

Table of Contents

Introduction.....	2
Description of Database:	2
Querying data: SELECT.....	3
Q.1 What are the first name, last name, and job title of each employee in the company? 3	
Sorting data: ORDER BY.....	3
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.	4
Filtering data: WHERE, DISTINCT, AND, OR, IN, NOT IN, BETWEEN, LIKE, LIMIT, IS NULL.....	4
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.	5
Grouping data: GROUP BY, HAVING, HAVING COUNT, ROLLUP	6
Q.4 How does the total sales revenue vary by order status and year?	6
Joining tables: INNER JOIN, LEFT JOIN, SELF JOIN, RIGHT JOIN, SELF JOIN, CROSS JOIN.....	7
• Inner join	8
• Left join	10
• Right join	11
• Cross join	11
Subqueries: SUBQUERY, DERIVED TABLE, EXISTS	12
Q.5 Which customers have made payments that are above the average payment amount?	12
Common Table Expressions (CTEs)	12
Q.6 Who was the top 5 sales representatives by total sales in 2003, and what were their total sales amounts?	13
Windows Functions	14
Q.7 Write a query to calculate the cumulative total of payments made by each customer , ordered by the payment date.....	15

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.

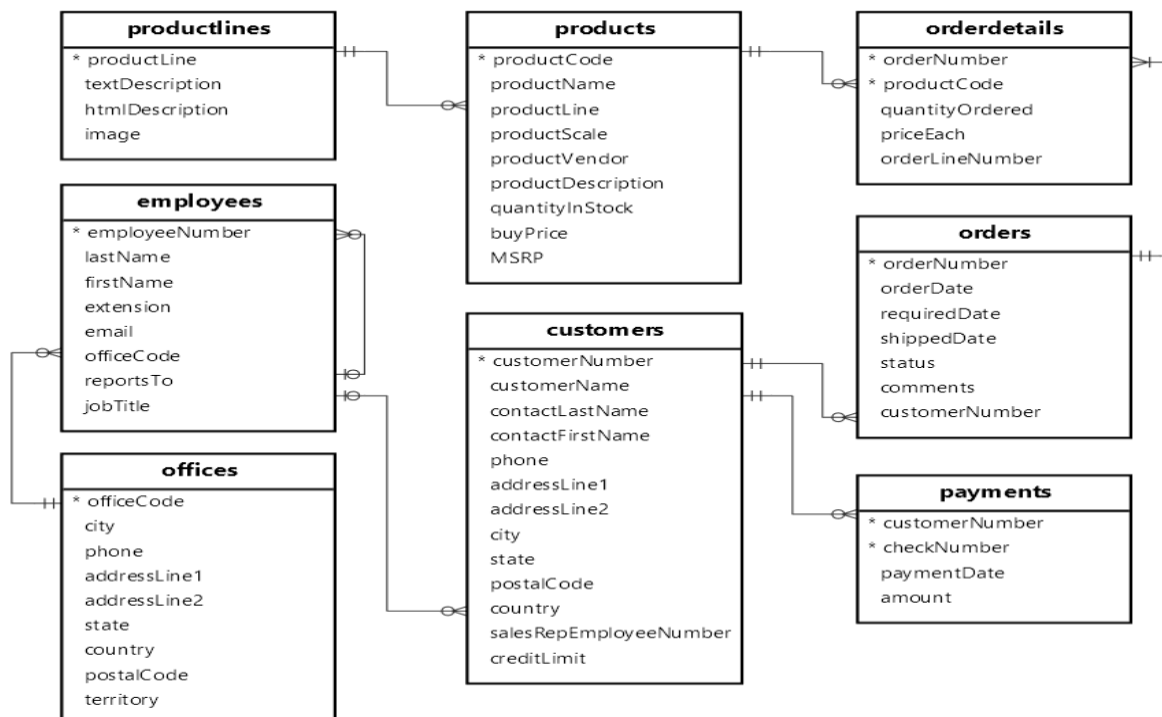
Description of Database:

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;

The screenshot shows a database management interface. On the left is a tree view of the database structure, including schemas like classicmodels, information_schema, and tables like customers, employees, orders, etc. The main area displays the results of a query. At the top, a status bar indicates 'Showing rows 0 - 22 (23 total, Query took 0.0002 seconds.)'. Below this, the SQL query is shown: `SELECT lastName, firstName, jobTitle FROM employees;`. There are links for 'Profiling', 'Edit inline', 'Edit', 'Explain SQL', 'Create PHP code', and 'Refresh'. Below the query, there are controls for 'Show all', 'Number of rows' (set to 25), and a 'Filter rows' search box. An 'Extra options' button is also present. The results are shown in a table with columns: `lastName`, `firstName`, and `jobTitle`. Each row has checkboxes and icons for 'Edit', 'Copy', and 'Delete'.

	lastName	firstName	jobTitle
<input type="checkbox"/>	Murphy	Diane	President
<input type="checkbox"/>	Patterson	Mary	VP Sales
<input type="checkbox"/>	Firrelli	Jeff	VP Marketing
<input type="checkbox"/>	Patterson	William	Sales Manager (APAC)
<input type="checkbox"/>	Bondur	Gerard	Sale Manager (EMEA)
<input type="checkbox"/>	Bow	Anthony	Sales Manager (NA)
<input type="checkbox"/>	Jennings	Leslie	Sales Rep
<input type="checkbox"/>	Thompson	Leslie	Sales Rep
<input type="checkbox"/>	Firrelli	Julie	Sales Rep
<input type="checkbox"/>	Patterson	Steve	Sales Rep
<input type="checkbox"/>	Tseng	Foon Yue	Sales Rep
<input type="checkbox"/>	Vanauf	George	Sales Rep
<input type="checkbox"/>	Bondur	Loui	Sales Rep
<input type="checkbox"/>	Hernandez	Gerard	Sales Rep
<input type="checkbox"/>	Castillo	Pamela	Sales Rep
<input type="checkbox"/>	Bott	Larry	Sales Rep
<input type="checkbox"/>	Jones	Barry	Sales Rep

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 sort 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

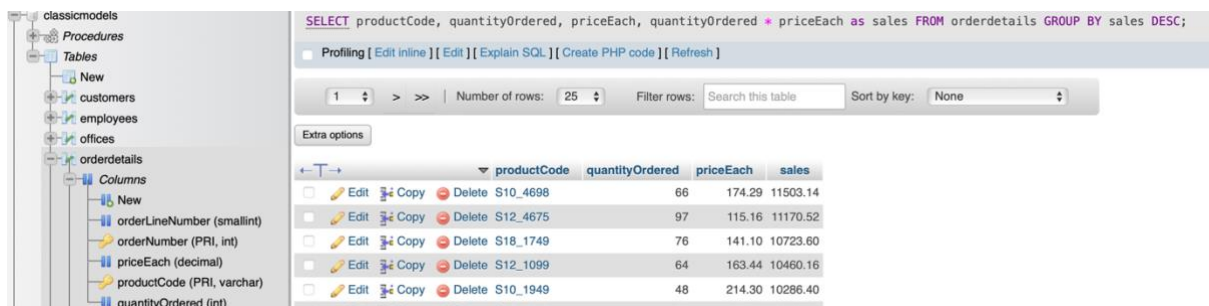
column1 [ASC|DESC],

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 sort in descending order.



The screenshot shows a MySQL database interface. On the left, a tree view displays the database structure, including tables like customers, employees, offices, and orderdetails. The 'orderdetails' table is selected, and its columns are listed: orderLineNumber (smallint), orderNumber (PRI, int), priceEach (decimal), productCode (PRI, varchar), and quantityOrdered (int). On the right, a SQL query is entered in the 'Edit' tab: `SELECT productCode, quantityOrdered, priceEach, quantityOrdered * priceEach as sales FROM orderdetails GROUP BY sales DESC;`. Below the query, a table of results is displayed, sorted by the calculated 'sales' column in descending order. The table has four columns: productCode, quantityOrdered, priceEach, and sales. The results show five rows of data for different product codes.

productCode	quantityOrdered	priceEach	sales
S10_4698	66	174.29	11503.14
S12_4675	97	115.16	11170.52
S18_1749	76	141.10	10723.60
S12_1099	64	163.44	10460.16
S10_1949	48	214.30	10286.40

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 WHERE clause.

