**SQL Documentation**

In SQL, there are four main types of language commands, each serving a specific purpose. These are DCL (Data Control Language), DDL (Data Definition Language), DML (Data Manipulation Language), and TCL (Transaction Control Language). Let's break down each one with definitions, examples, and sample SQL queries.

1. ***Data Control Language (DCL)***

DCL is used to control access to data in a database. It deals with permissions and access control.

1. **GRANT:** Gives a user access privileges to the database.

**>>** GRANT SELECT ON employees TO user\_name;

1. **REVOKE:** Removes user access privileges to the database.

**>>** REVOKE SELECT ON employees FROM user\_name;

1. ***Data Definition Language (DDL)***

DDL is used to define and modify the structure of database objects like tables, indexes, and schemas. It directly impacts the structure of the database.

1. **CREATE:** Creates new database objects (tables, views, indexes).

**>>** CREATE TABLE employees (id INT PRIMARY KEY, name VARCHAR(50), department VARCHAR(50), salary INT );

1. **ALTER:** Modifies the structure of existing database objects.

**>>** ALTER TABLE employees ADD date\_of\_joining DATE;

1. **TRUNCATE:** Removes all records from a table but keeps its structure.

**>>** TRUNCATE TABLE employees;

1. **DROP:** Deletes the complete data and structure.

**>>** DROP TABLE employees;

1. ***Data Manipulation Language (DML)***

DML is used for managing data within schema objects. It focuses on manipulating the data present in the database.

1. **INSERT:** Adds new data into a table.

**>>** INSERT INTO employees (id, name, department, salary) VALUES (1, ‘Adam’, ‘HR’, 50000);

1. **UPDATE:** Modifies existing data within a table.

**>>** UPDATE employees SET salary=55000 WHERE id=1;

1. **DELETE:** Removes existing data from a table.

**>>** DELETE FROM employees WHERE id=1;

1. **SELECT:** Select existing data within a table.

**>>** SELECT \* FROM employees;

1. ***Transaction Control Language (TCL)***

TCL is used to manage the changes made by DML statements. It ensures the integrity of the database by controlling transactions.

1. **COMMIT:** Save all changes made during the current transaction.

**>>** COMMIT;

1. **ROLLBACK:** Undoes all changes made during the current transaction.

**>>** ROLLBACK TO sp1;

1. **SAVEPOINT:** Sets a point within a transaction to which you can later roll back.

**>>** SAVEPOINT sp1;

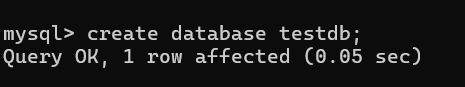
1. **SET TRANSACTION:** Set properties for the current transaction (like isolation level).

**>>** SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

**DATABASE**

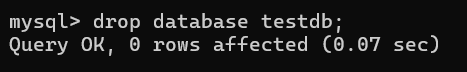
1. ***Create Database***
   * Used to create a new SQL database.

>> CREATE DATABASE testdb;



1. ***Drop Database***
   * Used to drop an existing SQL database.

>> DROP DATABASE testdb;



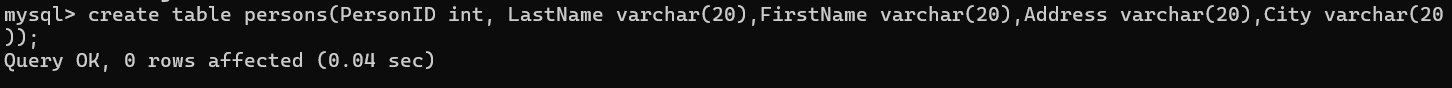
1. ***Select a specific database***
   * Used to select a database to manage tables.

>> USE fsds;



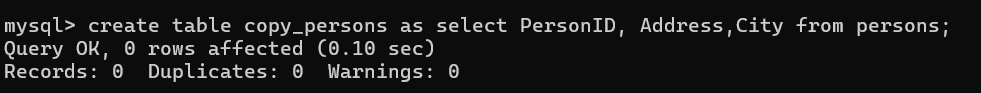
1. ***Create Table***
   * used to create a new table in a database.

>> CREATE TABLE persons(PersonID INT, LastName VARCHAR(20), FirstName VARCHAR(20), Address VARCHAR(20), City VARCHAR(20));



1. ***Create Table Using Another Table***
   * A copy of an existing table can also be created using CREATE TABLE. The new table gets the same column definitions. All columns or specific columns can be selected. If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

>> CREATE TABLE copy\_persons AS SELECT PersonID, Address, City FROM persons;



1. ***Drop Table***
   * The following SQL statement drops the existing table.

>> DROP TABLE copy\_persons;



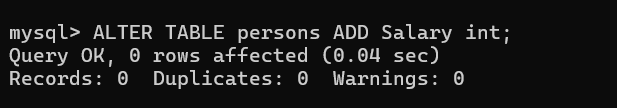
1. ***Truncate Table***
   * used to delete the data inside a table, but not the table itself.

>> TRUNCATE TABLE persons;



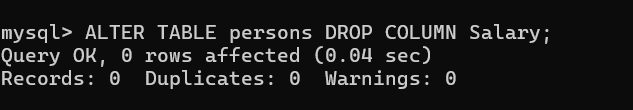
1. ***Alter Table- ADD Column***
   * To add a column in a persons table

>> ALTER TABLE persons ADD Salary int;



1. ***Alter Table- Drop Column***
   * To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column)

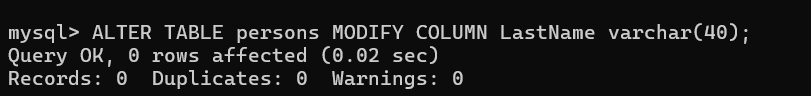
>> ALTER TABLE persons DROP COLUMN Salary;



1. ***Alter Table- Rename & Modify Column***
   * To rename & change the data type of a column in a table.

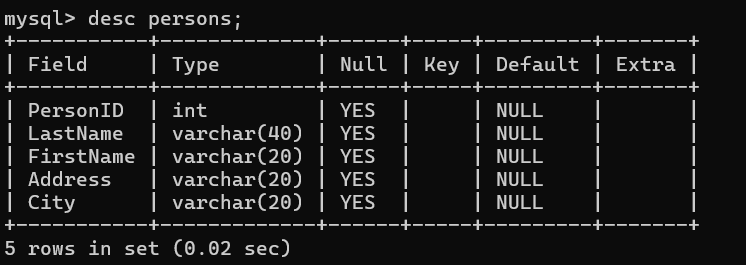
>> ALTER TABLE persons RENAME COLUMN old\_name TO new\_name;

>> ALTER TABLE persons MODIFY COLUMN LastName varchar(40);



1. ***Describe Table***

>> DESC persons;



