Extracting data using SQL

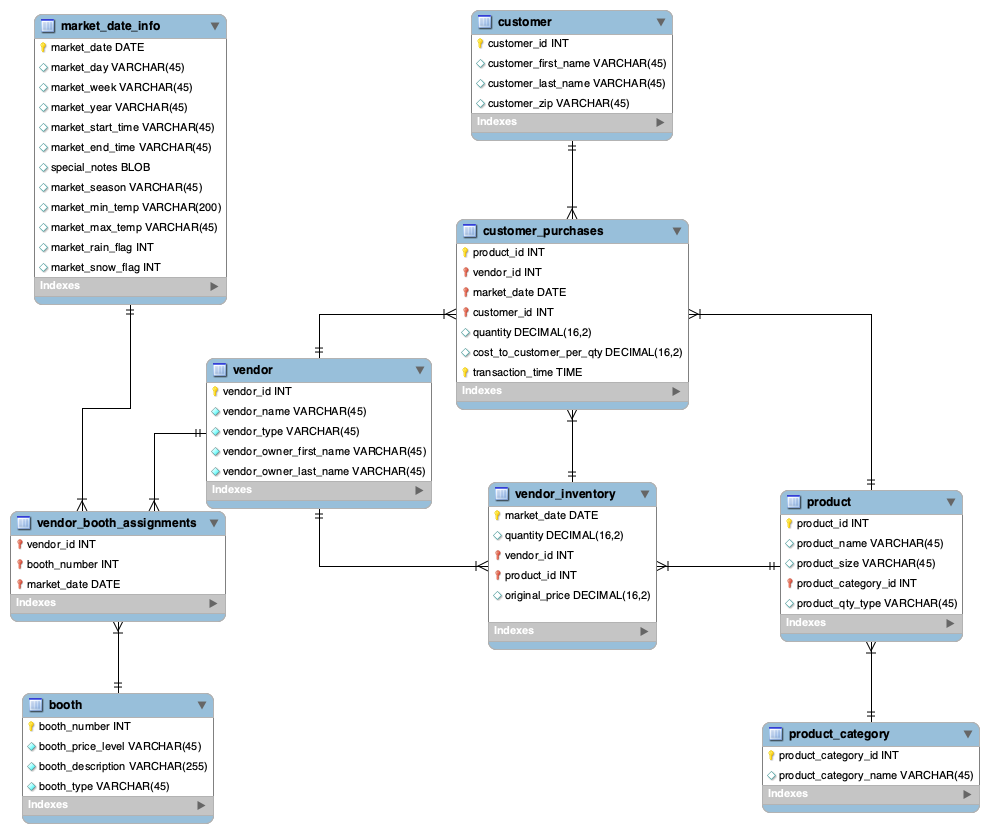
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Please note that any topics that are not covered in today's lecture will be covered in the next lecture.**