SELECT

select_list

FROM

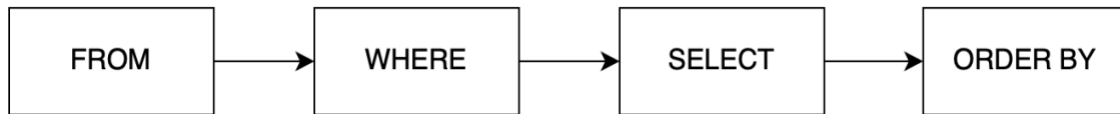
table_name

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;
```

The screenshot shows a database management tool with a sidebar on the left containing a tree view of database objects: classicmodels, Procedures, Tables, New, customers, employees, offices, orderdetails, orders, payments, productlines, products, information_schema, labdb, mysql, nit5130lab8, performance_schema, phpmyadmin, and test.

The main area displays a SQL query: `SELECT firstName, lastName, officeCode FROM employees WHERE officeCode BETWEEN 1 AND 3 ORDER BY officeCode;`

Below the query, there are controls for the query execution: Profiling, Edit inline, Edit, Explain SQL, Create PHP code, and Refresh. There are also options to Show all, Number of rows (set to 25), Filter rows (Search this table), and Sort by key (set to None).

The results are displayed in a table with columns: firstName, lastName, and officeCode. The table contains 10 rows of data, grouped by officeCode.

firstName	lastName	officeCode
Diane	Murphy	1
Leslie	Thompson	1
Leslie	Jennings	1
Anthony	Bow	1
Jeff	Firrelli	1
Mary	Patterson	1
Julie	Firrelli	2
Steve	Patterson	2
Foon Yue	Tseng	3
George	Vanauf	3

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;

Showing rows 0 - 11 (12 total. Query took 0.0046 seconds.)

```
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;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table

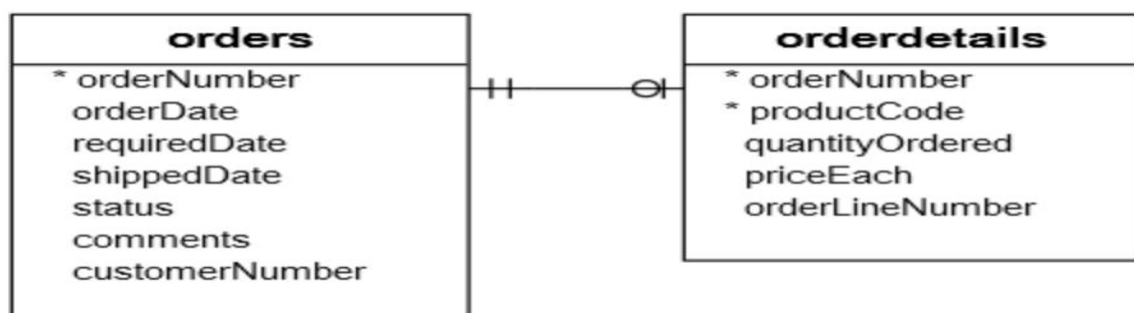
Extra options

year	status	total
2003	Cancelled	67130.69
2003	Resolved	27121.90
2003	Shipped	3223095.80
2004	Cancelled	171723.49
2004	On Hold	23014.17
2004	Resolved	20564.86
2004	Shipped	4300602.99
2005	Disputed	61158.78
2005	In Process	135271.52
2005	On Hold	146561.44
2005	Resolved	86549.12
2005	Shipped	1341395.85

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.