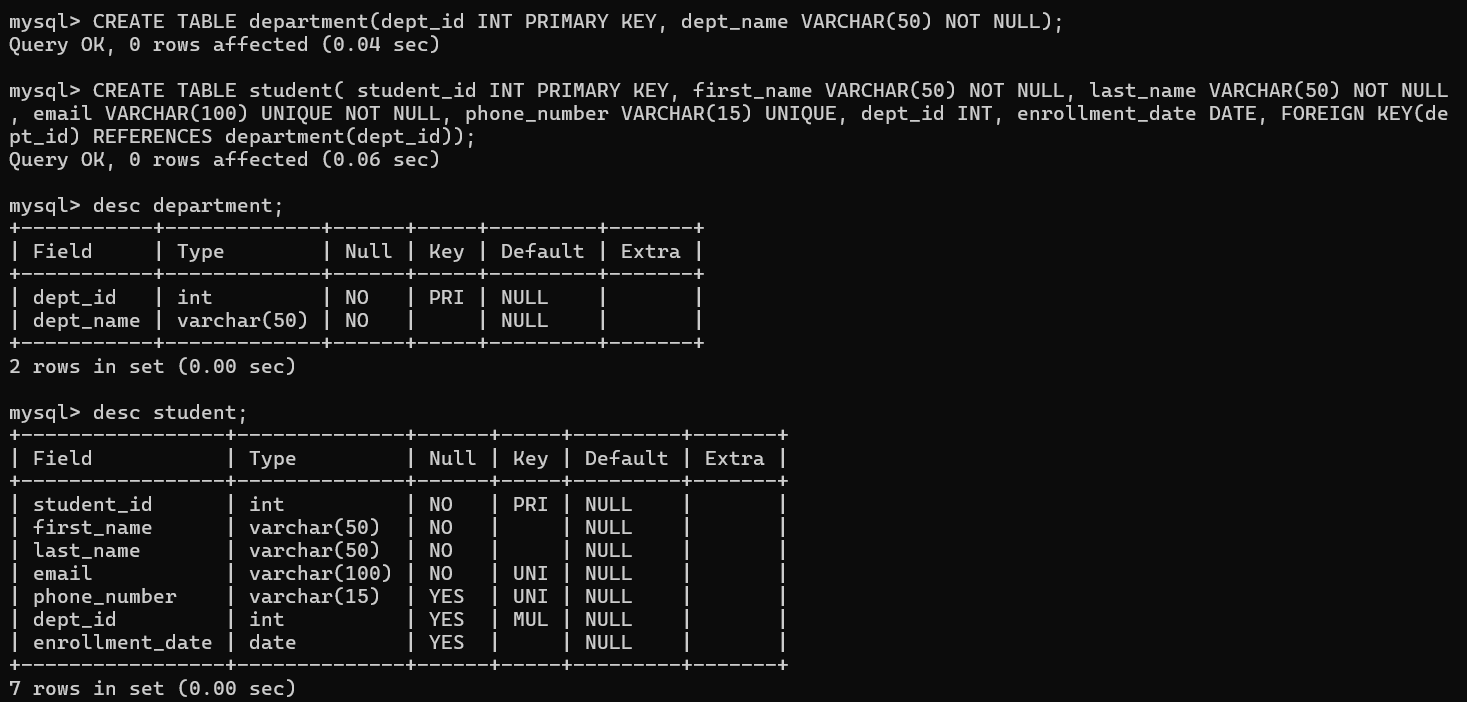
1. ***MySQL Constraints***
   * *NOT NULL:* Ensures that a column cannot have a NULL value
   * *UNIQUE:* Ensures that all values in a column are different
   * *PRIMARY KEY:* A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
   * *FOREIGN KEY:* Prevents actions that would destroy links between tables
   * *CHECK:* Ensures that the values in a column satisfy a specific condition
   * *DEFAULT:* Ensures that the values in a column satisfy a specific condition
   * *CREATE INDEX:* Used to create and retrieve data from the database very quickly

* Create the department table for foreign key reference

>> CREATE TABLE department(dept\_id INT PRIMARY KEY, dept\_name VARCHAR(50) NOT NULL);

* Create the student table

>> CREATE TABLE student(student\_id INT PRIMARY KEY, first\_name VARCHAR(50) NOT NULL, last\_name VARCHAR(50) NOT NULL, email VARCHAR(100) UNIQUE NOT NULL, phone\_number VARCHAR(15) UNIQUE, dept\_id INT, enrollment\_date DATE, FOREIGN KEY(dept\_id) REFERENCES department(dept\_id));



Explanation:

* student\_id: The primary key for the student table.
* first\_name and last\_name: Both fields are required and cannot be NULL.
* email: Must be unique and cannot be NULL.
* phone\_number: Must be unique but can be NULL if a student doesn’t provide it.
* dept\_id: A foreign key referencing the dept\_id in the department table, allowing us to associate a student with a specific department.

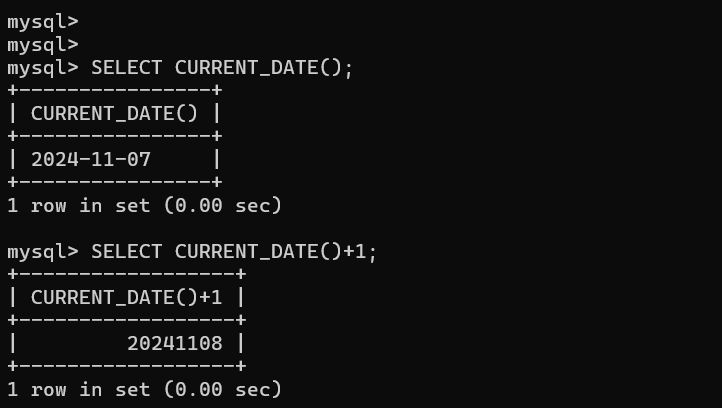
1. ***Auto Increment field***
   * Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

>> CREATE TABLE test(id INT NOT NULL AUTO\_INCREMENT, Name varchar(20), PRIMARY KEY(id));

1. ***Working with Dates***

>> SELECT CURRENT\_DATE();

>> SELECT CURRENT\_DATE()+1;

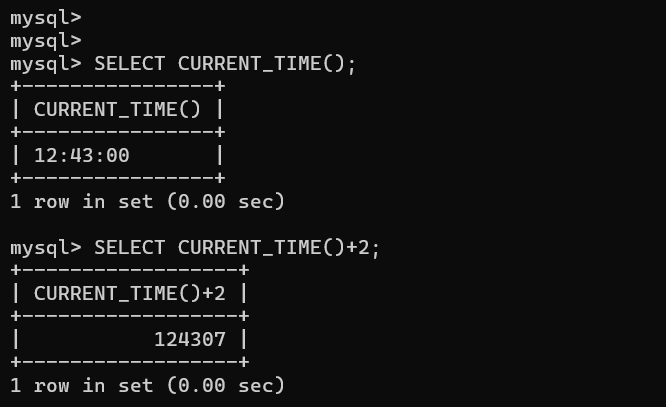


>> SELECT \* FROM table WHERE date=’2023-10-08’;

1. ***Working with time***

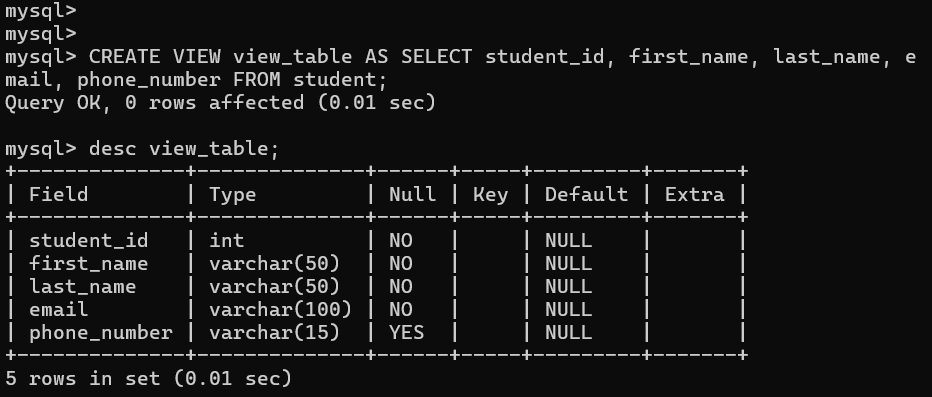
>> SELECT CURRENT\_TIME();

