Complete SQL With Notes

- 1. Introduction to SQL-What Is SQL & Database
- 2. Data Types, Primary-Foreign Keys & Constraints
 - a. Install postgresql and pgadmin4
- 3. Create Table In SQL & Create Database
- 4. INSERT UPDATE, DELETE & ALTER Table
- 5. SELECT Statement & WHERE Clause with Example
- 6. How To Import Excel File (CSV) to SQL
- 7. Functions in SQL & String Function
- 8. Aggregate Functions Types & Syntax
- 9. Group By and Having Clause
- 10. Time Stamp and Extract Function, Date Time Function
- 11. SQL JOINS Types & Syntax
- 12. SELF JOIN, UNION & UNION ALL
- 13. Subquery
- 14. Window Function Types & Syntax
- 15. Case Statement/Expression with examples
- 16. CTE- Common Table Expression with examples

WHAT IS SQL & DATABASE-INTRODUCTION

SQL Tutorial In Hindi-1

SQL By Rishabh Mishra

Introduction to SQL

- What is SQL
- It's applications
- SQL v/s NoSQL
- Types of SQL Commands
- What is Database
- Excel v/s Database in SQL

What is SQL?

SQL (Structured Query Language) is a programming language used to interact with database



SQL Application



CRUD is an acronym for CREATE, READ(SELECT), UPDATE, and DELETE statements in SQL

SQL v/s NoSQL

| Relational Database | Non-Relational Database |
|--|--|
| SQL database | NoSQL database |
| Data stored in tables | Data stored are either key-value pairs, document-based, graph databases or wide- column stores |
| These databases have fixed or static or predefined schema | They have dynamic schema |
| Low performance with huge volumes of data | Easily work with huge volumes of data |
| Eg: PostgreSQL, MySQL, MS SQL Server | Eg: MongoDB, Cassandra, Hbase |

SQL Commands

There are mainly 3 types of SQL commands:

- **DDL** (Data Definition Language): create, alter, and drop
- **DML** (Data Manipulation Language): select, insert, update and delete
- **DCL** (Data Control Language): grant and revoke permission to users

What is Database?

Database is a system that allow users to store and organise data



SQL By Rishabh Mishra

Excel v/s Database

| Excel | Database |
|---|---------------------------------|
| Easy to use- untrained person can work | Trained person can work |
| Data stored less data | Stores large amount of data |
| Good for one time analysis, quick charts | Can automate tasks |
| No data integrity due to manual operation | High data integrity |
| Low search/filter capabilities | High search/filter capabilities |

SQL Databases



Watch and learn this complete SQL tutorial on the YouTube channel

YouTube: https://www.youtube.com/@RishabhMishraOfficial



SQL Playlist: https://youtube.com/playlist?list=PLdOKnrf8EcP17p05q13WXbHO5Z_JfXNpw

DATA TYPES, PRIMARY & FOREIGN KEYS, CONSTRAINTS

SQL Tutorial In Hindi-2

SQL By Rishabh Mishra

SQL Structure



RDBMS

Database Diagram





int

int

Creating Database & Tables

- Data types
- Primary & Foreign keys
- Constraints
- SQL Commands
 - CREATE
 - INSERT
 - UPDATE
 - BACKUP
 - DELETE
 - ALTER
 - DROP, TRUNCATE

Data Types

- Data type of a column defines what value the column can store in table
- Defined while creating tables in database
- Data types mainly classified into three categories + most used

OString: char, varchar, etc

ONumeric: int, float, bool, etc

ODate and time: date, datetime, etc

Data Types

Commonly Used data types in SQL:

- **int:** used for the integer value
- float: used to specify a decimal point number
- **bool:** used to specify Boolean values true and false
- **char:** fixed length string that can contain numbers, letters, and special characters
- varchar: variable length string that can contain numbers, letters, and special characters
- date: date format YYYY-MM-DD
- datetime: date & time combination, format is YYYY-MM-DD hh:mm:ss

Primary and Foreign Keys:

