Table of Contents

Introduction

Structured Query Language (SQL) is a crucial component of data analytics, serving as a key tool for querying and managing relational databases. This report centres on the development and execution of SQL queries to engage with a detailed relational database. The main aim was to demonstrate SQL's proficiency in data manipulation and its capability to address relevant business questions. Utilizing diverse SQL techniques—including data retrieval, aggregation, and intricate joins—the project sought to derive valuable insights that facilitate informed decision-making.

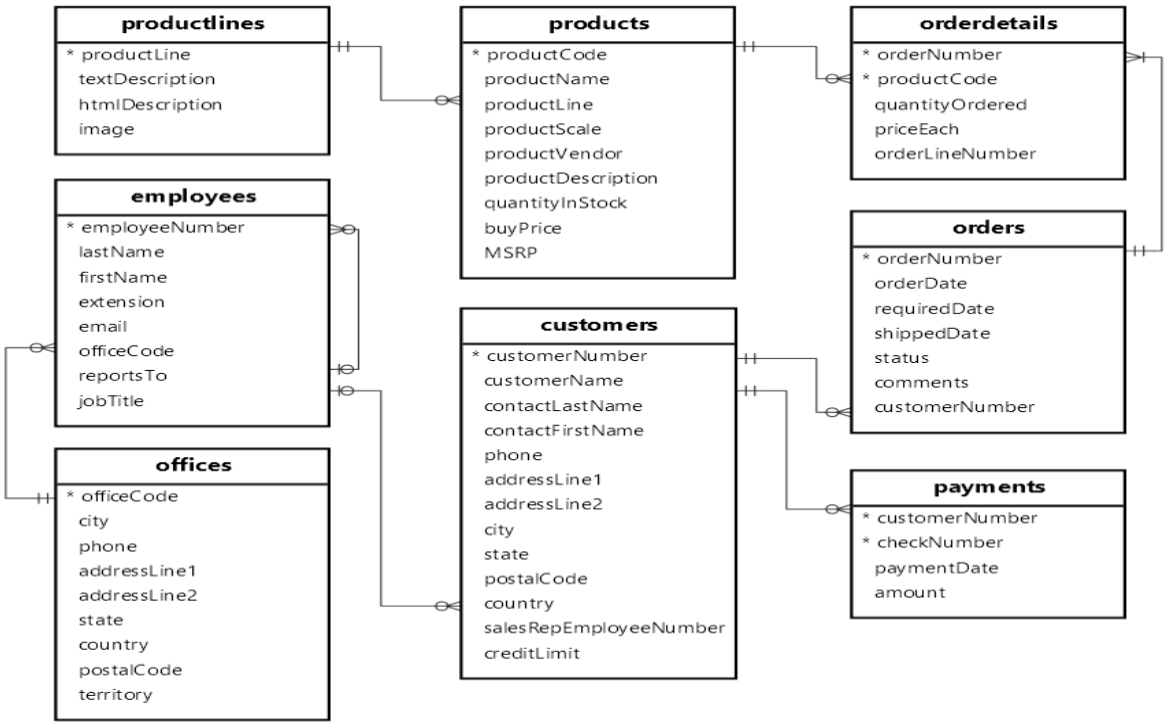
Description of Database:

The **classicmodels** database is a retailer of scale models of classic cars. It contains typical business data, including information about customers, products, sales orders, sales order line items, and more.

The database schema consists of the following tables:

- **customers**: stores customer's data.
- **products**: stores a list of scale model cars.
- **productlines**: stores a list of product lines.
- **orders**: stores sales orders placed by customers.
- **orderdetails**: stores sales order line items for every sales order.
- **payments**: stores payments made by customers based on their accounts.
- **employees**: stores employee information and the organization structure such as who reports to whom.
- **offices**: stores sales office data.

The following picture illustrates the ER diagram of the sample database:

