AGGREGATE FUNCTIONS

COUNT()	**Cuenta los valores no nulos de una columna.**
SUM()	**Suma lo valores de una columna.**
MIN()	**Retorna el mínimo de una columna.**
MAX()	**Retorna el máximo de una columna.**
AVG()	**Retorna el promedio de una columna.**

AGGREGATE FUNCTIONS

```
SELECT
    agg_fun(column)
FROM
    table_name;
```

```
SELECT
    COUNT(nombre)
FROM
    estudiantes_neoland;
```

```
SELECT
    MAX(edad)
FROM
    estudiantes_neoland;
```

```
SELECT
    COUNT(DISTINCT edad)
FROM
    estudiantes_neoland;
```

GROUP BY

GROUP BY: Agrupa resultados con respecto a uno o varios campos (columnas).

* GROUP BY debe colocarse inmediatamente después de WHERE (si aplica) y antes de ORDER BY.

```
SELECT
    *
FROM
    table_name
GROUP BY
    column_name;
```

```
SELECT
    *
FROM
    estudiantes_neoland
GROUP BY
    nombre;
```

RESUMEN

```
SELECT
    column_names
FROM
    table_name
WHERE
    conditions
GROUP BY
    column_names
ORDER BY
    column_names;
```

Aliases

Alias: Es usado para renombrar una selección del query.

```
SELECT
    nombre, COUNT(nombre) AS nombre_count
FROM
    estudiantes_neoland
GROUP BY
    nombre;
```

HAVING

HAVING: Funciona como un WHERE aplicado a GROUP BY.

* Con HAVING se pueden aplicar condiciones a las AGGREGATE FUNCTIONS, mientras que WHERE no tiene esta característica.

```
SELECT
    *
FROM
    table_name
GROUP BY
    column_name
HAVING
    conditions;
```

```
SELECT
    nombre, COUNT(nombre)
FROM
    estudiantes_neoland
GROUP BY
    nombre
HAVING
    COUNT(nombre) > 5;
```

LIMIT

LIMIT: funciona como .head() en pandas.

```
SELECT
    *
FROM
    table_name
LIMIT
    n;
```

```
SELECT
    nombre
FROM
    estudiantes_neoland
LIMIT
    5;
```

RESUMEN

```
SELECT
    column_names
FROM
    table_name
WHERE
    conditions
GROUP BY
    column_names
HAVING
    conditions
ORDER BY
    column_names
LIMIT
    n;
```

INSERT

INSERT: Se usa para agregar registros (filas) a las tablas. Se puede insertar una fila que no tenga todas las columnas, estos espacios vacíos se llenarán con Null.

```
INSERT INTO table_name (
        column_1,
        column_2,
        ...,
        column_n)
VALUES (
        value_1,
        value_2,
        ...,
        value_n);
```

```
INSERT INTO table_name
VALUES (
        value_1,
        value_2,
        ...,
        value_n);
```

* Si no especificamos las columnas, tendremos que insertar tantos valores como columnas hay en la tabla.

INSERT

También se puede usar **INSERT** para agregar nuevas filas a otra tabla. En este ejemplo, la tabla_2 toma las filas de la tabla_1 cuando se cumple cierta condición sobre la tabla_1.

```
INSERT INTO
        table_2 (column_1, column_2, ..., column_n)
SELECT
        column_1, column_2, ..., column_n
FROM
        table_1
WHERE
        conditions;
```

