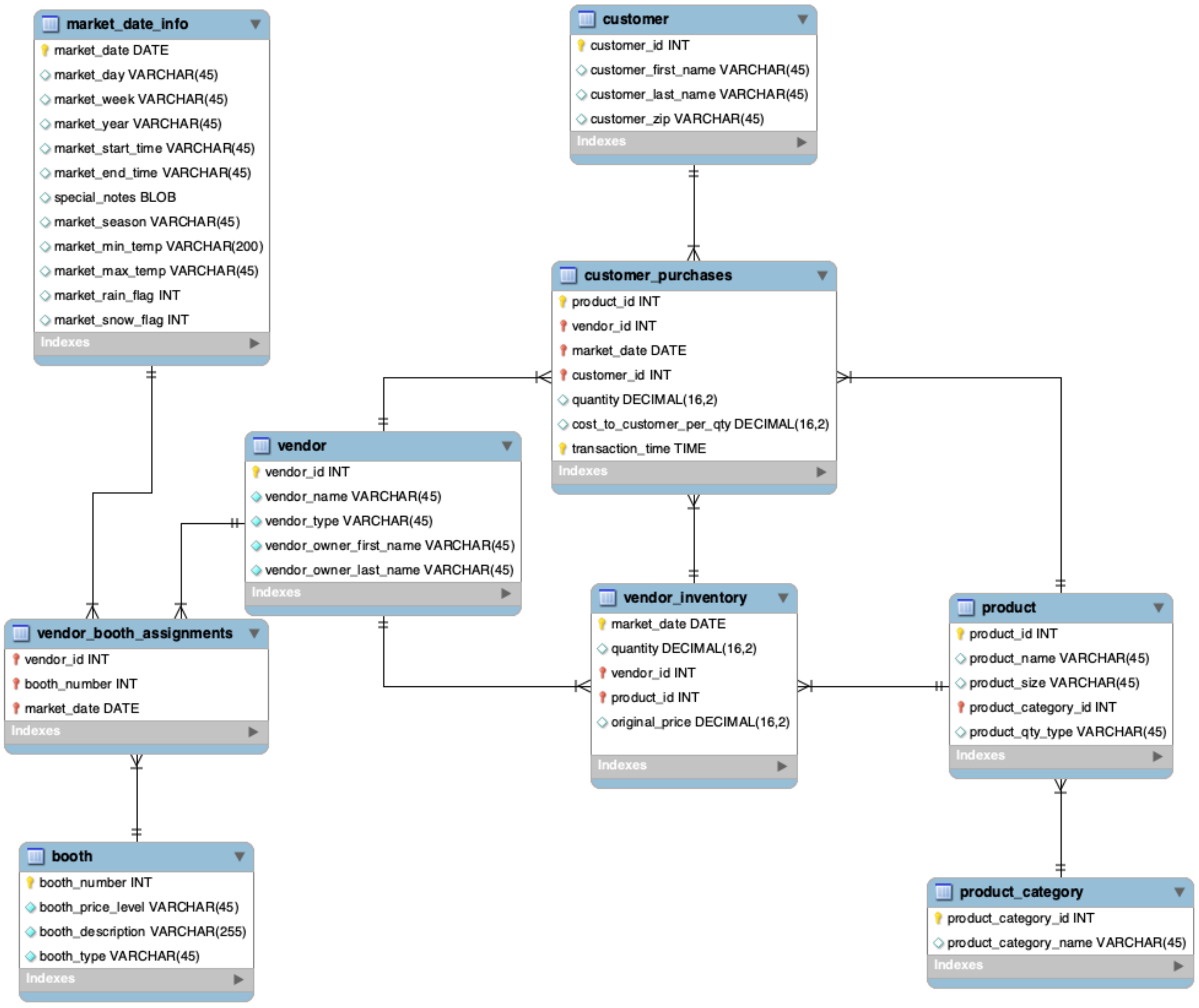
Joins

Problem Statement:

You are a Data Analyst at Amazon Fresh. You have been tasked to study the Farmer's Market.

Dataset: Farmer's Market database



So far...

- You have learned to select data from **a single database table** and filter to the rows you want.
- But you might wonder what to do if the data you need exists across multiple related tables in the database.

Question: Get details of all vendors selling products along with the name of each product they sell and the quantity of that product present in their inventory?

