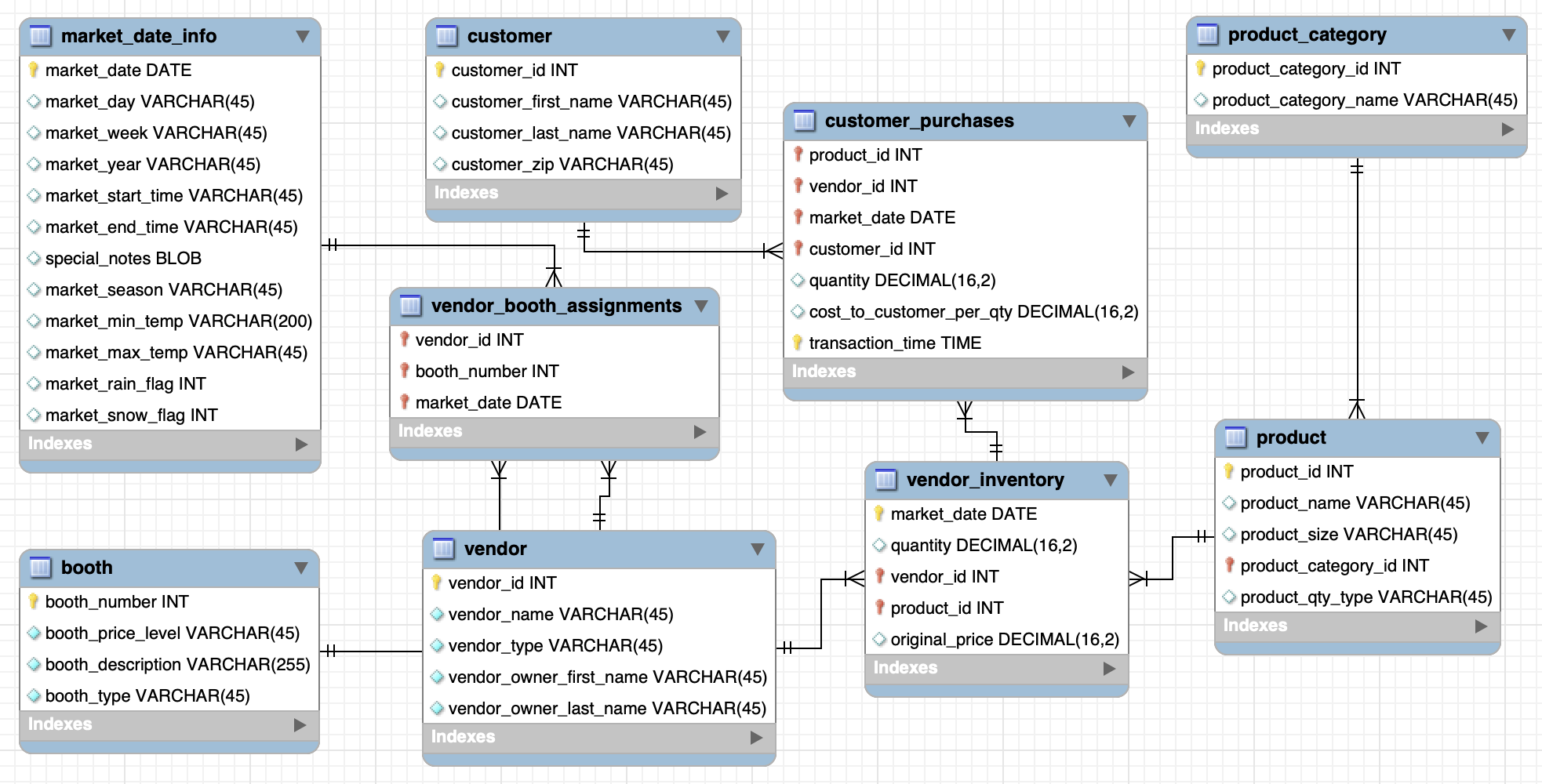
Joins

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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.

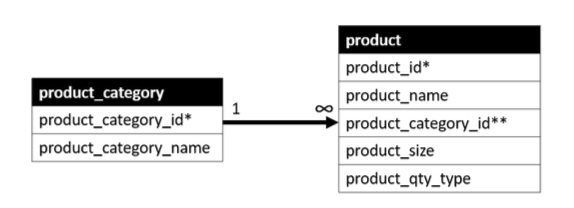
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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,

