**Q1) What are the level of abstraction in dbms?**

First What is data Abstraction?

=> Hiding the complexity and providing only the essential Details to the user.

There are 3 Level of abstraction in dbms:-

1. Physical Level.
2. Logical Level.
3. View Level.

1) Physical Level:-

* This is the lowest level of abstraction.
* This level Deals with how the data are stored in the database.

2) Logical Level:-

* This level Deals with what data are actually stored and what is the relationship exist among those data.

3) View Level:-

* This is the highest level of abstraction.
* Only a part of the actual database is viewed by the users.
* Users view data in the form of rows and columns.
* Users can just view the data and interact with the database, storage and implementation details are hidden from them.

**Q2) What are different types of relationships in SQL?**

* One-to-One (1:1) Relationship: In this type of relationship, each record in the first table is related to exactly one record in the second table, and vice versa. For example, consider a database where each employee has exactly one office assigned to them.
* One-to-Many (1:N) Relationship: In a one-to-many relationship, each record in the first table is related to one or more records in the second table, but each record in the second table is related to only one record in the first table. For instance, in a database where one department can have many employees, but each employee belongs to only one department.
* Many-to-Many (N:M) Relationship: In this type of relationship, multiple records in one table can be related to multiple records in another table. To represent a many-to-many relationship in a relational database, a junction table or associative entity is often used. For example, in a database modeling students and courses, where each student can enroll in multiple courses, and each course can have multiple students.

**Blog Link :- https://www.edureka.co/blog/normalization-in-sql/**

**Q3) What is Normalization and different type of normalization?**

* Normalization is the process of organizing the data in the database.
* It is used to reduce the data redundancy in the database. Data redundancy in dbms means having the same data at multiple place. It is necessary to remove the data redundancy because it causes anomalies in a database which make it very hard for a database administrator to maintain it.

First Normal Form (1NF) :-

* Using row order to convey information is not permitted.
* Mixing data types within the same column is not permitted.
* Having a table without a primary key is not permitted.
* Repeating groups are not permitted.

Second Normal Form (2NF) :-

* The table has to be in 1NF.
* The table also should not contain partial dependency.

Third Normal Form (3NF) :-

* The Table has to be in 2NF.
* There should be no transitive dependency for non-prime attributes.
* That means non-prime attributes (which doesn’t form a candidate key) should not be dependent on other non-prime attributes in a given table.

**Q4) What are different types of keys?**

1. Primary key
2. Foreign Key
3. Super key
4. Candidate key
5. Alternate key
6. Composite key
7. Unique key

1) Primary Key

* A primary key is a column or a set of columns that uniquely identifies each record (row) in a table.
* A table can contain only one primary key constraint.
* A primary key cannot have null value.
* There can be no duplicate value for a primary key.

2) Foreign Key:-

* A foreign key is a field in a table that is a primary key in another table. It establishes a relationship between two tables and enforces referential integrity.

**CREATE TABLE Customers (**

**CustomerID INT PRIMARY KEY,**

**CustomerName VARCHAR(100),**

**ContactNumber VARCHAR(15)**

**);**

**CREATE TABLE Orders (**

**OrderID INT PRIMARY KEY,**

**OrderDate DATE,**

**CustomerID INT,**

**FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)**

**);**

3) Super Key:-

* Super key is a single key or a group of multiple keys that can uniquely identify a tuple.
* Super key can contain null values.

Example :- Let's consider an **EMPLOYEE\_DETAIL** table example where we have the following attribute:

Emp\_ssn, Emp\_id, Emp\_name, Emp\_email.

Set of super keys obtained :-

* {Emp\_ssn}
* {Emp\_id}
* {Emp\_email}
* {Emp\_ssn, Emp\_id}
* {Emp\_id, Emp\_name}
* {Emp\_ssn, Emp\_id, Emp\_email}
* {Emp\_ssn, Emp\_name, Emp\_id}

4) Candidate Key :-

* A candidate key is a column or set of column that can uniquely identify each record in a table.
* A table could contain multiple candidate key.
* This key cannot store null values.
* It must contain unique values.
* Candidate key is a subset of super key.

Example :- Consider a table Employees with the following columns:

**CREATE TABLE Employees (**

**EmployeeID INT,**

**Email VARCHAR(100),**

**SocialSecurityNumber VARCHAR(20),**

**FirstName VARCHAR(50),**

**LastName VARCHAR(50)**

**);**

In this table:

* **EmployeeID** could be a candidate key.
* **Email** could be a candidate key.
* **SocialSecurityNumber** could be a candidate key.