>> SELECT CURRENT\_TIME()+2;



1. ***Create View***
   * In SQL, a view is a virtual table based on the result set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

>> CREATE VIEW view\_table AS SELECT student\_id, first\_name, last\_name, email, phone\_number FROM student;



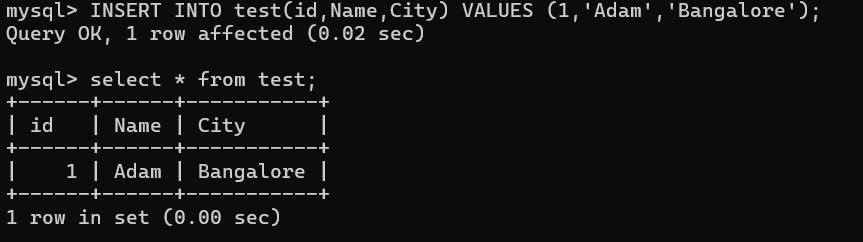
**SQL**

1. ***Insert***

Used to select data from a database

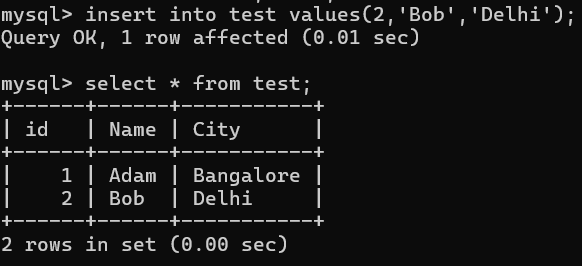
* Secure Insert

>> INSERT INTO test(id, Name, City) VALUES (1, ‘Adam’, ‘Bangalore’);

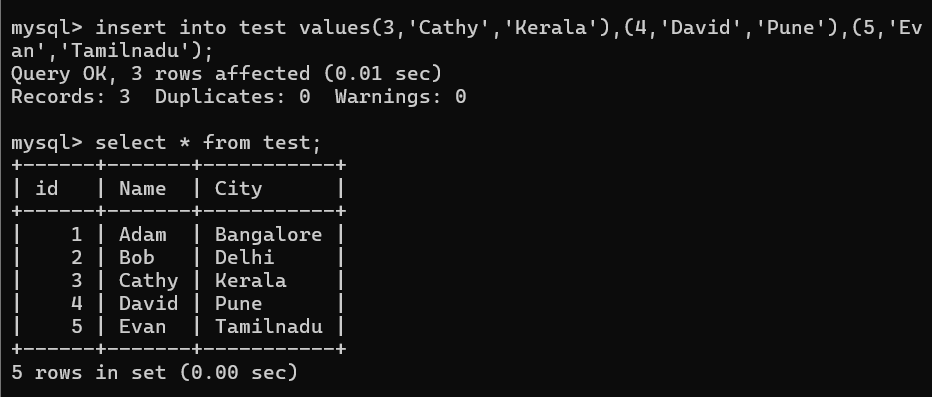


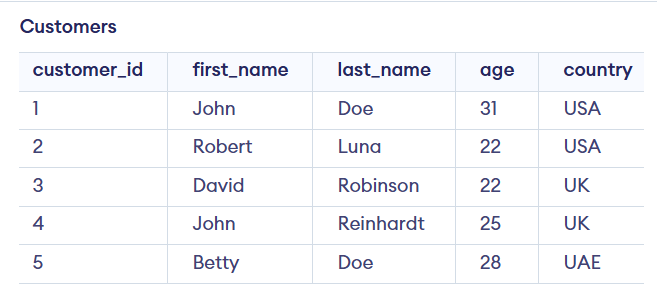
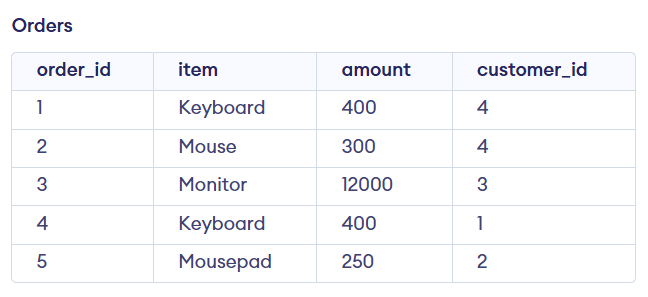
* Insecure Insert

>> INSERT INTO test VALUES(2,’Bob’,’Delhi);



>> INSERT INTO test VALUES(3,’Cathy’,’Kerala’),(4,’David’,’Pune’),(5,’Evan’,’Tamilnadu’);

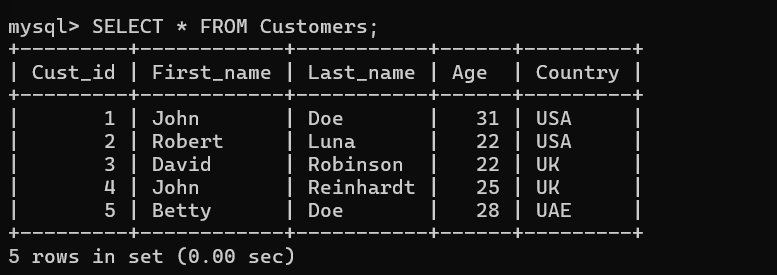


** **

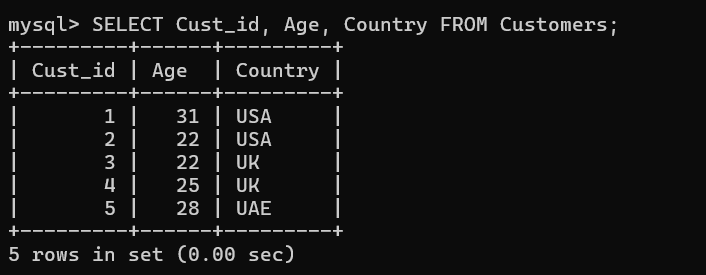
1. ***Select***

Used to select data from a database.

>> SELECT \* FROM Customers;

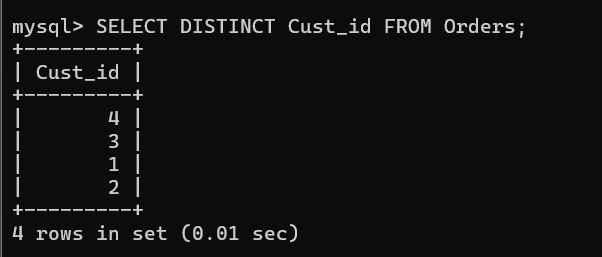


>> SELECT Cust\_id, Age, Country FROM Customers;



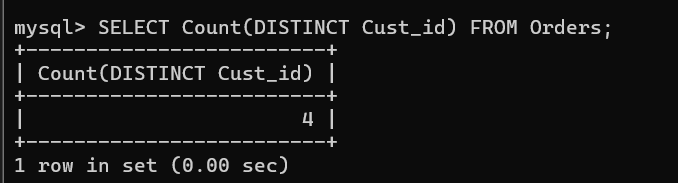
>> SELECT DISTINCT Cust\_id FROM Orders;

Gives the unique values of column id.