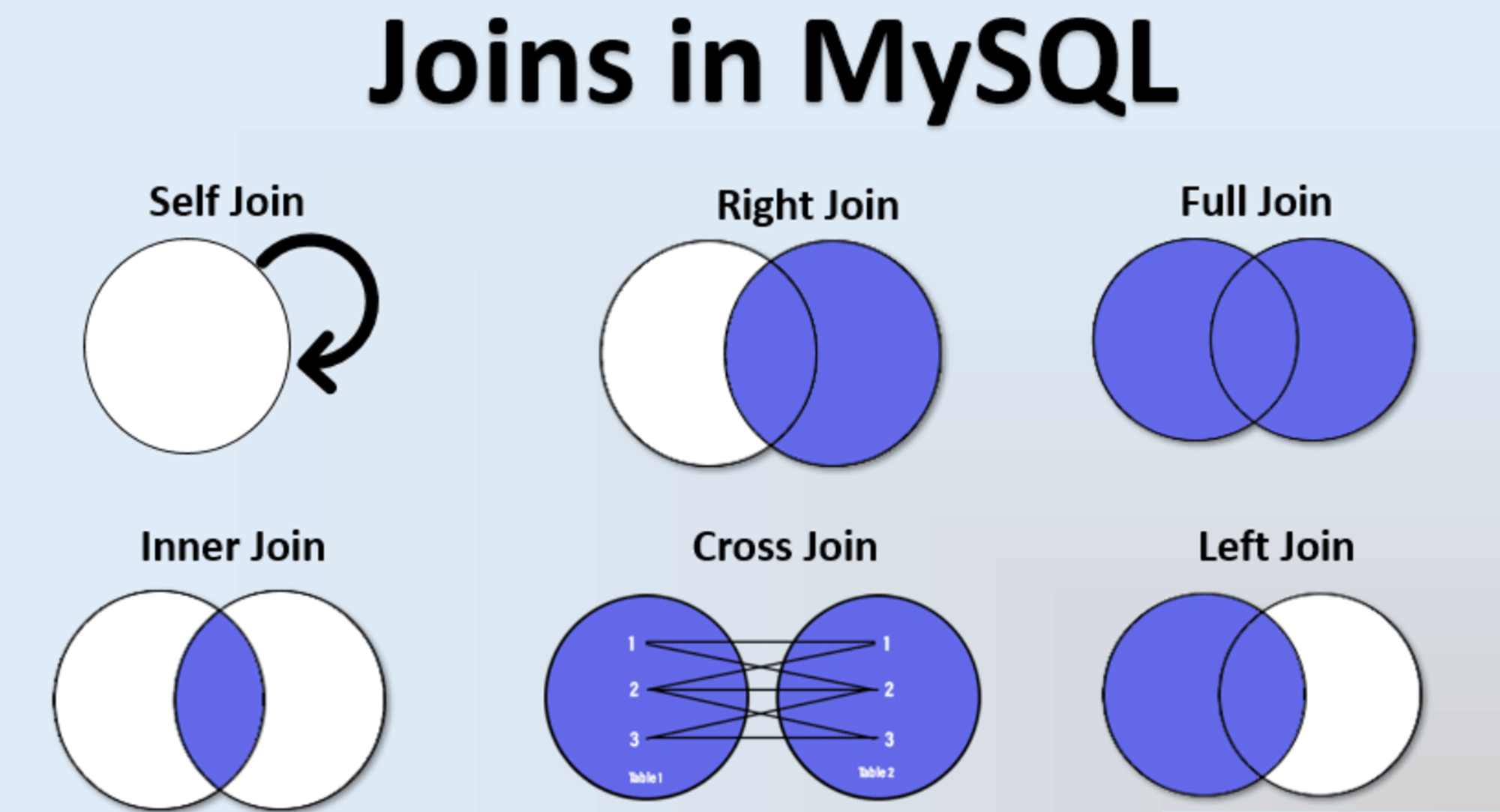
We have to combine the data in the product and product\_category tables together to generate this list.

****

* 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.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

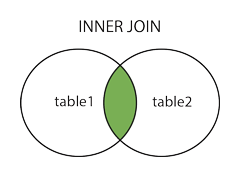
## Question: Get a list of customers' zip codes for customers who made a purchase on 2019-04-06.

Will you need a join here?

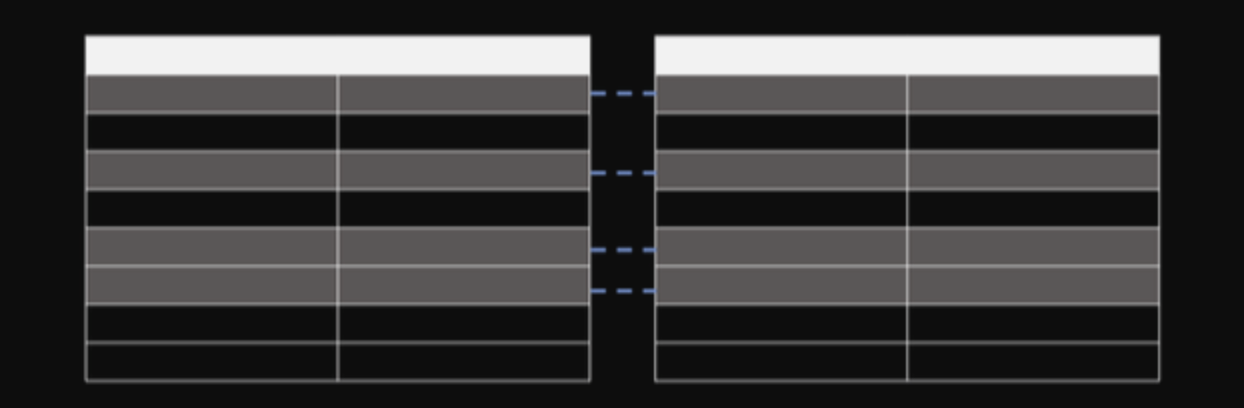
What type of join are we going to use here?

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

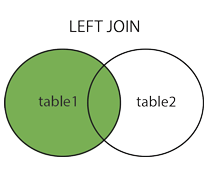
cp.market\_date='2019-04-06';

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.

