SQL Revision

What is SQL?

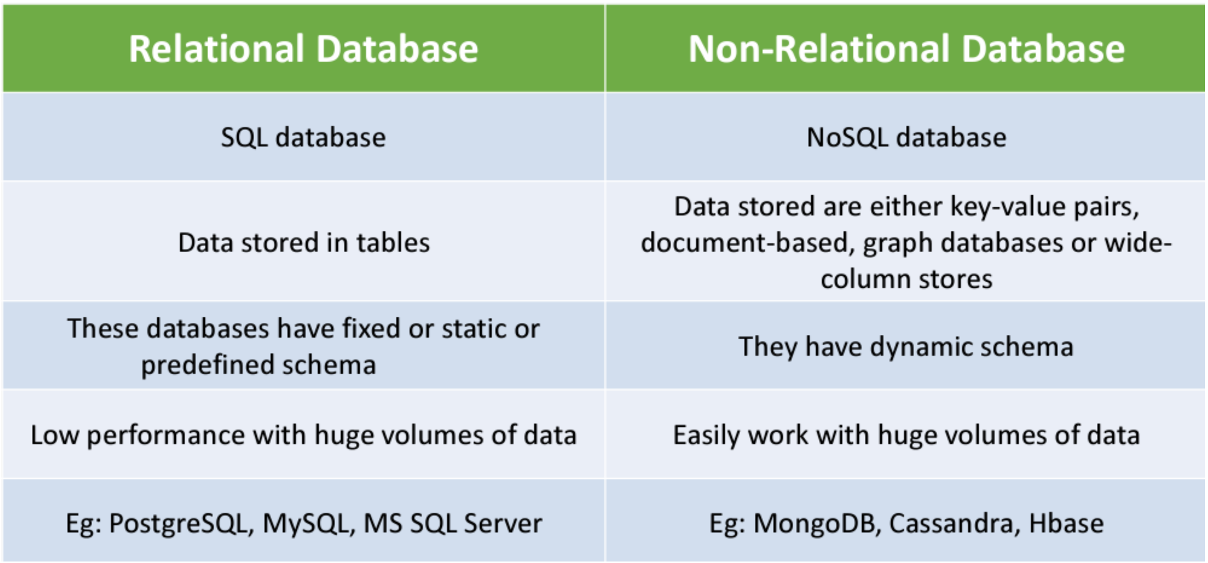
SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in relational database.

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

What is RDBMS:

RDBMS stands for Relational Database Management System. It is a type of database management system that organizes and stores data in a structured manner, using tables with rows and columns. RDBMS systems use a relational model to establish relationships between data elements, making it easier to query and manipulate data. Some popular RDBMS products include MySQL, PostgreSQL, Oracle, and Microsoft SQL Server.

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

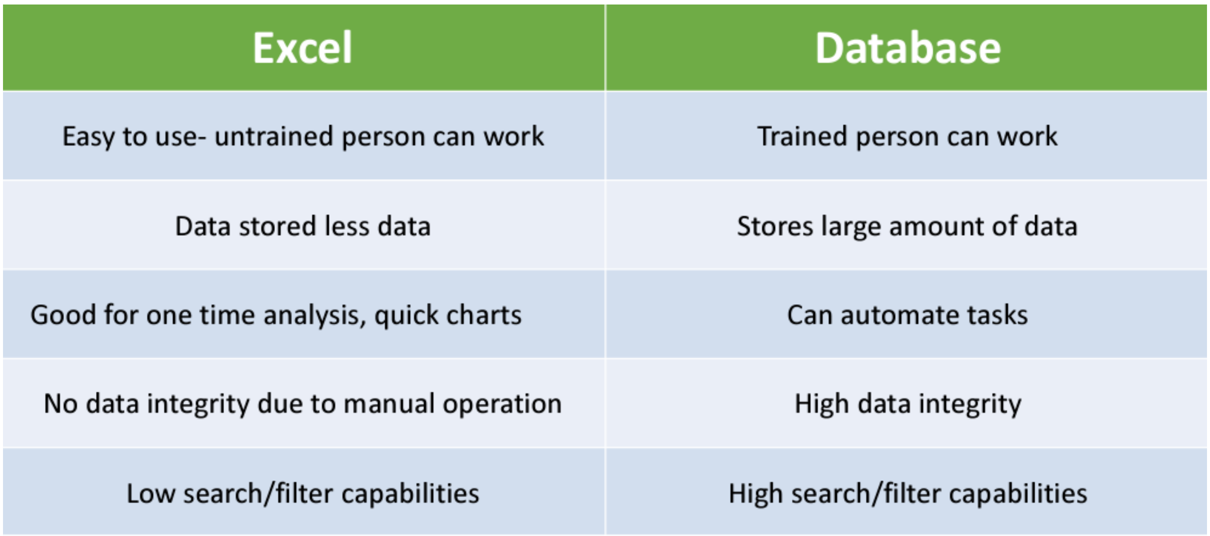
SQL Commands: There are mainly 3 types of SQL commands:

- 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.

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:

teradata.
IBM
DB2
Microsoft
SQL Server
SQL
Amazon RDS
ORACLE
DATABASE
MySQL
SQL
snowflake
PostgreSQL
SAP
S/4 HANA

SQL Constraints:

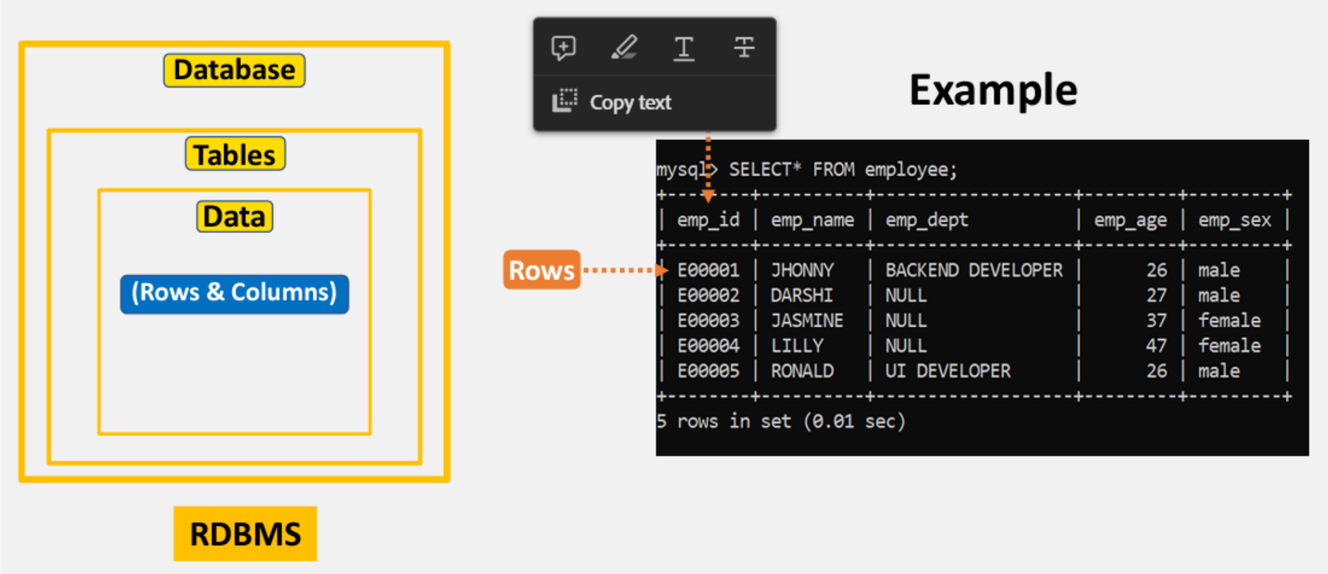
Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be column level or table level. Column level constraints are applied only to one column whereas table level constraints are applied to the whole table.