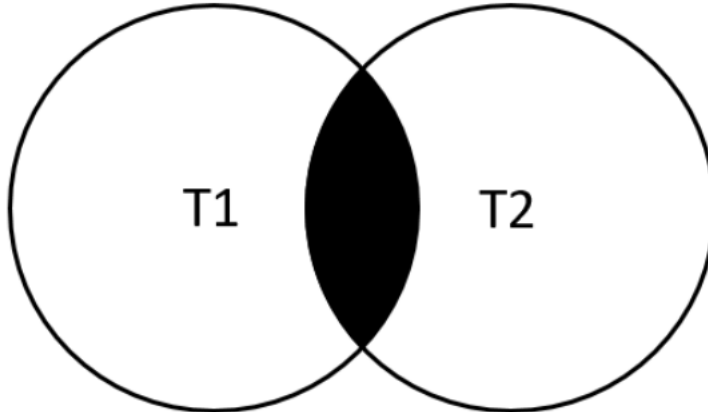
For example, orders and orderdetails are linked using orderNumber column:



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;
```

Showing rows 0 - 24 (110 total, Query took 0.0010 seconds.)

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

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

1 > >> Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

productCode	productName	textDescription
S10_1949	1952 Alpine Renault 1300	Attention car enthusiasts: Make your wildest car o...
S10_4757	1972 Alfa Romeo GTA	Attention car enthusiasts: Make your wildest car o...
S10_4962	1962 LanciaA Delta 16V	Attention car enthusiasts: Make your wildest car o...
S12_1099	1968 Ford Mustang	Attention car enthusiasts: Make your wildest car o...
S12_1108	2001 Ferrari Enzo	Attention car enthusiasts: Make your wildest car o...
S12_3148	1969 Corvair Monza	Attention car enthusiasts: Make your wildest car o...
S12_3380	1968 Dodge Charger	Attention car enthusiasts: Make your wildest car o...
S12_3891	1969 Ford Falcon	Attention car enthusiasts: Make your wildest car o...
S12_3990	1970 Plymouth Hemi Cuda	Attention car enthusiasts: Make your wildest car o...
S12_4675	1969 Dodge Charger	Attention car enthusiasts: Make your wildest car o...
S18_1129	1993 Mazda RX-7	Attention car enthusiasts: Make your wildest car o...
S18_1589	1965 Aston Martin DB5	Attention car enthusiasts: Make your wildest car o...
S18_1889	1948 Porsche 356-A Roadster	Attention car enthusiasts: Make your wildest car o...
S18_1984	1995 Honda Civic	Attention car enthusiasts: Make your wildest car o...
S18_2238	1998 Chrysler Plymouth Prowler	Attention car enthusiasts: Make your wildest car o...

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

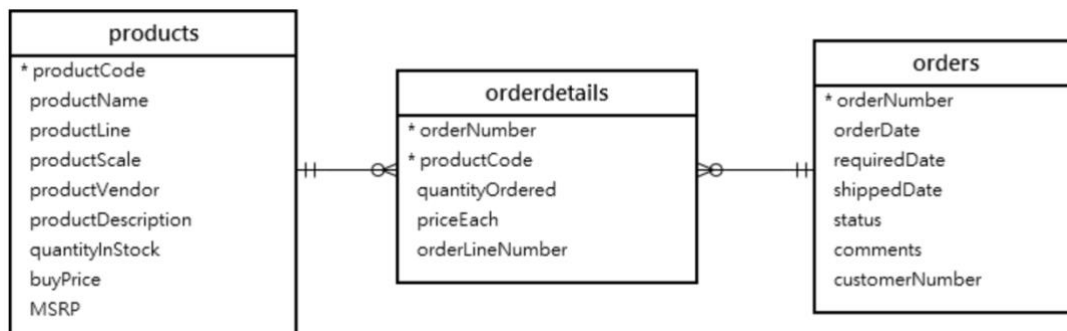
INNER JOIN productlines t2

USING (productline);

The screenshot shows a database management tool interface. On the left is a tree view of the database schema. The main area displays a query and its results. The query is: `SELECT productCode, productName, textDescription FROM products INNER JOIN productlines USING (productline);`. The results table has three columns: productCode, productName, and textDescription. It shows 25 rows of data, including car models like Alpine Renault 1300, Alfa Romeo GTA, Lancia Delta 16V, Ford Mustang, Ferrari Enzo, and Corvair Monza.

productCode	productName	textDescription
S10_1949	1952 Alpine Renault 1300	Attention car enthusiasts: Make your wildest car o...
S10_4757	1972 Alfa Romeo GTA	Attention car enthusiasts: Make your wildest car o...
S10_4962	1962 Lancia Delta 16V	Attention car enthusiasts: Make your wildest car o...
S12_1099	1968 Ford Mustang	Attention car enthusiasts: Make your wildest car o...
S12_1108	2001 Ferrari Enzo	Attention car enthusiasts: Make your wildest car o...
S12_3148	1969 Corvair Monza	Attention car enthusiasts: Make your wildest car o...
S12_3380	1968 Dodge Charger	Attention car enthusiasts: Make your wildest car o...

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;