>> SELECT COUNT(DISTINCT Cust\_id) FROM Orders;

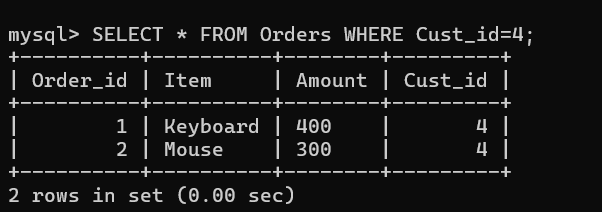
Gives the count of unique values of column id.



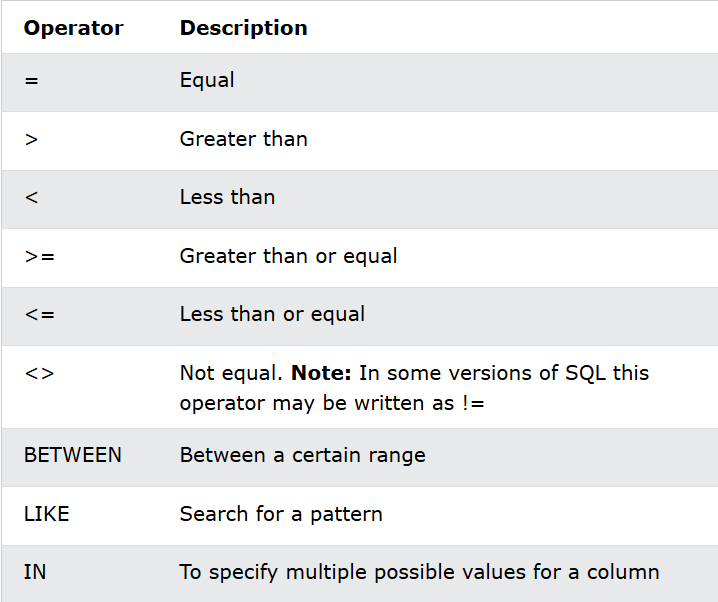
1. ***Where Clause***

Used to filter the records. The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc.!

>>SELECT \* FROM Orders WHERE Cust\_id=4;

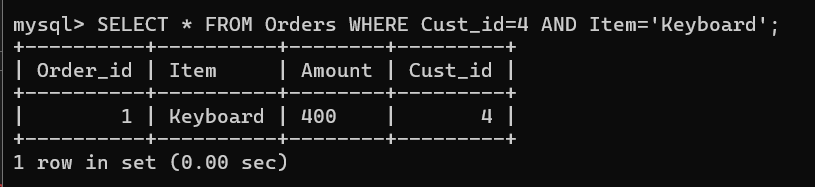


* Operators in Where clause

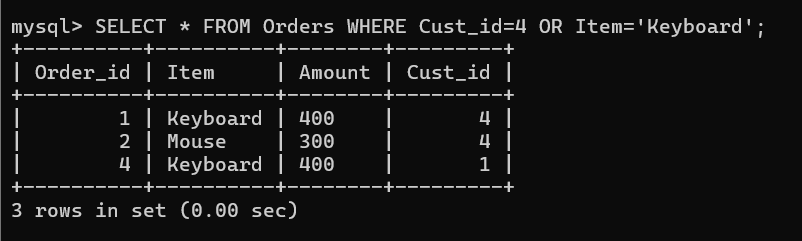


1. ***MySQL AND, OR and NOT operators***

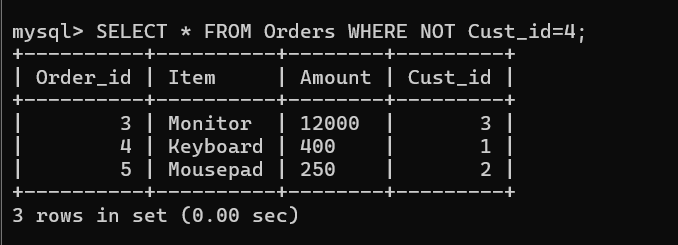
>>SELECT \* FROM Orders WHERE Cust\_id=4 AND Item=’Keyboard’;



>> SELECT \* FROM Orders WHERE Cust\_id=4 OR Item=’Keyboard’;



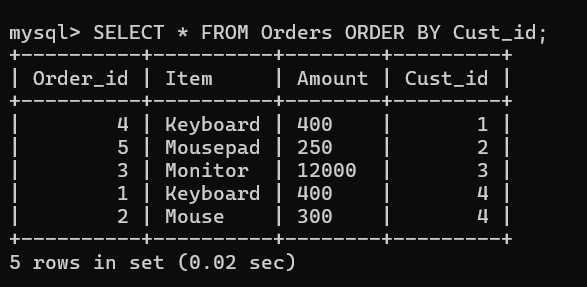
>>SELECT \* FROM Orders WHERE NOT Cust\_id=4;



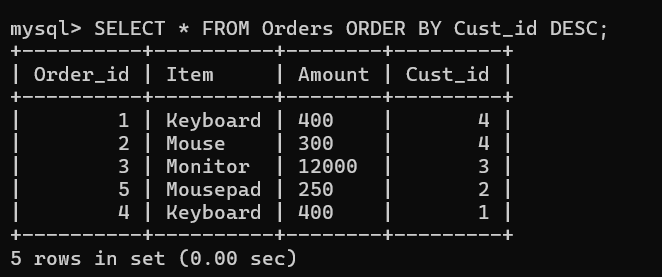
1. ***Order By Keyword***

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

>>SELECT \* FROM Orders ORDER BY Cust\_id;



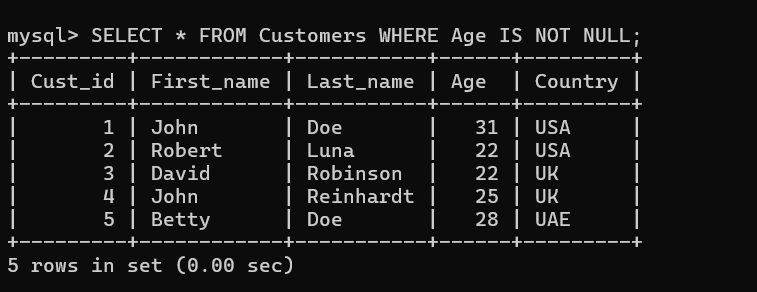
>>SELECT \* FROM Orders ORDER BY Cust\_id DESC;



1. ***NULL Values***

A field with a NULL value is a field with no value.

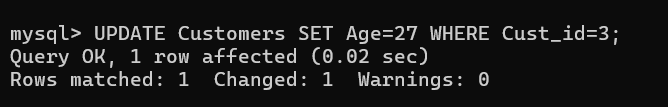
>>SELECT \* FROM Customers WHERE Age IS NOT NULL;



1. ***Update Statement***

Used to modify the existing records in a table.