Primary key (PK):

- A Primary key is a unique column we set in a table to easily identify and locate data in queries
- A table can have only one primary key, which should be unique and NOT NULL

Foreign keys (FK):

- A Foreign key is a column used to link two or more tables together
- A table can have any number of foreign keys, can contain duplicate and NULL values

Constraints

- Constraints are used to specify rules for data in a table
- This ensures the accuracy and reliability of the data in the table
- Constraints can be specified when the table is created with the CREATE TABLE statement, or
- after the table is created with the ALTER TABLE statement
- Syntax

CREATE TABLE table_name (column1 datatype constraint, column2 datatype constraint, column3 datatype constraint,

```
····
);
```

Constraints

Commonly used constraints in SQL:

- 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
- FOREIGN KEY Prevents actions that would destroy links between tables (used to link multiple tables together)
- CHECK Ensures that the values in a column satisfies a specific condition
- DEFAULT Sets a default value for a column if no value is specified
- CREATE INDEX Used to create and retrieve data from the database very quickly

Install Postgre SQL (Click Here)

SQL By Rishabh Mishra

×

Creating Database & Tables SQL Tutorial In Hindi-3

SQL By Rishabh Mishra

Create Table

The CREATE TABLE statement is used to create a new table in a database

• Syntax

```
CREATE TABLE table_name
(
    column_name1 datatype constraint,
    column_name2 datatype constraint,
    column_name3 datatype constraint,
);
```

• Example

```
CREATE TABLE customer
(
CustID int8 PRIMARY KEY,
CustName varchar(50) NOT NULL,
Age int NOT NULL,
City char(50),
Salary numeric
);
```

Insert, Update, Delete Values in Table + Alter, Drop & Truncate Table

SQL Tutorial In Hindi-4

SQL By Rishabh Mishra

Insert Values In Table

The INSERT INTO statement is used to insert new records in a table

• Syntax

INSERT INTO TABLE_NAME

(column1, column2, column3,...columnN)

VALUES

(value1, value2, value3,...valueN);

• Example

INSERT INTO customer (CustID, CustName, Age, City, Salary) VALUES (1, 'Sam', 26, 'Dolbi', 9000)

(1, 'Sam', 26, 'Delhi', 9000),

(2, 'Ram', 19, 'Bangalore', 11000),

(3, 'Pam', 31, 'Mumbai', 6000),

(4, 'Jam', 42, 'Pune', 10000);

Update Values In Table

The UPDATE command is used to update existing rows in a table

• Syntax

```
UPDATE TABLE_NAME
SET "Column_name1" = 'value1', "Column_name2" = 'value2'
WHERE "ID" = 'value'
```

• Example

UPDATE customer SET CustName = 'Xam', Age= 32 WHERE CustID = 4;

ALTER Table

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table

• ALTER TABLE - ADD Column Syntax

ALTER TABLE table_name ADD COLUMN column_name ;

• ALTER TABLE - DROP COLUMN Syntax

ALTER TABLE table_name DROP COLUMN column_name;

• ALTER TABLE - ALTER/MODIFY COLUMN Syntax

ALTER TABLE table_name ALTER COLUMN column_name datatype;

ALTER Table Example

- ADD Column Syntax: Adding new 'Gender' column to customer table ALTER TABLE customer ADD COLUMN Gender varchar(10);
- ALTER/MODIFY COLUMN Syntax: changing Gender column data type from varchar(10) to char(10)

ALTER TABLE customer

ALTER COLUMN Gender char(10);

• DROP COLUMN Syntax: Deleting Gender column from customer table

ALTER TABLE customer **DROP COLUMN** Gender;

Delete Values In Table

The DELETE statement is used to delete existing records in a table

• Syntax

DELETE FROM table_name WHERE condition;

• Example

DELETE FROM customer WHERE CustID = 3;

Drop & Truncate Table

The DROP TABLE command deletes a table in the database

• Syntax

DROP TABLE table_name;