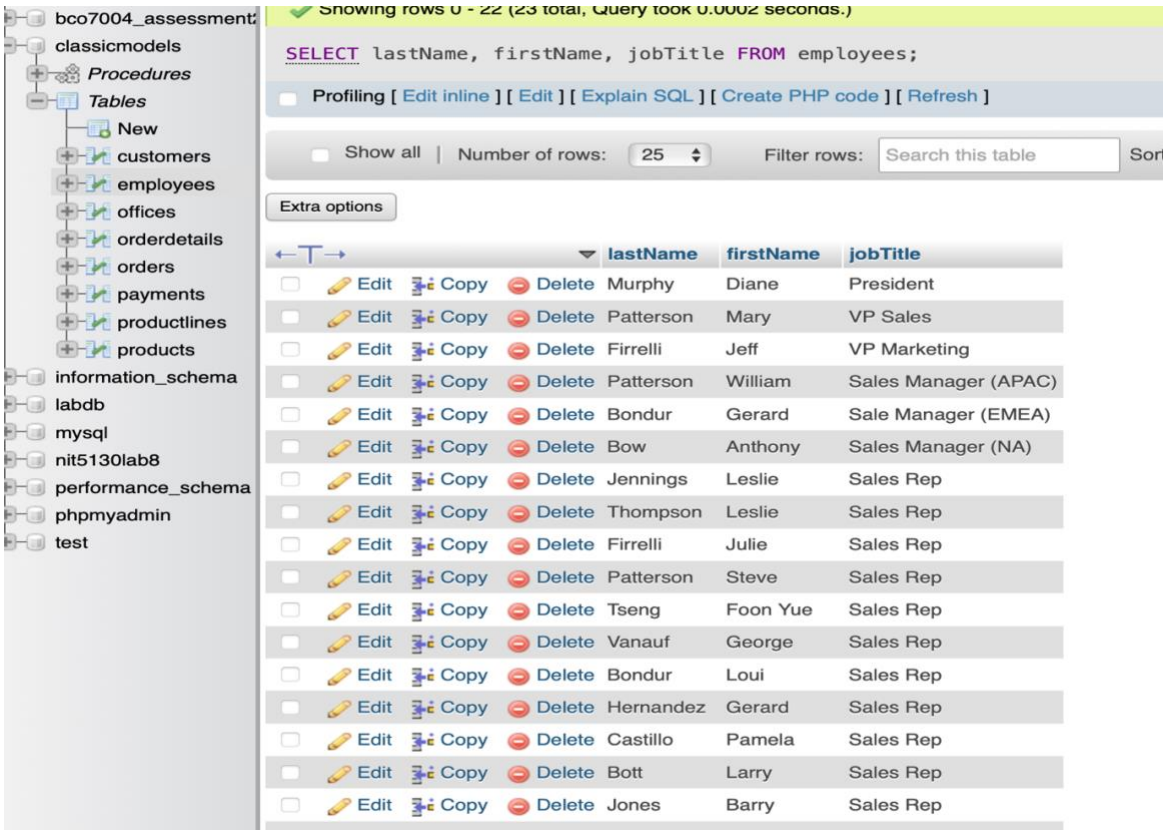


Querying data: SELECT

Select statement allows to select data from *one or more* tables. When executing the SELECT statement, MySQL evaluates the FROM clause before the SELECT.

Q.1 What are the first name, last name, and job title of each employee in the company?

```
SELECT
    lastName,
    firstName,
    jobTitle
FROM
    employees;
```



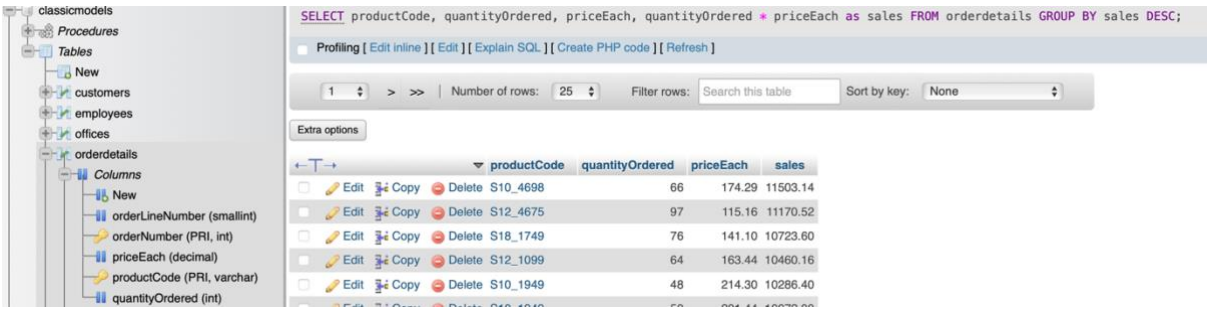
Sorting data: ORDER BY

When using SELECT statement to query data from a table, the order of the rows in the result set is unspecified. To short row in the result set, we need to add the ORDER BY clause to the SELECT statement.

```
SELECT
    select_list
FROM
    table_name
ORDER BY
    column1 [ASC|DESC],
    column2 [ASC|DESC],
    …;
```

When executing the SELECT statement with an ORDER BY clause, MySQL always evaluates the ORDER BY clause after the FROM and SELECT clause.

Q.2 Write an SQL query to retrieve product code, quantity ordered and price, calculate total of each product as sales and short in descending order.