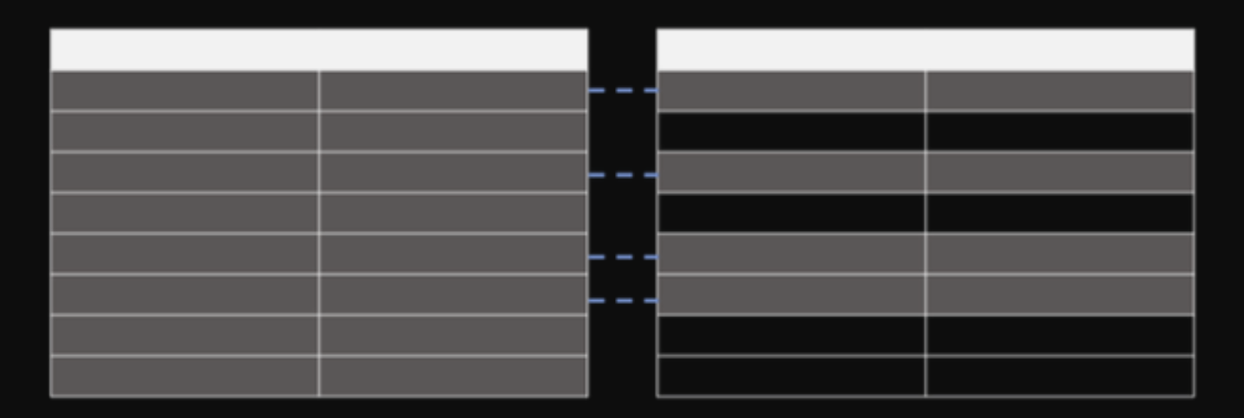
If the columns have different names or if you need to perform more complex joins, you should stick to the ON keyword to specify the join condition explicitly.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**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

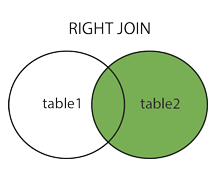
ORDER BY pc.product\_category\_name, p.product\_name;

Breaking down **Left JOIN** :

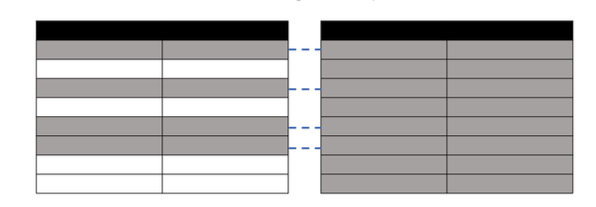
* The Left JOIN indicates that we want all rows from the **product** table (which is listed on the left side of the JOIN keyword) and
* only the associated rows from the **product\_category** table. So, if a category is not associated with any products, it will not be included in the results.
* If a 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, 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?

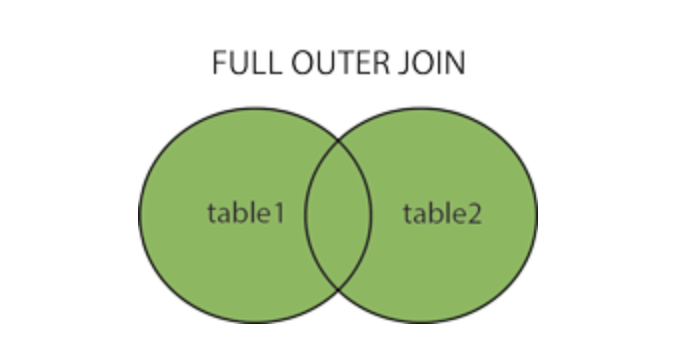
## For customers who are new to the market, we’d need the **`customer`** table because that’s where the new customers are (those who haven’t made any purchase yet).