SQL Syntax:

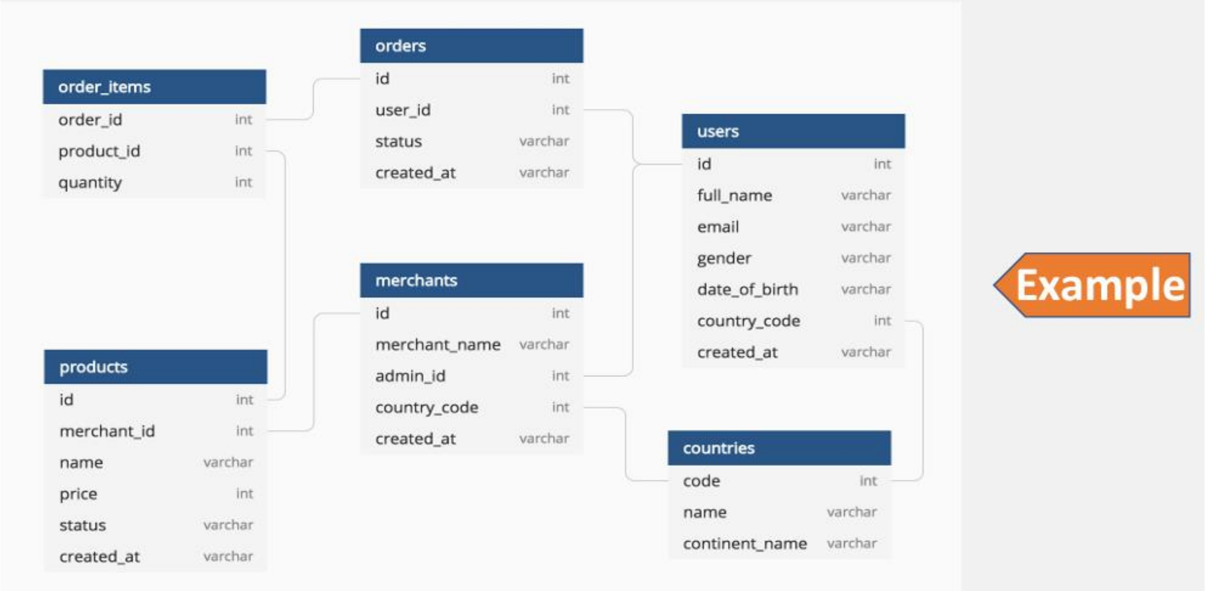
SQL is followed by unique set of rules and guidelines called Syntax. This tutorial gives you a quick start with SQL by listing all the basic SQL Syntax: All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and all the statements end with a semicolon ;. Important point to be noted is that SQL is case insensitive which means SELECT and select have same meaning in SQL statements but MySQL make difference in table names. So if you are working with MySQL then you need to give table names as they exist in the database.

SQL Structure:



```
mysql> SELECT* FROM employee;
+--------+----------+-------------------+---------+---------+
| emp_id | emp_name | emp_dept          | emp_age | emp_sex |
+--------+----------+-------------------+---------+---------+
| E00001 | JHONNY   | BACKEND DEVELOPER |      26 | male    |
| E00002 | DARSHI   | NULL              |      27 | male    |
| E00003 | JASMINE  | NULL              |      37 | female  |
| E00004 | LILLY    | NULL              |      47 | female  |
| E00005 | RONALD   | UI DEVELOPER      |      26 | male    |
+--------+----------+-------------------+---------+---------+
5 rows in set (0.01 sec)
```

Database Diagram:



order_items	
order_id	int
product_id	int
quantity	int

orders	
id	int
user_id	int
status	varchar
created_at	varchar

users	
id	int
full_name	varchar
email	varchar
gender	varchar
date_of_birth	varchar
country_code	int
created_at	varchar

merchants	
id	int
merchant_name	varchar
admin_id	int
country_code	int
created_at	varchar

products	
id	int
merchant_id	int
name	varchar
price	int
status	varchar
created_at	varchar

countries	
code	int
name	varchar
continent_name	varchar

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:

1. String: char, varchar, etc
2. Numeric: int, float, bool, etc
3. Date and time: date, datetime, etc

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

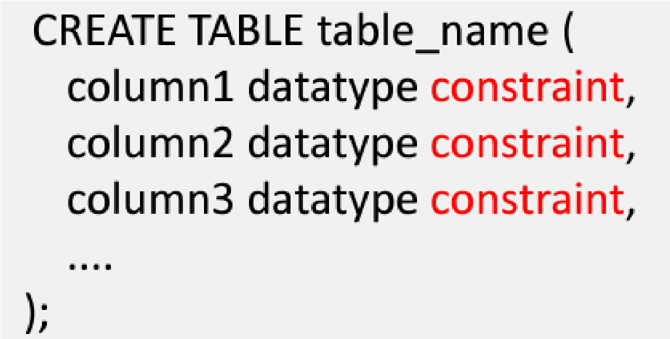
Foreign keys (FK):

• 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,
  ....
);
```

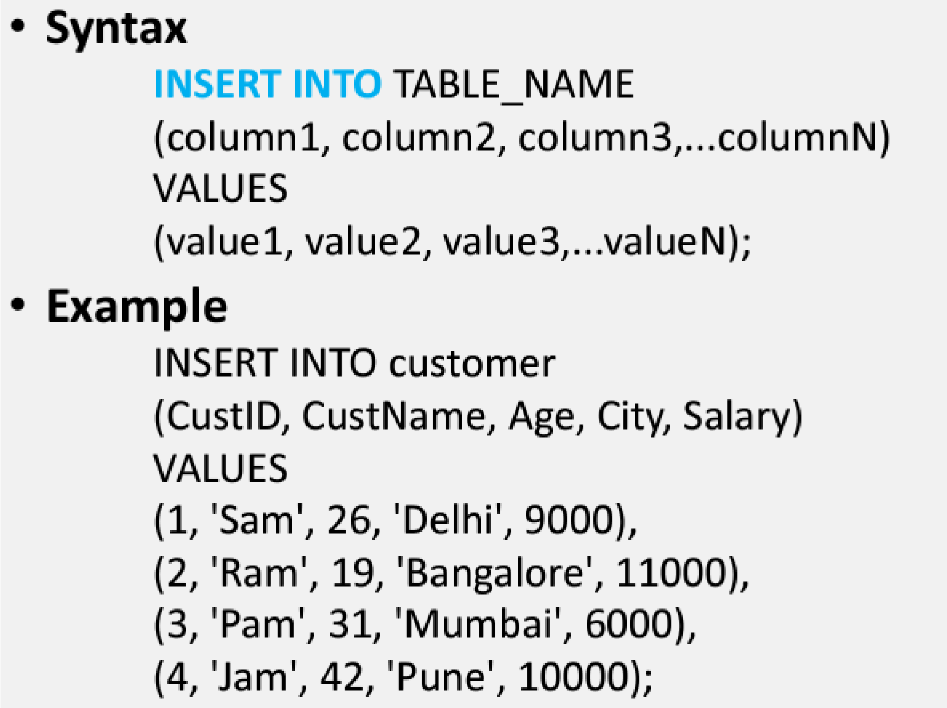
Constraints

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

Insert Values In Table

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



- **Syntax**

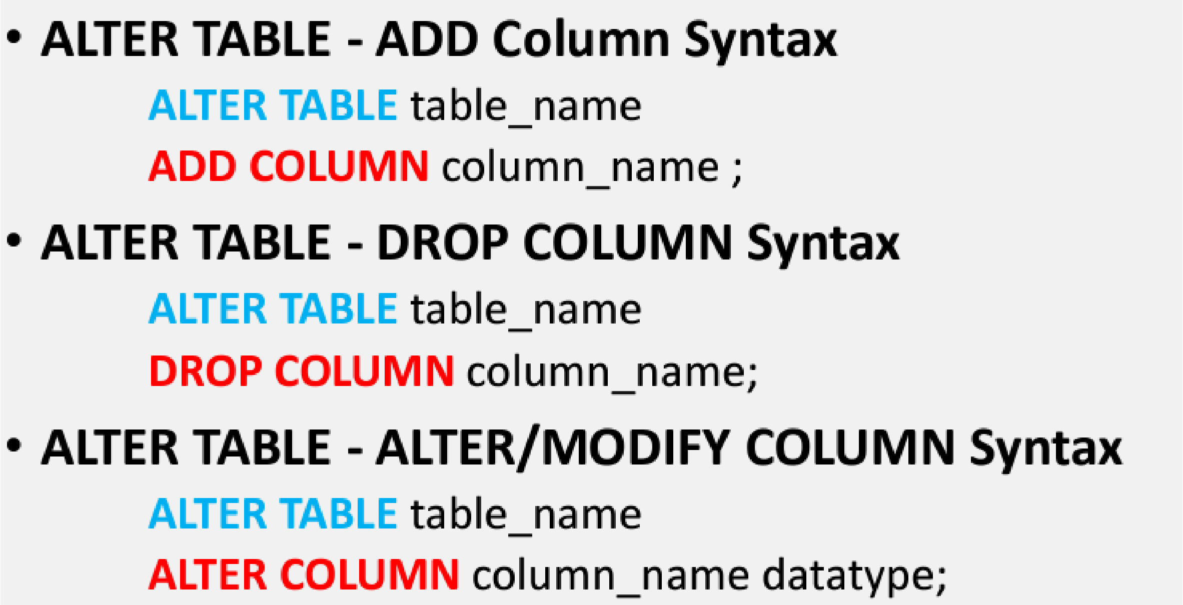
```
INSERT INTO TABLE_NAME
(column1, column2, column3,...columnN)
VALUES
(value1, value2, value3,...valueN);
```

- **Example**

```
INSERT INTO customer
(CustID, CustName, Age, City, Salary)
VALUES
(1, 'Sam', 26, 'Delhi', 9000),
(2, 'Ram', 19, 'Bangalore', 11000),
(3, 'Pam', 31, 'Mumbai', 6000),
(4, 'Jam', 42, 'Pune', 10000);
```

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 ;
```