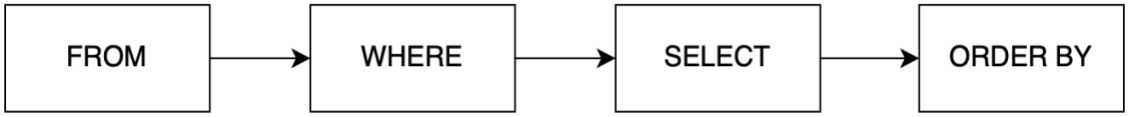


Filtering data: WHERE, DISTINCT, AND, OR, IN, NOT IN, BETWEEN, LIKE, LIMIT, IS NULL

The WHERE clause allows to specify a search condition for the rows returned by a query. Below is the syntax for filtering data using WHARE clause.

```
SELECT
    select_list
FROM
    table_name
WHERE
    search_condition;
```

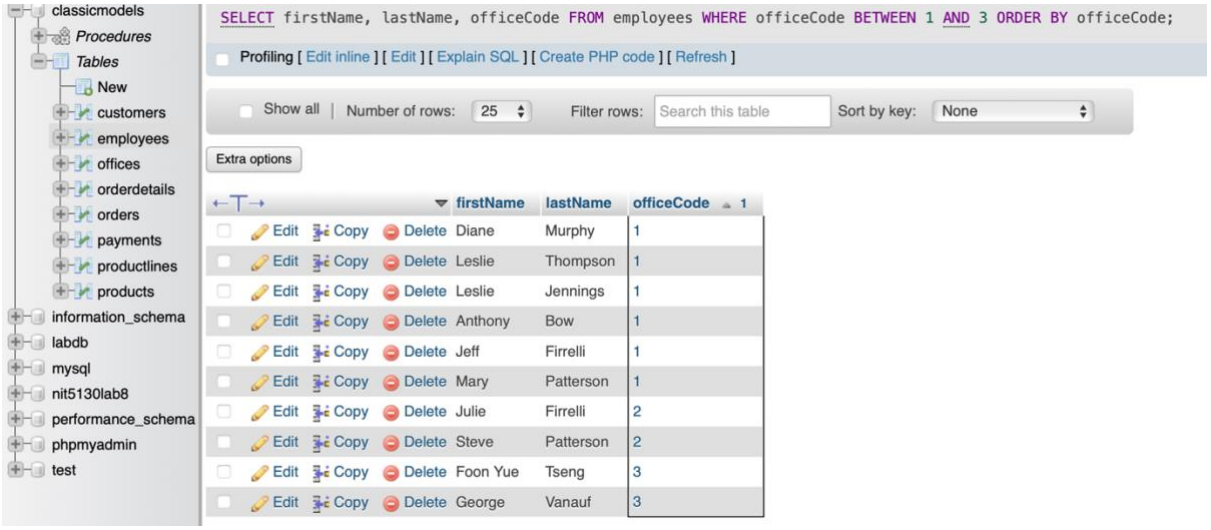
The search_condition is a combination of one or more *expressions* using the logical operator AND, OR and NOT.
When executing a SELECT statement with a WHERE clause, MySQL evaluates the WHERE clause after the FROM clause and before the SELECT and ORDER BY clauses:



Operator	Description
=	Equal to. You can use it with almost any data type.
<> or !=	Not equal to
<	Less than. You typically use it with numeric and date/time data types.
>	Greater than.
<=	Less than or equal to
>=	Greater than or equal to

Q.3 Write a SQL query to retrieve the first names, last names, and office codes of employees who are in offices with office codes between 1 and 3. The results should be ordered by office code.

```
SELECT
    firstName,
    lastName,
    officeCode
FROM
    employees
WHERE
    officeCode BETWEEN 1 AND 3
ORDER BY officeCode;
```



Grouping data: GROUP BY, HAVING, HAVING COUNT, ROLLUP

The GROUP BY Statement in SQL is used to arrange identical data into groups with the help of some functions. i.e. if a particular column has the same values in different rows, then it will arrange these rows in a group.