Intuition:

Now, what tables do we require?

1. to get the vendor details like vendor name & type - the ***vendor*** table.
2. to get the details about each specific item, including product name & size - the ***product*** table.
3. to find out the quantity of the product for each vendor - the ***vendor_inventory*** table.

So you need all this combined information from 3 tables here.

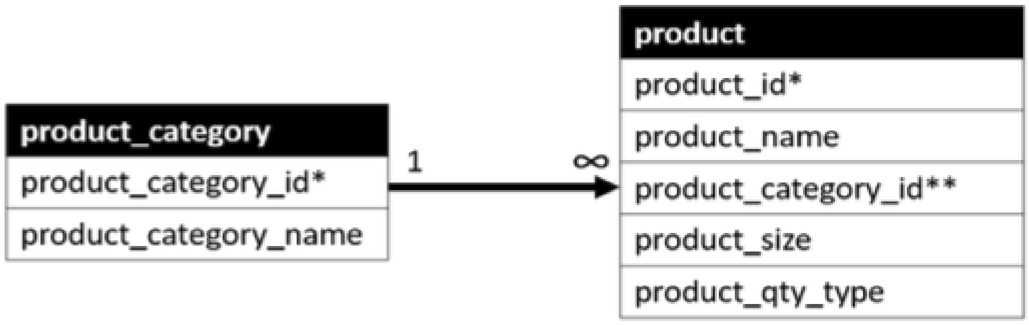
This is where **SQL JOINs** come in.

Note that ER Diagrams are crucial in identifying which tables we can join and which key fields connect them.

Question: List all the products along with their product category name.

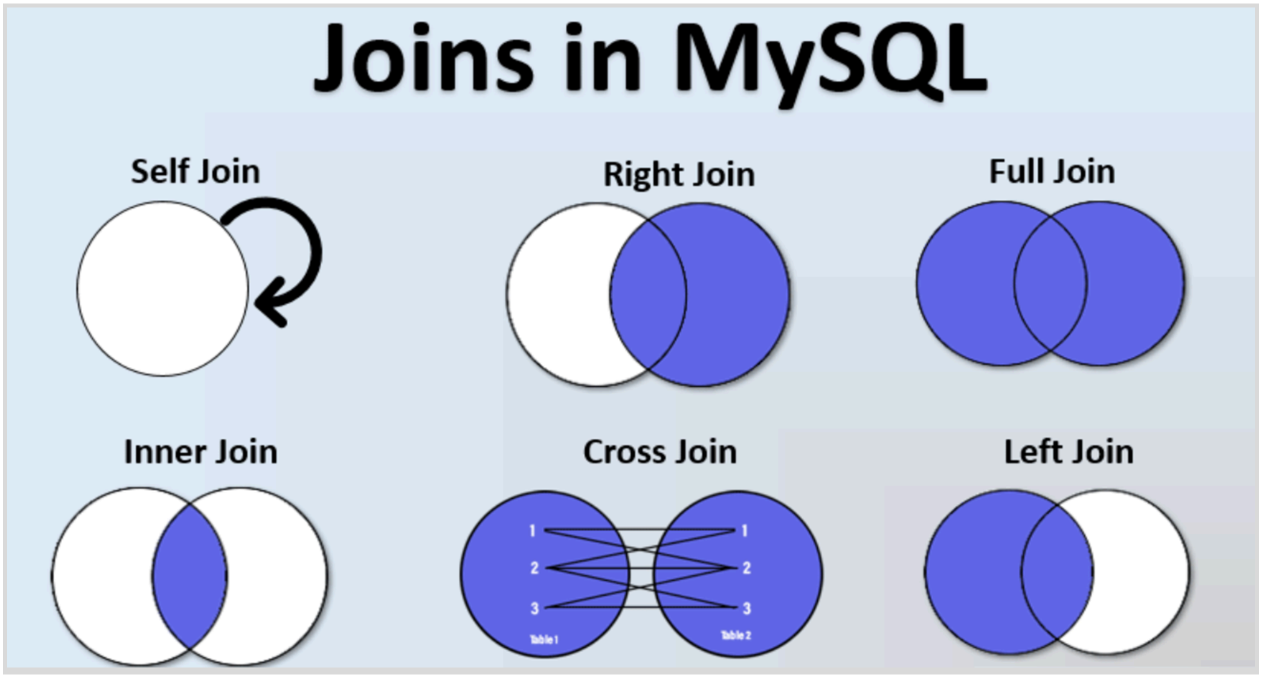
Since only the ID of the product category exists in the ***product*** table, and the product category's name is in the ***product_category*** table,

We have to combine the data in the product and product_category tables together to generate this list.



