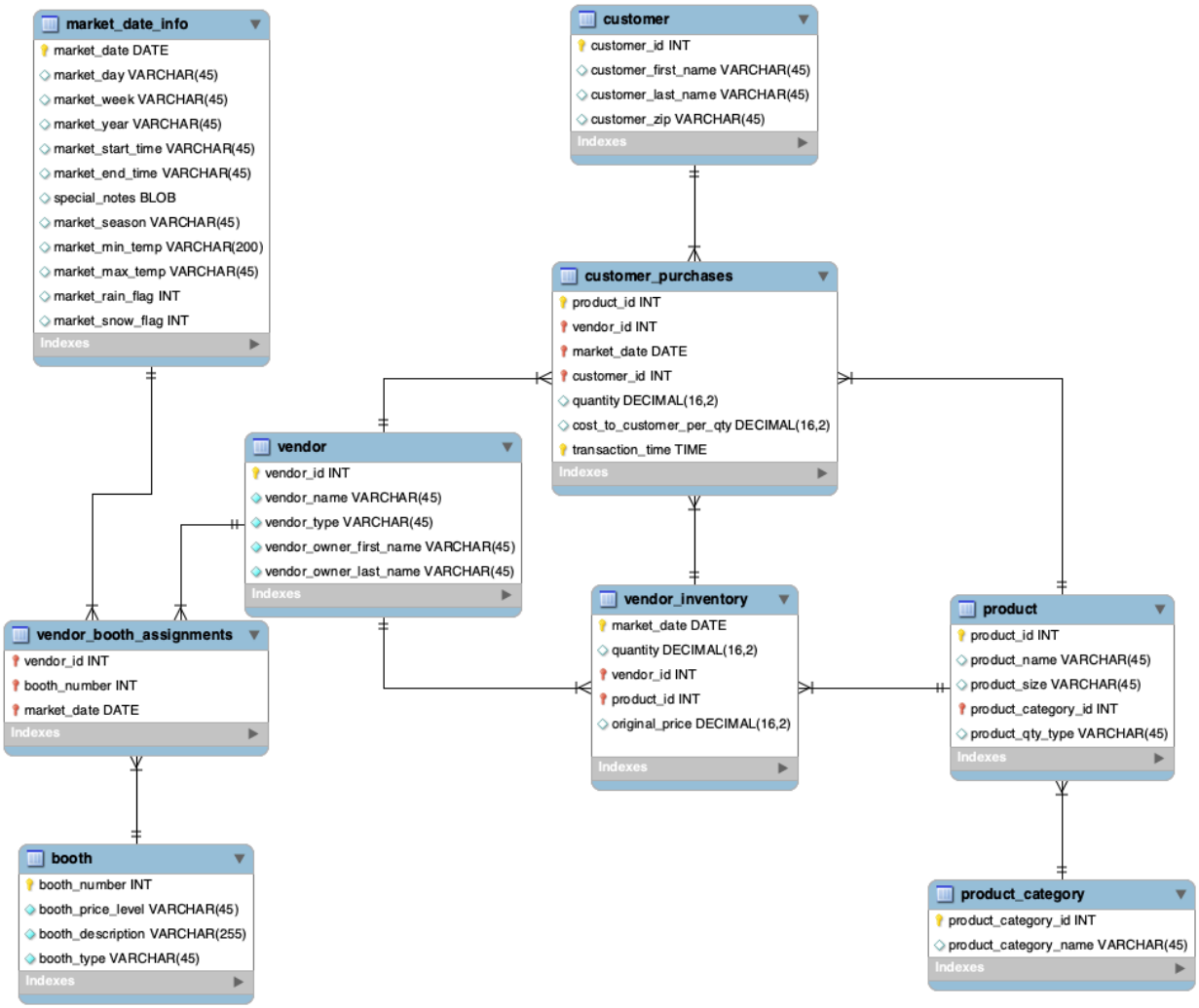
Joins contd.