## 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.

# 

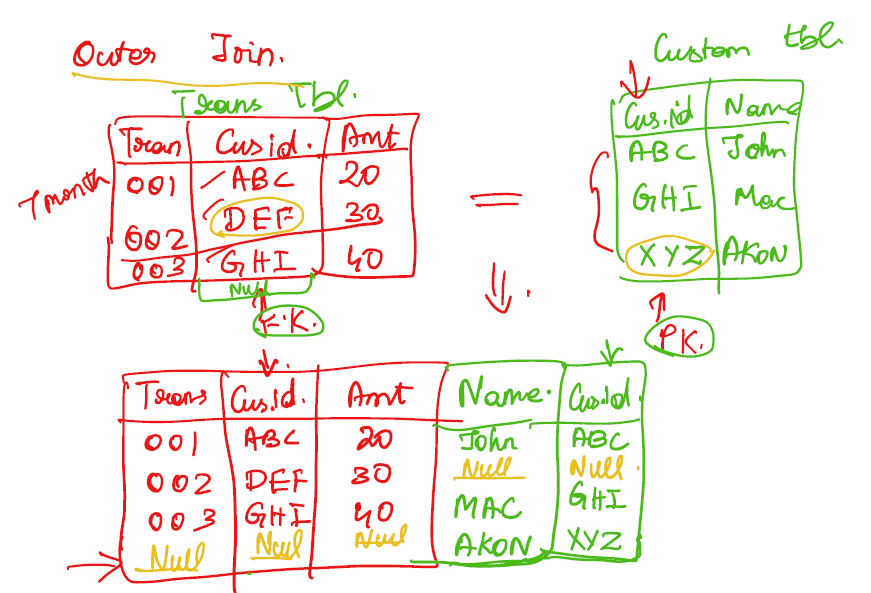
# 

# **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.

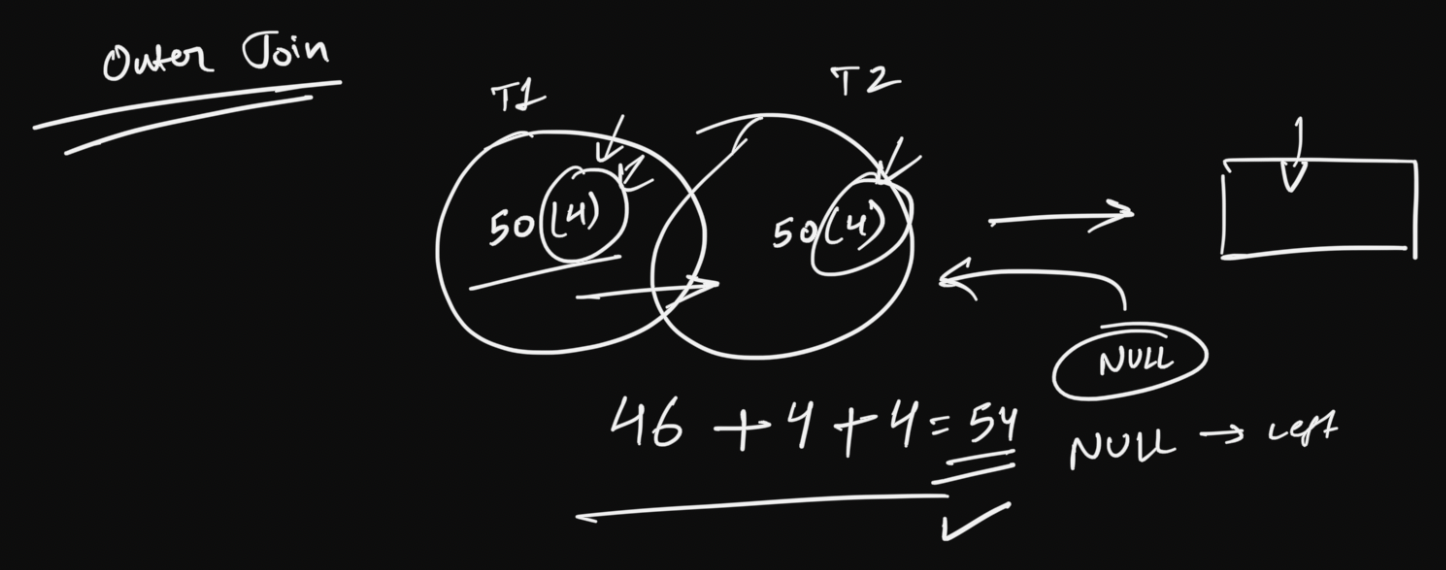
**Example -**



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Q1. Get all the customers who haven’t purchased anything from the market yet.

What type of JOIN should we use here?

Answer: Left JOIN

**Query:**

SELECT \*

FROM farmers\_market.customer AS c

LEFT JOIN farmers\_market.customer\_purchases AS cp

ON c.customer\_id = cp.customer\_id;

* There can be customers without any purchases.
* The customer table has details of all the customers regardless of their purchase history.
* Since we did a LEFT JOIN, we’re getting a list of all customers, and their associated purchases, if there are any.
* Customers with multiple purchases will show up in the output multiple times for each item purchased.
* Customers without purchases will have NULL values in all fields displayed from the customer\_purchases table.

**To get the list of customers that did not purchase anything.**

**Query 1:**

SELECT c.\* -- select columns from customer table only

FROM customer AS c

LEFT JOIN customer\_purchases AS cp

ON c.customer\_id = cp.customer\_id

WHERE cp.customer\_id IS NULL;

* Here we only selected columns from the customer table, using c.\*,
* because all of the columns on the customer\_purchases side of the relationship will be NULL (since we’re filtering to NULL customer\_id, and there are no purchases in the customer\_purchases table without a customer\_id, since it is a required field).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Q2. Get all the customers who have deleted their account from the market.

What type of JOIN should we use here?

Answer: Right JOIN

**Query:**

SELECT \*

FROM farmers\_market.customer AS c

RIGHT JOIN farmers\_market.customer\_purchases AS cp

ON c.customer\_id = cp.customer\_id

**To get the list of customers who deleted their account.**

**Query 2:**

SELECT cp.\* -- select columns from customer\_purchases table

FROM customer AS c

RIGHT JOIN customer\_purchases AS cp

ON c.customer\_id = cp.customer\_id

WHERE c.customer\_id IS NULL;

* Here we only selected columns from the customer\_purchases table, using cp.\*,
* because all of the columns on the customer side of the relationship will be NULL (since we’re filtering to NULL customer\_id, and there are no customers in the customers table without a customer\_id).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Now let’s combine these two queries i.e. Query 1 & Query 2, to obtain the final solution query.

**Soln using UNION -**

SELECT c.customer\_id,

"New Customer" AS customer\_type

FROM farmers\_market.customers AS c

LEFT JOIN farmers\_market.customer\_purchases AS cp

ON c.customer\_id = cp.customer\_id

WHERE cp.customer\_id IS NULL

UNION DISTINCT

SELECT cp.customer\_id,

"Deleted Customer" AS customer\_type

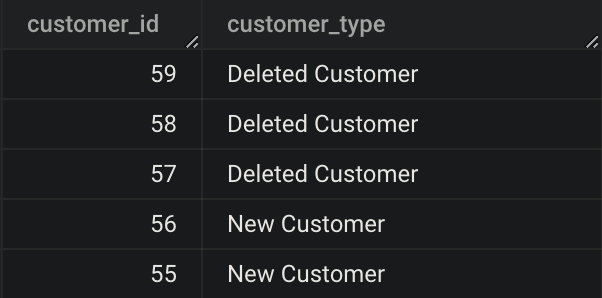
FROM farmers\_market.customers AS c

RIGHT JOIN farmers\_market.customer\_purchases AS cp

ON c.customer\_id = cp.customer\_id

WHERE c.customer\_id IS NULL;