The TRUNCATE TABLE command deletes the data inside a table, but not the table itself

• Syntax

TRUNCATE TABLE table_name;

SELECT & WHERE CLAUSE

SQL Tutorial In Hindi-5

SQL By Rishabh Mishra

Creating a Classroom dataset for practice

```
CREATE TABLE classroom
```

```
rollno int8 PRIMARY KEY,
name varchar(50) NOT NULL,
house char(12) NOT NULL,
grade char(1)
);
```

```
INSERT INTO classroom
(rollno, name, house, grade)
VALUES
(1, 'Sam', 'Akash', 'B'),
```

- (2, 'Ram', 'Agni', 'A'),
- (3, 'Shyam', 'Jal', 'B'),
- (4, 'Sundar', 'Agni', 'A'),
- (5, 'Ram', 'Yayu', 'B');

SELECT Statement

The SELECT statement is used to select data from a database.

• Syntax

SELECT column_name FROM table_name;

To select all the fields available in the table

• Syntax

SELECT * FROM table_name;

To select distinct/unique fields available in the table

• Syntax

SELECT DISTINCT Column_name FROM table_name;

WHERE Clause

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition

• Syntax

SELECT column_name FROM table_name WHERE conditions;

• Example

SELECT name **FROM** classroom **WHERE** grade='A';

Operators In SQL

The SQL reserved words and characters are called operators, which are used with a WHERE clause in a SQL query

Most used operators:

- **1. Arithmetic operators :** arithmetic operations on numeric values Example: Addition (+), Subtraction (-), Multiplication (*), Division (/), Modulus (%)
- 2. Comparison operators: compare two different data of SQL table
 - Example: Equal (=), Not Equal (!=), Greater Than (>), Greater Than Equals to (>=)
- 3. Logical operators: perform the Boolean operations
 - Example: ALL, IN, BETWEEN, LIKE, AND, OR, NOT, ANY
- 4. Bitwise operators: perform the bit operations on the Integer values
 - Example: Bitwise AND (&), Bitwise OR(|)

LIMIT Clause

The LIMIT clause is used to set an upper limit on the number of tuples returned by SQL.

Example: below code will return 5 rows of data

SELECT column_name FROM table_name LIMIT 5;

ORDER BY Clause

The ORDER BY is used to sort the result-set in ascending (ASC) or descending order (DESC).

Example: below code will sort the output data by column name in ascending order

SELECT column_name FROM table_name

ORDER BY column_name e **ASC**;
IMPORT CSV FILE SQL Tutorial In Hindi-6

SQL By Rishabh Mishra

CSV files

customer csv file: <u>https://bit.ly/3KLPb2P</u> payment csv file: <u>https://bit.ly/41kIYjW</u>

SQL Tutorial In Hindi-7

SQL By Rishabh Mishra

Functions In SQL

Functions in SQL are the database objects that contains a set of SQL statements to perform a specific task. A function accepts input parameters, perform actions, and then return the result.

Types of Function:

- 1. System Defined Function : these are built-in functions
 - Example: rand(), round(), upper(), lower(), count(), sum(), avg(), max(), etc
- 2. User-Defined Function : Once you define a function, you can call it in the same way as the built-in functions

Most Used String Functions

String functions are used to perform an operation on input string and return an output string

- **UPPER()** converts the value of a field to uppercase
- LOWER() converts the value of a field to lowercase
- LENGTH() returns the length of the value in a text field
- **SUBSTRING()** extracts a substring from a string
- **NOW()** returns the current system date and time
- FORMAT() used to set the format of a field
- **CONCAT()** adds two or more strings together
- **REPLACE()** Replaces all occurrences of a substring within a string, with a new substring
- **TRIM()** removes leading and trailing spaces (or other specified characters) from a string

AGGREGATE FUNCTION

SQL By Rishabh Mishra

Most Used Aggregate Functions

Aggregate function performs a calculation on multiple values and returns a single value.

