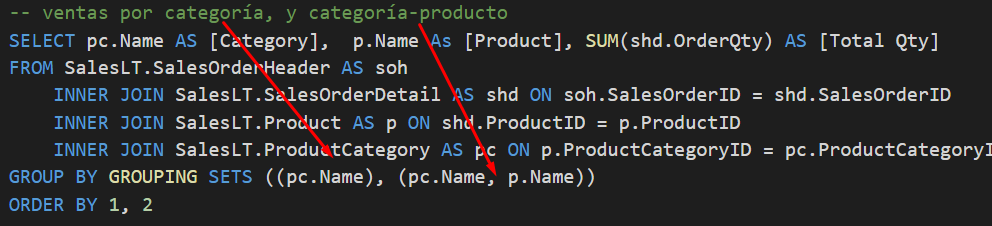
EJERCICIO BBDD de AZURE

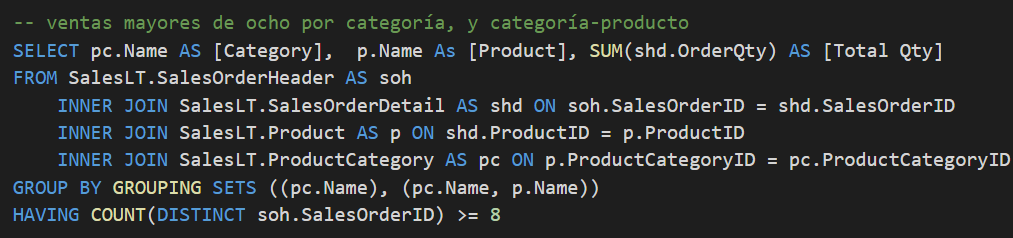


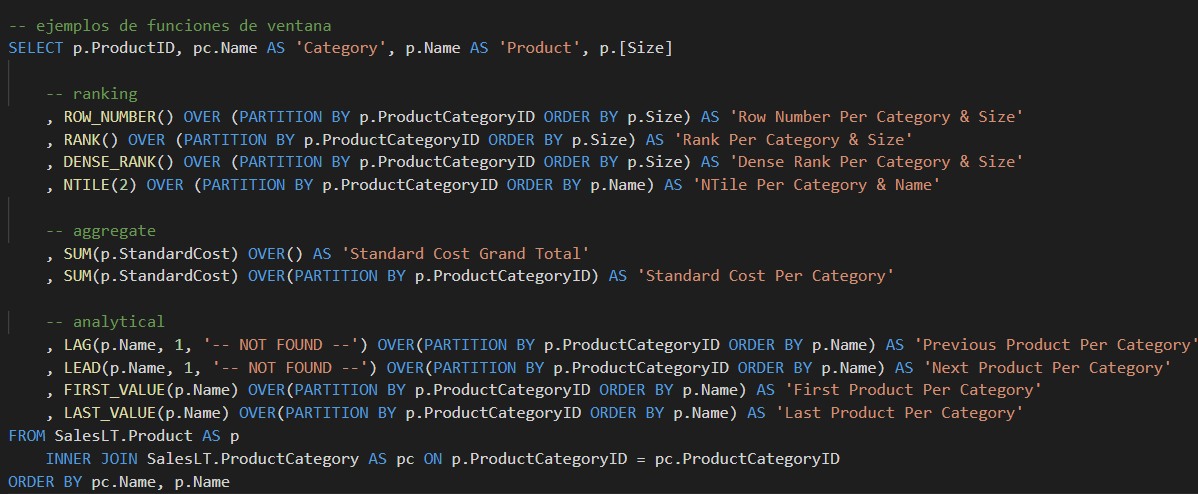
→ Como los registros son de las ventas, cada count será una venta, por eso ponemos count(\*)