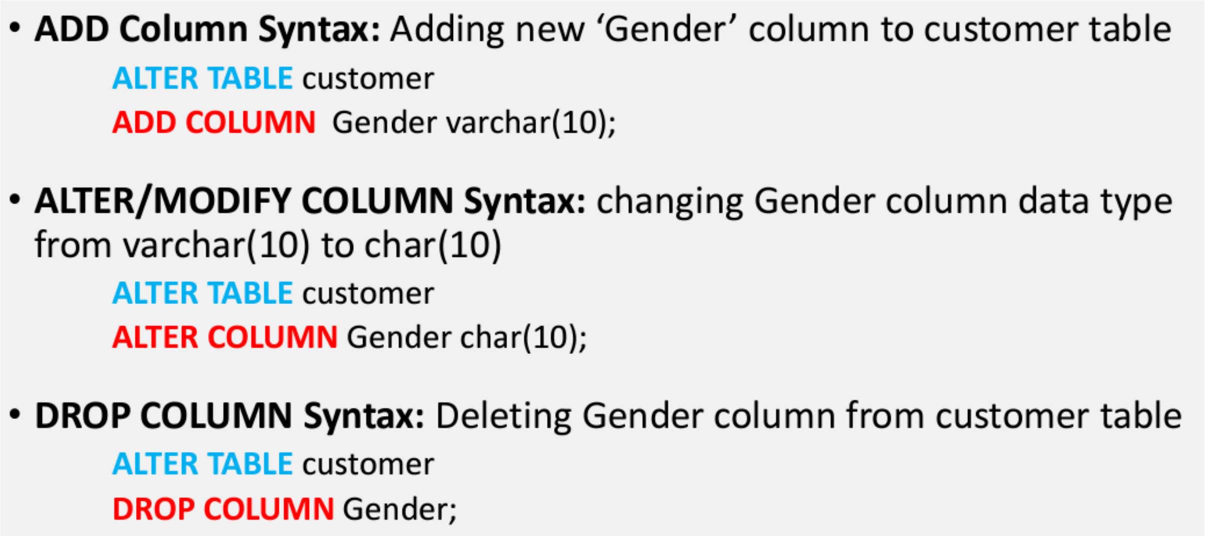
- **ALTER TABLE - DROP COLUMN Syntax**

```
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)

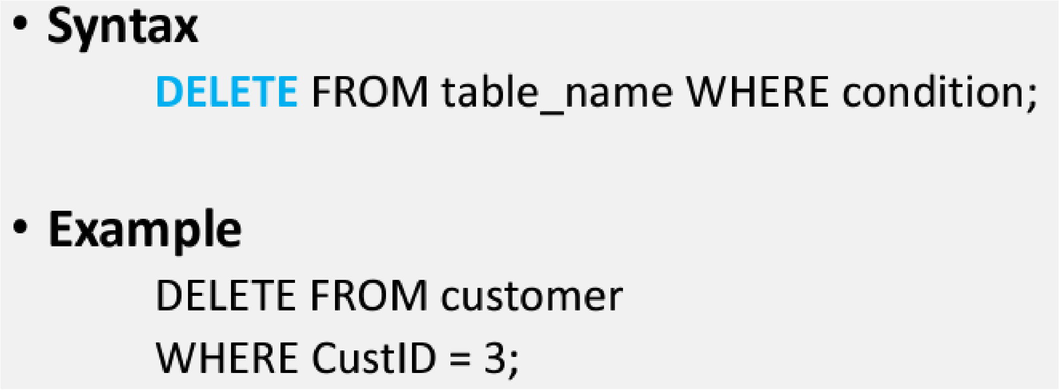
```
ALTER TABLE customer
ALTER COLUMN Gender char(10);
```

- **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**

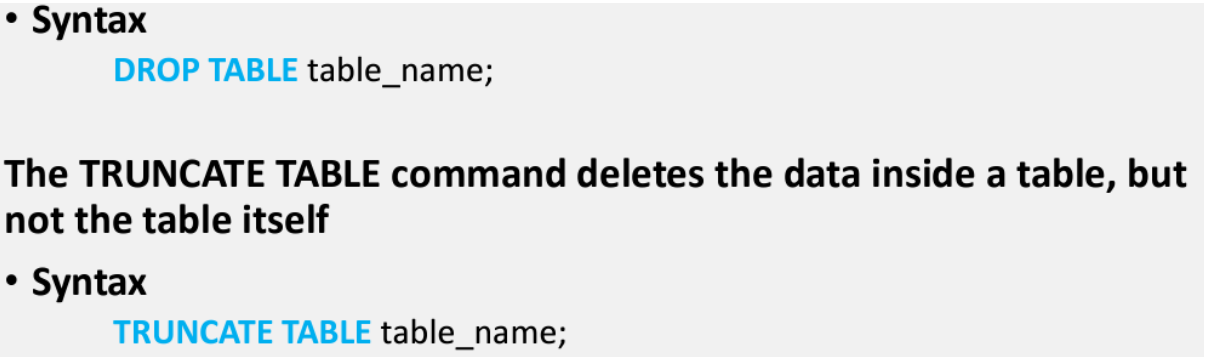
```
DELETE FROM table_name WHERE condition;
```

- **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;
```

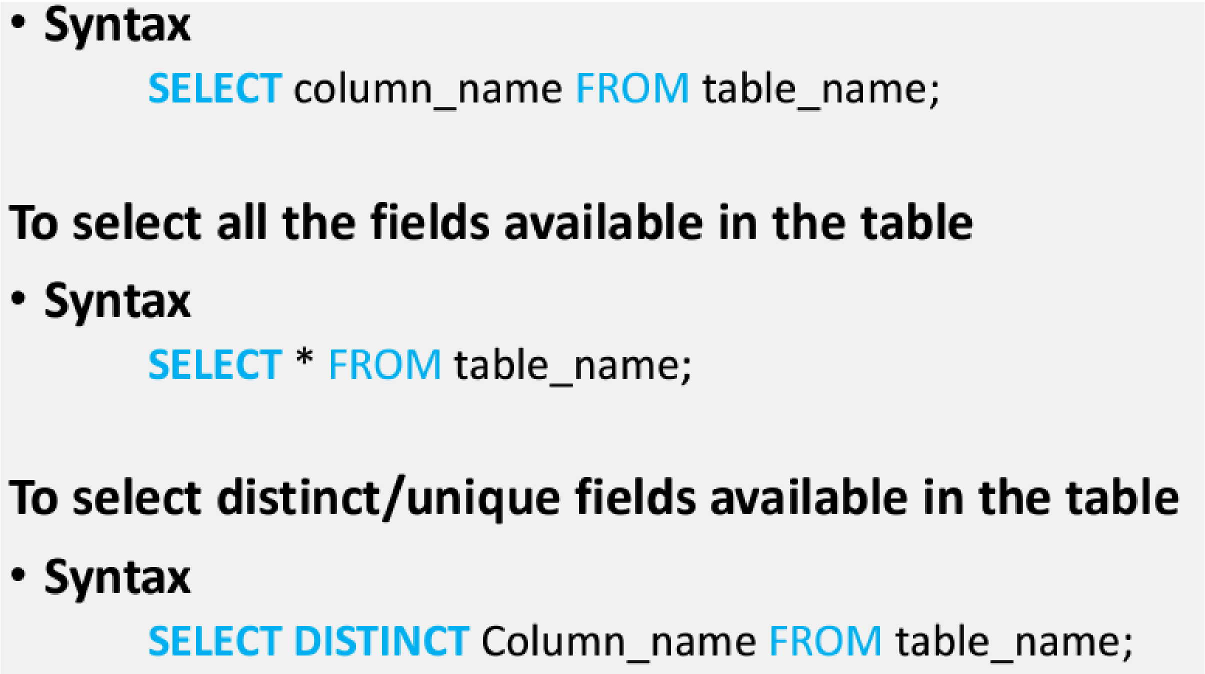
The TRUNCATE TABLE command deletes the data inside a table, but not the table itself

- **Syntax**

```
TRUNCATE TABLE table_name;
```

SELECT 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;
```

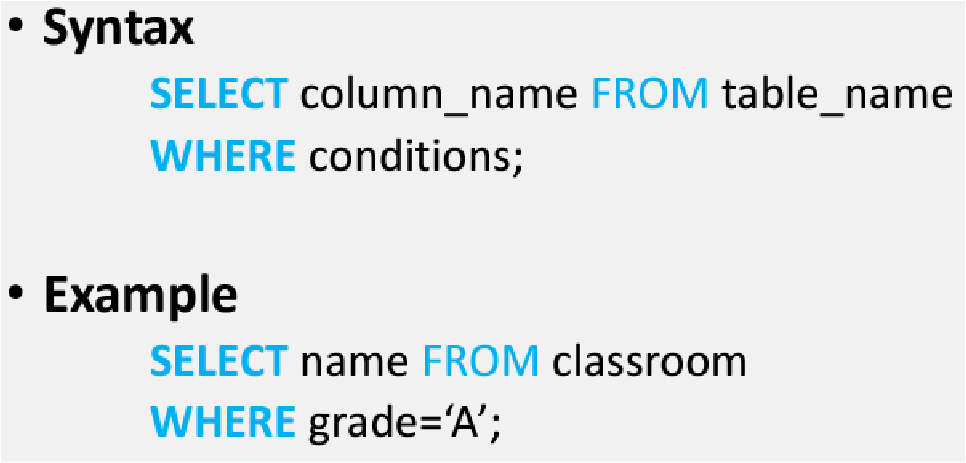
To select distinct/unique fields available in the table

- **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 (>=)**

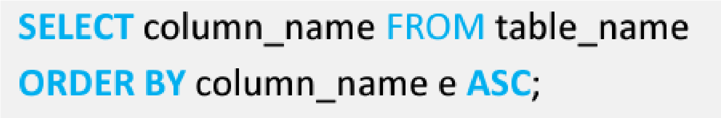
3. Logical operators: perform the Boolean operations •Example**: ALL, IN, BETWEEN, LIKE, AND, OR, NOT, ANY**

4. Bitwise operators: perform the bit operations on the Integer values •Example**: Bitwise AND (&), Bitwise OR(|)**

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;
```