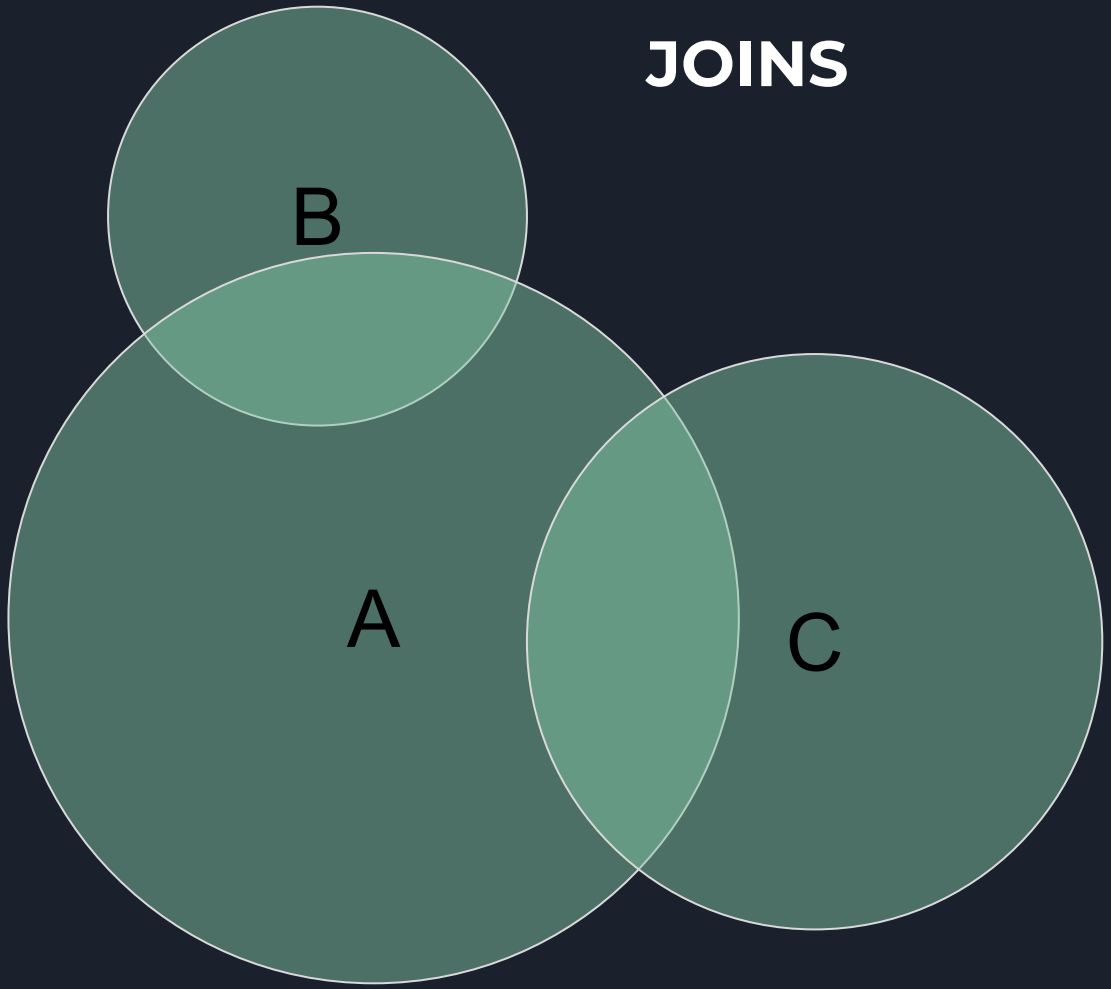
ROUND

ROUND: redondea los números de la columna, tiene como parámetro opcional la cantidad de decimales, si este parámetro se omite entonces redondea sin número decimal.

```
SELECT
    ROUND(column, n)
FROM
    table;
```

```
SELECT
    ROUND(AVG(edad)
FROM
    estudiantes_neoland;
```