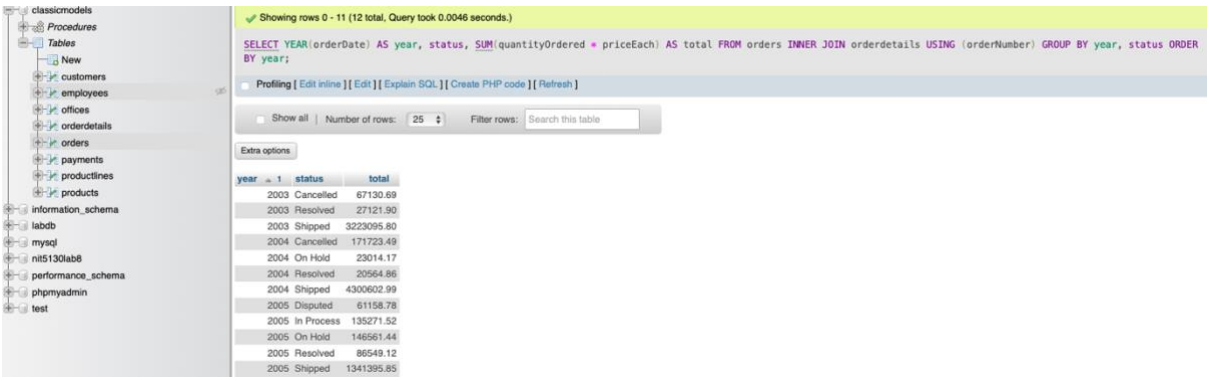
Features

- GROUP BY clause is used with the SELECT statement.
- In the query, the GROUP BY clause is placed after the WHERE clause.
- In the query, the GROUP BY clause is placed before the ORDER BY clause if used.
- In the query, the Group BY clause is placed before the Having clause.
- Place condition in the having clause.

Q.4 How does the total sales revenue vary by order status and year?

```
SELECT
  YEAR(orderDate) AS year,
  status,
  SUM(quantityOrdered * priceEach) AS total
FROM
  orders
  INNER JOIN orderdetails USING (orderNumber)
GROUP BY
  year,
  status
```

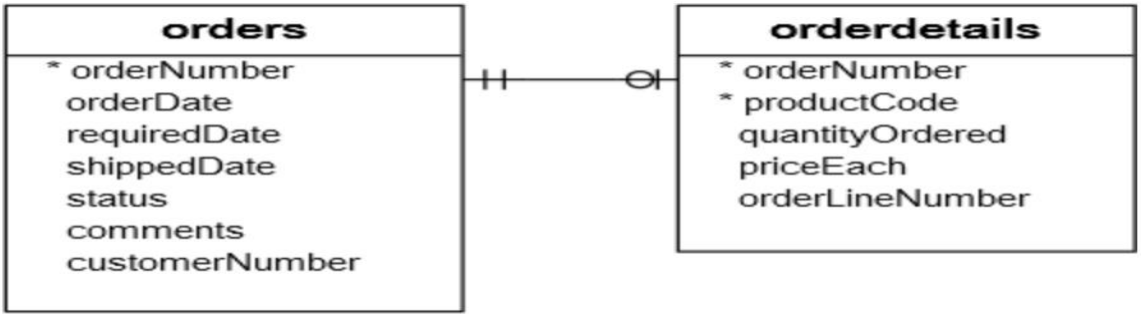
```
ORDER BY
  year;
```



Joining tables: INNER JOIN, LEFT JOIN, SELF JOIN, RIGHT JOIN, SELF JOIN, CROSS JOIN

A relational database is a structured collection of multiple interrelated tables that are linked through common columns known as foreign keys. This design is referred to as the relational model, which is developed based on business requirements or specific scenarios. For a deeper understanding, refer to the concept of Relational Data Modeling in SQL, which illustrates how relationships between tables are structured to support efficient querying and data management.

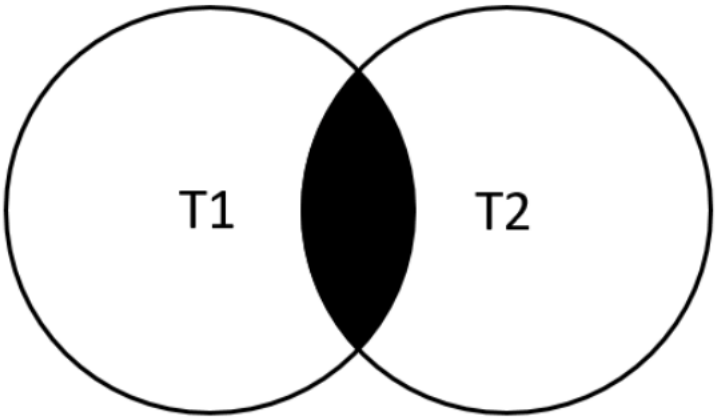
For example, orders and orderdetails are linked using orderNumber column:



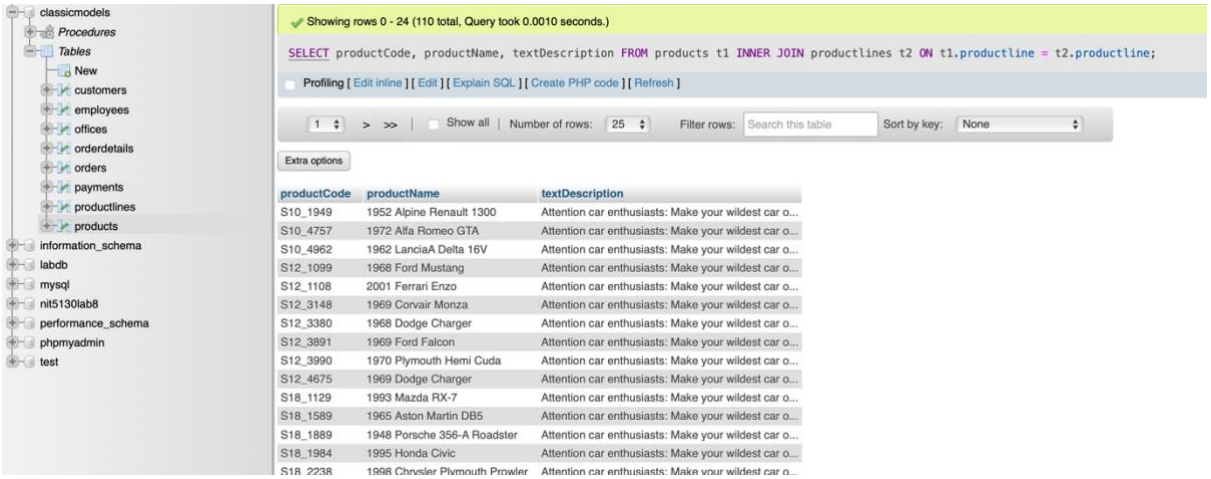
To get complete order information, we need to query data from both orders and orderdetails tables. This is where Joins come into the play. There are different types of joins; Inner join, Left join, Right join, Cross join, self-join.

- Inner join

An INNER JOIN in SQL is a type of join that retrieves rows from two or more tables based on a matching condition between specified columns. It only returns rows where there is a match in both tables.

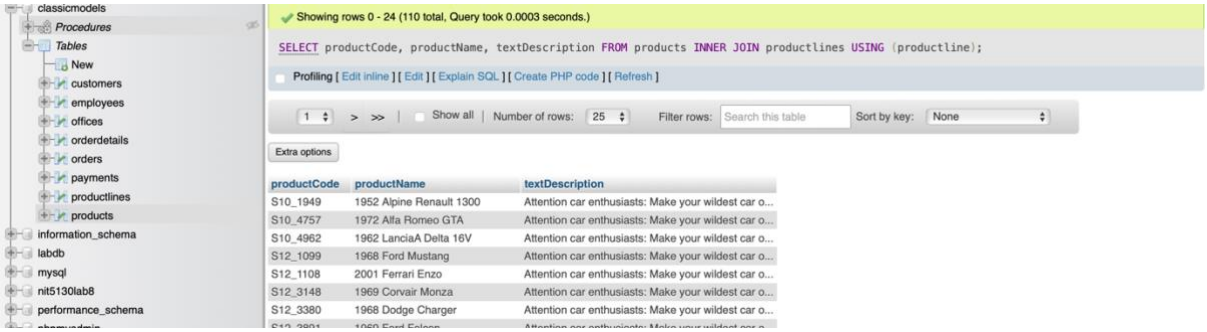



```
SELECT
    productCode,
    productName,
    textDescription
FROM
    products t1
INNER JOIN productlines t2
    ON t1.productline = t2.productline;
```

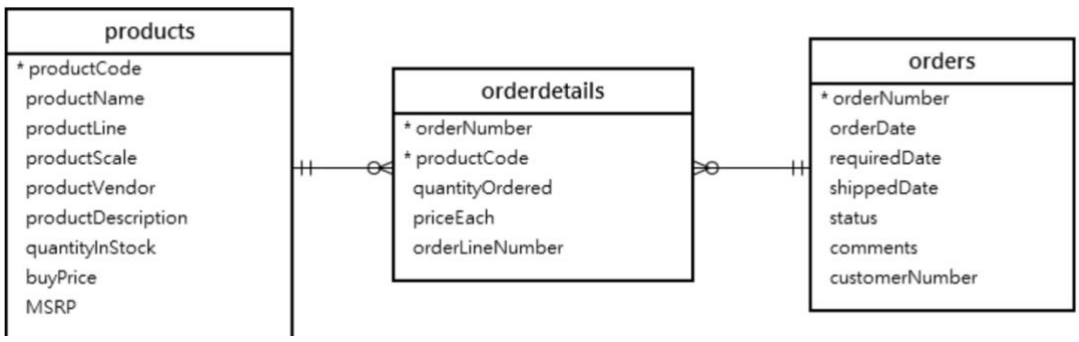


Alternatively, we can join by clause “USING” if the two table have the same column names present in both tables.