>>UPDATE Customers SET Age=27 WHERE Cust\_id=3;



1. **Delete Statement**

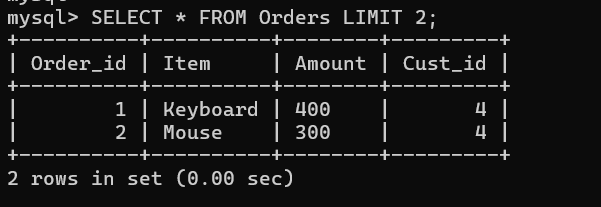
Used to delete existing records in a table. While deleting make sure to check the WHERE clause. If you omit the WHERE clause, all records in the table will be deleted.

>>DELETE FROM Customers WHERE Cust\_id=5

1. ***Limit Statement***

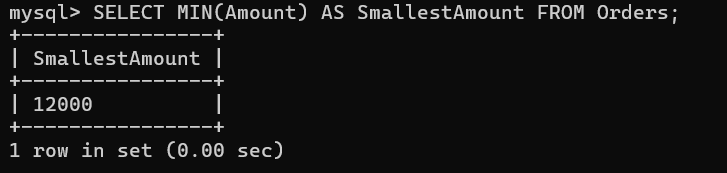
This clause is used to specify the number of records to return. Useful on large tables with thousand of records.

>>SELECT \* FROM Customers LIMIT 2;

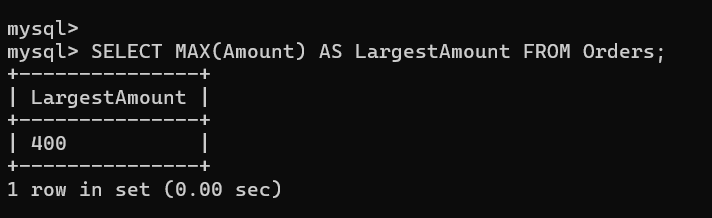


1. ***Min & Max Functions***

>>SELECT MIN(Amount) AS SmallestAmout FROM Orders;

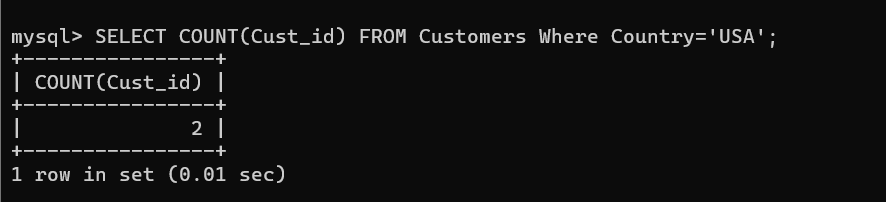
****

>>SELECT MAX(Amount) AS LargestAmount FROM Orders;

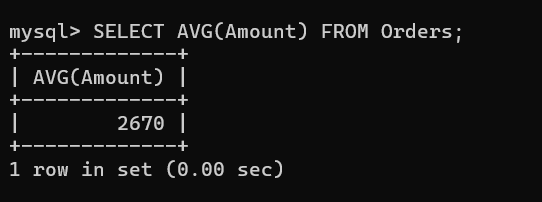
****

1. ***Count, Average, Sum Functions***

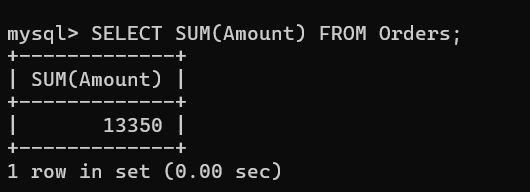
>>SELECT COUNT(Cust\_id) FROM Customers Where Country=’USA’;



>>SELECT AVG(Amount) FROM Orders;



>>SELECT SUM(Amount) FROM Orders;



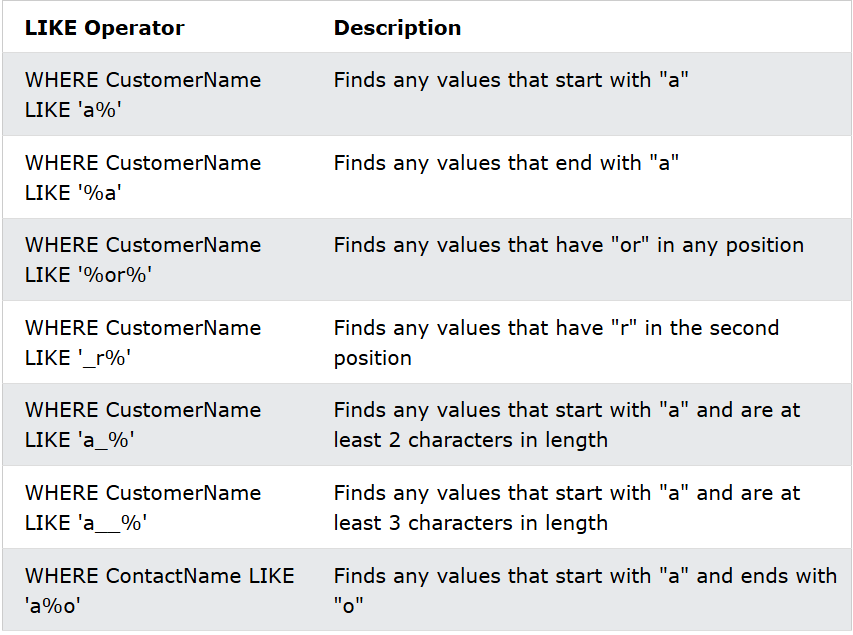
1. ***Wild Cards (Like Operator)***

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

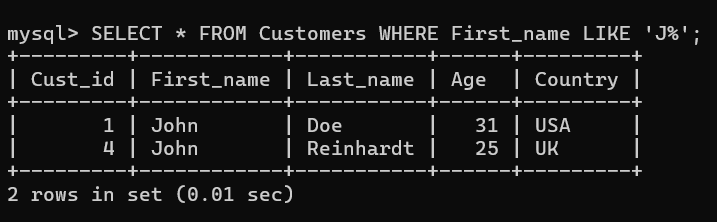
* The percent sign (%) represents zero, one, or multiple characters
* The underscore sign (\_) represents one, single character

The percent sign and the underscore can also be used in combinations!

******

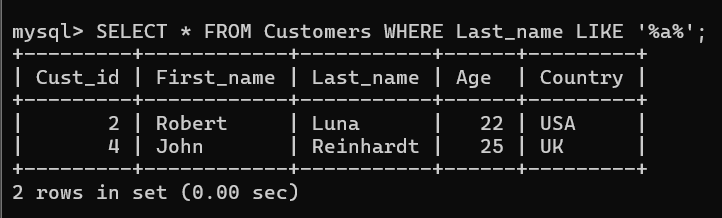
>>SELECT \* FROM Customers WHERE First\_name LIKE ‘J%’;

Selects all Customers with First\_name starting with ‘a’.



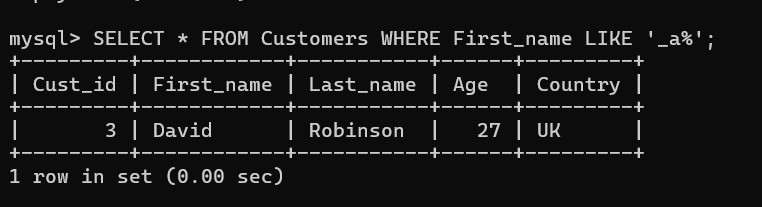
>>SELECT \* FROM Customers WHERE Last\_name LIKE ‘%a%’;

Selects all Customers with First\_name having ‘a’ in any position.