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

Most Used Aggregate Functions

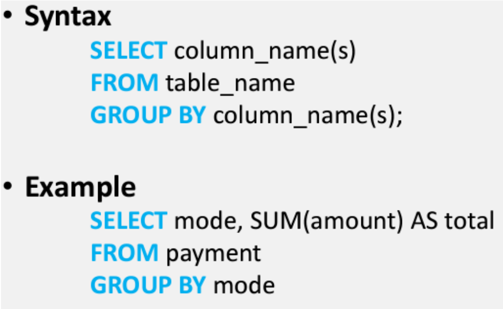
Aggregate function performs a calculation on multiple values and returns a single value and Aggregate functions are often used with GROUP BY & SELECT statement

- **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 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);
```

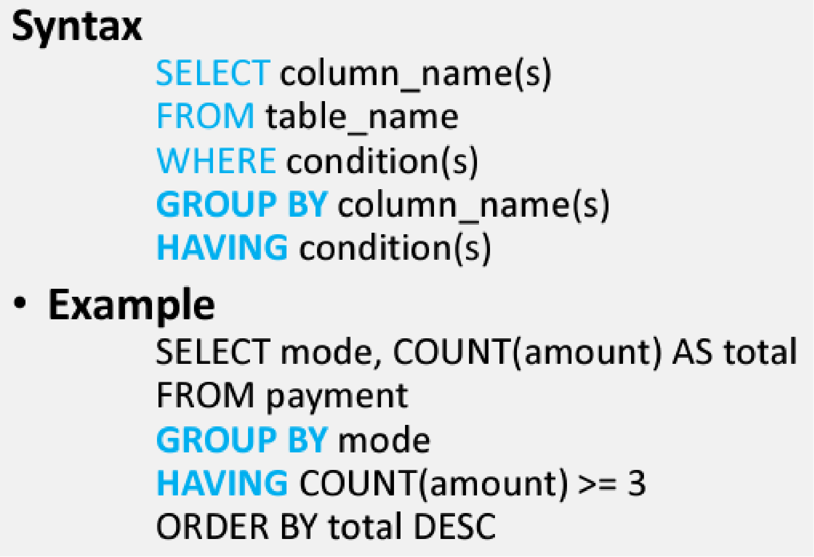
- **Example**

```
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 WHEREclause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause

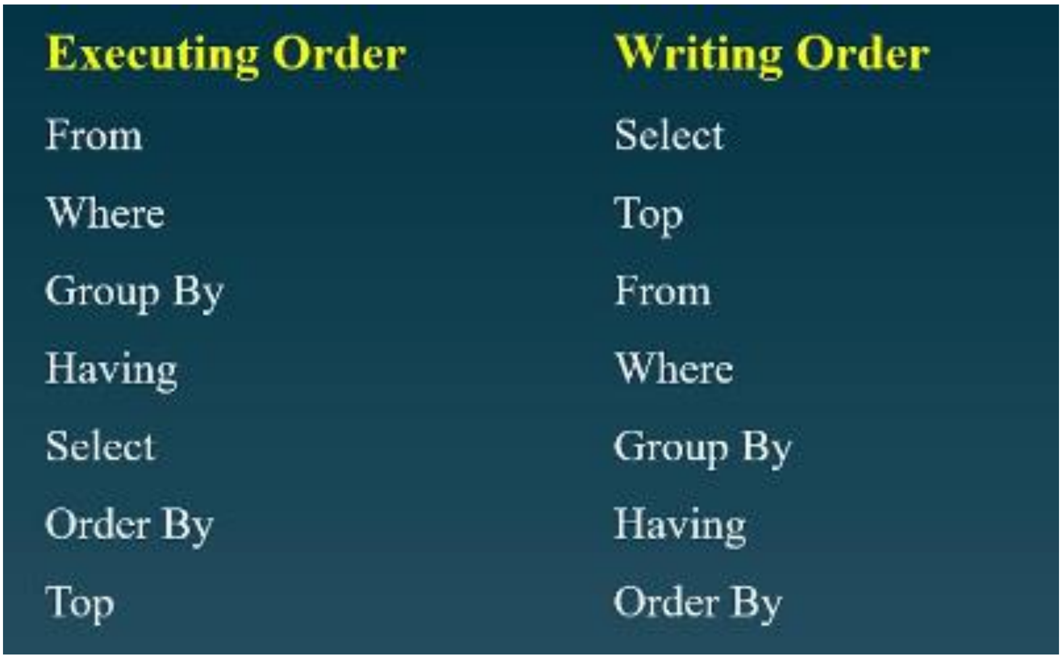


Syntax

```
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
```



Executing Order	Writing Order
From	Select
Where	Top
Group By	From
Having	Where
Select	Group By
Order By	Having
Top	Order By

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(MONTHFROM date_field) FROM Table