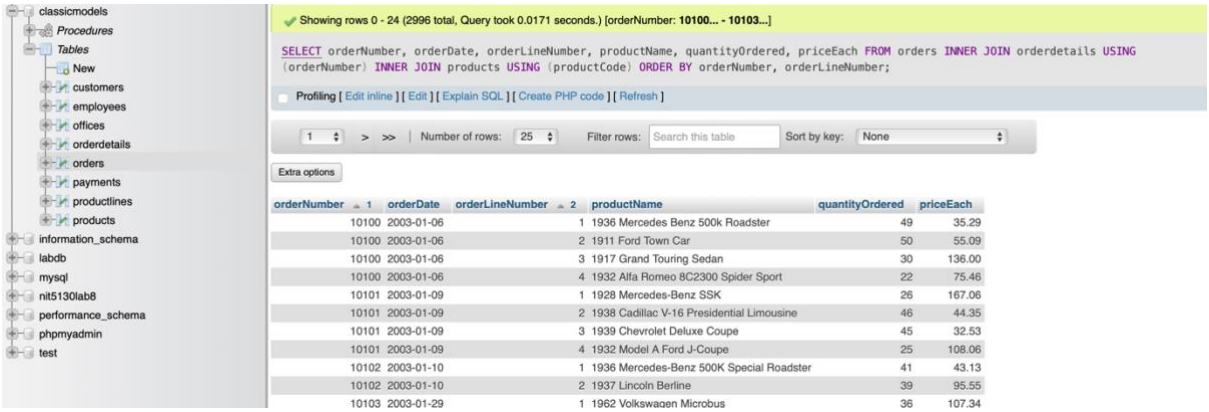
```
SELECT
    productCode,
    productName,
    textDescription
FROM
    products t1
INNER JOIN productlines t2
    USING (productline);
```



INNER JOIN – join three tables example

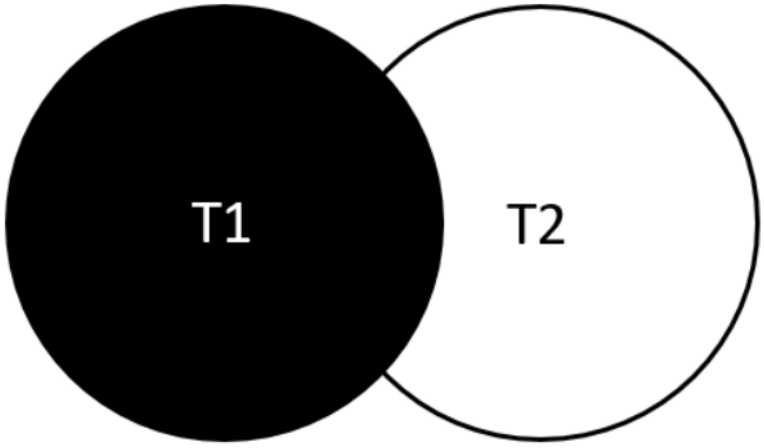


```
SELECT
    orderNumber,
    orderDate,
    orderLineNumber,
    productName,
    quantityOrdered,
    priceEach
FROM
    orders
INNER JOIN
    orderdetails USING (orderNumber)
INNER JOIN
    products USING (productCode)
ORDER BY
    orderNumber,
    orderLineNumber;
```



- ## Left join

A LEFT JOIN (also known as a LEFT OUTER JOIN) in SQL returns all rows from the left table (the first table), and the matched rows from the right table (the second table). If there is no match, the result will contain NULL values for columns from the right table.




```
SELECT
    select_list
FROM
    t1
LEFT JOIN t2 ON
    join_condition;
```

- Right join

A RIGHT JOIN (or RIGHT OUTER JOIN) in SQL returns all rows from the right table (second table), and the matched rows from the left table (first table). If there is no match, the result will contain NULL values for columns from the left table.

- Cross join

A CROSS JOIN in SQL returns the Cartesian product of two tables, meaning it combines every row from the first table with every row from the second table. This join doesn't require any condition and generates all possible combinations of rows between the two tables.