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



JOINs with more than two tables -

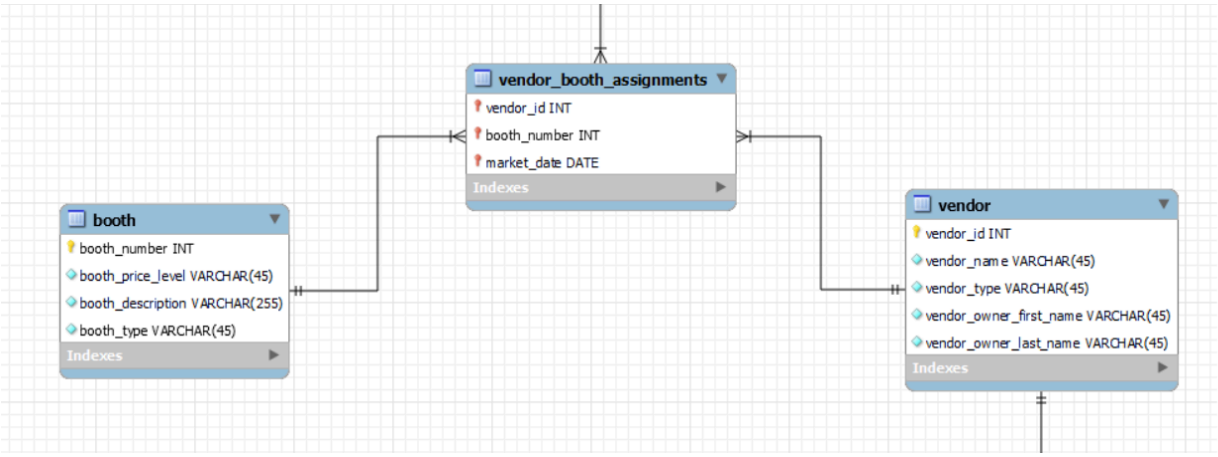
Question: Let's say we want details about all farmer's market booths and every vendor booth assignment for every market date.

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



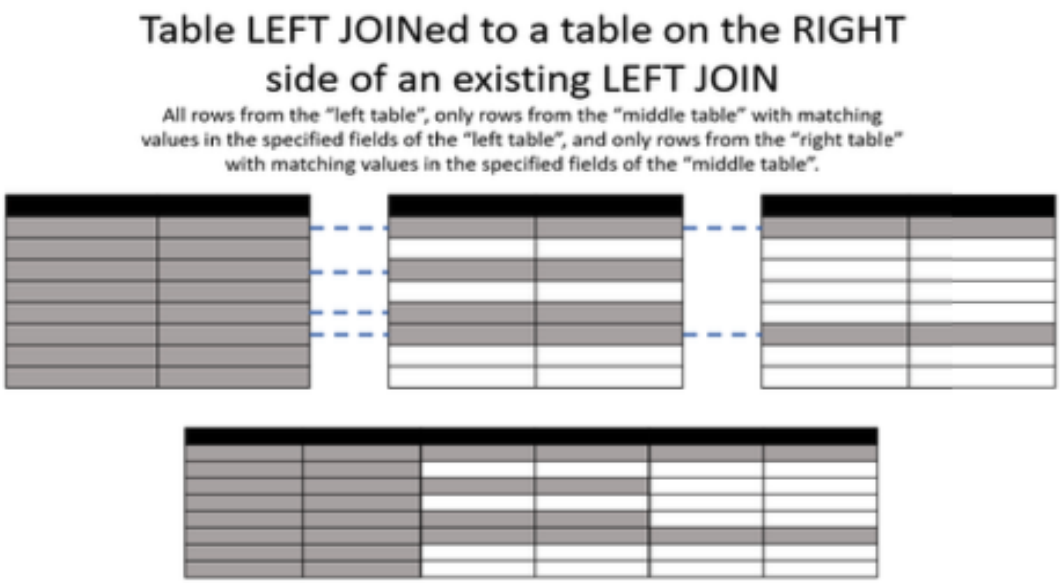
What kind of JOINs do you think we could use to ensure that all booths are included, even if they aren't assigned to a vendor yet, and all vendors assigned to booths are included?

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.

employeeNumber	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
1002	Murphy	Diane	x5800	dmurphy@classicmodelcars.com	1	NULL	President
1056	Patterson	Mary	x4611	mpatterso@classicmodelcars.com	1	1002	VP Sales
1076	Firrelli	Jeff	x9273	jfirrelli@classicmodelcars.com	1	1002	VP Marketing
1088	Patterson	William	x4871	wpatterson@classicmodelcars.com	6	1056	Sales Manager (APAC)
1102	Bondur	Gerard	x5408	gbondur@classicmodelcars.com	4	1056	Sale Manager (EMEA)
1143	Bow	Anthony	x5428	abow@classicmodelcars.com	1	1056	Sales Manager (NA)
1165	Jennings	Leslie	x3291	ljennings@classicmodelcars.com	1	1143	Sales Rep
1166	Thompson	Leslie	x4065	lthompson@classicmodelcars.com	1	1143	Sales Rep