And Aggregate functiona are often used with GROUP BY & SELECT statement

- **COUNT()** returns number of values
- **SUM()** returns sum of all values
- AVG() returns average value
- MAX() returns maximum value
- MIN() returns minimum value
- **ROUND()** Rounds a number to a specified number of decimal places

GROUP BY & HAVING CLAUSE

SQL Tutorial In Hindi-9

SQL By Rishabh Mishra

GROUP BY Statement

The GROUP BY statement group rows that have the same values into summary rows.

It is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns

• Syntax

SELECT column_name(s)
FROM table_name
GROUP BY column_name(s);

• Example

SELECT mode, SUM(amount) AS total FROM payment GROUP BY mode

HAVING Clause

The **HAVING** clause is used to apply a filter on the result of **GROUP BY** based on the specified condition.

The **WHERE** clause places conditions on the selected columns, whereas the **HAVING** clause places conditions on groups created by the **GROUP BY** clause

Syntax

SELECT column_name(s) FROM table_name WHERE condition(s) GROUP BY column_name(s) HAVING condition(s)

• Example

SELECT mode, COUNT(amount) AS total FROM payment GROUP BY mode HAVING COUNT(amount) >= 3 ORDER BY total DESC

Quick Assignment: 01

Order of execution in SQL: SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY, LIMIT

Answer in video's comment (no cheating)

SQL By Rishabh Mishra

TIMESTAMPS & EXTRACT

SQL Tutorial In Hindi-10,

SQL By Rishabh Mishra

TIMESTAMP

The **TIMESTAMP** data type is used for values that contain both date and time parts

- **TIME** contains only time, format HH:MI:SS
- **DATE** contains on date, format YYYY-MM-DD
- YEAR contains on year, format YYYY or YY
- **TIMESTAMP** contains date and time, format YYYY-MM-DD HH:MI:SS
- **TIMESTAMPTZ** contains date, time and time zone

TIMESTAMP functions/operators Below are the TIMESTAMP functions and operators in SQL:

- SHOW TIMEZONE
- SELECT NOW()
- SELECT TIMEOFDAY()
- SELECT CURRENT_TIME
- SELECT CURRENT_DATE

EXTRACT Function

The **EXTRACT()** function extracts a part from a given date value.

Syntax: SELECT **EXTRACT**(MONTH FROM date_field) FROM Table

- YEAR
- QUARTER
- MONTH
- WEEK
- DAY
- HOUR
- MINUTE
- DOW day of week
- DOY day of year

JOINS

SQL Tutorial In Hindi-11_{*}

SQL By Rishabh Mishra

×

TOPICS IN JOIN

- WHAT IS JOIN?
- USE OF JOIN
- JOIN TYPES
- WHICH JOIN TO USE
- JOIN SYNTAX
- EXAMPLES IN SQL

SQL JOIN

- JOIN means to combine something.
- A JOIN clause is used to combine data from two or more tables, based on a related column between them
- Let's understand the joins through an example:



JOIN Example

Question: How much amount was paid by customer 'Madan', what was mode and payment date?



JOIN Example

| customer_id [PK] bigint | first_name character varying (50) | last_name character varying (50) | address_id bigint |
|----------------------------|--------------------------------------|-------------------------------------|--------------------|
| 1 | Mary | Smith | 5 |
| 2 | Madan | Mohan | 6 |
| 3 | Linda | Williams | 7 |
| 4 | Barbara | Jones | 8 |
| 5 | Elizabeth | Brown | 9 |

Question: How much amount was paid by customer 'Madan', what was mode and payment date?

| customer_id [PK] bigint | amount bigint | mode character varying (50) | payment_date / |
|----------------------------|------------------|--------------------------------|-----------------|
| 1 | 60 | Cash | 2020-09-24 |
| 2 | 30 | Credit Card | 2020-04-27 |
| 3 | 90 | Credit Card | 2020-07-07 |
| 4 | 50 | Debit Crad | 2020-02-12 |
| 5 | 40 | Mobile Payment | 2020-11-20 |