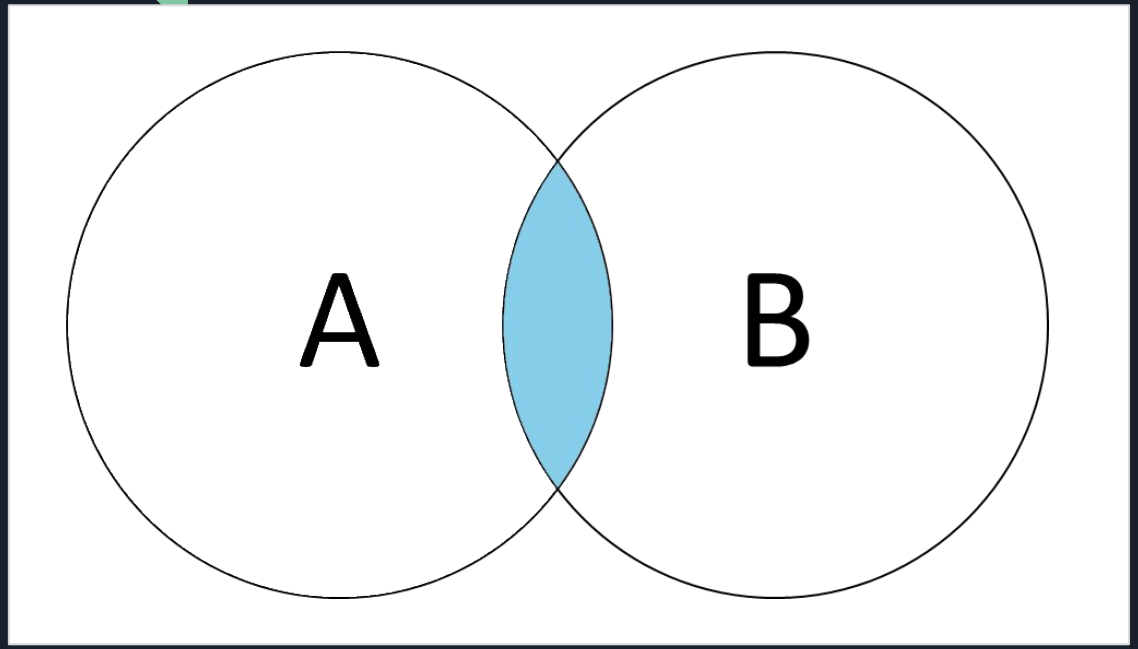
JOINS



JOIN

- JOIN: Es la herramienta que nos permitirá construir relaciones entre objetos.
- Para hacer JOIN entre 2 tablas debemos encontrar la columna que los relaciona.
- Cuando se haya hecho JOIN, podremos seleccionar columnas de ambas tablas.

INNER JOIN



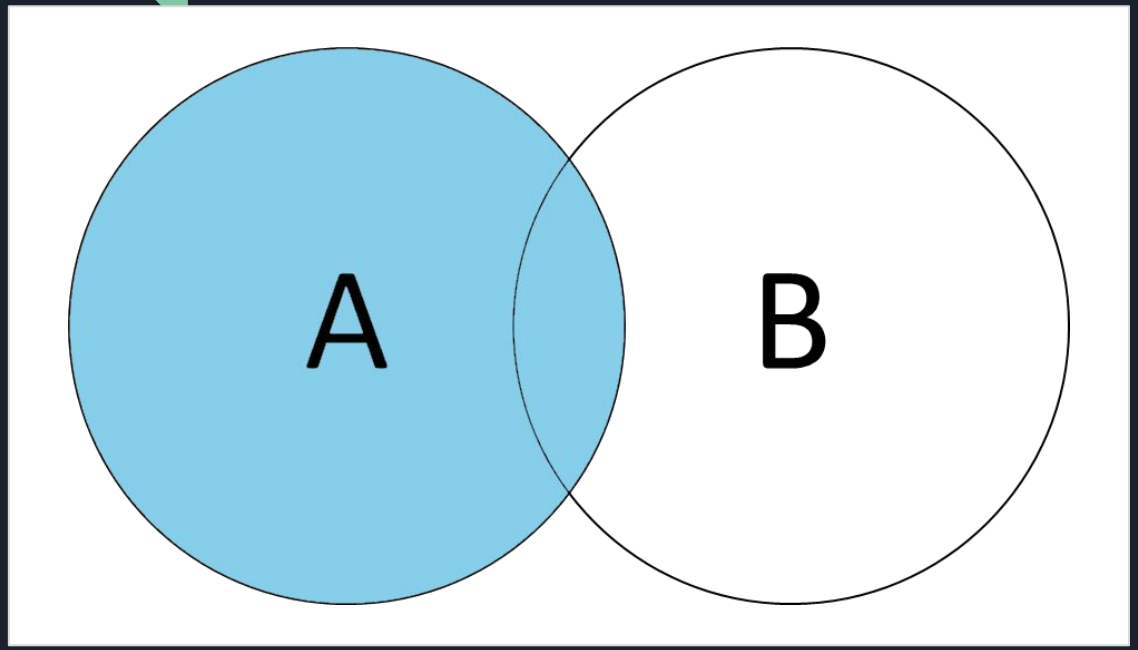
Inner joins extraerá solo las filas en la que los valores de la columna relacionada coincida.