**Output:**



Now let’s also try to write the solution query using a Full Outer Join

**Soln using Full Outer Join -**

SELECT

c.customer\_id AS new\_customer,

cp.customer\_id AS deleted\_customer

FROM farmers\_market.customers AS c

FULL OUTER JOIN

farmers\_market.customer\_purchases AS cp

ON c.customer\_id = cp.customer\_id

WHERE

c.customer\_id IS NULL

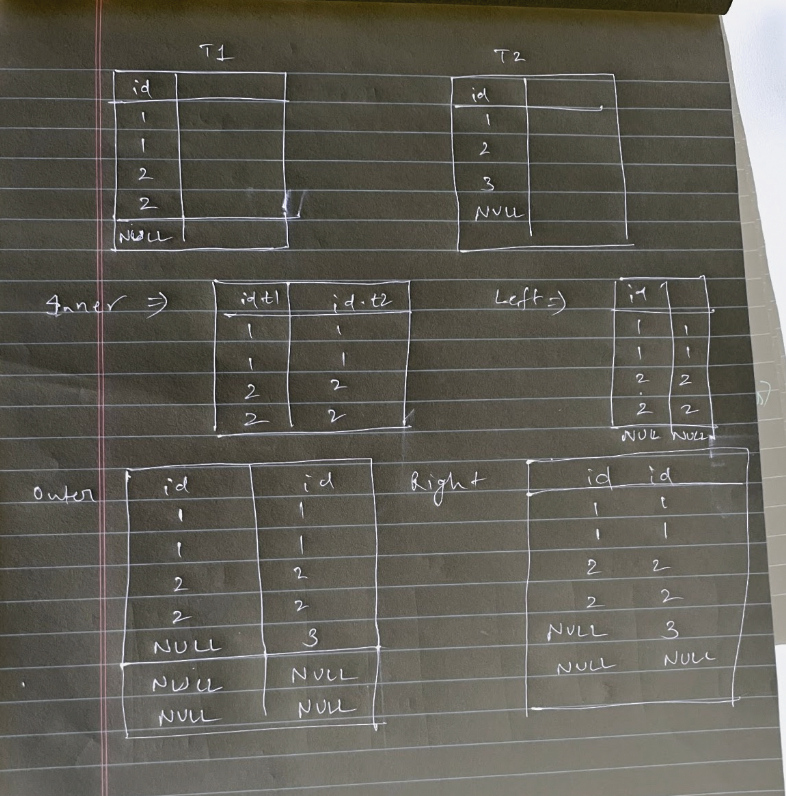
OR cp.customer\_id IS NULL;

**Output:**



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Example of different type of JOINs

****

## **JOINs with more than two tables -**

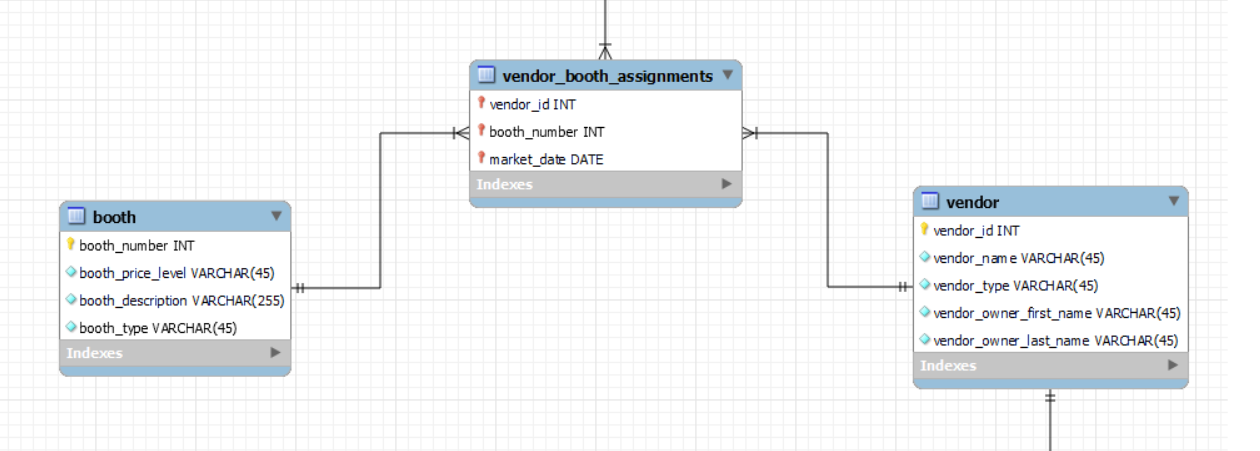
## Question: Let’s say we want details about all farmer’s market booths and every vendor booth assignment for every market date.

Imagine that you’re building an interactive report that lets you filter to a booth, a vendor, or a date, to see the resulting list of booth assignments with additional booth and vendor details.

So we need a merged dataset that contains all of their records.

This requires joining the three tables:

1. booth
2. vendor\_booth\_assignment
3. vendor



What kind of JOINs do you think we could use to ensure that all booths are included, even if they aren’t assigned to a vendor yet, and all vendors assigned to booths are included?

We can LEFT JOIN the booth table to the vendor\_booth\_assignments, including all of the booths, and LEFT JOIN vendor to vendor\_booth\_assignments in the results.