>>SELECT \* FROM Customers WHERE First\_name LIKE ‘\_a%’;

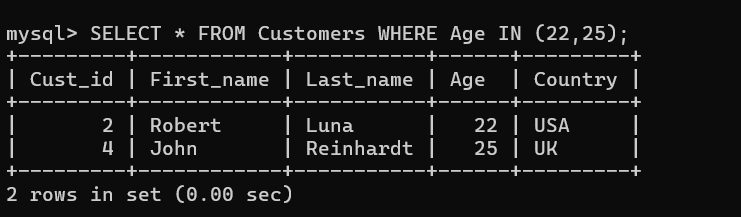
Selects all Customers with First name that have ‘a’ in the second position.



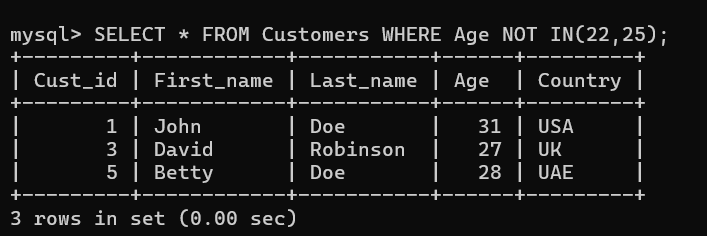
1. ***IN Operator***

The IN operator allows you to specify multiple values in a WHERE clause. This operator is a shorthand for multiple OR condition.

>>SELECT \* FROM Customers WHERE Age IN (22,25);



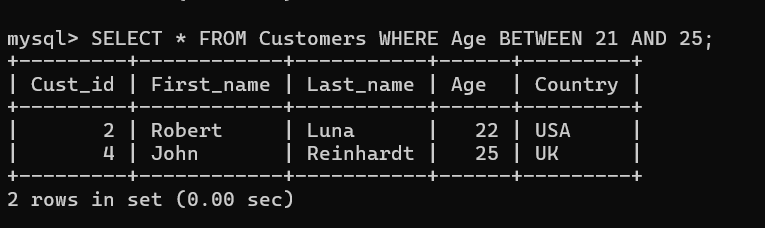
>>SELECT \* FROM Customers WHERE Age NOT IN(22,25);



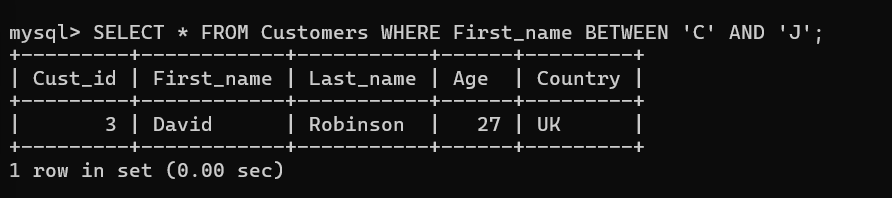
1. ***Between Operator***

Selects values within a given range. The values can be numbers, text or dates. This operator is inclusive of begin and end values are included.

>>SELECT \* FROM Customers WHERE Age BETWEEN 21 AND 25;



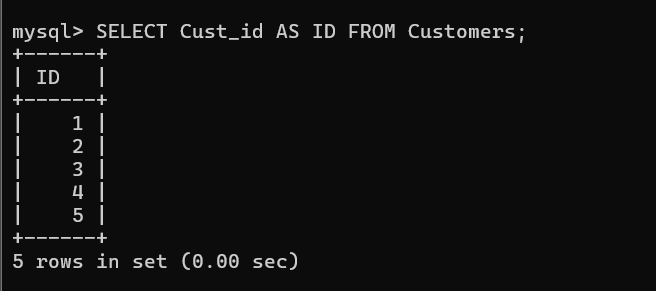
>>SELECT \* FROM Customers WHERE First\_name BETWEEN ‘C’ AND ‘J’;



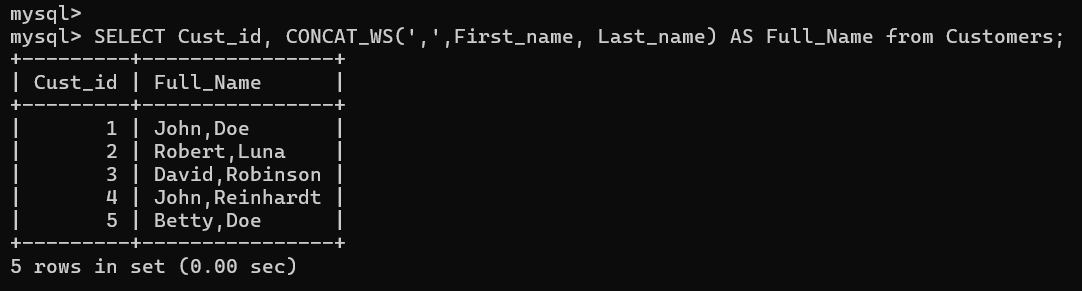
1. ***Aliases***

Used to give a table, or a column in a table, a temporary name for more readable.

>>SELECT Cust\_id AS ID FROM Customers;



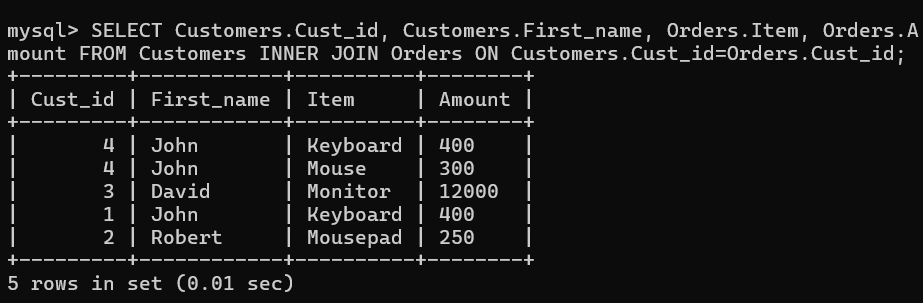
>>SELECT Cust\_id, CONCAT\_WS(‘,’,First\_name, Last\_name) AS Full\_Name from Customers;



1. ***INNER Join***

An INNER JOIN returns records that have matching values in both tables. If there’s no match, the records are not included.

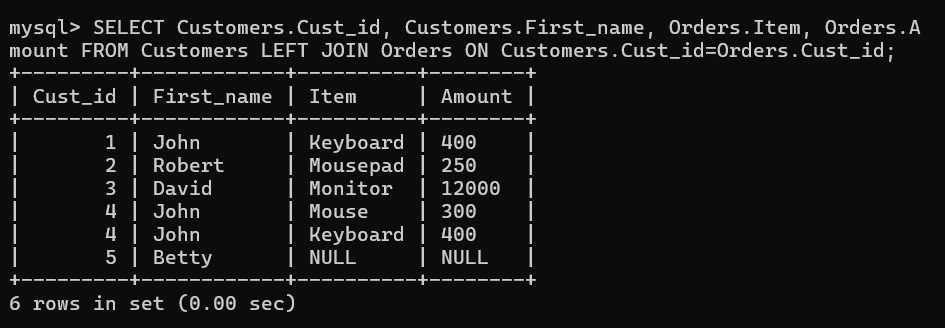
>>SELECT Customers.Cust\_id, Customers.First\_name, Orders.Item, Orders.Amount FROM Customers INNER JOIN Orders ON Customers.Cust\_id=Orders.Cust\_id;



1. ***LEFT Join (or LEFT OUTER Join)***

A LEFT JOIN returns all records from the left table (Customers), and the matched records from the right table (Orders). If there's no match, NULL values are shown for columns from the right table.