Estos valores existen en ambas tablas, por lo que valores que solo existan en una de las tablas no se mostrarán.

INNER JOIN

```
SELECT
    t1.*,
    t2.*
FROM
    table_1 t1
    JOIN
    table_2 t2 ON t1.column_key = t2.column_key;
```

LEFT JOIN

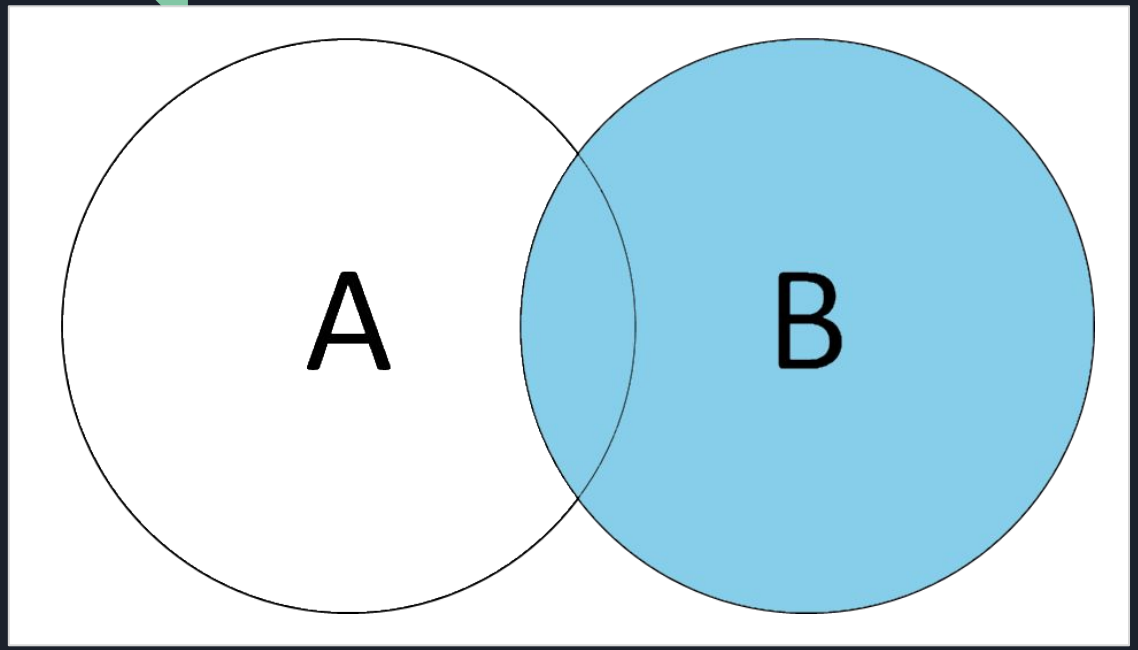


Left joins extraerá las filas del conjunto de la izquierda agregando las columnas del conjunto de la derecha, si existe la coincidencia de valores entonces los agrega a la fila, en caso contrario llena la fila con NULL.