•YEAR

•QUARTER

•MONTH

•WEEK

•DAY

•HOUR

•MINUTE

•DOW–day of week

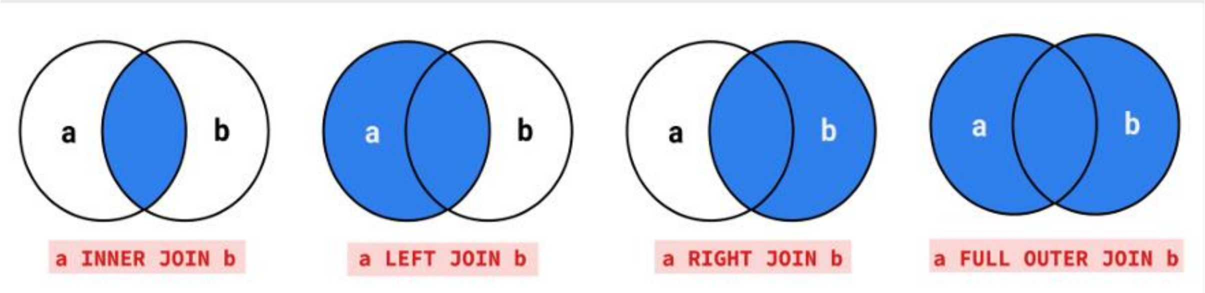
•DOY–day of year

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

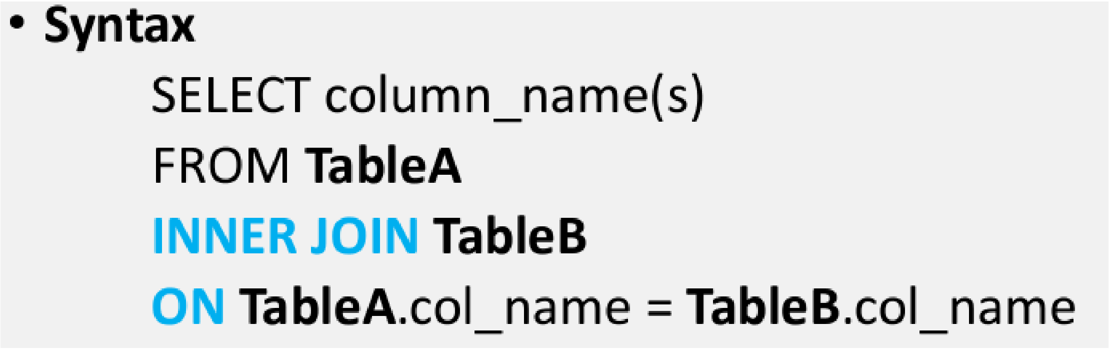
TYPES OF JOINS

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



INNER JOIN

Returns records that have matching values in both tables

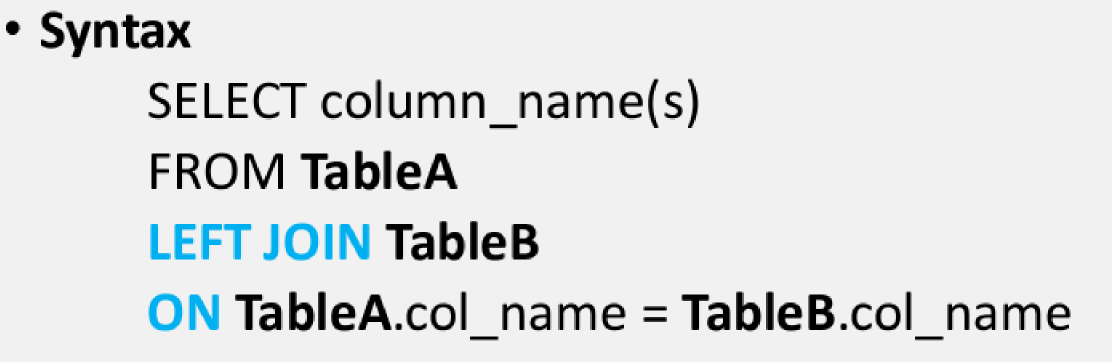


- **Syntax**

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

LEFT JOIN

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

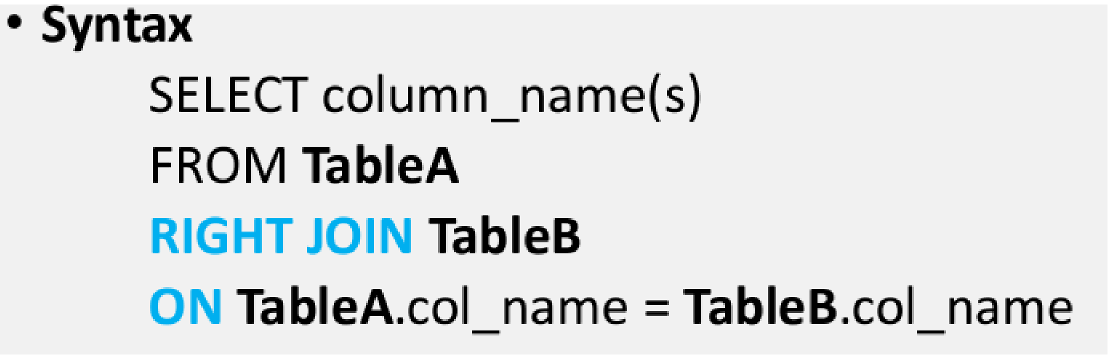


- **Syntax**

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

RIGHT JOIN

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

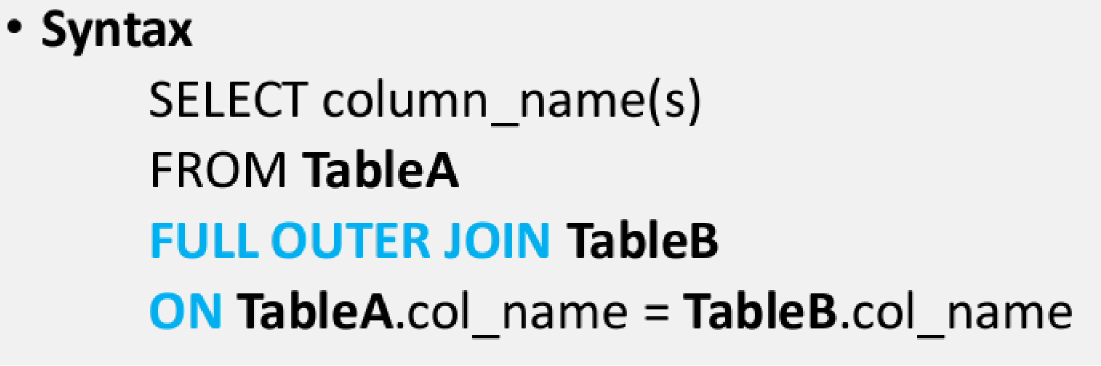


- **Syntax**

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

FULL JOIN

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



- **Syntax**

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

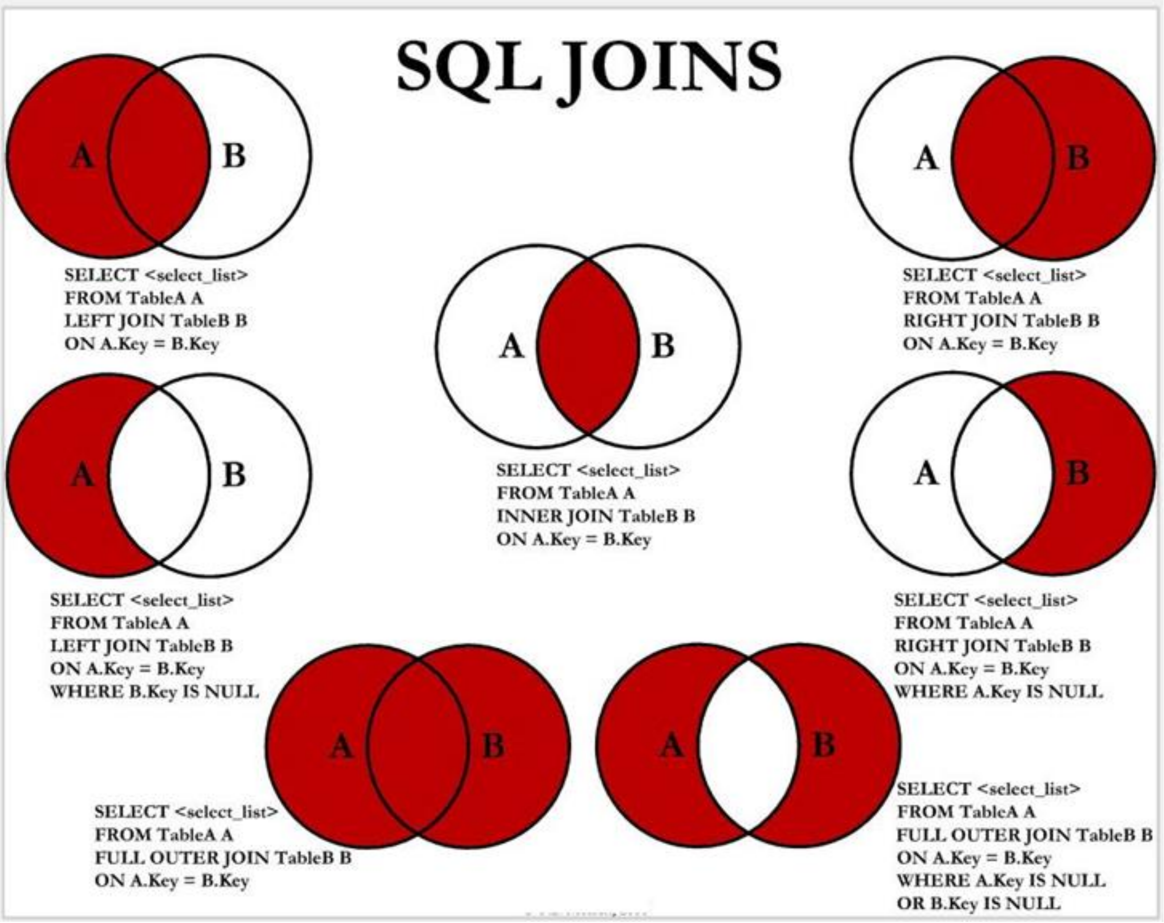
Which JOIN To Use

•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



```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

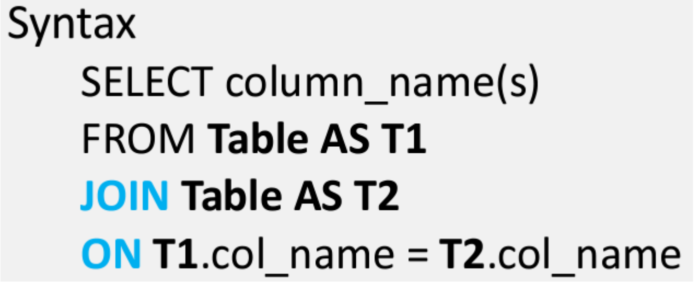
```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

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
```

UNION

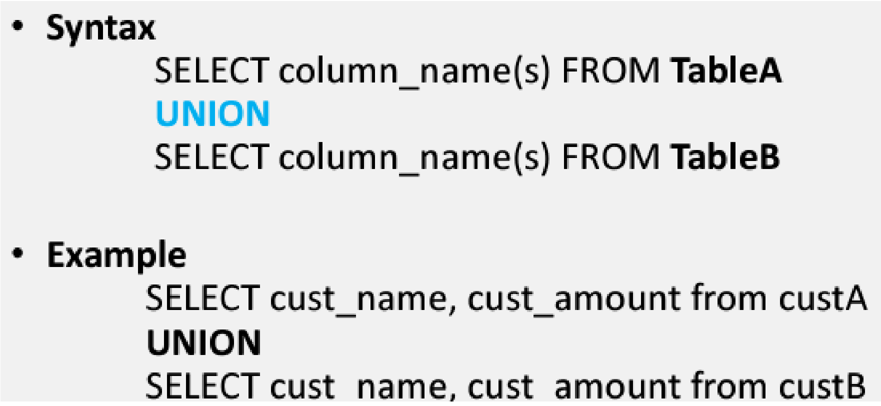
The SQL UNION clause/operator is used to combine/concatenate the results of two or more SELECT statements without returning any duplicate rows and keeps unique records

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**

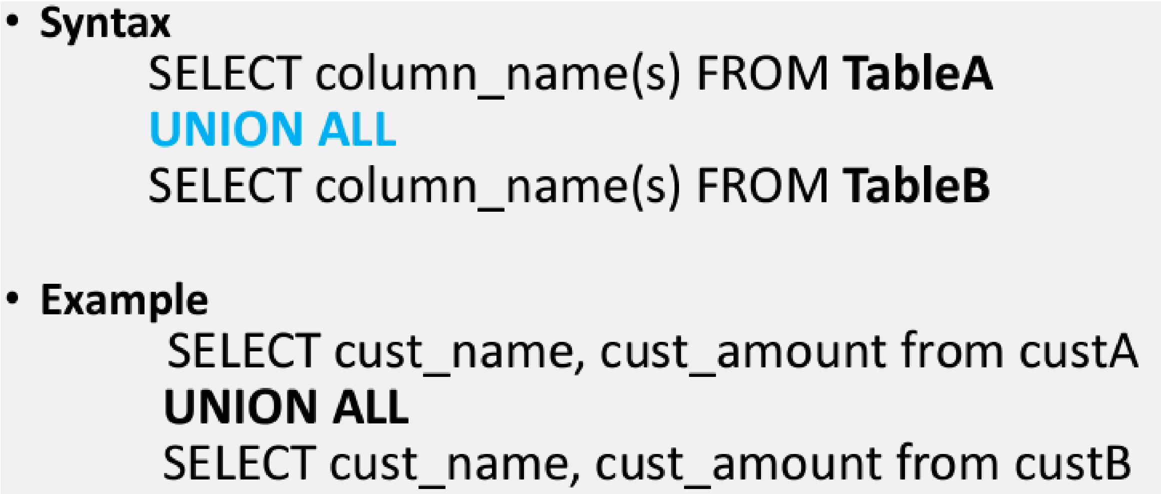
```
SELECT column_name(s) FROM TableA
UNION
SELECT column_name(s) FROM TableB
```

- **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
```