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

### **Problem Statement**

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



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

# **Relationships in a Schema**

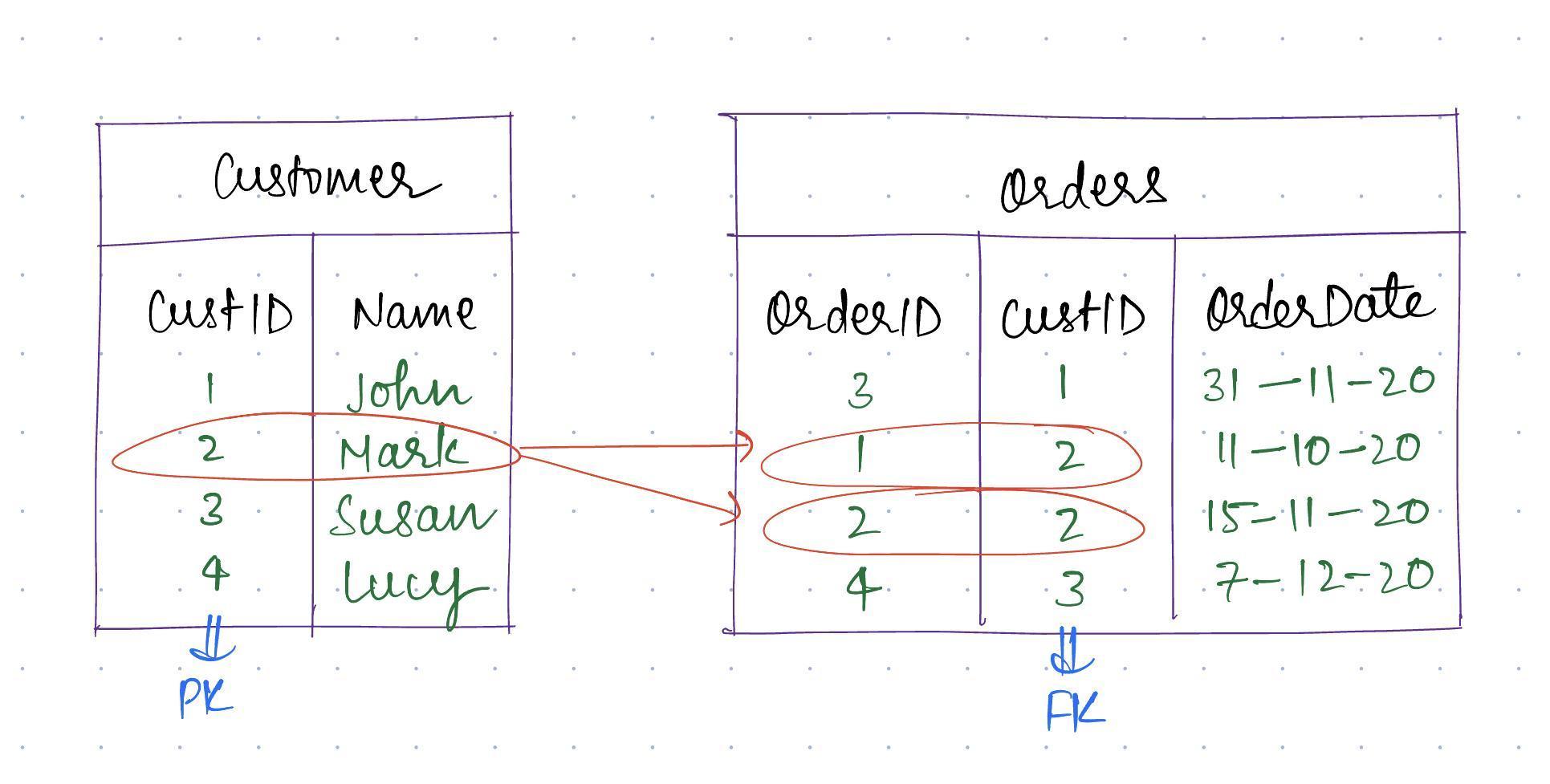
In the previous lecture, we learnt about different types of keys.

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.

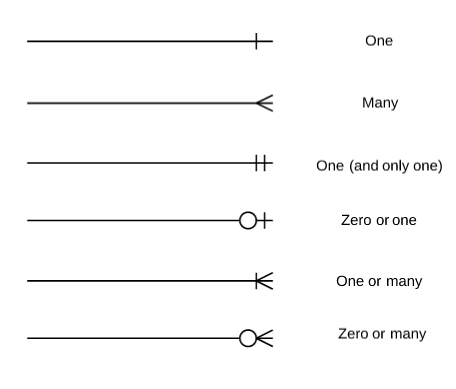
#### **Different types of relationships -**

* 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.
      * However, one order can only be placed by a single customer.



* 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** - the relationship between customers and suppliers
    - One customer can purchase from many suppliers.
    - One supplier can sell to many customers.

