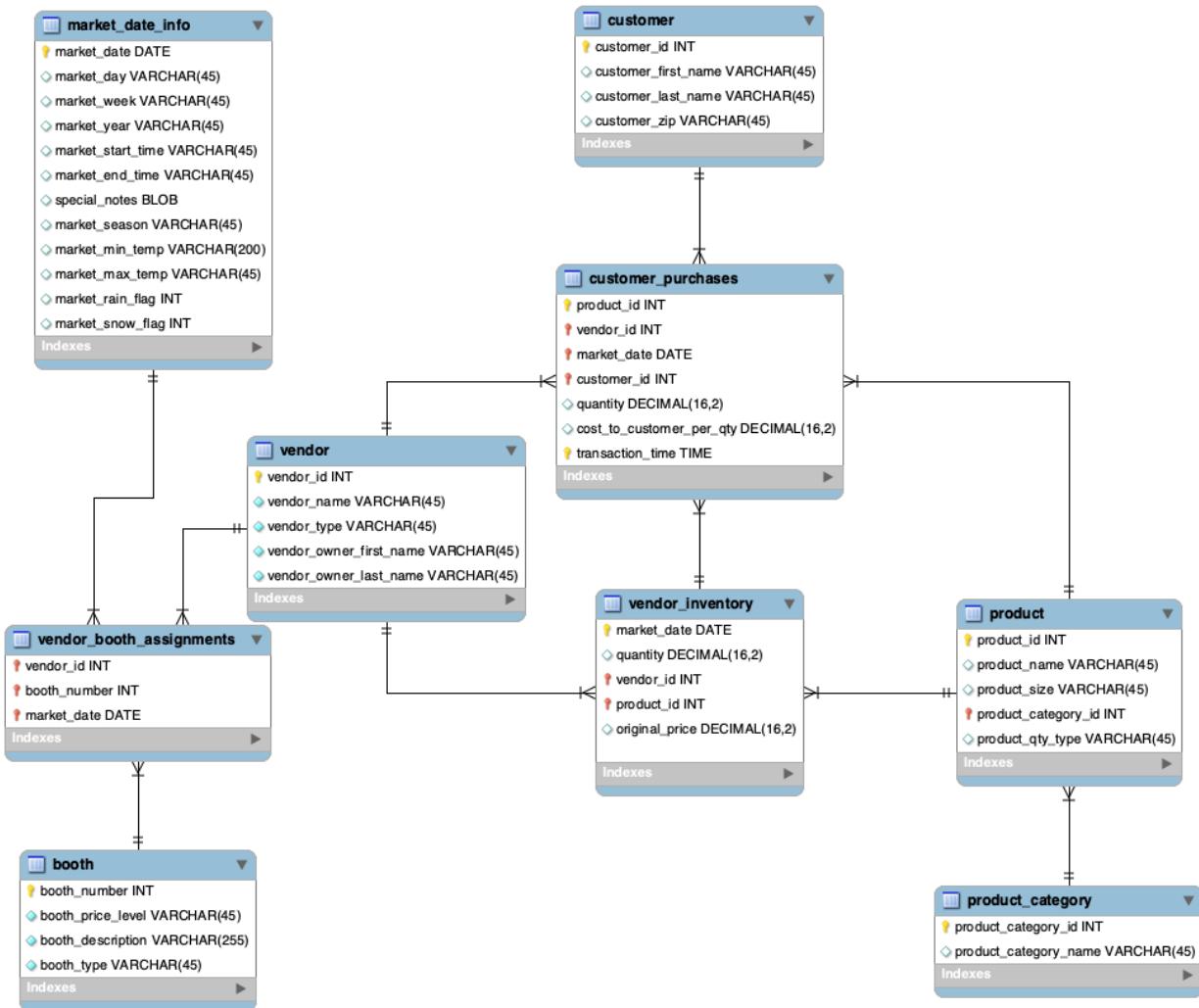


Extracting Data using SQL

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



Relationships in a Schema

In the previous lecture, we learnt about different types of keys.

Now let's look at how we define the relationships between two or more tables.

There are multiple types of relationships that two entities (or tables) can share with each other.