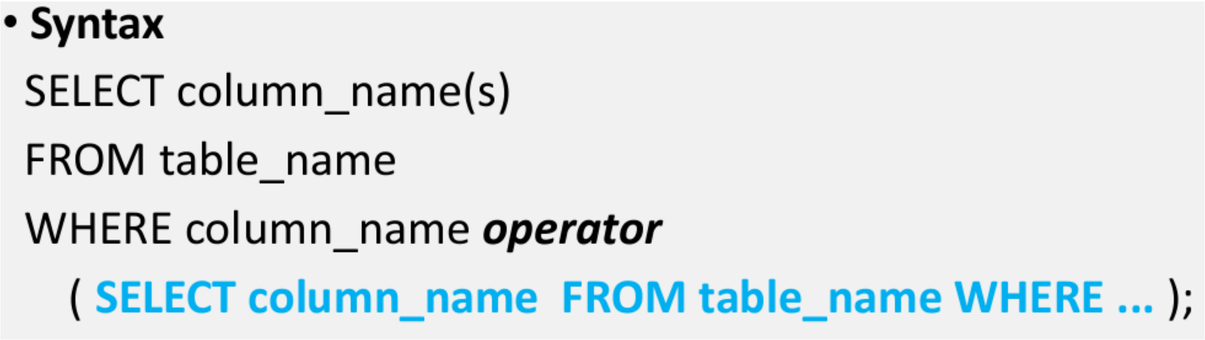
- **Example**

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

SUB QUERY

A Subqueryor, 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



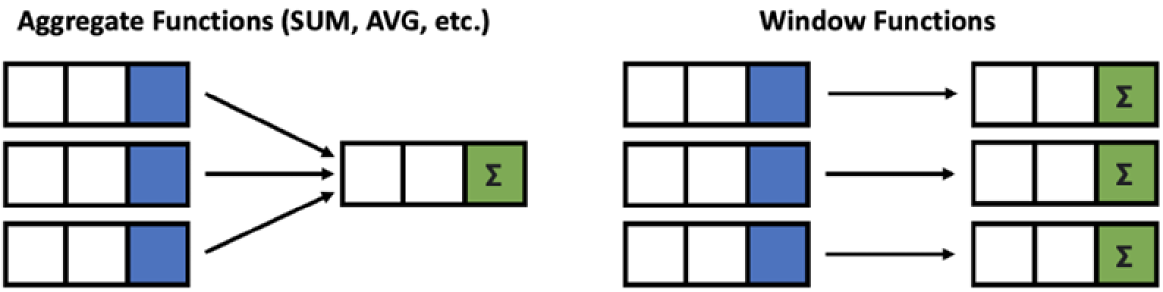
• **Syntax**

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator
   ( SELECT column_name  FROM table_name WHERE ... );
```

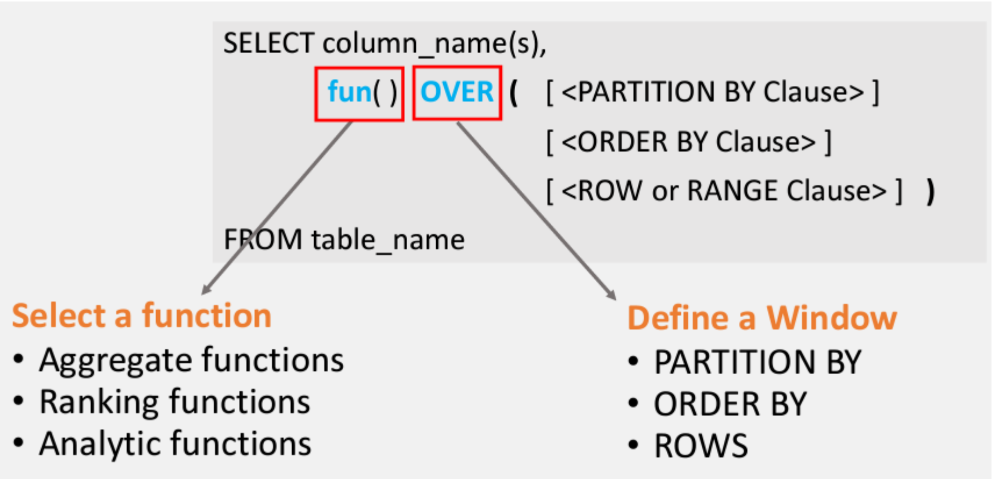
WINDOW FUNCTION

• Window functions applies aggregate, ranking and analytic functions over a particular window (set of rows).

• And OVERclause is used with window functions to define that window.



WINDOW FUNCTION SYNTAX