Answer: Amount = 30, Mode = Credit Card, Date = 2020-04-27

TYPES OF JOINS

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN



INNER JOIN

 Returns records that have matching values in both tables





INNER JOIN

• Syntax

SELECT column_name(s) FROM TableA INNER JOIN TableB ON TableA.col_name = TableB.col_name



• Example

SELECT * FROM customer AS c INNER JOIN payment AS p ON c.customer_id = p.customer_id

LEFT JOIN

• Returns all records from the left table, and the matched records from the right table





LEFT JOIN

• Syntax

SELECT column_name(s) FROM TableA LEFT JOIN TableB ON TableA.col_name = TableB.col_name



• Example

SELECT * FROM customer AS c LEFT JOIN payment AS p ON c.customer_id = p.customer_id

RIGHT JOIN

• Returns all records from the right table, and the matched records from the left table





RIGHT JOIN

• Syntax

SELECT column_name(s) FROM TableA RIGHT JOIN TableB ON TableA.col_name = TableB.col_name



• Example

SELECT * FROM customer AS c RIGHT JOIN payment AS p ON c.customer_id = p.customer_id

FULL JOIN

 Returns all records when there is a match in either left or right table





FULL JOIN

• Syntax

SELECT column_name(s) FROM TableA FULL OUTER JOIN TableB ON TableA.col_name = TableB.col_name



• Example

SELECT * FROM customer AS c FULL OUTER JOIN payment AS p ON c.customer_id = p.customer_id

Which JOIN To Use

- **INNER JOIN:** Returns records that have matching values in both tables
- LEFT JOIN: Returns all records from the left table, and the matched records from the right table
- **RIGHT JOIN:** Returns all records from the right table, and the matched records from the left table
- FULL JOIN: Returns all records when there is a match in either left or right table



JOIN CHEAT SHEET



SQL Tutorial In Hindi-12

SQL By Rishabh Mishra

SELF JOIN

- A **self join** is a regular join in which a table is joined to itself
- SELF Joins are powerful for comparing values in a column of rows with the same table
- Syntax

SELECT column_name(s)
FROM Table AS T1
JOIN Table AS T2
ON T1.col_name = T2.col_name

SELF JOIN example

| empid [PK] bigint 🖍 | empname character varying (50) | manager_id bigint | 1 |
|------------------------|-----------------------------------|-----------------------------|---|
| 1 | Agni | | 3 |
| 2 | Akash | | 4 |
| 3 | Dharti | | 2 |
| 4 | Vayu | | 3 |



• Find the name of respective managers for each of the employees?

SELF JOIN example

| empid [PK] bigint | empname character varying (50) | manager_id bigint | | mngr character varying (50) |
|----------------------|-----------------------------------|--------------------|---------------|--------------------------------|
| 1 | Agni | 3 | | Dharti |
| 2 | Akash | 4 | \rightarrow | Vayu |
| 3 | Dharti | 2 | | Akash |
| 4 | Vayu | 3 | | Dharti |

SELECT T2.empname, T1.empname FROM emp AS T1 JOIN emp AS T2 ON T1.empid = T2.manager_id

UNION

The SQL **UNION** clause/operator is used to combine/concatenate the results of two or more SELECT statements without returning any duplicate rows and keeps **unique records**

To use this UNION clause, each SELECT statement must have

- The same number of columns selected and expressions
- The same data type and
- Have them in the same order
- Syntax

SELECT column_name(s) FROM **TableA** UNION SELECT column_name(s) FROM **TableB**

• Example

SELECT cust_name, cust_amount from custA UNION SELECT cust_name, cust_amount from custB
UNION ALL

In **UNION ALL** everything is same as **UNION**, it combines/concatenate two or more table but keeps all records, **including duplicates**

Syntax
 SELECT column_name(s) FROM TableA
 UNION ALL
 SELECT column_name(s) FROM TableB

• Example

SELECT cust_name, cust_amount from custA UNION ALL SELECT cust_name, cust_amount from custB

