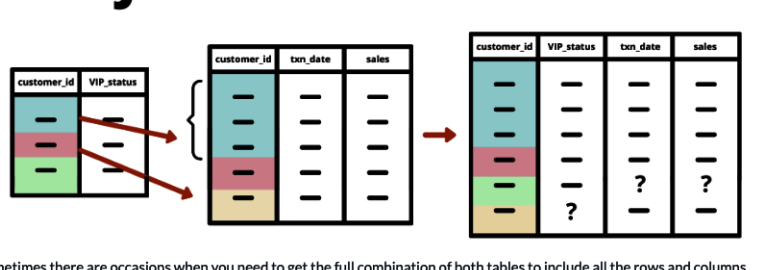
→ Temporary table + CTE

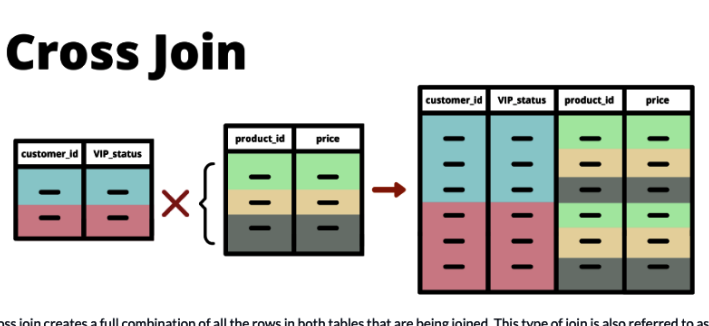
```
DROP TABLE IF EXISTS <table_name>;
CREATE TEMP TABLE <table_name> AS (

    WITH <cte_name> (col_1, col2, ..., coln) AS (

        VALUES
        (   ,   ,   ,   ),
        (   ,   ,   ,   ),
        (   ,   ,   ,   )

    )
)
```

Full Join



Sometimes there are occasions when you need to get the full combination of both tables to include all the rows and columns.

Cross Join



A cross join creates a full combination of all the rows in both tables that are being joined. This type of join is also referred to as a cartesian join which is taken from the original mathematical transformation - the cartesian product of two sets!

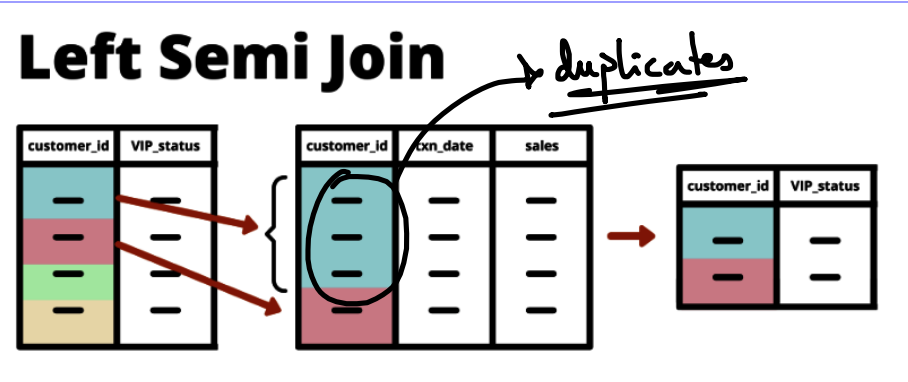
→ FROM t1 CROSS JOIN t2 / FROM t1, t2

- `LEFT SEMI JOIN` or `WHERE EXISTS`
- `ANTI JOIN` or `WHERE NOT EXISTS`

The main purpose of using these advanced joins is to remove/exclude rows from a table where specific column values based off a separate lookup table.

They also have the added benefit of helping eliminate some unexpected behaviour when there are duplicates in the "target" table or in join operations.

Left Semi Join



A left semi join is actually really similar to an `INNER JOIN` where it captures only the matching records between left and right tables BUT it differs in one very key way: it only returns records from the left table - no columns or rows from the right table are included in the output.

→ Target table acts as a filter

Syntax:

ORDER OF TABLES IMPORTANT

```
SELECT
  ~~
FROM <base_table>
WHERE EXISTS (
  SELECT 1
  FROM <target_table>
  WHERE ~~
);
```

Doesn't Return any data (→ SELECT 1)

```
SELECT
  ~
FROM <base_table>
LEFT SEMI JOIN <target_table>
ON ~~
```