# ER Diagram edge notations:

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

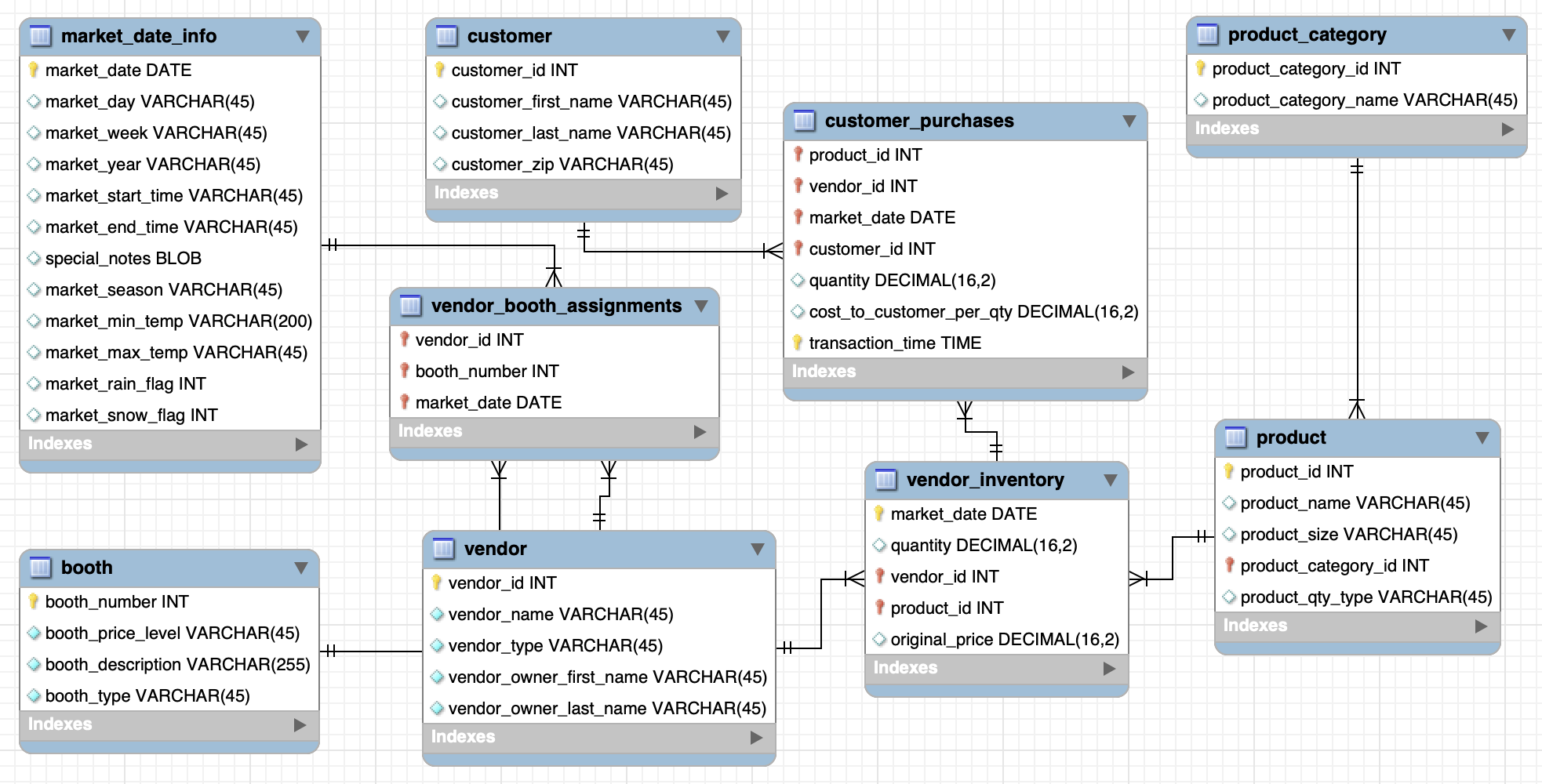
# Understanding the ER diagram of the Farmer’s Market database.

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:
   1. customer and customer\_purchase
   2. vendor and vendor\_inventory
   3. 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.

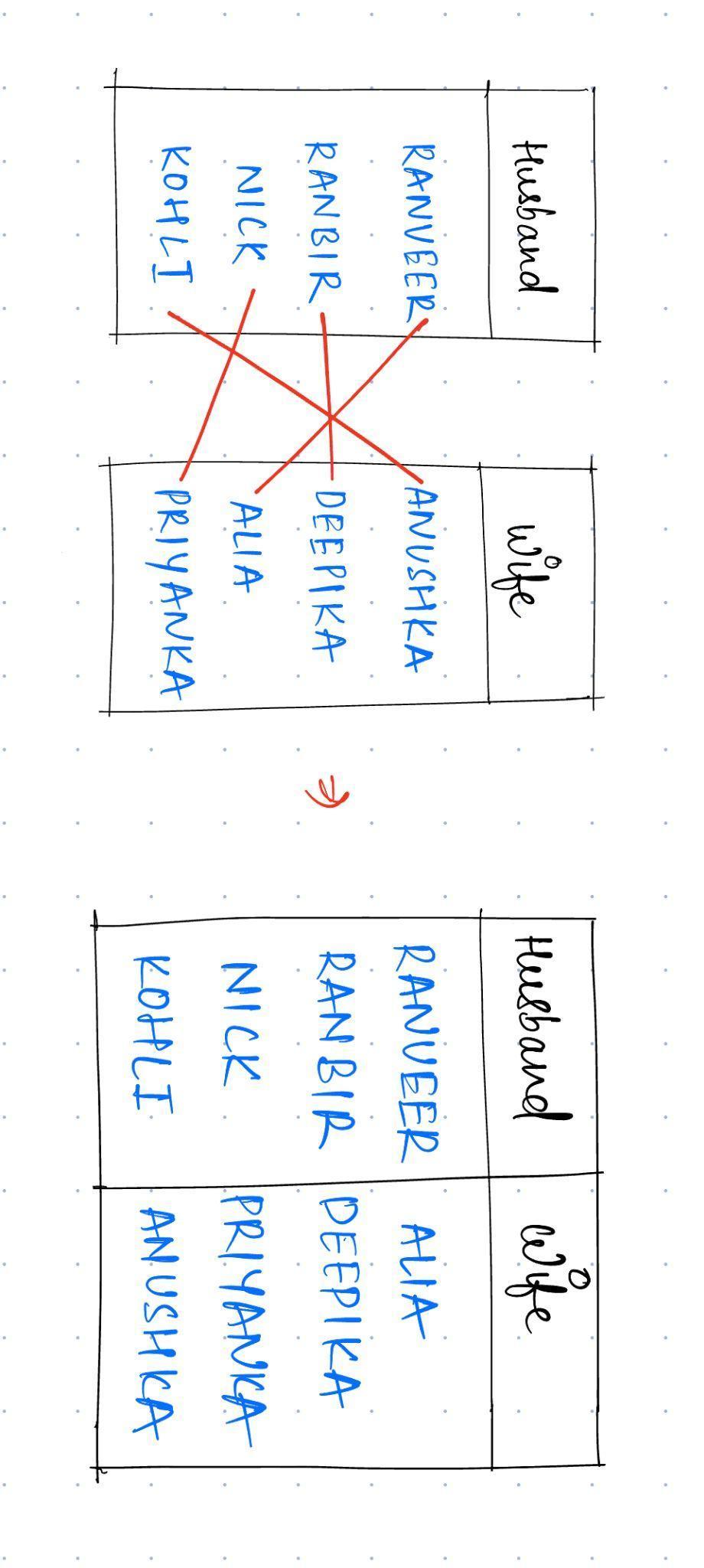
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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?