**The query to accomplish these joins looks like this -**

SELECT

b.booth\_number, b.booth\_type,

vba.market\_date,

v.vendor\_id, v.vendor\_name, v.vendor\_type

FROM farmers\_market.booth AS b

LEFT JOIN farmers\_market.vendor\_booth\_assignments AS vba

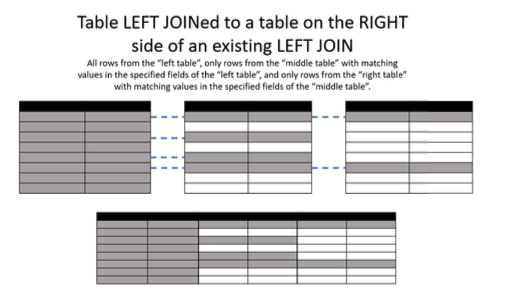
ON b.booth\_number = vba.booth\_number

LEFT JOIN farmers\_market.vendor AS v

ON v.vendor\_id = vba.vendor\_id

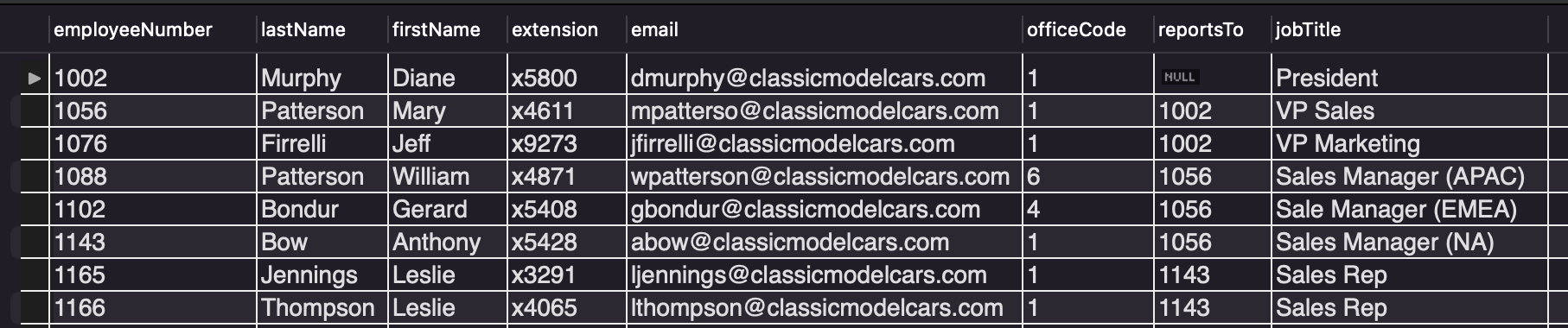
ORDER BY b.booth\_number, vba.market\_date;

* You can think of the second JOIN as being merged into the result of the first JOIN.
* Because in this case the *vendor\_id* field in the third table, **vendor**, is joined to the *vendor\_id* field in the second table, **vendor\_booth\_assignments**, only vendors that exist in the **vendor\_booth\_assignments** table will be included, resembling something like this:



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Question: For each employee in the given table, find out the name of their manager.



Here, we have the employee table in which for every employee we have the ‘reportsTo’ column where the employee ID of the manager is recorded.

Now, how would you solve this problem?

Note: For Murphy, we have NULL in reportsTo, so he must be the TOP manager/CEO.

**Self JOIN**

**Query:**

SELECT

CONCAT(e.lastName, ' ', e.firstName) AS Employee

CONCAT(m.lastName, ' ', m.firstName) AS Manager,

FROM

employees e

JOIN employees m

ON m.employeeNumber = e.reportsTo

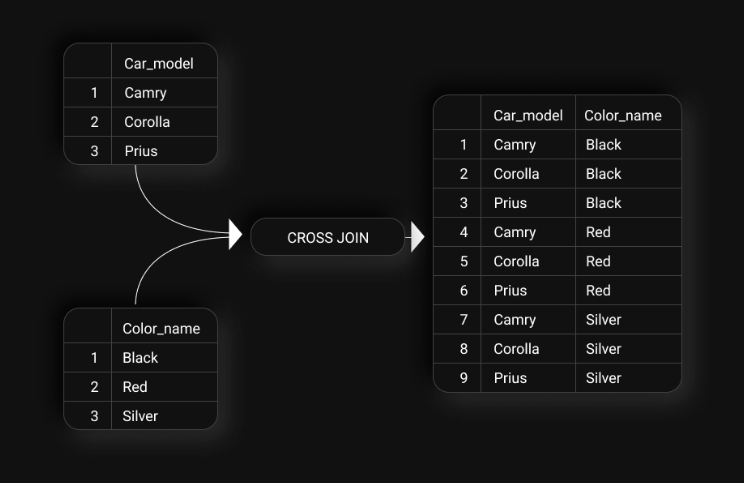
ORDER BY Manager;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# **Cross JOIN**

A Cross JOIN basically matches each record from one table (say T1) with each record from the other table (say T2).

So if we have ‘n’ no. of rows in T1 and ‘m’ no. of rows in T2 then the resulting table (say T3) will have ‘nxm’ no. of rows.



**Query:**

SELECT

c.Car\_model,

c1.Color\_name

FROM Cars c

CROSS JOIN Colors c1;

OR

SELECT

c.Car\_model,

c1.Color\_name

FROM Cars c, Colors c1;

**Another example of Cross JOINs could be**: You want to calculate the total sales of a product in every store in a country. For every product, we’d need to create a combination of product and store mapping.

Thus, we’d need to use cross-join for that.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# **Equi & Non-equi Joins**

We have seen equi-joins which are the regular joins.

* **Non-equi Join** is also a type of Join in which we need to retrieve data from multiple tables.
* **Non-Equi Join** matches the column values from different tables based on an inequality based on the operators like, **<,>,<=,>=,!=, BETWEEN**, etc.