UNION Example

Table: custA

Table: custB

| cust_name character (30) | cust_amount bigint |
|-----------------------------|-----------------------|
| Madan Mohan | 2100 |
| Gopi Nath | 1200 |
| Govind Dev | 5000 |

| cust_name character (30) | cust_amount bigint |
|-----------------------------|-----------------------|
| Gopal Bhat | 1500 |
| Madan Mohan | 2100 |

SQL Tutorial In Hindi-13

SQL By Rishabh Mishra

SUB QUERY

A **Subquery** or Inner query or a Nested query allows us to create complex query on the output of another query

• Sub query syntax involves two SELECT statements

• Syntax

SELECT column_name(s)

FROM table_name

WHERE column_name *operator*

(SELECT column_name FROM table_name WHERE ...);

SUB QUERY Example

Question: Find the details of customers, whose payment amount is more than the average of total amount paid by all customers

Divide above question into two parts:

- 1. Find the average amount
- Filter the customers whose amount > average amount

| | customer_id [PK] bigint | amount bigint | mode character varying (50) | payment_date |
|---|----------------------------|------------------|--------------------------------|--------------|
| 1 | 1 | 60 | Cash | 2020-09-24 |
| 2 | 2 | 30 | Credit Card | 2020-04-27 |
| 3 | 8 | 110 | Cash | 2021-01-26 |
| 4 | 10 | 70 | mobile Payment | 2021-02-28 |
| 5 | 11 | 80 | Cash | 2021-03-01 |

WINDOWS FUNCTION SQL Tutorial In Hindi-14

SQL By Rishabh Mishra

WINDOW FUNCTION

- Window functions applies aggregate, ranking and analytic functions over a particular window (set of rows).
- And OVER clause is used with window functions to define that window.



Give output one row per aggregation

The rows maintain their separate identities

WINDOW FUNCTION SYNTAX



WINDOW FUNCTION TERMS

Let's look at some definitions:

- Window function applies aggregate, ranking and analytic functions over a particular window; for example, sum, avg, or row_number
- Expression is the name of the column that we want the window function operated on. This may not be necessary depending on what window function is used
- **OVER** is just to signify that this is a window function
- PARTITION BY divides the rows into partitions so we can specify which rows to use to compute the window function
- ORDER BY is used so that we can order the rows within each partition. This is optional and does not have to be specified
- ROWS can be used if we want to further limit the rows within our partition. This is optional and usually not used

WINDOW FUNCTION TYPES

There is no official division of the SQL window functions into categories but high level we can divide into three types



SELECT new_id, new_cat,

SUM(new_id) OVER(PARTITION BY new_cat ORDER BY new_id) AS "Total", AVG(new_id) OVER(PARTITION BY new_cat ORDER BY new_id) AS "Average", COUNT(new_id) OVER(PARTITION BY new_cat ORDER BY new_id) AS "Count", MIN(new_id) OVER(PARTITION BY new_cat ORDER BY new_id) AS "Min", MAX(new_id) OVER(PARTITION BY new_cat ORDER BY new_id) AS "Max" FROM test_data

| new_id | new_cat | Total | Average | Count | Min | Max |
|--------|---------|-------|-----------|-------|-----|-----|
| 100 | Agni | 300 | 150 | 2 | 100 | 200 |
| 200 | Agni | 300 | 150 | 2 | 100 | 200 |
| 500 | Dharti | 1200 | 600 | 2 | 500 | 700 |
| 700 | Dharti | 1200 | 600 | 2 | 500 | 700 |
| 200 | Vayu | 1000 | 333.33333 | 3 | 200 | 500 |
| 300 | Vayu | 1000 | 333.33333 | 3 | 200 | 500 |
| 500 | Vayu | 1000 | 333.33333 | 3 | 200 | 500 |