Different types of relationships -

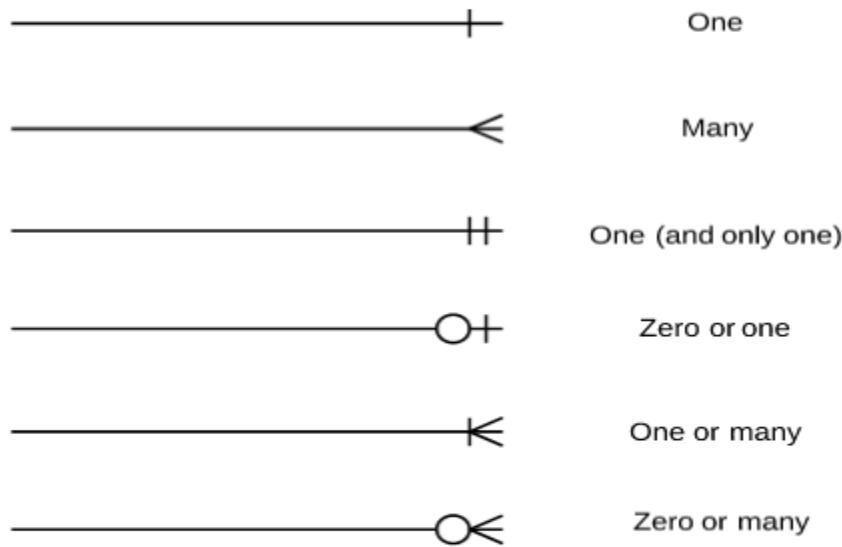
- One-to-One Relationship:
 - One row in Table A will be related to one and only one row in Table B.
 - **Example** - marriage between a husband and his wife
 - One husband can have only one wife and vice versa.
- One-to-Many Relationship:
 - A single row in Table A is related to multiple rows of Table B.
 - A single row in Table B is related to a single row in Table A.
 - **Example** -
 - relationship between a customer and his orders
 - A customer can place multiple orders.
 - But one order can only be placed by a single customer.

Customer		Orders		
CustID	Name	OrderID	CustID	OrderDate
1	John	3	1	31-11-20
2	Mark	1	2	11-10-20
3	Susan	2	2	15-11-20
4	Lucy	4	3	7-12-20

↓ PK ↓ FK

- Many-to-Many Relationship:
 - One row in Table A is related to many rows in Table B.
 - One row in Table B is related to many rows in Table A.
 - **Example** - relationship between customers and suppliers
 - One customer can purchase from many suppliers.
 - One supplier can sell to many customers.

ER Diagram edge notations:

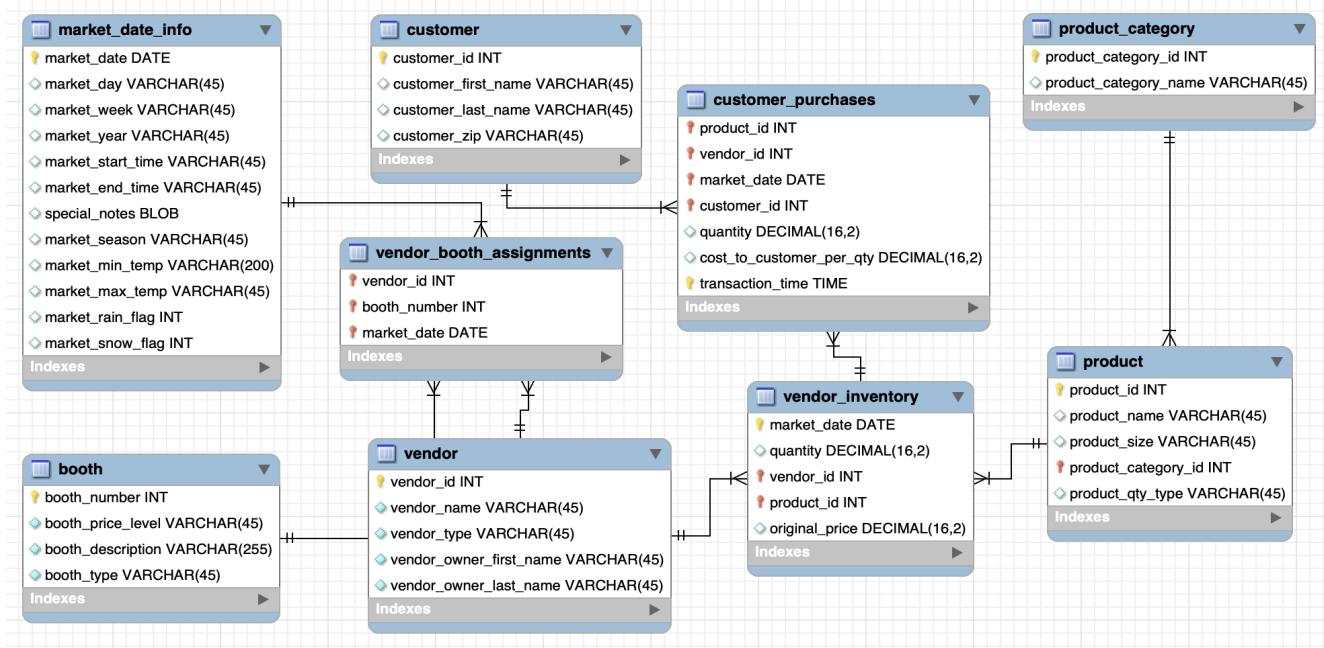


Understanding the ER diagram of the Farmer's Market database.

You should have a clear understanding of ER diagrams now.
Let's go back to the original task that your manager has given you:

Looking at the diagram, answer the following questions:

1. What is the primary key in the customer, product and vendor tables?
2. What is the relationship between the following tables:
 - a. customer and customer_purchase
 - b. vendor and vendor_inventory
 - c. product and product_category
3. Which attribute is the foreign key in customer_purchase, vendor_inventory and product_category tables?



So, we have defined multiple entities and how they are related to each other using these diagrams and edges to connect them.

This diagram that shows the entities along with their attributes, keys, and relationships is called an Entity-Relationship diagram.

Most of the time, you'll be given this **ER diagram** which is interchangeably referred to as the **DB schema** and you'll have to extract the insights from the tables.

Issues with implementing **1-1** and **m-m** relationships:

Did you notice that we don't have any, **One-to-One** or **Many-to-Many** relationships in our ED diagram?

Let's understand why that's the case.

1. One-to-One

- a. With one to one relationships, do we even need two separate tables?
- b. No. Instead, we can keep this data in two different columns of a single table.

The diagram illustrates the transformation of a one-to-one relationship between 'Husband' and 'Wife' into a single-table representation.

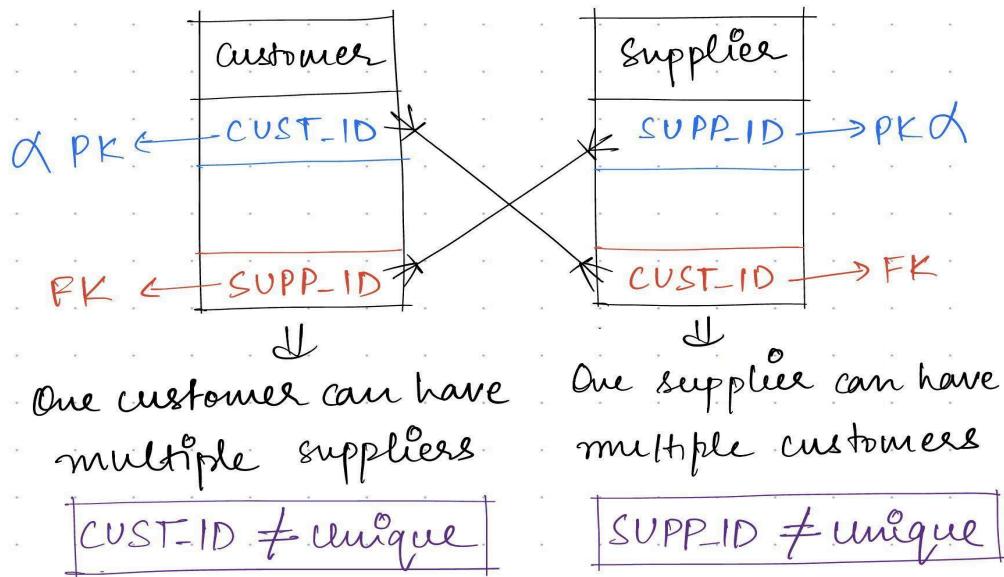
Husband	Wife
RANVEER	ANUSHKA
RANBIR	DEEPIKA
NICK	ALIA
KOHLI	PRIYANKA