LEFT JOIN

```
SELECT
    t1.*,
    t2.*
FROM
    table_1 t1
    LEFT JOIN
    table_2 t2 ON t1.column_key = t2.column_key;
```

RIGHT JOIN

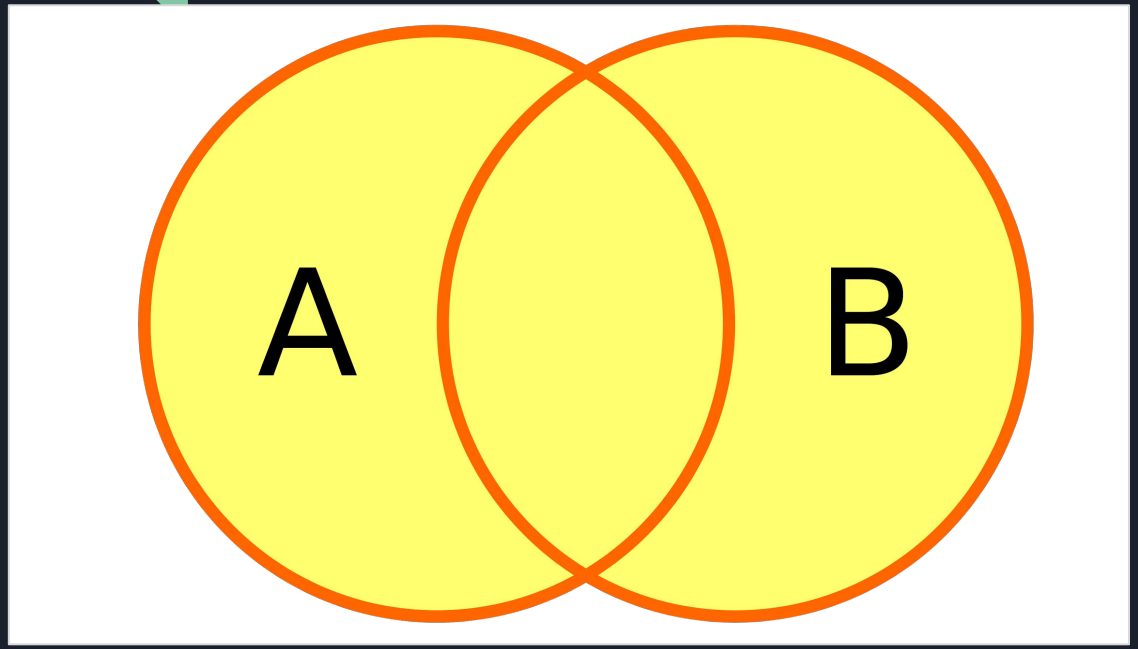


Right joins extraerá las filas del conjunto de la derecha agregando las columnas del conjunto de la izquierda, si existe la coincidencia de valores entonces los agrega a la fila, en caso contrario llena la fila con NULL.

RIGHT JOIN

```
SELECT
    t1.*,
    t2.*
FROM
    table_1 t1
    RIGHT JOIN
    table_2 t2 ON t1.column_key = t2.column_key;
```