AGGREGATE FUNCTION Example **SELECT** new_id, new_cat,

SUM(new_id) OVER(ORDER BY new_id ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS "Total", AVG(new_id) OVER(ORDER BY new_id ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS "Average", COUNT(new_id) OVER(ORDER BY new_id ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS "Count", MIN(new_id) OVER(ORDER BY new_id ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS "Min", MAX(new_id) OVER(ORDER BY new_id ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS "Max" FROM test_data

| new_id | new_cat | Total | Average | Count | Min | Max |
|--------|---------|-------|-----------|-------|-----|-----|
| 100 | Agni | 2500 | 357.14286 | 7 | 100 | 700 |
| 200 | Agni | 2500 | 357.14286 | 7 | 100 | 700 |
| 200 | Vayu | 2500 | 357.14286 | 7 | 100 | 700 |
| 300 | Vayu | 2500 | 357.14286 | 7 | 100 | 700 |
| 500 | Vayu | 2500 | 357.14286 | 7 | 100 | 700 |
| 500 | Dharti | 2500 | 357.14286 | 7 | 100 | 700 |
| 700 | Dharti | 2500 | 357.14286 | 7 | 100 | 700 |

AGGREGATE FUNCTION Example

NOTE: Above we have used: "**ROWS** BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING" which will give a SINGLE output based on all INPUT Values/RARTITION (if used)

SELECT new_id, ROW_NUMBER() OVER(ORDER BY new_id) AS "ROW_NUMBER", RANK() OVER(ORDER BY new_id) AS "RANK", DENSE_RANK() OVER(ORDER BY new_id) AS "DENSE_RANK", PERCENT_RANK() OVER(ORDER BY new_id) AS "PERCENT_RANK" FROM test_data

| new_id | ROW_NUMBER | RANK | DENSE_RANK | PERCENT_RANK |
|--------|------------|------|------------|--------------|
| 100 | 1 | 1 | 1 | 0 |
| 200 | 2 | 2 | 2 | 0.166 |
| 200 | 3 | 2 | 2 | 0.166 |
| 300 | 4 | 4 | 3 | 0.5 |
| 500 | 5 | 5 | 4 | 0.666 |
| 500 | 6 | 5 | 4 | 0.666 |
| 700 | 7 | 7 | 5 | 1 |

RANKING FUNCTION Example SELECT new_id, FIRST_VALUE(new_id) OVER(ORDER BY new_id) AS "FIRST_VALUE", LAST_VALUE(new_id) OVER(ORDER BY new_id) AS "LAST_VALUE", LEAD(new_id) OVER(ORDER BY new_id) AS "LEAD", LAG(new_id) OVER(ORDER BY new_id) AS "LAG" FROM test_data

| new_id | FIRST_VALUE | LAST_VALUE | LEAD | LAG |
|--------|-------------|------------|------|------|
| 100 | 100 | 100 | 200 | null |
| 200 | 100 | 200 | 200 | 100 |
| 200 | 100 | 200 | 300 | 200 |
| 300 | 100 | 300 | 500 | 200 |
| 500 | 100 | 500 | 500 | 300 |
| 500 | 100 | 500 | 700 | 500 |
| 700 | 100 | 700 | null | 500 |

NOTE: If you just want the single last value from whole column, use: "**ROWS** BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING" SQL By Rishabh Mishra

Quick Assignment: WINDOW FUNCTION

Offset the LEAD and LAG values by 2 in the output columns ?

| INPUT | | OUTPUT | |
|--------|--------|--------|------|
| new_id | new_id | LEAD | LAG |
| 100 | 100 | 200 | NULL |
| 200 | 200 | 300 | NULL |
| 200 | 200 | 500 | 100 |
| 300 | 300 | 500 | 200 |
| 500 | 500 | 700 | 200 |
| 500 | 500 | NULL | 300 |
| 700 | 700 | NULL | 500 |

SELECT new_id, LEAD(new_id, 2) OVER(ORDER BY new_id) AS "LEAD_by2", LAG(new_id, 2) OVER(ORDER BY new_id) AS "LAG_by2" FROM test_data

| new_id | LEAD_by2 | LAG_by2 |
|--------|----------|---------|
| 100 | 200 | null |
| 200 | 300 | null |
| 200 | 500 | 100 |
| 300 | 500 | 200 |
| 500 | 700 | 200 |
| 500 | null | 300 |
| 700 | null | 500 |