Given the **User** table and the **Country** table.

We have data of several users, the countries they can visit and the countries they’ve already visited.

[User](https://drive.google.com/file/d/18MGleVRmghLy2wTQM0DtGTShqC3qDE7a/view?usp=share_link) table -

| user | country\_visited |
| --- | --- |
| John | India |
| Susan | USA |
| Mark | Japan |

[Country](https://drive.google.com/file/d/1miFPIl1v6-gq0VR4QWxJL3bUKlyRpNQo/view?usp=sharing) table -

| travel\_to |
| --- |
| India |
| USA |
| Japan |

Question: Recommend new countries to the users which they have not visited yet.

**Query:**

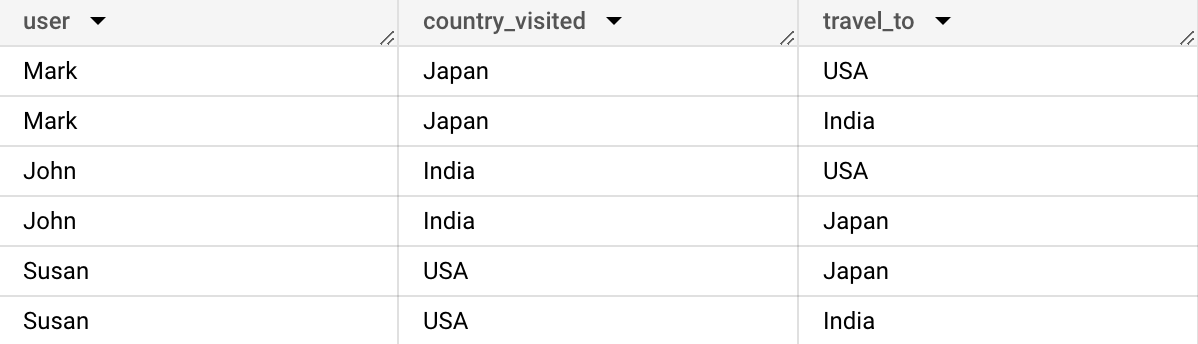
SELECT \*

FROM dataset.User T1

JOIN dataset.Country T2

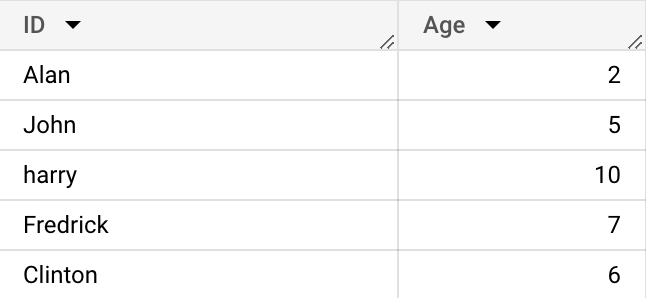
ON T1.country\_visited <> T2.travel\_to

**Output:**

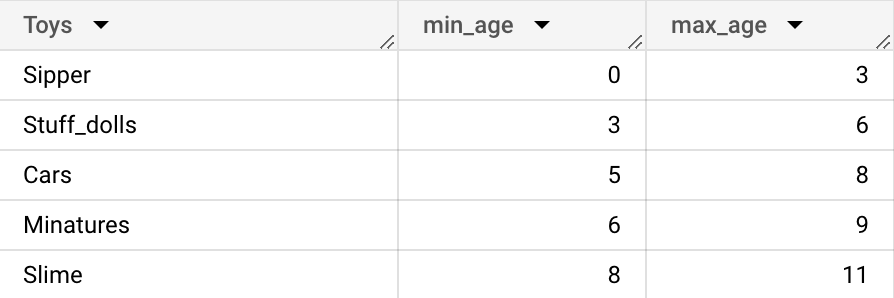
****

Given the **Kids\_info** table and the **Toys\_info** table.

[Kids\_info](https://drive.google.com/file/d/1RXoiWFiztQRO3lvYsQhrszAF6FfWBQ9V/view?usp=share_link) table-



[Toys\_info](https://drive.google.com/file/d/1HLqKaco6w8u7bkGPEut5oG7lCvDWR2DD/view?usp=share_link) table -



Question: Recommend toys to each kid who is above the minimum required age to play with those toys.