```
SELECT column_name(s),
        fun( ) OVER (  [ <PARTITION BY Clause> ]
                       [ <ORDER BY Clause> ]
                       [ <ROW or RANGE Clause> ]  )
FROM table_name
```

Select a function
- Aggregate functions
- Ranking functions
- Analytic functions

Define a Window
- PARTITION BY
- ORDER BY
- ROWS

WINDOW FUNCTION TERMS

- Window function applies aggregate, ranking and analytic functions over a particular window; for example, sum, avg, or row_number

- Expressionis 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

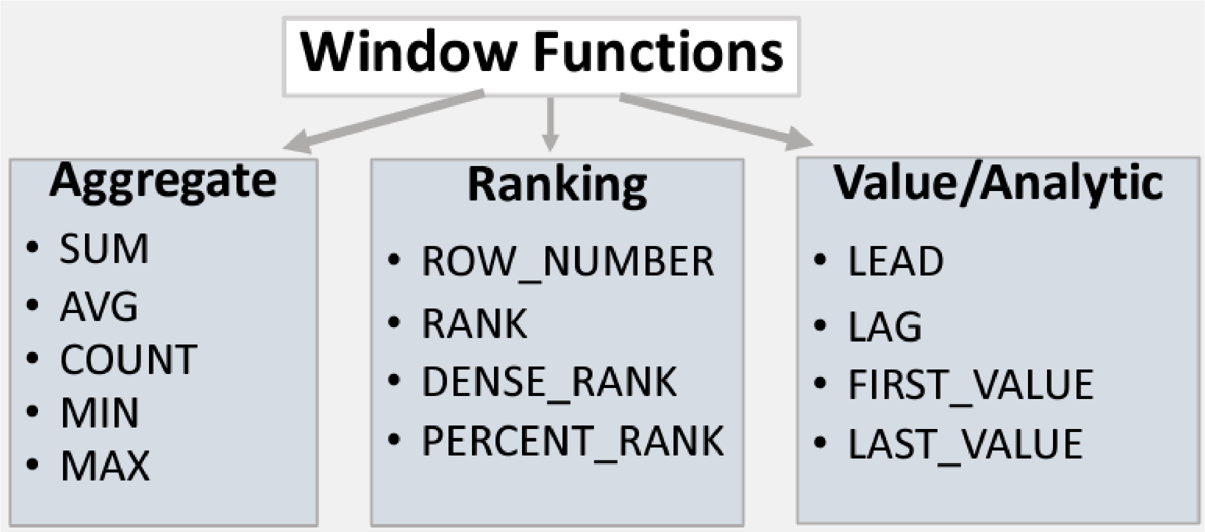
- PARTITION BY divides the rows into partitions so we can specify which rows to use to compute the window function

- 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

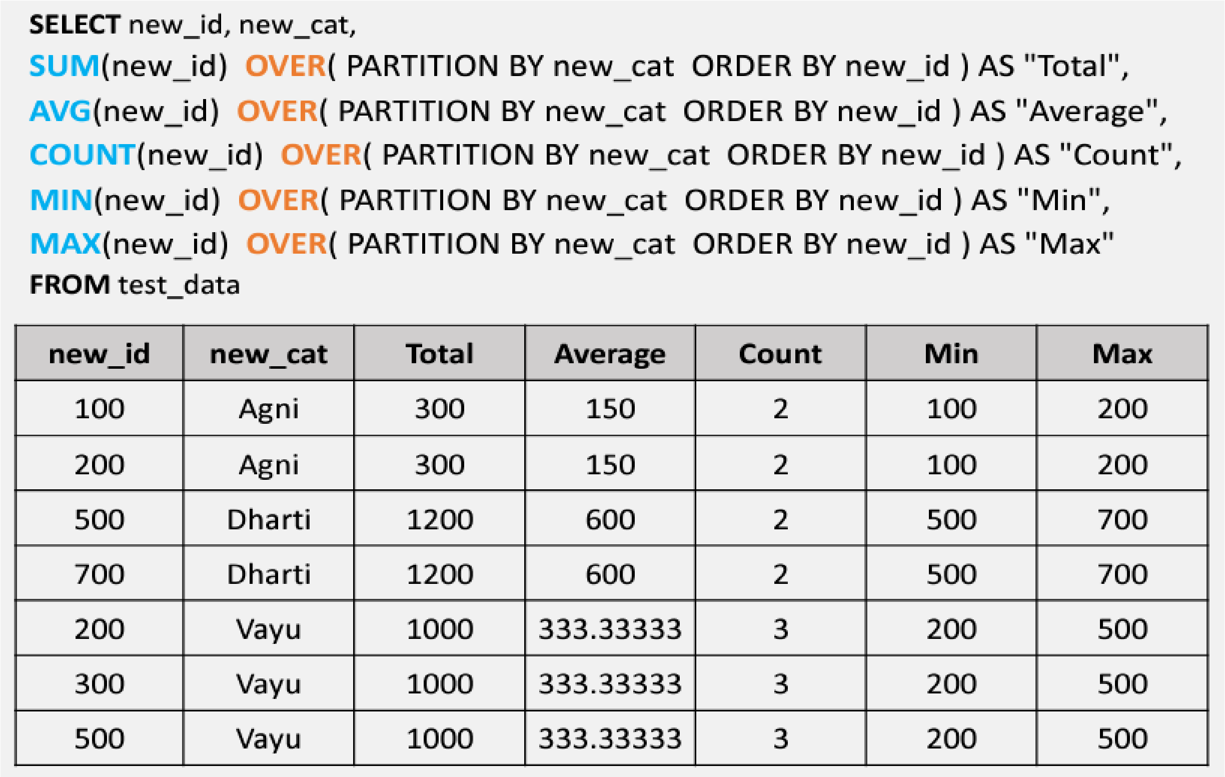


Window Functions

Aggregate
- SUM
- AVG
- COUNT
- MIN
- MAX

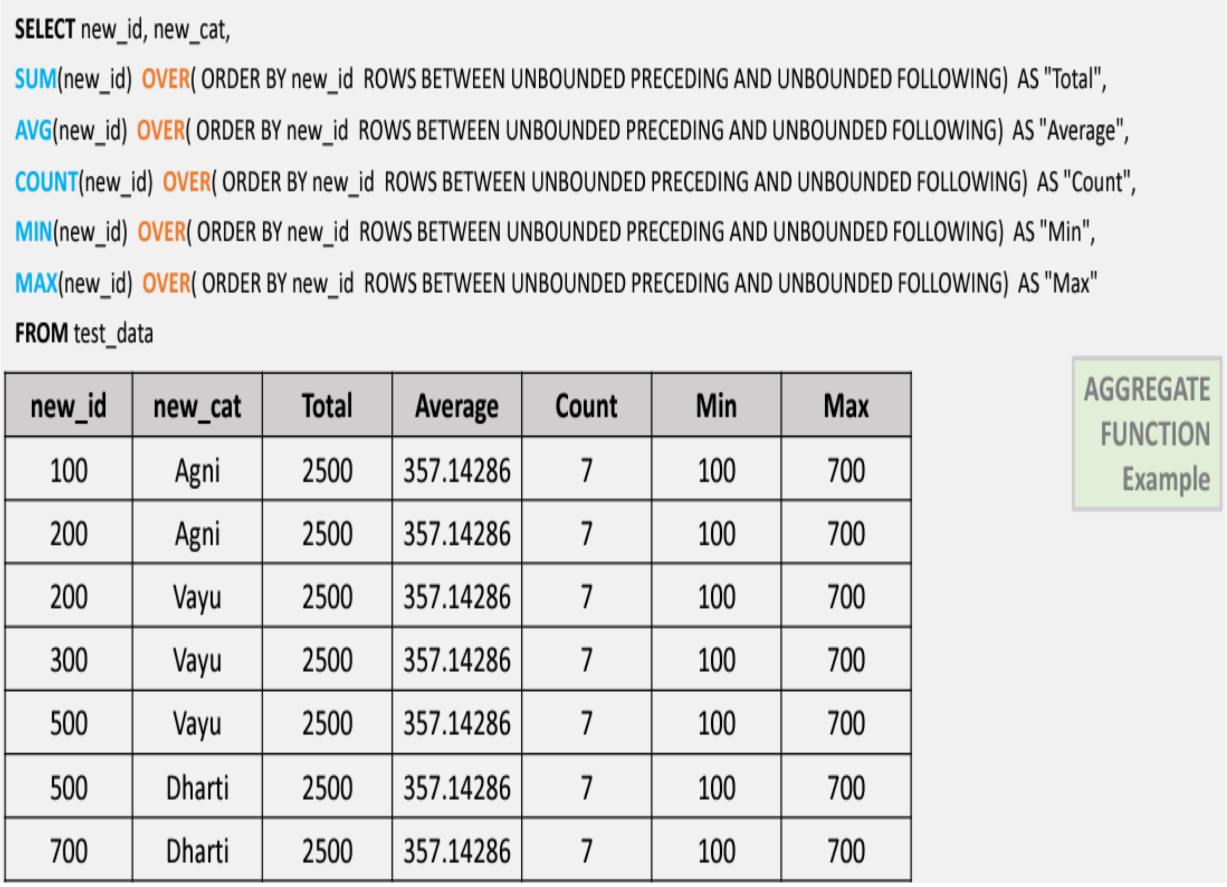
Ranking
- ROW_NUMBER
- RANK
- DENSE_RANK
- PERCENT_RANK

Value/Analytic
- LEAD
- LAG
- FIRST_VALUE
- LAST_VALUE



```
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
```

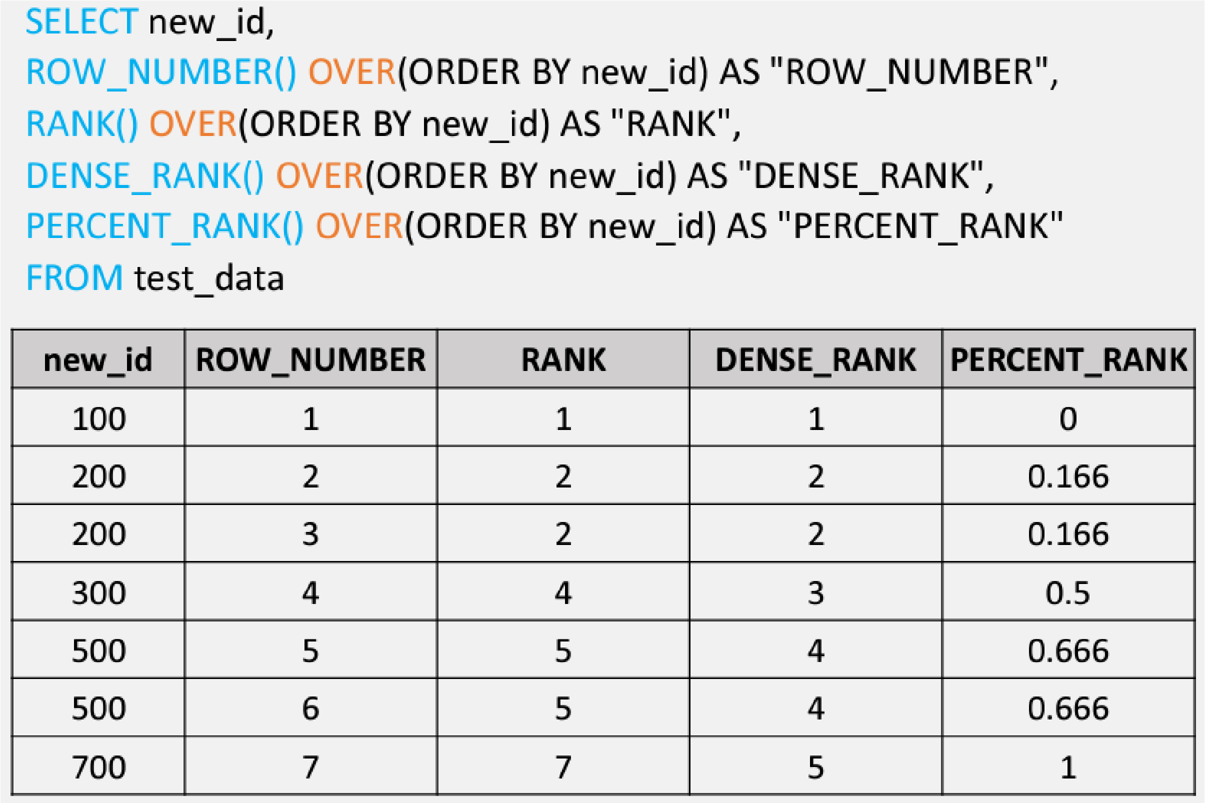
new_id	new_cat	Total	Average	Count	Min	Max
100	Agni	300	150	2	100	200
200	Agni	300	150	2	100	200
500	Dharti	1200	600	2	500	700
700	Dharti	1200	600	2	500	700
200	Vayu	1000	333.33333	3	200	500
300	Vayu	1000	333.33333	3	200	500
500	Vayu	1000	333.33333	3	200	500