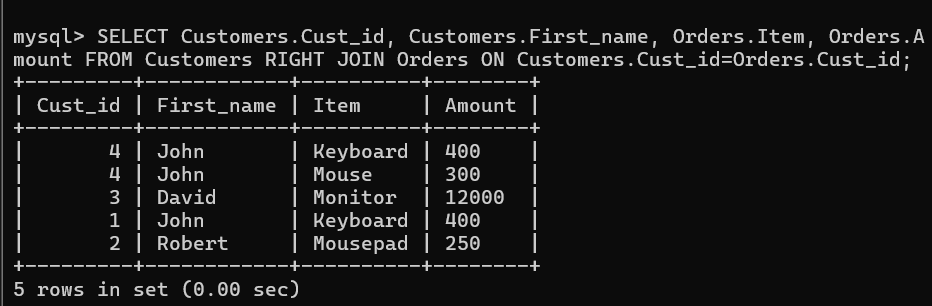
>>SELECT Customers.Cust\_id, Customers.First\_name, Orders.Item, Orders.Amount FROM Customers LEFT JOIN Orders ON Customers.Cust\_id=Orders.Cust\_id;



1. ***RIGHT JOIN (or RIGHT OUTER JOIN)***

A RIGHT JOIN returns all records from the right table (Orders), and the matched records from the left table (Customers). If there's no match, NULL values are shown for columns from the left table.

>>SELECT Customers.Cust\_id, Customers.First\_name, Orders.Item, Orders.Amount FROM Customers RIGHT JOIN Orders ON Customers.Cust\_id=Orders.Cust\_id;



1. ***FULL OUTER JOIN***

The FULL OUTER JOIN returns all the records when there is a match in left table or right table records. FULL OUTER JOIN and FULL JOIN are the same.

>>SELECT Customers.Cust\_id, Customers.First\_name, Orders.Item, Orders.Amount FROM Customers FULL OUTER JOIN Orders ON Customers.Cust\_id=Orders.Cust\_id;

1. ***UNION Operator***

The UNION operator is used to combine the result-set of two or more SELECT statements.

* Every SELECT statement within UNION must have the same number of columns.
* The columns must also have similar data types.
* The columns in every SELECT statement must also be in the same order.

>>SELECT \* FROM Customers UNION SELECT \* FROM Orders;

>>SELECT \* FROM Customers UNION ALL SELECT \* FROM Orders;

It allows the duplicate values.

1. ***GROUP BY***

This statement groups rows that have the same values into summary rows. This statement is often used with aggregate functions ( COUNT(), SUM(), MAX(), MIN(), AVG()) to group the result-set by one or more columns.

>>SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country;

This statement lists the number of customers in each country.

>>SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country ORDER BY COUNT(CustomerID) DESC;

This statement lists the number of customers in each country, sorted high to low.

1. ***HAVING CLAUSE***

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregated functions.

>>SELECT *column\_name(s)* FROM *table\_name* WHERE *condition* GROUP BY *column\_name(s)* HAVING *condition* ORDER BY *column\_name(s);*

>>SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country HAVING COUNT(CustomerID) > 5 ORDER BY COUNT(CustomerID) DESC;

>>SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM (Orders INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID) GROUP BY LastName HAVING COUNT(Orders.OrderID) > 10;

1. ***EXISTS Operator***

This operator tests the existence of any record in a subquery. It returns TRUE if the subquery returns one or more records.

>>SELECT *column\_name(s)* FROM *table\_name* WHERE EXISTS (SELECT *column\_name*FROM *table\_name* WHERE *condition*);

>>SELECT SupplierName FROM Suppliers WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price = 22);

The following SQL statement returns TRUE and lists the suppliers with a product price equal to 22:

1. ***ANY Operators***

This means that the condition will be true if the operation is true for any of the values in the range.

>>SELECT ProductName FROM Products WHERE ProductID = ANY (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity equal to 10 (this will return TRUE because the Quantity column has some values of 10).

1. ***ALL Operators***

This means that the condition will be true only of the operation is true for all values in the range.

>>SELECT ProductName FROM Products WHERE ProductID = ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

The following SQL statement lists the ProductName if ALL the records in the OrderDetails table has Quantity equal to 10. This will of course return FALSE because the Quantity column has many different values (not only the value of 10).

1. ***SELECT INTO Statement***

This statement copies data from one table into a new table.

>>SELECT *column1*, *column2*, *column3*, ... INTO *newtable* [IN *externaldb*]  
FROM *oldtable* WHERE *condition;*

>>SELECT \* INTO CustomersBackup2017 IN 'Backup.mdb'  
FROM Customers;

**Windows Functions**

This allows you to perform calculations across a specific set of rows, known as a window while keeping the detailed data intact. OVER clause is used with window functions to define that window. OVER clause does two things:

* Partitions rows to form a set of rows. (PARTITION BY clause is used).
* Orders rows within those partitions into a particular order. (ORDER BY clause is used).

Syntax:

>>SELECT  col\_name1,

window\_function(col\_name2)

OVER ([PARTITION BY col\_name1] [ORDER BY col\_name3]) AS new\_col

FROM table\_name;

*window\_function*: an aggregate or ranking function

*col\_name1*: column to be selected

*col\_name2:* column on which window function is to be applied

*col\_name3:* column on whose basis partition of rows is to be done.

*new\_col:* Name of new column

*table\_name:* Name of table



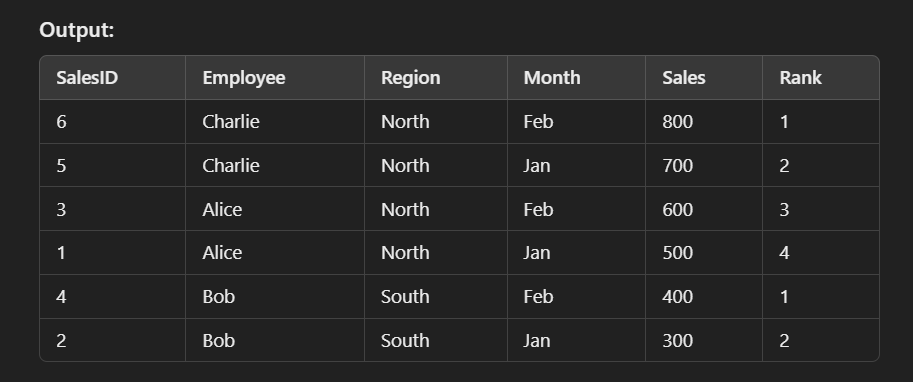
1. **ROW\_NUMBER() – Ranking sales by employee in each region.**

* Gives unique ranks (no ties)

>> SELECT SalesID, Employee, Region, Month, Sales,

ROW\_NUMBER() OVER (PARTITION BY Region ORDER BY Sales DESC) AS Rank

FROM Sales;