**Query:**

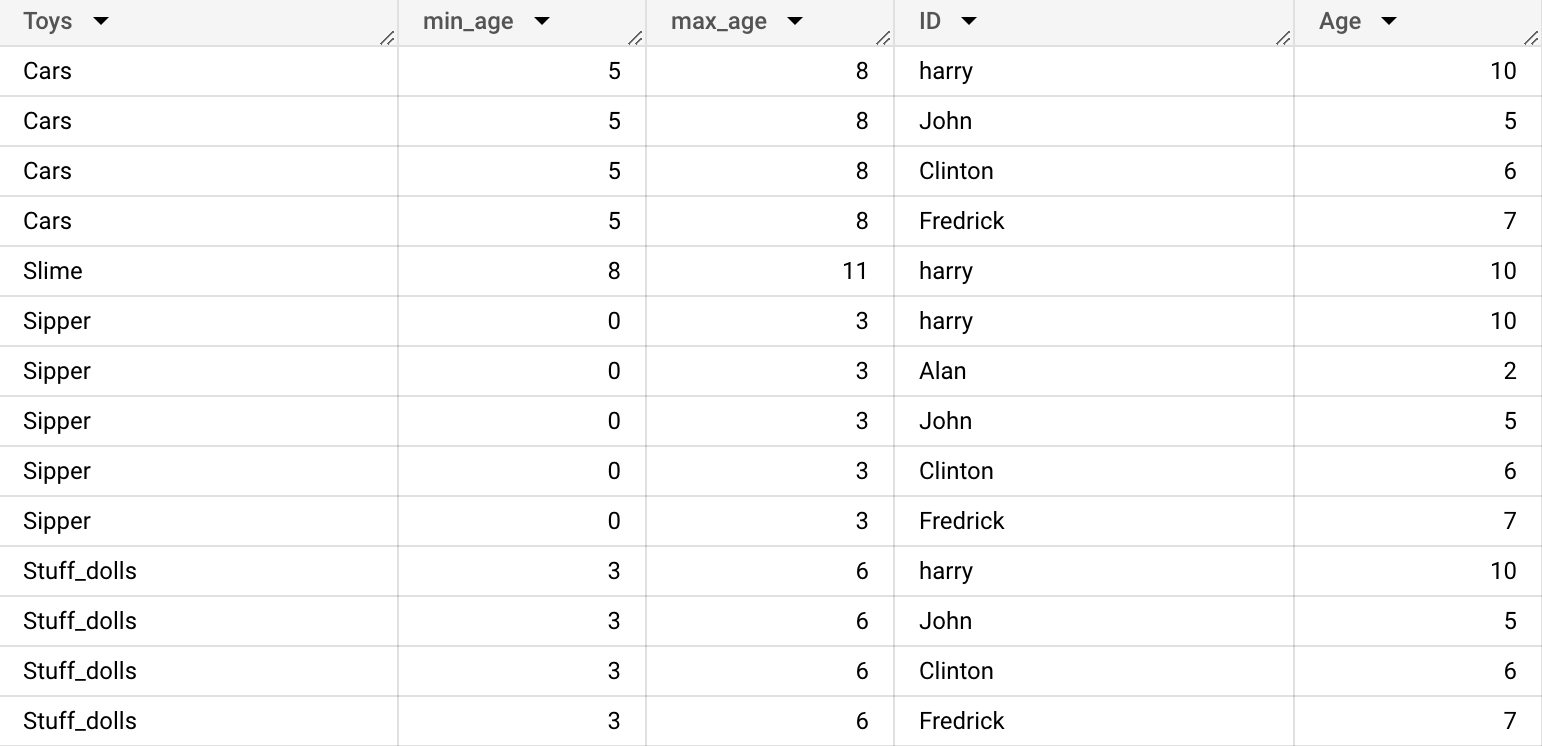
SELECT \*

FROM dataset.Toys\_info T

JOIN dataset.Kids\_info K

ON K.Age >= T.min\_age

**Output:**



Question: Recommend toys to each kid who falls within the right age group to play with those toys.

**Query:**

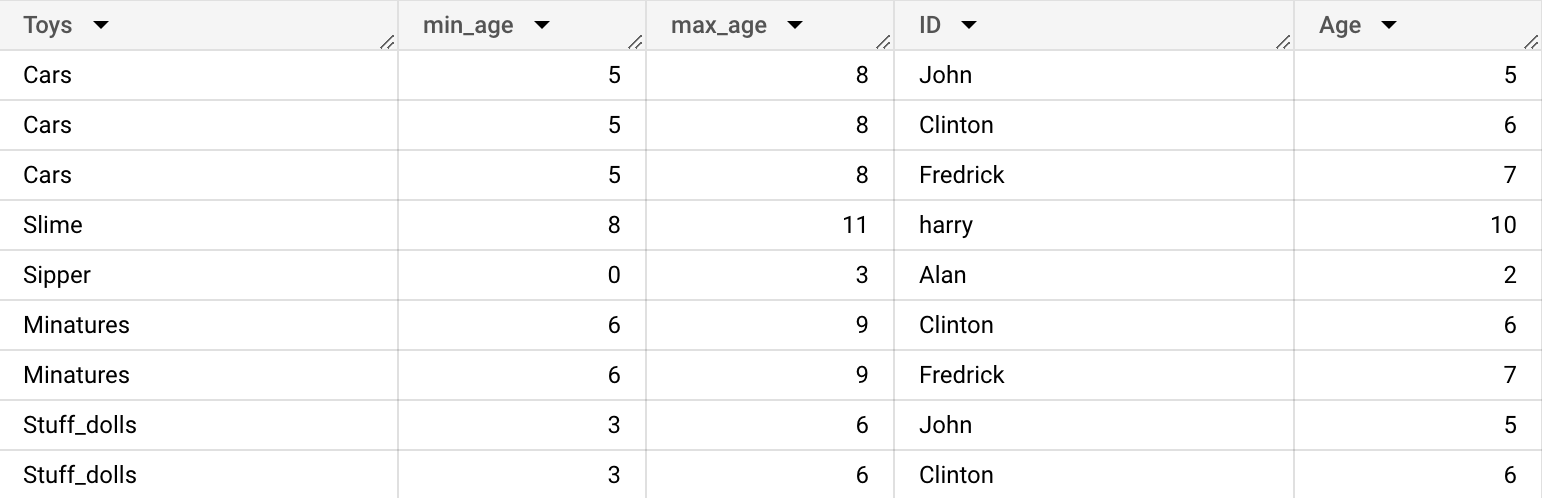
SELECT \*

FROM dataset.Toys\_info T

Join dataset.Kids\_info K

ON K.Age BETWEEN T.min\_age AND T.max\_age

**Output:**



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_