```
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
```

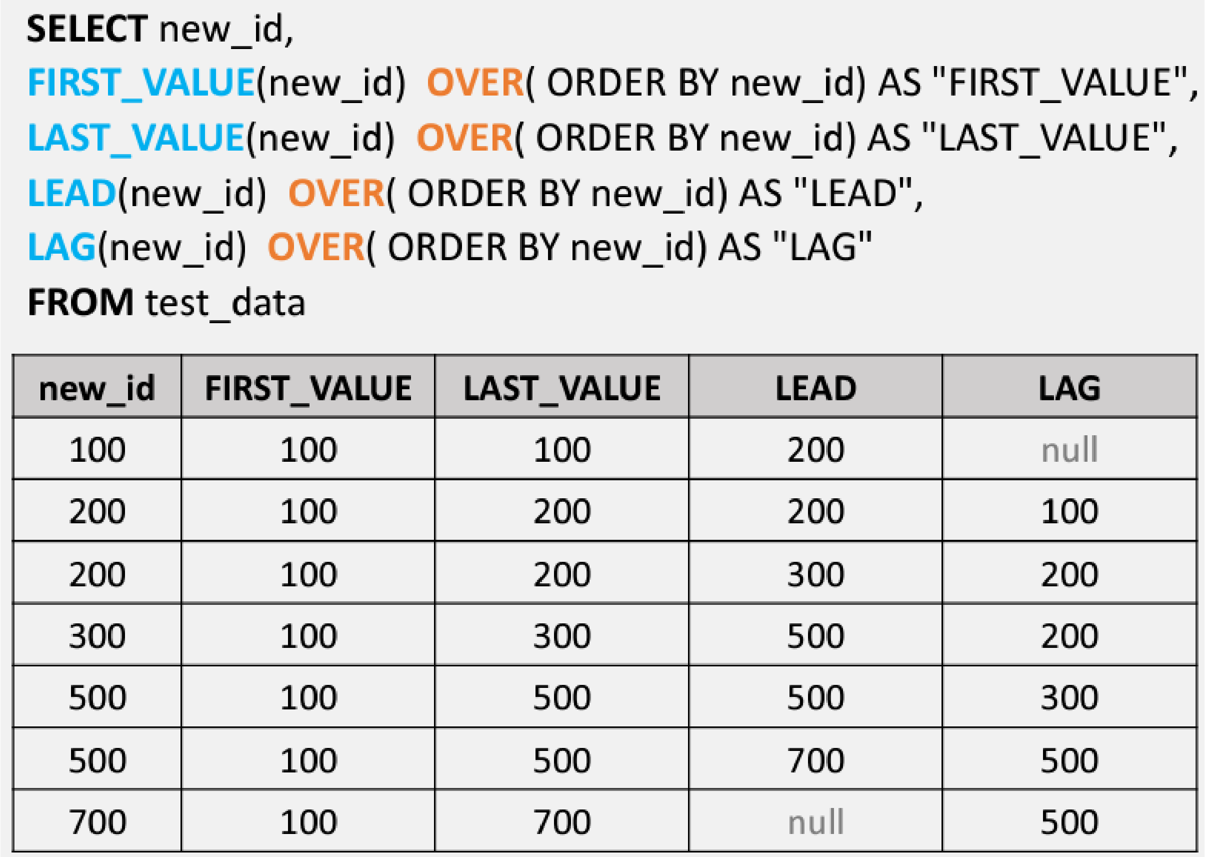
new_id	new_cat	Total	Average	Count	Min	Max
100	Agni	2500	357.14286	7	100	700
200	Agni	2500	357.14286	7	100	700
200	Vayu	2500	357.14286	7	100	700
300	Vayu	2500	357.14286	7	100	700
500	Vayu	2500	357.14286	7	100	700
500	Dharti	2500	357.14286	7	100	700
700	Dharti	2500	357.14286	7	100	700

AGGREGATE FUNCTION Example



```
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



```
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
```

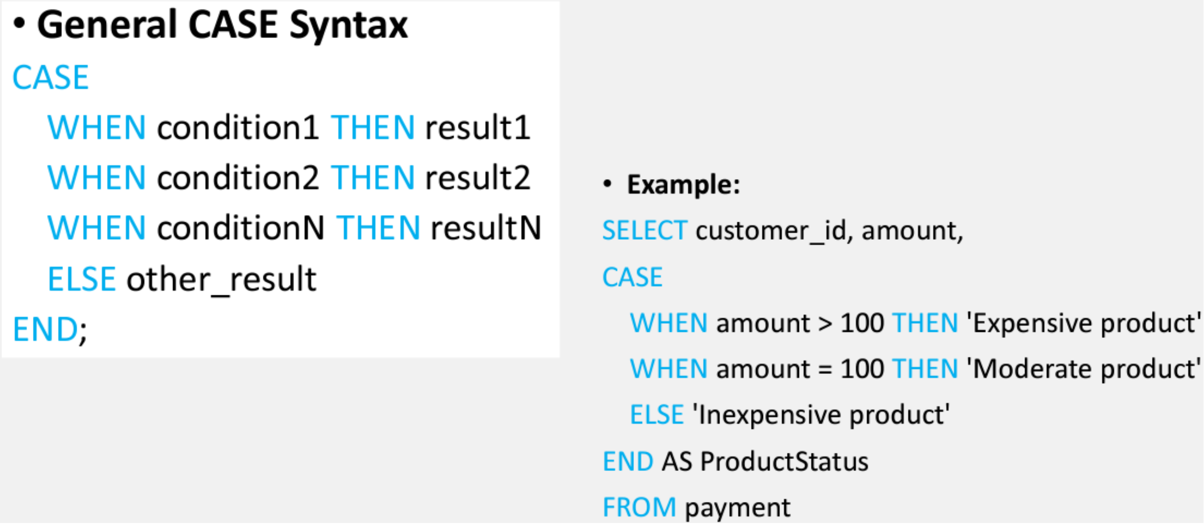
new_id	FIRST_VALUE	LAST_VALUE	LEAD	LAG
100	100	100	200	null
200	100	200	200	100
200	100	200	300	200
300	100	300	500	200
500	100	500	500	300
500	100	500	700	500
700	100	700	null	500

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



• **General CASE Syntax**

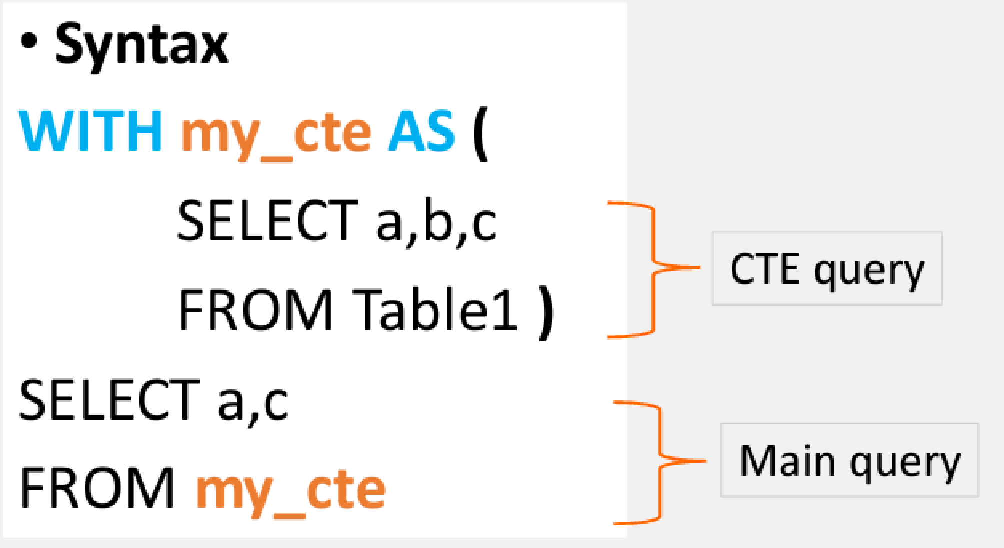
```
CASE
   WHEN condition1 THEN result1
   WHEN condition2 THEN result2
   WHEN conditionN THEN resultN
   ELSE other_result
END;
```

• **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
```

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



• **Syntax**

```
WITH my_cte AS (
        SELECT a,b,c
        FROM Table1 )
SELECT a,c
FROM my_cte
```