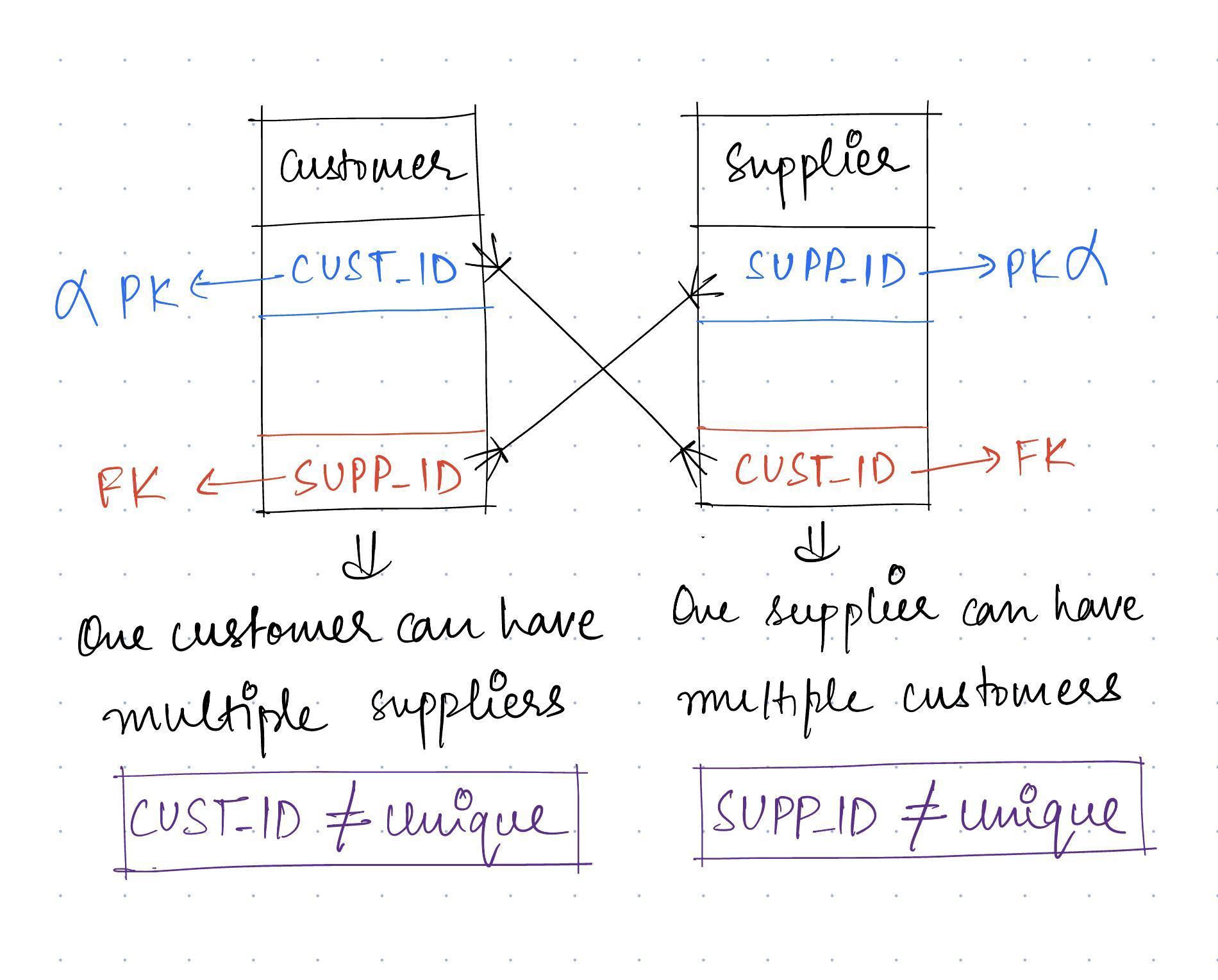
Let’s understand why that’s the case.