```

Showing rows 0 - 24 (2996 total, Query took 0.0171 seconds.) [orderNumber: 10100... - 10103...]

```

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

```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

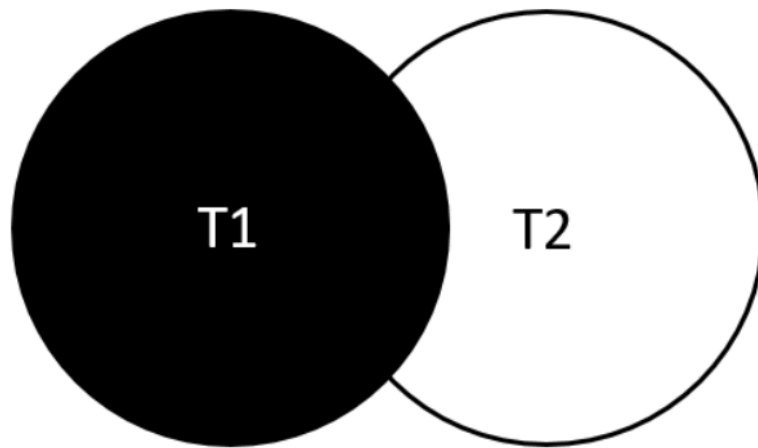
1 > >> | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

orderNumber	orderDate	orderLineNumber	productName	quantityOrdered	priceEach
10100	2003-01-06	1	1936 Mercedes Benz 500k Roadster	49	35.29
10100	2003-01-06	2	1911 Ford Town Car	50	55.09
10100	2003-01-06	3	1917 Grand Touring Sedan	30	136.00
10100	2003-01-06	4	1932 Alfa Romeo 8C2300 Spider Sport	22	75.46
10101	2003-01-09	1	1928 Mercedes-Benz SSK	26	167.06
10101	2003-01-09	2	1938 Cadillac V-16 Presidential Limousine	46	44.35
10101	2003-01-09	3	1939 Chevrolet Deluxe Coupe	45	32.53
10101	2003-01-09	4	1932 Model A Ford J-Coupe	25	108.06
10102	2003-01-10	1	1936 Mercedes-Benz 500K Special Roadster	41	43.13
10102	2003-01-10	2	1937 Lincoln Berline	39	95.55
10103	2003-01-29	1	1962 Volkswagen Microbus	36	107.34