- The figure shows the one-to-many relationship between these tables.
- Their primary keys are each identified with an asterisk and the foreign key with a double asterisk.
- Each row in the ***product_category*** table can be associated with many rows in the product table, but each row in the ***product*** table is associated with only one row in the ***product_category*** table.
- The fields that connect the two tables are *product_category.product_ category_id* and *product.product_category_id*.

Now, there are multiple ways of joining these tables.



Syntax for joining tables:

```
SELECT [columns to return]
FROM [left table]
[JOIN TYPE] [right table]
ON [left table].[field in left table to match] = [right table].[field in
right table to match]
```

Order of Execution of a SQL query:

- **FROM** - The database gets the data from tables in FROM clause and if necessary, performs the JOINs.
- **JOIN** - Depending on the type of JOIN used in the query and conditions specified for joining the tables in the **ON** clause, the database engine matches rows from the virtual table created in the FROM clause.
- **WHERE** - After the JOIN operation, the data is filtered based on the conditions specified in the WHERE clause. Rows that do not meet the criteria are excluded.

- **GROUP BY** - If the query includes a GROUP BY clause, the rows are grouped based on the specified columns and aggregate functions are applied to the groups created.
- **HAVING** - The HAVING clause filters the groups of rows based on the specified conditions
- **SELECT** - After grouping and filtering is done, the SELECT statement determines which columns to include in the final result set.
- **ORDER BY** - It allows you to sort the result set based on one or more columns, either in ascending or descending order.
- **OFFSET** - The specified number of rows are skipped from the beginning of the result set.
- **LIMIT** - After skipping the rows, the LIMIT clause is applied to restrict the number of rows returned.

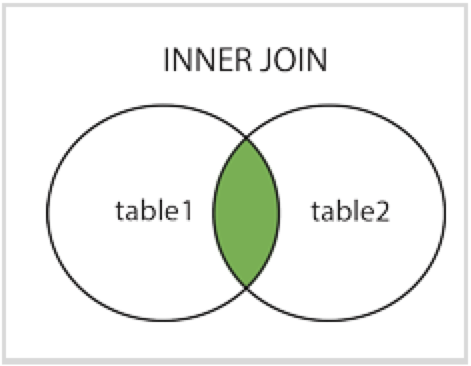
Question: Get a list of customers' zip codes for customers who made a purchase on 2019-04-06.

Will you need a join here?

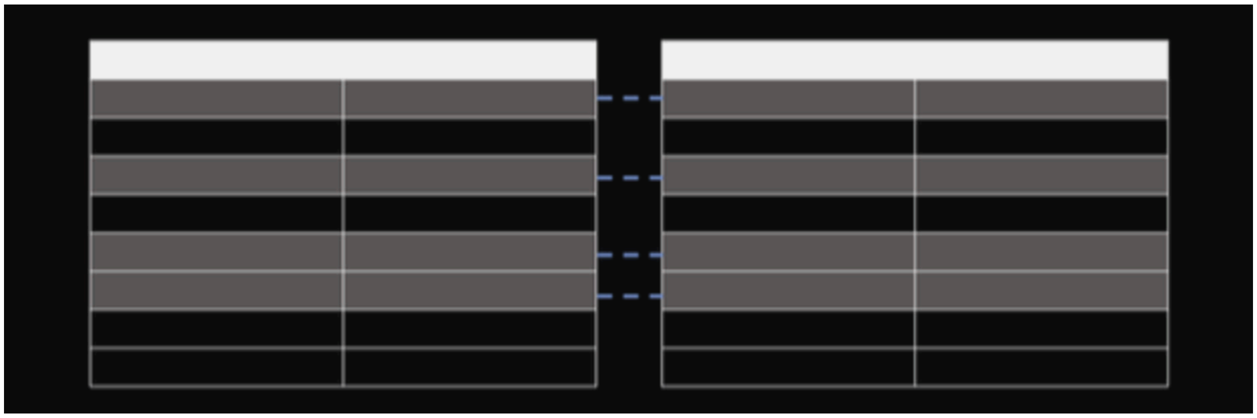
What type of join are we going to use here?