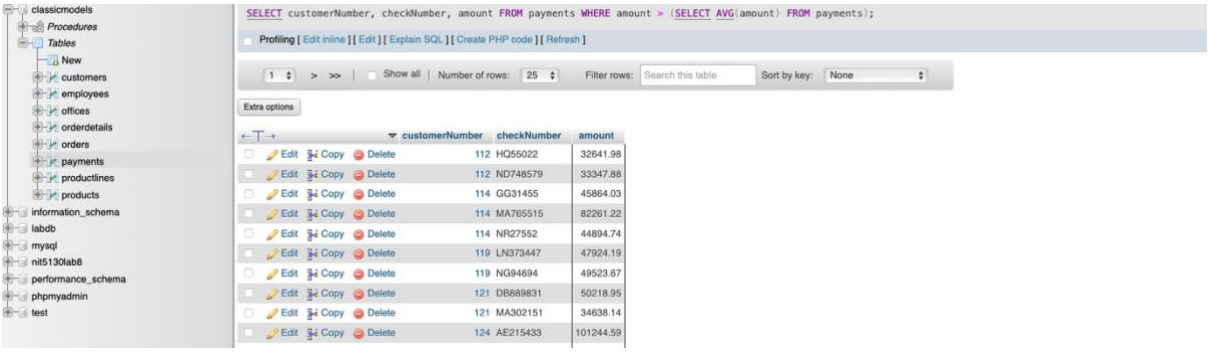
Subqueries: SUBQUERY, DERIVED TABLE, EXISTS

A subquery (also known as an inner query or nested query) is a SQL query embedded within another query. The subquery is executed first, and its result is used by the outer query. Subqueries can be placed in various parts of a SQL statement, including the SELECT, FROM, WHERE, and HAVING clauses.

In simple terms, a subquery is a query within another query. When the subquery is independent of the main query, it executes first. However, in cases where the subquery is correlated with the outer query, the database engine decides the order of execution dynamically, using the subquery's result accordingly. Subqueries must always be enclosed in parentheses and are typically found on the right side of a comparison operator.

Q.5 Which customers have made payments that are above the average payment amount?

```
SELECT
    customerNumber,
    checkNumber,
    amount
FROM
    payments
WHERE
    amount > (SELECT
            AVG(amount)
        FROM
            payments);
```



Common Table Expressions (CTEs)

Common Table Expressions (CTEs) are temporary result sets in SQL that you can define within a query. They are like subqueries but are easier to read and maintain, especially

for complex queries. CTEs are defined using the WITH keyword and exist only for the duration of the query they are used in.

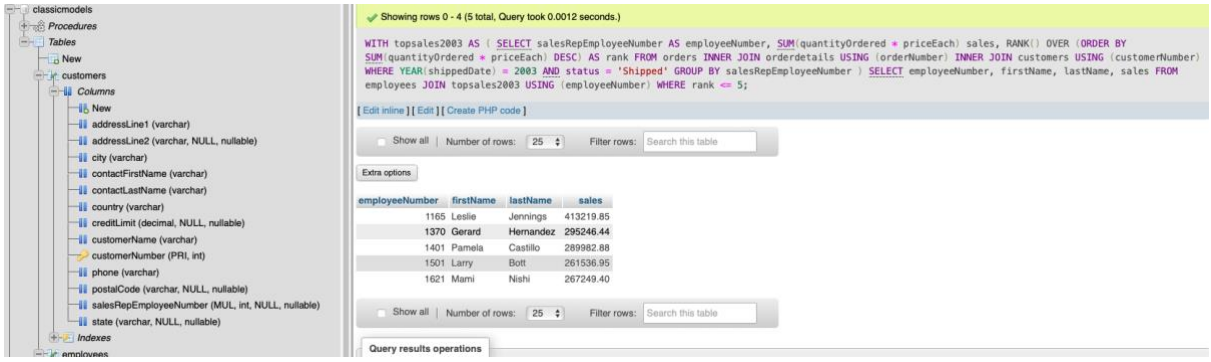
CTEs are useful for:

- Simplifying complex queries by breaking them into smaller, logical parts.
- Improving readability and making queries easier to debug.
- Referencing the result multiple times within the same query.

Q.6 Who was the top 5 sales representatives by total sales in 2003, and what were their total sales amounts?

```
WITH topsales2003 AS (
    SELECT
        salesRepEmployeeNumber AS employeeNumber,
        SUM(quantityOrdered * priceEach) sales,
        RANK() OVER (ORDER BY SUM(quantityOrdered * priceEach) DESC) AS rank
    FROM
        orders
            INNER JOIN
        orderdetails USING (orderNumber)
            INNER JOIN
        customers USING (customerNumber)
    WHERE
        YEAR(shippedDate) = 2003
            AND status = 'Shipped'
    GROUP BY salesRepEmployeeNumber
)
SELECT
    employeeNumber,
    firstName,
    lastName,
    sales
FROM
    employees
        JOIN
    topsales2003 USING (employeeNumber)
WHERE
```

rank <= 5;



Windows Functions

In SQL, a window function performs a calculation across a set of table rows that are related to the current row, but it doesn't group the results into a single output row like aggregate functions do. Instead, it maintains the individual rows while providing a result that is computed over a "window" of data.