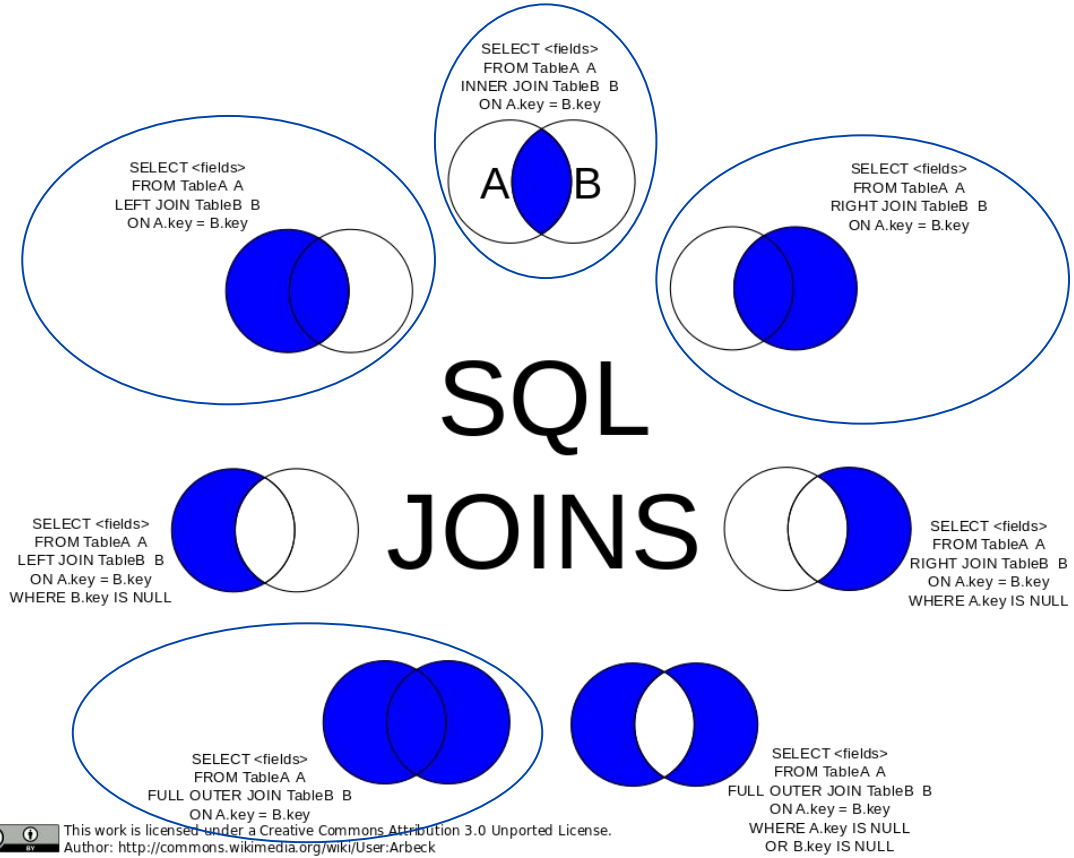
FULL JOIN



Full joins muestra todas las filas de ambas tablas, sin importar que hayan coincidencias en la columna clave. Al igual que LEFT y RIGHT JOIN llenará con NULL cuando no exista coincidencia.

FULL JOIN

```
SELECT
    t1.*,
    t2.*
FROM
    table_1 t1
    FULL JOIN
    table_2 t2 ON t1.column_key = t2.column_key;
```



LEFT JOIN II

```
SELECT
    t1.*,
    t2.*
FROM
    table_1 t1
    LEFT JOIN
    table_2 t2 ON t1.column_key = t2.column_key
WHERE t2.column_key IS NULL;
```

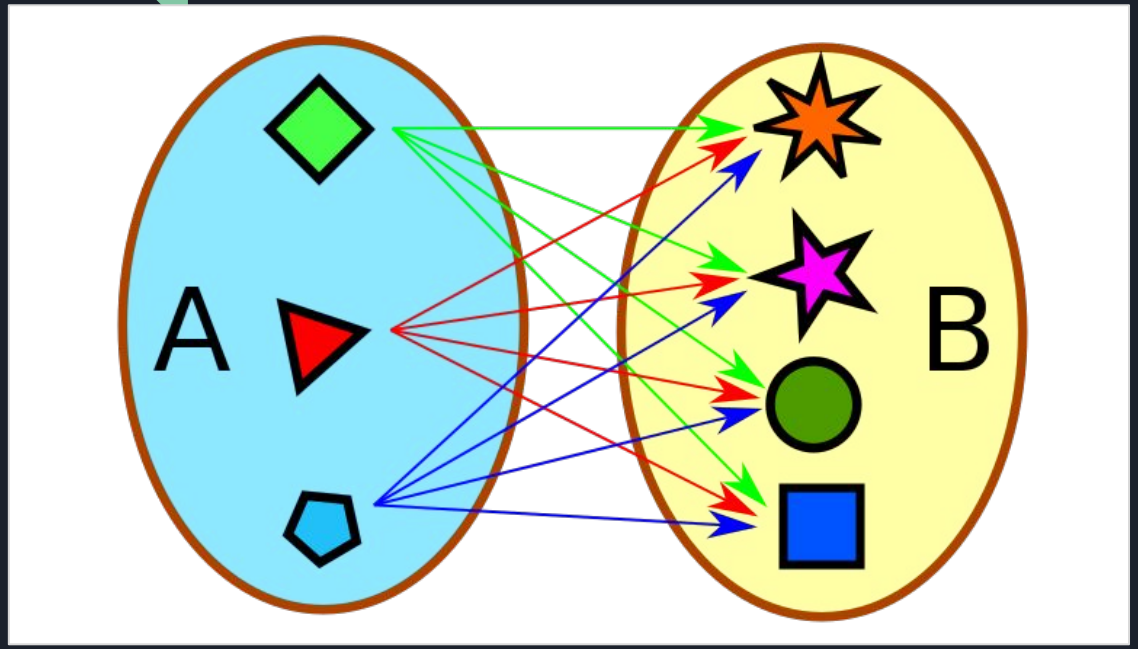
RIGHT JOIN II

```
SELECT
    t1.*,
    t2.*
FROM
    table_1 t1
    RIGHT JOIN
    table_2 t2 ON t1.column_key = t2.column_key
WHERE t1.column_key IS NULL;
```