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

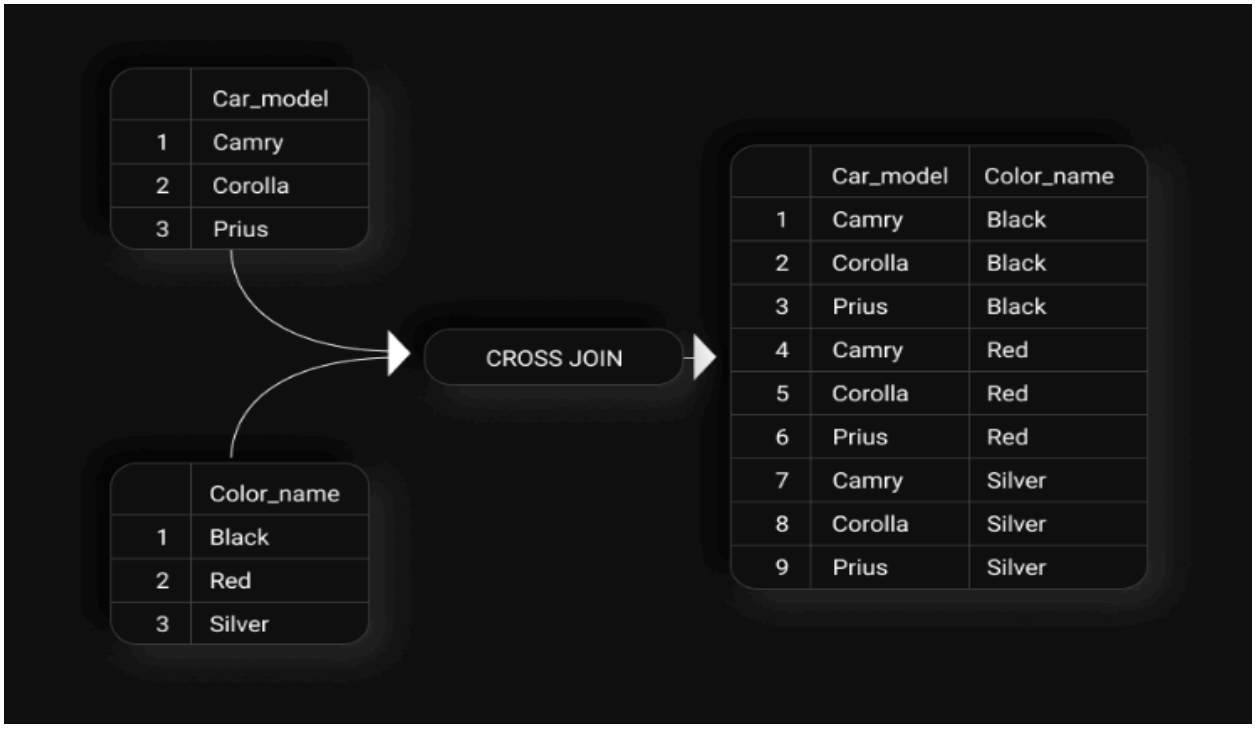
Query:

```
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 table -

user	country_visited
John	India
Susan	USA
Mark	Japan

Country 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:

user	country_visited	travel_to
Mark	Japan	USA
Mark	Japan	India
John	India	USA
John	India	Japan
Susan	USA	Japan
Susan	USA	India

Given the **Kids_info** table and the **Toys_info** table.

Kids_info table-

ID	Age
Alan	2
John	5
harry	10
Fredrick	7
Clinton	6

Toys_info table -

Toys	min_age	max_age
Sipper	0	3
Stuff_dolls	3	6
Cars	5	8
Minatures	6	9
Slime	8	11

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:

Toys	min_age	max_age	ID	Age
Cars	5	8	harry	10
Cars	5	8	John	5
Cars	5	8	Clinton	6
Cars	5	8	Fredrick	7
Slime	8	11	harry	10
Sipper	0	3	harry	10
Sipper	0	3	Alan	2
Sipper	0	3	John	5
Sipper	0	3	Clinton	6
Sipper	0	3	Fredrick	7
Stuff_dolls	3	6	harry	10
Stuff_dolls	3	6	John	5
Stuff_dolls	3	6	Clinton	6
Stuff_dolls	3	6	Fredrick	7

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:

Toys	min_age	max_age	ID	Age
Cars	5	8	John	5
Cars	5	8	Clinton	6
Cars	5	8	Fredrick	7
Slime	8	11	harry	10
Sipper	0	3	Alan	2
Minatures	6	9	Clinton	6
Minatures	6	9	Fredrick	7
Stuff_dolls	3	6	John	5
Stuff_dolls	3	6	Clinton	6