- 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

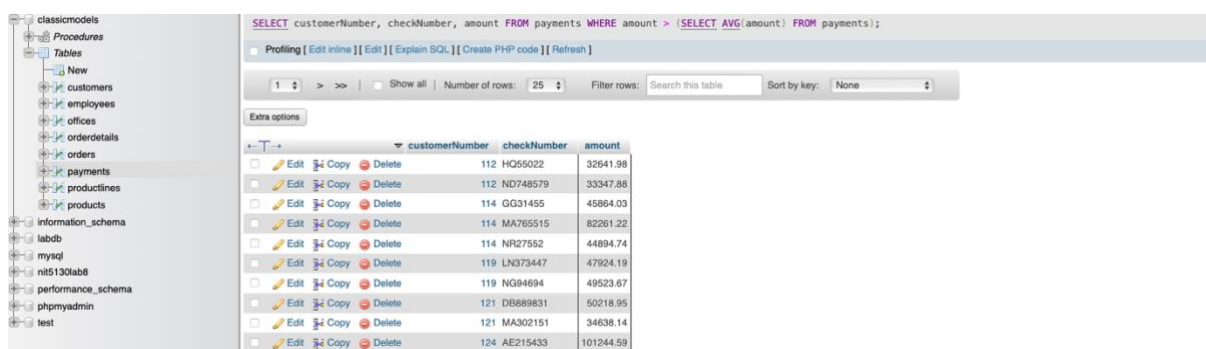
customerNumber,
checkNumber,
amount

FROM

payments

WHERE

amount > (SELECT
AVG(amount)
FROM
payments);



	customerNumber	checkNumber	amount
<input type="checkbox"/>	112	HQ55022	32641.98
<input type="checkbox"/>	112	ND748579	33347.88
<input type="checkbox"/>	114	GG31455	45864.03
<input type="checkbox"/>	114	MA765515	82261.22
<input type="checkbox"/>	114	NR27552	44894.74
<input type="checkbox"/>	119	LN373447	47924.19
<input type="checkbox"/>	119	NG94694	49523.67
<input type="checkbox"/>	121	DB889831	50218.95
<input type="checkbox"/>	121	MA302151	34638.14
<input type="checkbox"/>	124	AE215433	101244.59

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;

The screenshot shows a database management interface. On the left, a tree view displays the database structure, including tables like 'customers' and 'employees'. The main area displays a SQL query that uses a window function to rank sales by employee. The query is as follows:

```
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;
```

Below the query, the results are displayed in a table with columns: employeeNumber, firstName, lastName, and sales. The results show the top 5 sales for each employee in 2003.

employeeNumber	firstName	lastName	sales
1165	Leslie	Jennings	413219.85
1370	Gerard	Hernandez	295246.44
1401	Pamela	Castillo	289982.88
1501	Larry	Bott	261536.95
1621	Mami	Nishi	267249.40

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;

The screenshot shows a MySQL database interface. On the left, a tree view displays the database schema with tables 'customers' and 'employees'. The 'customers' table is selected, showing its columns: addressLine1, addressLine2, city, contactFirstName, contactLastName, country, creditLimit, customerName, customerNumber (primary key), phone, postalCode, salesRepEmployeeNumber, and state. The main area displays the results of a SQL query. The query is: `SELECT customerNumber, paymentDate, amount, SUM(amount) OVER (PARTITION BY customerNumber ORDER BY paymentDate) AS cumulativeTotal FROM payments ORDER BY customerNumber, paymentDate;` The results show 25 rows of payment data, grouped by customer number. The columns are: customerNumber, paymentDate, amount, and cumulativeTotal. The cumulativeTotal column shows the running sum of payments for each customer, ordered by payment date.

customerNumber	paymentDate	amount	cumulativeTotal
103	2003-06-05	14571.44	14571.44
103	2004-10-19	6066.78	20638.22
103	2004-12-18	1676.14	22314.36
112	2003-06-06	32641.98	32641.98
112	2004-08-20	33347.88	65989.86
112	2004-12-17	14191.12	80180.98
114	2003-05-20	45864.03	45864.03
114	2003-05-31	7565.08	53429.11
114	2004-03-10	44894.74	98323.85
114	2004-12-15	82261.22	180585.07
119	2004-08-08	47924.19	47924.19
119	2004-11-14	19501.82	67426.01

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].