If we need zip codes of all the customers who made a purchase, we only require an **intersection of customers** whose details are present in the customer tables.

Inner JOIN



An INNER JOIN only returns records that have matches in both tables.



Query:

```
SELECT
        DISTINCT cp.customer_id,
        c.customer_zip
FROM farmers_market.customer c
INNER JOIN farmers_market.customer_purchases cp
ON cp.customer_id = c.customer_id
WHERE
        cp.market_date='2019-04-06';
```

Breaking down **Inner JOIN** :

- An inner join is a type of join in SQL that returns only the rows from both tables that have matching values in the specified columns.

- In this case, the inner join is performed on the "customer" and "customer_purchases" tables using the "customer_id" column as the matching column.

- The "INNER JOIN" clause specifies that we want to retrieve only the rows that have matching values in both tables. In other words, we only want to retrieve the details of customers who have made a purchase on the specified date.

- The "ON" clause specifies the condition for the join. In this case, we want to match the "customer_id" column in both tables. By joining on this column, we can link each purchase to the corresponding customer.

- Once the join is performed, we can retrieve the zip codes of the customers who made a purchase on the specified date by selecting the "customer_zip" column from the "customer" table.
- The "DISTINCT" keyword is used to ensure that we only get one row per customer, even if they made multiple purchases on the specified date.

USING keyword

The USING keyword simplifies the join syntax when you want to join tables based on a common column name.

It specifies which column should be used in the join condition.

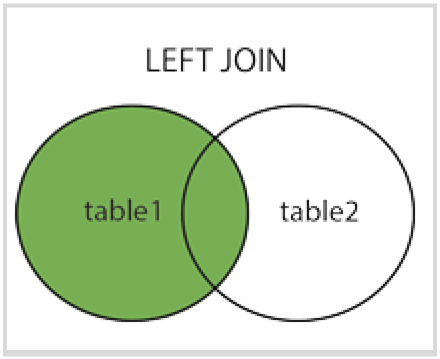
The above query can also be written in the following manner -

```
SELECT
        DISTINCT cp.customer_id,
        c.customer_zip
FROM farmers_market.customer c
INNER JOIN farmers_market.customer_purchases cp
USING (customer_id)
WHERE
        cp.market_date='2019-04-06';
```

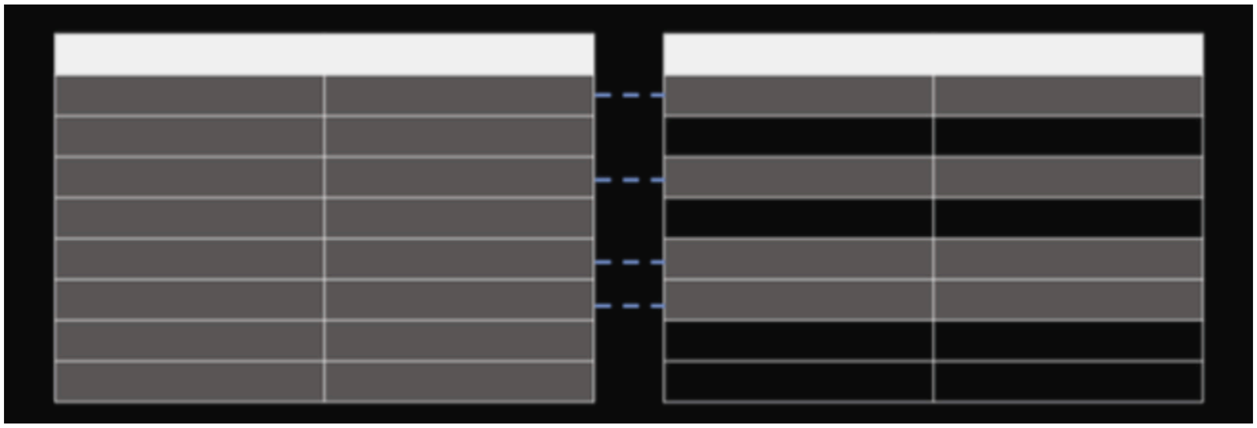
Please note that the USING keyword can only be used when joining columns with the same name in both tables, and it's typically used for inner joins and natural joins.