Red X marks are placed over the entries in the 'Wife' column for RANBIR and NICK. An arrow points from this state to the final representation:

Husband	Wife
RANVEER	ALIA
RANBIR	DEEPIKA
NICK	PRIYANKA
KOHLI	ANUSHKA

2. Many-to-Many

A many to many relationship is hard to maintain. Why?



That's why to resolve this issue, a **junction table** is introduced.

- The junction table serves as an intermediary table that connects the two tables, representing the relationships between them.
- By introducing a junction table, you can break down the many-to-many relationship into two one-to-many relationships.
- Each record in the junction table represents a unique combination of a customer and a supplier, linking them together.

Question: What's the relationship between vendor and product tables?

Many-to-Many relationship established using a **junction table** (vendor_inventory).

How will you extract data from each of these tables?

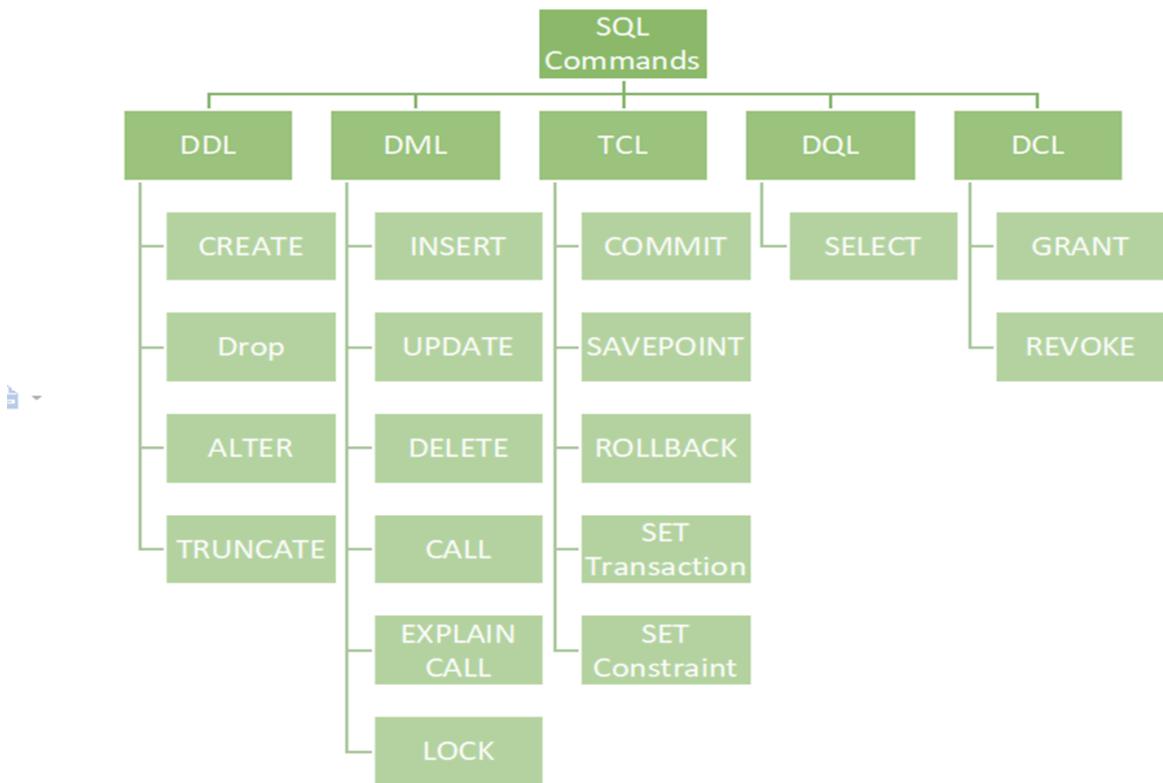
For that, we have a specific language to interact with this relational DBMS, called **SQL**.