**LAB1:**

- Authors → au

- Discounts → dc

- Employee → ep

- Jobs → jb

- Pub\_info → pbi

- Publishers → pbs

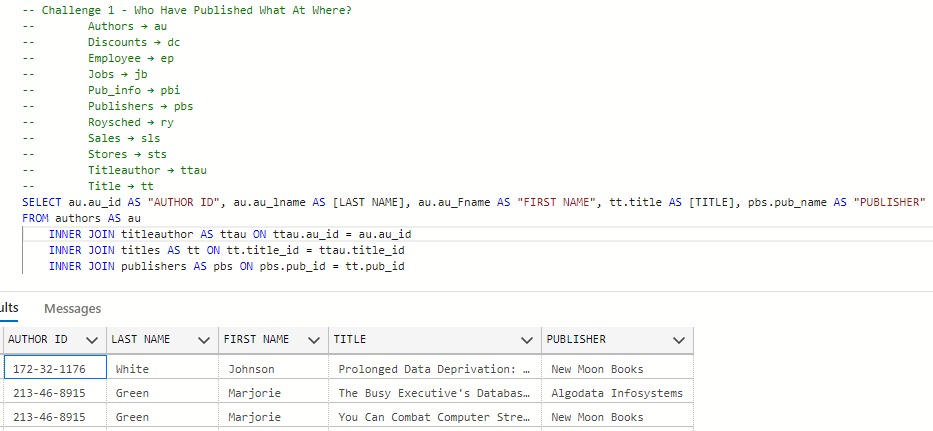
- Roysched → ry

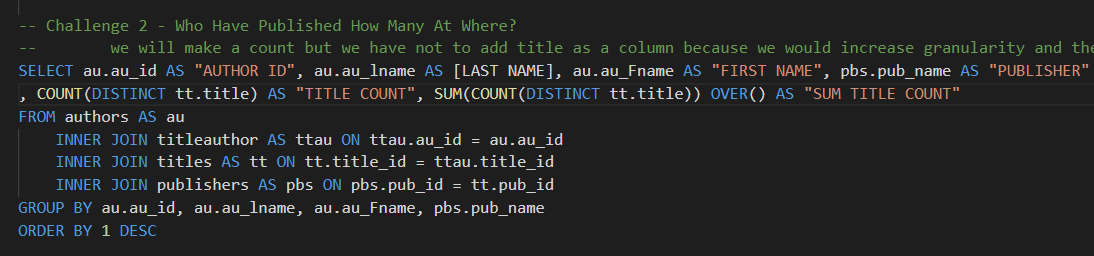
- Sales → sls

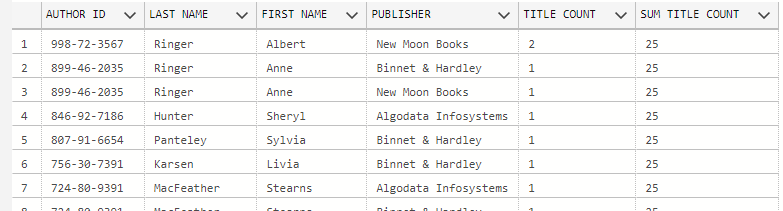
- Stores → sts

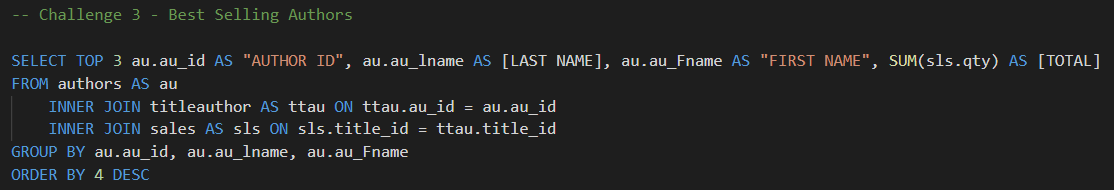
- Titleauthor → ttau

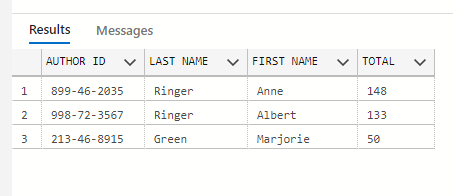
- Title → tt

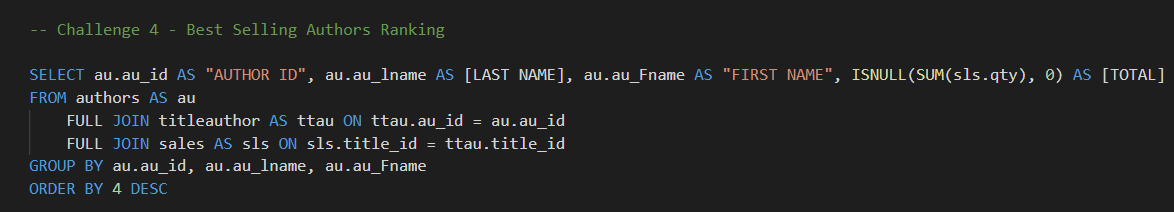














**otro ejemplo:**