Each of these columns can uniquely identify a row in the Employees table, assuming they have unique values and do not contain NULLs.

5) Alternate Key :-

* Any candidate key that is not a part of primary key is an alternate key.
* For Example :- we have an employee table having multiple candidate key like id, name, salary, age, gender. If we make a column id as a primary key then rest of the candidate keys are called alternate keys.

### **6) Composite Key**

Sometimes, a table might not have a single column/attribute that uniquely identifies all the records of a table. To uniquely identify rows of a table, a combination of two or more columns/attributes can be used.  It still can give duplicate values in rare cases. So, we need to find the optimal set of attributes that can uniquely identify rows in a table.

* It acts as a primary key if there is no primary key in a table
* Two or more attributes are used together to make a [composite key](https://www.geeksforgeeks.org/composite-key-in-sql/).
* Different combinations of attributes may give different accuracy in terms of identifying the rows uniquely.

7) Unique Key :-

* Unique key is a column or group of columns in a table that uniquely identify every row in that table.
* Unique key can’t contain duplicated values.
* A table can have more than one unique key.
* Unique key field can allow only one null value in a column.

Example :- Let's consider an **EMPLOYEE** table example where we have the following attribute:

Emp\_id, mobile\_No., licence\_No., Registration\_No., Passport\_No., First\_name, Last\_name, Age

Unique Keys are :- Emp\_id, mobile\_NO, licence\_NO, Registration\_NO, Passport\_No.

**Q5) What are join and explain different types of joins.**

What are joins in sql?

=> The purpose of joins is to combine data from multiple table and show it to the single table.

There are 4 types of joins in sql.

1. Inner Join
2. Left Join
3. Right Join
4. Full Join

**1) Inner Join:-**

An inner join returns only the rows that have matching values in both tables.

Example :-

SELECT \*

FROM CUSTOMER AS C

INNER JOIN PAYMENT AS P

ON C.CUSTOMER\_ID = P.CUSTOMER\_ID

-----------------------------------

**2) Left Join:-**

A left join returns all the rows from the left table, and the matching rows from the right table. If there is no match in the right table, it returns NULL values.

Example :-

SELECT \*

FROM CUSTOMER AS C

LEFT JOIN PAYMENT AS P

ON C.CUSTOMER\_ID = P.CUSTOMER\_ID

**3) Right Join:-**

A right join returns all the rows from the right table and the matching rows from the left table. If there is no match in the left table, it returns NULL values.

Example :-

SELECT \*

FROM CUSTOMER AS C

RIGHT JOIN PAYMENT AS P

ON C.CUSTOMER\_ID = P.CUSTOMER\_ID

**4) Full Join:-**

A full outer join returns all the rows from both tables. It includes the matching rows from both tables and the non-matching rows with NULL values for the missing columns.

Example :-

SELECT \*

FROM CUSTOMER AS C

FULL OUTER JOIN PAYMENT AS P

ON C.CUSTOMER\_ID = P.CUSTOMER\_ID

**Q6) What is subquery?**

A subquery is a query within another query.

**Blog Link** :- <https://www.geeksforgeeks.org/sql-subquery/>

**Q7) How to do Transaction Control?**

* Commit
* Rollback
* Savepoint

**Commit:**

* It is used to save the transaction on the database.

**Rollback:**

* It is used to restore the database to original since the last Commit.

**Q8) What is difference between drop, truncate and delete commands?**

**Drop :-**

* A SQL DROP TABLE statement is used to delete a table definition and all data from a table.

**Truncate :-**

* A truncate SQL statement is used to remove all rows (complete data) from a table. It is similar to the DELETE statement with no WHERE clause.

**Delete :-**

* The DELETE statement is used to delete rows from a table. If you want to remove a specific row from a table you should use WHERE condition.

**Q9) What is Denormalization?**

* It combines the data and organizes it in a single table.
* Denormalization is a process of adding redundant data to the normalized relational database to optimize its performance.