If the columns have different names or if you need to perform more complex joins, you should stick to the ON keyword to specify the join condition explicitly.

Left JOIN



This tells the DBMS to pull all records from the table on the "left side" of the JOIN, and only the matching records (based on the criteria specified in the JOIN clause) from the table on the "right side" of the JOIN.



As per our question,

Question: List all the products along with their product category name.

Ques. Which table should we use as the left table if we use LEFT JOIN?
Ans. The product table should be on the left and product_categories should be on the right.

Actual Query:

```
SELECT * FROM
farmers_market.product
LEFT JOIN farmers_market.product_category
        ON product.product_category_id =
product_category.product_category_id
```

NOTE: You may have noticed two columns called **product_category_id** in the output.

That is because we selected all fields using the asterisk(*), and there are fields in both tables with the same name.

To remedy this, **we could either specify the list of fields to be returned** and only include the **product_category_id** from one of the tables or **alias the column names to indicate which table each came from**.

Query:

```
SELECT
   p.product_id,
   p.product_name,
    pc.product_category_id,
    pc.product_category_name
FROM farmers_market.product AS p
LEFT JOIN farmers_market.product_category AS pc
ON p.product_category_id = pc.product_category_id
ORDER BY pc.product_category_name, p.product_name;
```

Breaking down **Left JOIN** :

- The Left JOIN indicates that we want all rows from the **product** table (which is listed on the left side of the JOIN keyword) and

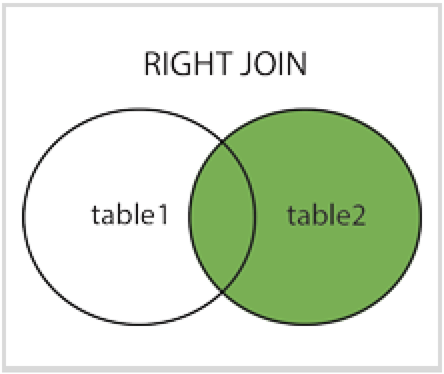
- only the associated rows from the **product_category** table. So, if a category is not associated with any products, it will not be included in the results.

- If a product were without a category, it would be included in the results, with the fields on the **product_category** side being NULL.

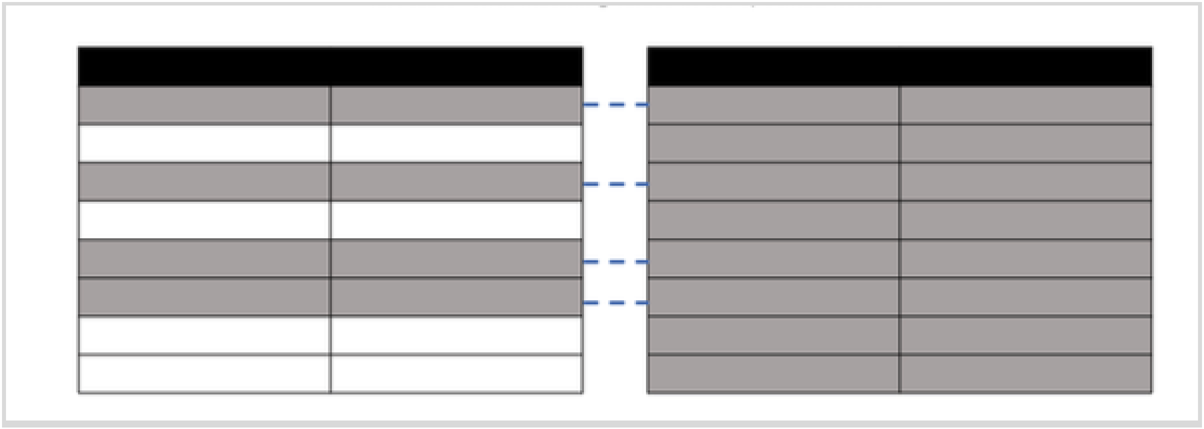
- The ON part of the JOIN clause tells the query to match up the rows in the two tables using each table's values in the product_category_id field.