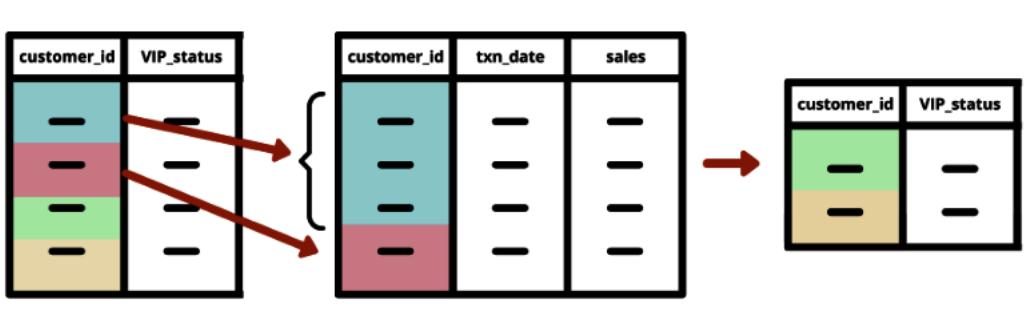
→ Depending on the columns chosen Distinct may not remove duplicates

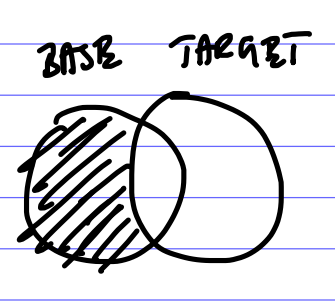
```
SELECT DISTINCT
  base_table. ~~
FROM <base_table>
LEFT JOIN <target_tables>
ON ~~
WHERE target_table. ~~ IS NOT NULL ;
```

INEFFICIENT WHEN WE WANT ONLY RESULTS FROM BASE TABLE.

```
SELECT DISTINCT
  base_table. ~~
FROM <base_table>
INNER JOIN <target_table>
ON ~~
```

Anti Join





An `ANTI JOIN` is the opposite of a `LEFT SEMI JOIN` where only records from the left which do not appear on the right are returned - it is expressed as a `WHERE NOT EXISTS` clause in PostgreSQL but similar to the `LEFT SEMI JOIN` some SQL flavours support a direct `ANTI JOIN` syntax also.

Syntax:

```
SELECT
  ~~
FROM <base_table>
WHERE NOT EXISTS (
  SELECT 1
  FROM <target_table>
  WHERE ~~
);
```

OR

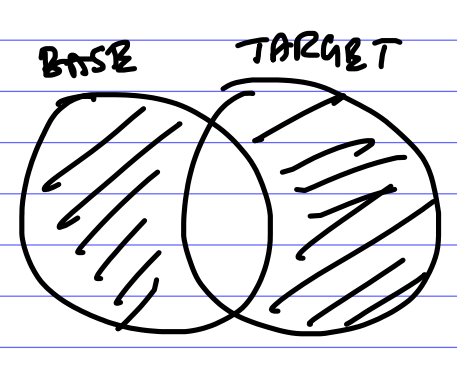
```
SELECT
  ~
FROM <base_table>
ANTI JOIN <target_table>
ON ~~
```

← MORE EFFICIENT

```
SELECT
  ~
FROM <base_table>
LEFT JOIN <target_table>
ON ~~
WHERE <target_table>. ~~ IS NULL;
```

* If we are joining tables which have NULL in the JOINING COLUMN THE RESULTS DON'T ACT AS EXPECTED.

NULL ≠ NULL




```
SELECT
  5 > null AS col1,
  5 < null AS col2,
  5 != null AS col3,
  5 IN (null) AS col4,      -- this is our unknown value
  5 NOT IN (null) AS col5,  -- this is our unknown value
  5 IS NULL AS col6,
  5 IS NOT NULL AS col7;
```

col1	col2	col3	col4	col5	col6	col7
null	null	null	null	null	false	true

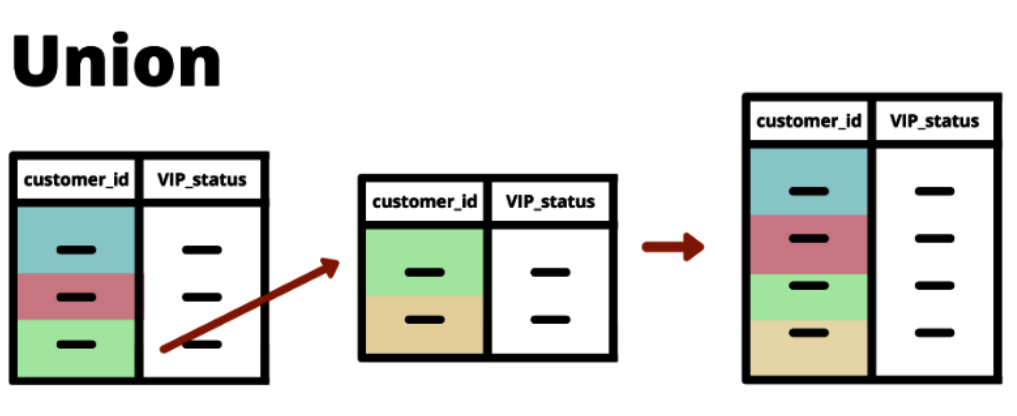
```
1 SELECT
2     NULL IN (NULL) AS col1,
3     NULL NOT IN (NULL) AS col2;
```

col1	col2
NULL	NULL

To keep things simple and less prone to issues - try to always stick with `LEFT SEMI JOIN` and `ANTI JOIN` instead of `IN` and `NOT IN` where you do not know what exact values will appear for the `IN` subquery.