ANALYTIC FUNCTION Assignment

CASE EXPRESSION

SQL By Rishabh Mishra

CASE Expression

- The CASE expression goes through conditions and returns a value when the first condition is met (like if-then-else statement). If no conditions are true, it returns the value in the ELSE clause.
- If there is no ELSE part and no conditions are true, it returns NULL.
- Also called CASE STATEMENT

CSV file

payment csv file: https://bit.ly/41kJLRW

CASE Statement Syntax

• General CASE Syntax

CASE

WHEN condition1 THEN result1 WHEN condition2 THEN result2 WHEN conditionN THEN resultN ELSE other_result END;

• Example:

SELECT customer_id, amount, CASE WHEN amount > 100 THEN 'Expensive product'

WHEN amount = 100 THEN 'Moderate product'

ELSE 'Inexpensive product'

END AS ProductStatus

FROM payment

CASE Expression Syntax

- CASE Expression Syntax
- CASE Expression

WHEN value1 THEN result1 WHEN value2 THEN result2 WHEN valueN THEN resultN ELSE other_result END;

Example:
SELECT customer_id,
CASE amount
WHEN 500 THEN 'Prime Customer'
WHEN 100 THEN 'Plus Customer'
ELSE 'Regular Customer'
END AS CustomerStatus
FROM payment

COMMON TABLE EXPRESSION

SQL Tutorial In Hindi-16_{*}

SQL By Rishabh Mishra

Common Table Expression (CTE)

- A common table expression, or CTE, is a temporary named result set created from a simple SELECT statement that can be used in a subsequent SELECT statement
- We can define CTEs by adding a WITH clause directly before SELECT, INSERT, UPDATE, DELETE, or MERGE statement.
- The **WITH** clause can include one or more CTEs separated by commas

CSV files

- Grustomer csv file: <u>https://bit.ly/3xGABBR</u>
- payment csv file: https://bit.ly/3Zc0GUV

Common Table Expression (CTE)

• Syntax



The name of this CTE is my_cte, and the CTE query is SELECT a,b,c FROM Table1. The CTE starts with the WITH keyword, after which you specify the name of your CTE, then the content of the query in parentheses. The main query comes after the closing parenthesis and refers to the CTE. Here, the main query (also known as the outer query) is SELECT a,c FROM my_cte

CTE- Example

1. Example EASY

WITH my_cte AS (

SELECT *, AVG(amount) OVER(ORDER BY p.customer_id) AS "Average_Price",

COUNT(address_id) OVER(ORDER BY c.customer_id) AS "Count"

FROM payment as p

INNER JOIN customer AS c

ON p.customer_id = c.customer_id

)

SELECT first_name, last_name

FROM my_cte

1. Example Multiple CTEs

WITH my_cp AS (

SELECT *, AVG(amount) OVER(ORDER BY p.customer_id) AS "Average_Price",

"Count" COUNT(address_id) OVER(ORDER BY c.customer_id) AS

FROM payment as p

INNER JOIN customer AS c

ON p.customer_id = c.customer_id

my_ca AS (

),

SELECT *

FROM customer as c

INNER JOIN address AS a

ON a.address_id = c.address_id

INNER JOIN country as cc

ON cc.city_id = a.city_id

SELECT cp.first_name, cp.last_name, ca.city, ca.country, cp.amount FROM my_ca as ca , my_cp as cp

2. <u>Example Advance</u>

WITH my_cte AS (
 SELECT mode, MAX(amount) AS highest_price,
SUM(amount) AS total_price
 FROM payment
 GROUP BY mode
)
SELECT payment.*, my.highest_price, my.total_price
FROM payment
JOIN my_cte my
 ON payment.mode = my.mode
 ORDER BY payment.mode