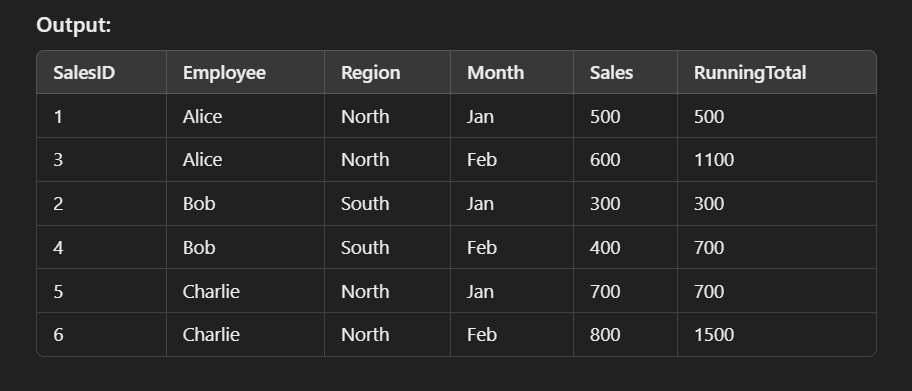
1. **SUM() – Running total of sales for each employee**

* Used for cumulative totals

>> SELECT SalesID, Employee, Region, Month, Sales,

SUM(Sales) OVER (PARTITION BY Employee ORDER BY Month) AS RunningTotal

FROM Sales;

****

1. **LEAD() and LAG() – Comparing sales of consecutive months**

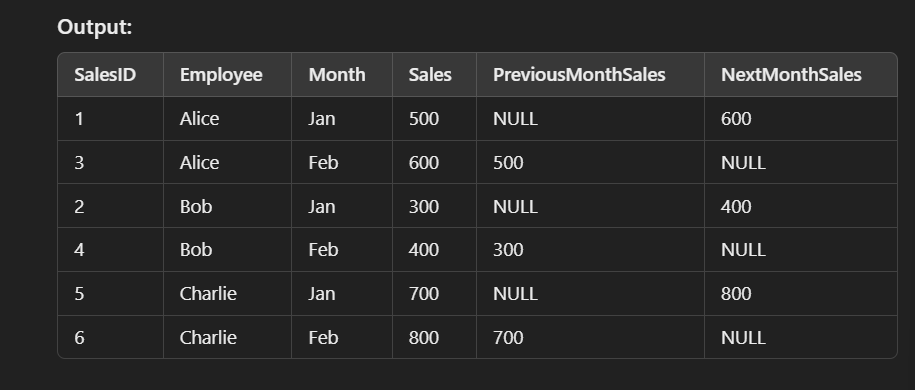
* Compare values across rows within a partition

>> SELECT SaledID, Employee, Month, Sales,

LAG(Sales) OVER (PARTITION BY Employee ORDER BY Month) AS PreviousMonthSales

LEAD(Sales) OVER (PARTITION BY Employee ORDER BY Month) AS NextMonthSales

FROM Sales;

****

1. **RANK() vs DENSE\_RANK()**

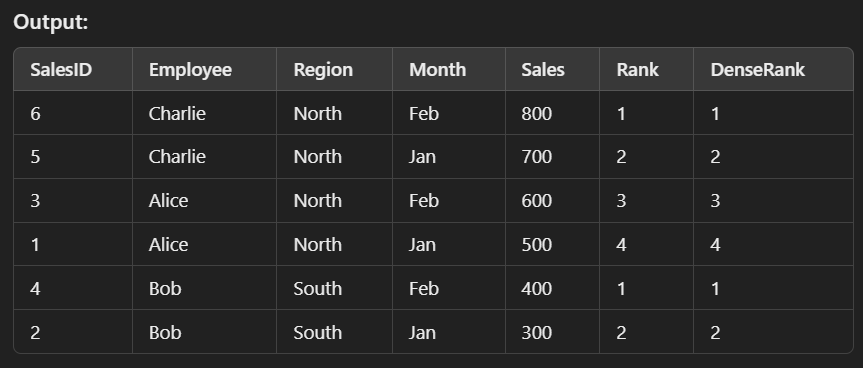
* RANK(): Skips ranks for ties.
* DENSE\_RANK(): No skipped ranks for ties.

>> SELECT SalesID, Employee, Region, Month, Sales,

RANK() OVER (PARTITION BY Region ORDER BY Sales DESC) AS Rank,

DENSE\_RANK() OVER (PARTITION BY Region ORDER BY Sales DESC) AS DenseRank

FROM Sales;

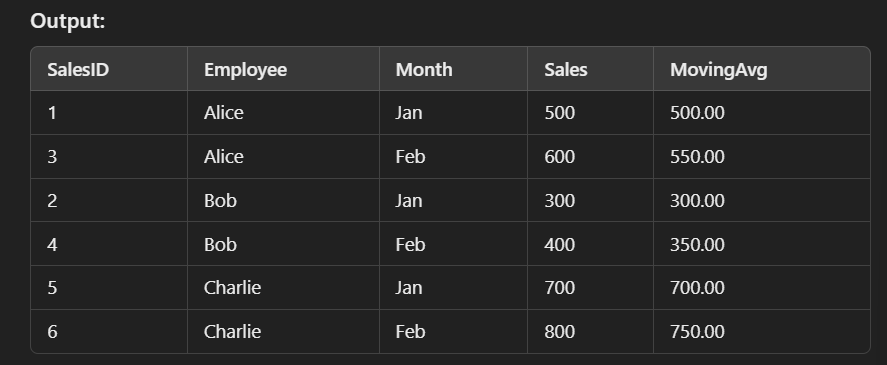
****

1. **Moving Averages – Calculate the moving average of sales for each employee.**

>> SELECT SalesID, Employee, Month, Sales,

AVG(Sales) OVER (PARTITION BY Employee ORDER BY Month ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS MovingAvg

FROM Sales;

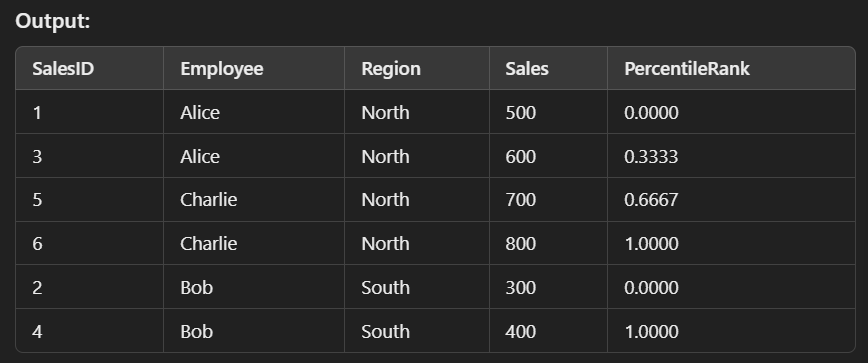


1. **Calculating Percentile Ranks**

>> SELECT SalesID, Employee, Region, Sales,

PERCENT\_RANK() OVER (PARTITION BY Region ORDER BY Sales) AS PercentileRank

FROM Sales;



1. **Difference Between Current and Previous Sales**

>> SELECT SalesID, Employee, Month, Sales,

Sales – LAG(Sales) OVER (PARTITION BY Employee ORDER BY Month) AS SalesDifference

FROM Sales;