What is SQL?

- SQL stands for **Structured Query Language**
- There are different types of SQL commands depending on the type of task you want to perform.

Types of SQL commands -

- **DDL** - Data Definition Language
- **DML** - Data Manipulation Language
- **TCL** - Transaction Control Language
- **DQL** - Data Query Language - most important
- **DCL** - Data Control Language



Now, If I ask you to get me all the products from the farmer's market database, how will you do that?

Question: Get all the products available in the market.

The Fundamental Syntax Structure of a **SELECT** Query

In SQL, we have these SELECT statements. **SELECT:** A SELECT statement is the SQL code that retrieves data from the database.

SQL SELECT queries follow this basic syntax, though most of the clauses are optional:

```
SELECT [columns to return]
FROM [schema.table]
WHERE [conditional filter statements]
GROUP BY [columns to group on]
HAVING [conditional filter statements that are run after grouping]
ORDER BY [columns to sort on]
LIMIT [first x number of rows to be selected]
```

To answer our question, The SELECT and FROM clauses are generally required because they indicate which columns to select and from what table.

For example,

```
SELECT * FROM `farmers_market.product`
```

can be read as:

- “Select all columns from the product table in the farmers_market schema.”
-

Tip: Problems with (*)

- Even if you want all columns returned, it’s good practice to list the names of the columns instead of using the asterisk.
- **Production pipelines:** Especially if the query will be used as part of a data pipeline (if the query is automatically run nightly and the results are used as an input into the next step of a series of code functions without human review).
- This is because returning “all” columns may result in a different output **if the underlying table is modified**, such as when a new column is added, or columns appear in a different order, which

could break your automated data pipeline.

- Although it's convenient to query all columns using *, **it increases the unnecessary data transferred between the database server and the application** because the application may need only partial data from the table.
-

To specify which columns you want to be returned, list the column names immediately after SELECT, separated by commas, instead of using the asterisk (*).

For example, select the product IDs and their corresponding product names & sizes from the `product` table.

```
SELECT product_id, product_name, product_size  
FROM `farmers_market.product`
```

How to sort data in SQL?

This is where the **Order By** clause comes into play.

- The ORDER BY clause is used to sort the output rows.
- In it, you list the columns by which you want to sort the results, in order, separated by commas.
- You can also specify whether you want the sorting to be done in ascending (**ASC**) or descending (**DESC**) order.
- The sort order is **ascending by default**, so the ASC keyword is optional.

If we want to sort this entire data in **descending order** of a particular column, in this case by **product_id** -

```
SELECT product_id, product_name, product_size  
FROM `farmers_market.product`  
ORDER BY product_id DESC
```

You can also sort data by multiple columns with different sort order in the following manner -

Syntax:

```
SELECT <col1>, <col2>, ...  
FROM `dataset.table_name`  
ORDER BY  
    <col1> ORDER,  
    <col2>,  
    <col3>, ...
```

The sorting order is applicable for one single column only.

Here the column written first is given priority so first the data will be sorted on the basis of <col1> then by <col2> and at the end by <col3>

Let's look at the `customer_purchases` table -

```
SELECT *  
FROM `farmers_market.customer_purchases`
```

Question: How to sort the purchases by market date (most recent) and transaction time both?

For each **market_date** (in descending order), we'll sort the data by **transaction_time** in ascending order.

```
SELECT  
    market_date,  
    transaction_time,  
    quantity,  
    cost_to_customer_per_qty  
FROM `farmers_market.customer_purchases`  
ORDER BY market_date DESC, transaction_time
```

NOTE:

- **For strings**, ASC sorts text alphabetically and numeric values from low to high, and DESC sorts them in reverse order.
 - **In MySQL**, NULL values appear first when sorting is done in the ascending order.
-