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



1. **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.

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]**

**OFFSET [number of rows to skip]**

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 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 (\*).

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

How to sort data in SQL?

This is wherethe **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 to 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>**

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

# Question: What if the manager asks you to fetch the 2nd & 3rd most recent purchase (skip the 1st one)?

SELECT

market\_date,

quantity

FROM `farmers\_market.customer\_purchases`

ORDER BY market\_date DESC

LIMIT 2

OFFSET 1

**LIMIT**

By using LIMIT, you can get a preview of your current query’s results without having to wait for all of the results to be compiled.

# **OFFSET**

OFFSET allows you to **omit a specified number of rows before the beginning** of the result set.

**NOTE:**

**BigQuery** uses the LIMIT and OFFSET clauses separately.

* The general syntax is LIMIT <count> OFFSET <offset>,
* where <count> specifies the maximum number of rows to return,
* and <offset> specifies the number of rows to skip.
* Eg. LIMIT 2 OFFSET 1

In **MySQL**,

* The OFFSET clause can either be used separately (as shown above).
* Or we can combine it with the LIMIT clause (as shown below).
  + The general syntax is LIMIT <offset>, <count>
    - where <offset> specifies the number of rows to skip
    - and <count> specifies the maximum number of rows to return.
    - E.g., LIMIT 1, 2.

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

**Tips:**

* Line breaks and tabs don’t matter to SQL code execution and are treated as spaces, so you can indent your SQL and break it into multiple lines for readability without affecting the output.
* SQL reserved keywords like SELECT, FROM, WHERE, etc. are case insensitive. Do not confuse the column names, dataset names or table names to be case insensitive as well.
* Semicolon (;) is not required when you’re writing a single query in the editor. In case you have multiple queries in the same tab, then only you need to add a semicolon at the end of a query.

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

**Order of Execution** of a SQL query:

* **FROM** - The database gets the data from tables in the FROM clause.
* **SELECT** - It 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.

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

## How do you do calculations on the go? - **Inline Calculations**

### Question: In the customer purchases, we have quantity and cost per qty separate, query the total amount that the customer has paid along with the date, customer ID, vendor\_id, qty, cost per qty and the total amount and round off all the values in the result set to 2 decimal places?

In the `customer\_purchases` table, we have a ***quantity*** column and a ***cost\_to\_customer\_per\_qty*** column, so we can multiply those to get the total amount the customer has paid.

**Functions in SQL**

* A SQL function is a piece of code that takes inputs that you give it (which are called parameters), performs some operation on those inputs, and returns a value.
* You can use **inline functions** in your query to modify the raw values from the database tables before displaying them in the output.

**Function Syntax**

**FUNCTION\_NAME([parameter 1],[parameter 2], . . . .[parameter n])**

* Each bracketed item shown is a placeholder for an input parameter.
* Parameters go inside the parentheses following the function name and are separated by commas.
* The input parameter might be a field name or a value.

We can easily do this by using the **ROUND()** function.

**Query:**

SELECT

market\_date,

customer\_id,

vendor\_id,

ROUND(quantity \* cost\_to\_customer\_per\_qty, 2) AS price

FROM farmers\_market.customer\_purchases

* To give the calculated column a meaningful name, we can create an *alias* by adding the keyword **AS** after the calculation and then specifying the new name.
* If your alias includes spaces, it should be surrounded by single quotes.

Here, we’ll give the alias “**price**” to the result of the “quantity times cost\_ to\_customer\_per\_qty” calculation.

**TIP:**

The ROUND() function can also accept negative numbers for the second parameter, to **round digits that are to the left of the decimal point**.

For example, SELECT ROUND(1245, -2) will return a value of 1200.

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