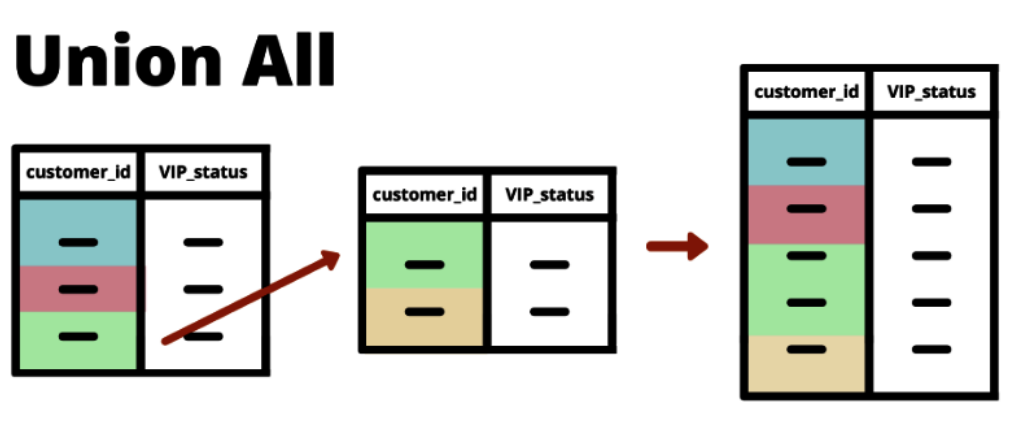
The behaviour when `null` values are present in the subquery for the filtering is not always stable and may lead to incorrect values!

Union

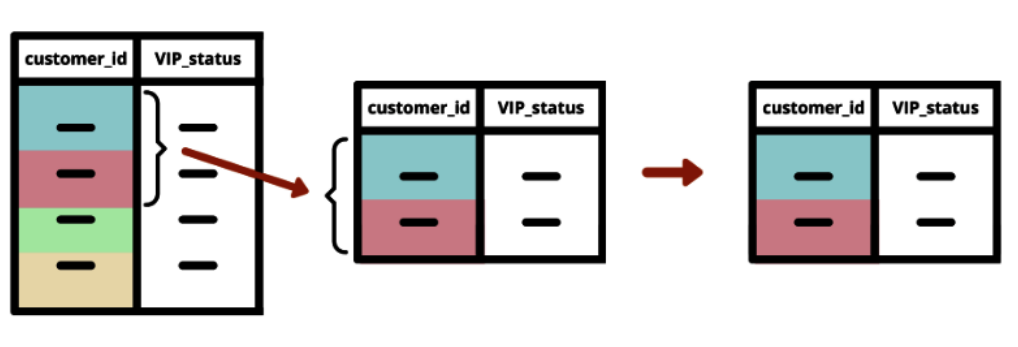


Simply put - `UNION` will be the union between two sets or `SELECT` results.

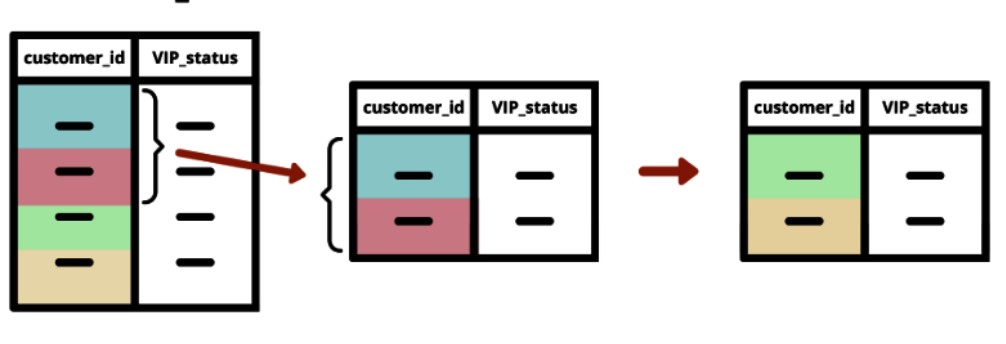
Union All



Intersect



Except



- =, >=, <=, >, < or `BETWEEN` conditions can be used for the `ON` clause for table joins

```
→ UPDATE <table>
  SET
    <col> = ____ ,
    <col> = ____
  WHERE ~~
  RETURNING *
```

(RETURNING *) → shows all updated rows as query output

→ Familiarizing with new database using the information_schema

- we can check datatypes using information_schema.columns table

```
Select
    table_name
    column_name
    data_type
From information_schema.columns
Where table_name in ( ~~ );
```

High lvl overview of the dataset we will be working with

```
Select
    schema_name,
    table_name,
    column_name
from information_schema.columns
Where column_name ILIKE '___';
```