FULL JOIN II

```
SELECT
    t1.*,
    t2.*
FROM
    table_1 t1
    FULL JOIN
    table_2 t2 ON t1.column_key = t2.column_key
WHERE t1.column_key IS NULL OR t2.column_key IS NULL;
```

CROSS JOIN



Cross joins retorna el resultado de multiplicar las filas de la tabla A con las filas de la tabla B, este tipo de resultado es llamado producto Cartesiano.

Este tipo de joins se usa muy poco.

CROSS JOIN

```
SELECT
    t1.*,
    t2.*
FROM
    table_1 t1
    CROSS JOIN
    table_2 t2 ON t1.column_key = t2.column_key
```

JOIN con más de 2 tablas

```
SELECT
    t1.*, t2.*, t3.*
FROM
    table_1 t1
    JOIN
    table_2 t2 ON t1.column_key_1 = t2.column_key_1
    JOIN
    table_3 t3 ON t2.column_key_2 = t3.column_key_2
    ...
```

UNION ALL

UNION ALL:

- Se usa para combinar SELECT statements en un mismo output.
- Para usarlo, debemos seleccionar el mismo número de columnas de cada tabla.
- Estas columnas deben tener el mismo nombre, deben estar en el mismo orden y deben de tener el mismo tipo de dato.
- En pandas, UNION ALL es el equivalente a pd.concat()
- Puede retornar filas duplicadas

UNION ALL

```
SELECT
    N columns
FROM
    table_1
UNION ALL SELECT
    N columns
FROM table_2;
```

UNION ALL

```
SELECT
        city_id,
        city,
        country_id,
        NULL AS country,
        last_update
FROM    city
UNION ALL SELECT
        NULL AS city_id,
        NULL AS city,
        country_id,
        country,
        last_update
FROM    country;
```

En este ejemplo, las tablas no tienen el mismo número de columnas, ni comparten los mismo nombres, pero con NULL AS podemos “crear” columnas ficticias** para lograr hacer el UNION ALL.

**Hacer trampa

UNION

UNION :

- Hace lo mismo que UNION ALL
- Se diferencian en que UNION NO retorna filas duplicadas
- Toma más tiempo para ejecutarse

UNION

```
SELECT
    N columns
FROM
    table_1
UNION SELECT
    N columns
FROM table_2;
```

UNION

```
SELECT
        city_id,
        city,
        country_id,
        NULL AS country,
        last_update
FROM    city
UNION SELECT
        NULL AS city_id,
        NULL AS city,
        country_id,
        country,
        last_update
FROM    country;
```

En este ejemplo, las tablas no tienen el mismo número de columnas, ni comparten los mismo nombres, pero con NULL AS podemos “crear” columnas ficticias** para lograr hacer el UNION ALL.

**Hacer trampax2