The term "window" refers to the set of rows used for the calculation. we define the window by using the OVER clause, which specifies how the rows in the window are partitioned and ordered.

Key Components of Window Functions:

- Function: The calculation you want to perform, such as ROW_NUMBER (), RANK(), SUM(), AVG(), etc.

- OVER clause: Defines the window by specifying partitioning (PARTITION BY) and ordering (ORDER BY).

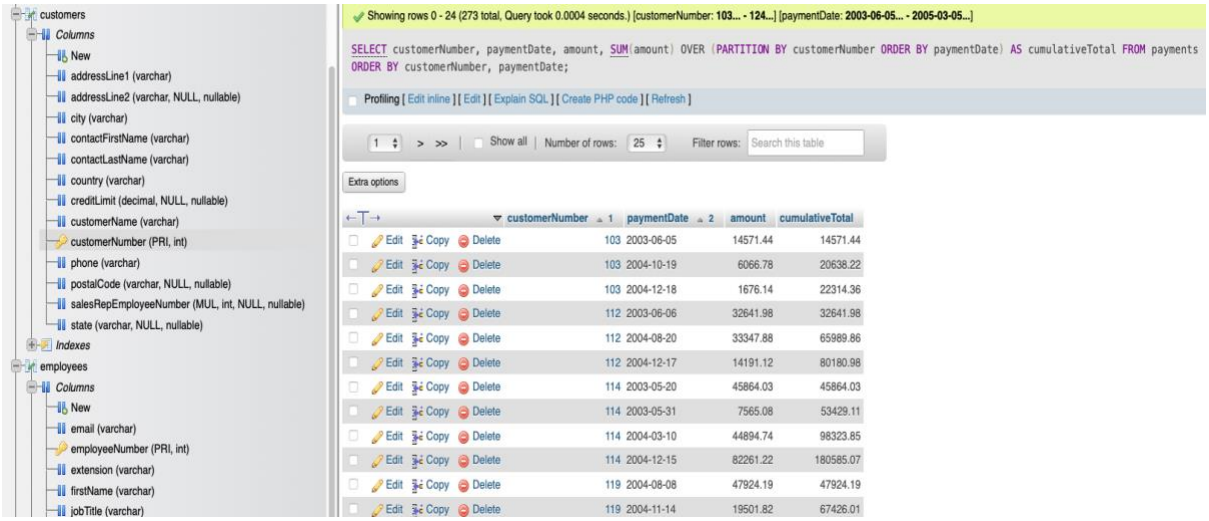
Benefits of subqueries in analytics:

1. *Preserve Row-Level Detail*: Maintain original row details while performing calculations across a set of rows.
2. *Perform Complex Calculations*: Enable advanced operations like ranking, cumulative sums, and moving averages.
3. *Efficiency*: Simplify queries and improve performance by eliminating the need for multiple subqueries or joins.
4. *Partitioned Calculations*: Analyse data within specific segments using the `PARTITION BY` clause for group-specific insights.
5. *Order-Sensitive Calculations*: Conduct order-based computations, such as ranking and time-based comparisons.
6. *Cohort and Time Series Analysis*: Facilitate lag, lead, and running total functions essential for time-based and cohort analysis.
7. *Flexibility*: Combine multiple window functions in a single query to gain comprehensive insights without restructuring data.

8. *Improve Readability*: Simplify SQL code and enhance readability by reducing complex joins and subqueries.

Q.7 Write a query to calculate the cumulative total of payments made by each customer , ordered by the payment date.

```
SELECT
    customerNumber,
    paymentDate,
    amount,
    SUM(amount) OVER (PARTITION BY customerNumber ORDER BY paymentDate) AS cumulativeTotal
FROM
    payments
ORDER BY
    customerNumber, paymentDate;
```



Reference:

MySQL Tutorial. (n.d.). *MySQL Tutorial – Learn MySQL Fast, Easy and Fun.* [online] Available at: https://www.mysqltutorial.org.

GeeksforGeeks (2023). *SQL Server Subquery*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/sql-server-subquery/ [Accessed 6 Sep. 2024].