- We can specify which table each column is from since it’s possible to have identically named columns in different tables.

Right JOIN



In a RIGHT JOIN, all of the rows from the “right table” are returned, along with only the matching rows from the “left table,” using the fields specified in the ON part of the query.



To write the same query (the one we saw earlier) using a RIGHT JOIN, you can simply reverse the order of the tables and use a RIGHT JOIN instead of a LEFT JOIN.

Query:

```
SELECT
  p.product_id,
  p.product_name,
  pc.product_category_id,
  pc.product_category_name
```

```
FROM farmers_market.product_category AS pc
RIGHT JOIN farmers_market.product AS p
ON p.product_category_id = pc.product_category_id
ORDER BY pc.product_category_name, p.product_name;
```

- In this query, the "product_category" table is on the left side of the RIGHT JOIN, and the "product" table is on the right side.
- The rest of the query remains the same as in your original LEFT JOIN query.
- This RIGHT JOIN query will return all product categories, including those without associated products, and it will display products when they have a matching category.

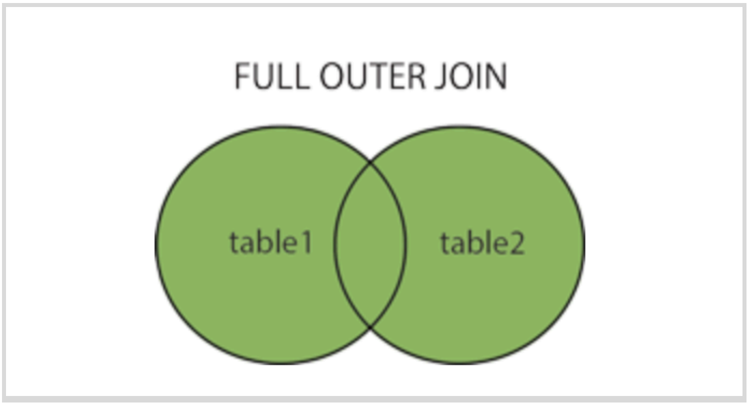
Question: Find out the customers who are either new to the market or have deleted their account from the market.

What information is required to answer this?

1. For customers who are new to the market, we'd need the **`customer`** table because that's where the new customers are (those who haven't made any purchase yet).

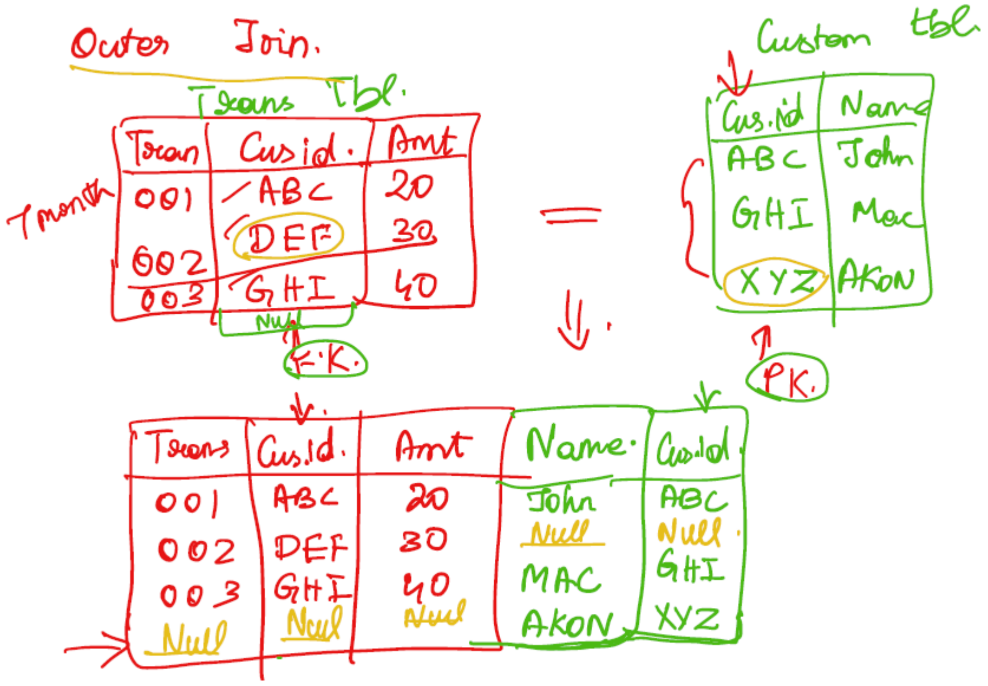
2. For the customers that have left, they can only be found in the purchase history i.e. the **`customer_purchases`** table as their records are deleted from the `customer` table.

Full Outer Join



A full outer join returns all rows from both the tables whether there is a match in left (table1) or right (table2) table records.

Example -



Trans Tbl.

Tran	Cus id.	Amt
001	ABC	20
002	DEF	30
003	GHI	40

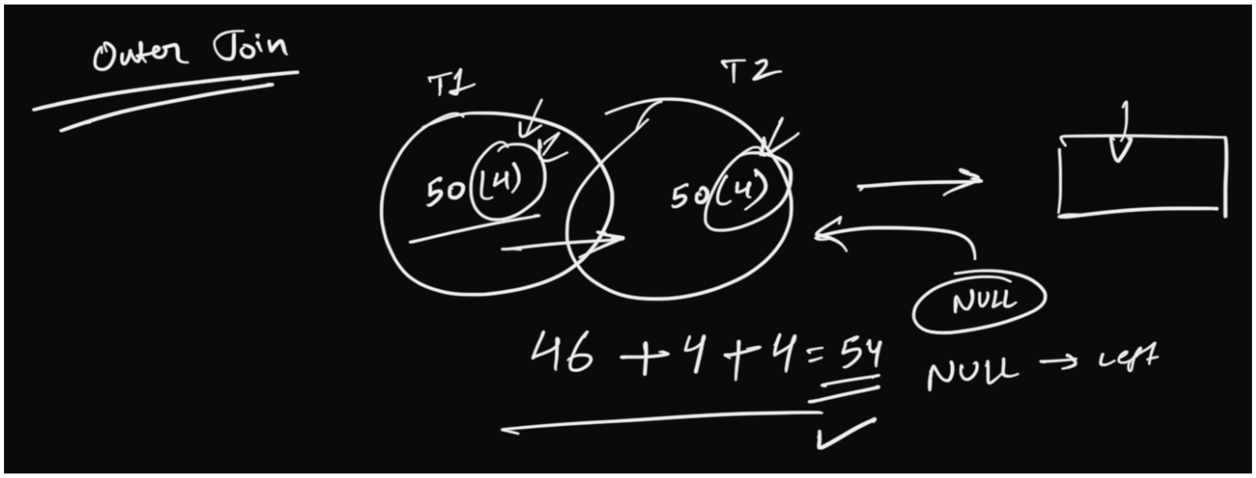
Custom tbl

Cus.id	Name
ABC	John
GHI	Mac
XYZ	AKON

Trans	Cus.Id.	Amt	Name.	Cus.id
001	ABC	20	John	ABC
002	DEF	30	Null	Null
003	GHI	40	MAC	GHI
Null	Null	Null	AKON	XYZ

Suppose you have two tables, T1 and T2, both with 50 records each.

When we try to match these two tables, we find that there are 4 mismatching rows.



How many rows are we gonna get after joining these two tables using a full outer join?
Ans. 54

Important:

- MySQL does not support FULL OUTER JOIN, so you have to combine LEFT JOIN, **UNION** and RIGHT JOIN to get an equivalent.
- It gives the results of A union B. It returns all records from both tables. Those columns which exist in only one table will contain NULL in the opposite table.

Let's learn about another important clause, **UNION**.

- Using a UNION, you can combine any two queries that result in the **same number of columns** with the **same data types**.
- The columns must be in the **same order** in both queries.

Example: If you want to look at revenue from all the different types of sales - Uber Cabs, Uber Eats. How will you get that?

How to do it in MySQL:

```
SELECT *
FROM A
LEFT JOIN B
        ON A.id = B.id
UNION
SELECT *
FROM A
        RIGHT JOIN B
ON A.id = B.id;
```

How to use UNION in BigQuery:

- UNION is only supported in MySQL, not in BigQuery.
- For BigQuery you'll have to use **UNION_DISTINCT**.
- Note that UNION_DISTINCT gets rid of any duplicate records.
- If you wish to include the duplicates, use **UNION_ALL** instead.