**Q10) What are Triggers in Sql?**

1. Triggers are stored program/procedures, which are automatically executed or fired when some events occurs.
2. Events can be any of the following :-

* Database Manipulation (DML) :- Statement like Delete, Insert, update
* Database Definition (DDL) :- Statement like create, alter, drop
* Database Operation :- like :- LOGON, LOGOFF, STARTUP OR SHUTDOWN

1. Triggers can be defined on table, view, Schema or database with which the event is associated.

Syntex Of Triggers :-

CREATE [OR REPLACE] TIRGGER trigger\_name

{BEFORE | AFTER}

{INSERT [OR] | UPDATE [OR] | DELETE}

ON table\_name

[FOR EACH ROW]

DECLEAR

Declaration-statement

BEGIN

Executable-statement

END

**Stored Procedure :-**

If we have some piece of code that we want to reuse again and again.

**Cast Statement :-**

**General Case Syntex :-**

SELECT column1, column2

CASE

WHEN condition1 THEN result1

WHEN condition2 THEN result2

WHEN condition3 THEN result3

ELSE other\_result

END AS alias\_name

FROM table\_name;

Example :-

We have a table having two columns :-

**Input :-**

Payment :-

|  |  |
| --- | --- |
| customer\_id | amount |
| 1 | 60 |
| 2 | 70 |
| 3 | 80 |
| 4 | 500 |
| 5 | 100 |

**Query :-**

SELECT customer\_id, amount

CASE

WHEN amount > 100 THEN ‘Expensive product’

WHEN amount = 100 THEN ‘Moderate product’

ELSE ‘Inexpensive product’

END AS product\_status

FROM payment;

**Output:-**

|  |  |  |
| --- | --- | --- |
| customer\_id | amount | product\_status |
| 1 | 60 | Inexpensive product |
| 2 | 70 | Inexpensive product |
| 3 | 80 | Inexpensive product |
| 4 | 500 | Expensive product |
| 5 | 100 | Moderate product |

**Case Expression Syntex :-**

SELECT column1, column2

CASE Expression

WHEN value1 THEN result1

WHEN value2 THEN result2

WHEN value3 THEN result3

ELSE other\_result

END AS alias\_name

FROM table\_name;

Example :-

We have a table having two columns :-

**Input :-**

Payment:-

|  |  |
| --- | --- |
| customer\_id | amount |
| 1 | 60 |
| 2 | 70 |
| 3 | 80 |
| 4 | 500 |
| 5 | 100 |

Query:-

SELECT customer\_id

CASE amount

WHEN 500 THEN ‘Prime Customer’

WHEN 100 THEN ‘Plus Customer’

ELSE ‘Regular Customer’

End AS customer\_status

FORM payment;

**Output:-**

|  |  |  |
| --- | --- | --- |
| customer\_id | amount | customer\_status |
| 1 | 60 | Regular Customer |
| 2 | 70 | Regular Customer |
| 3 | 80 | Regular Customer |
| 4 | 500 | Prime Customer |
| 5 | 100 | Plus Customer |

IF Clause :-

SELECT column1, column2

IF(condition, TRUE result, FALSE result)

AS alias\_name

FROM table\_name;

Example :-

We have 3 columns in Students table having id, name, percentage columns.

**INPUT :-**

**Students :-**

|  |  |  |
| --- | --- | --- |
| **Id** | **Name** | **Percentage** |
| 1 | Ram | 45 |
| 2 | Sarita | 85 |
| 3 | Salman | 29 |
| 4 | Juhi | 47 |
| 5 | Aditya | 33 |
| 6 | John | 22 |
| 7 | Sumit | 120 |

**Query :-**

SELECT Id, name

IF(percentage >= 33, ‘Pass’, ‘Fail’)

AS Result

FROM students;

**OUTPUT :-**

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Name** | **Percentage** | **Result** |
| 1 | Ram | 45 | Pass |
| 2 | Sarita | 85 | Pass |
| 3 | Salman | 29 | Fail |
| 4 | Juhi | 47 | Pass |
| 5 | Aditya | 33 | Pass |
| 6 | John | 22 | Fail |
| 7 | Sumit | 120 | Pass |

**Update Example :-**

UPDATE students SET

percentage = (

CASE id

WHEN 3 THEN 39

WHEN 7 THEN 62

END)

WHERE id IN (3,6);

**Window Function in SQL :-**

Aggregate

Function

COUNT(expr)

SUM(expr)

AVG(expr)

Max(expr)

Min(expr)

ROW\_NUMBER()

RANK()

DENSE\_RANK()

CUME\_DIST()

PERCENT\_RANK()

NTILE(n)

Rank

Function

Window

Function

Value

(Analytics)

Function

LEAD(expr, offset, default)

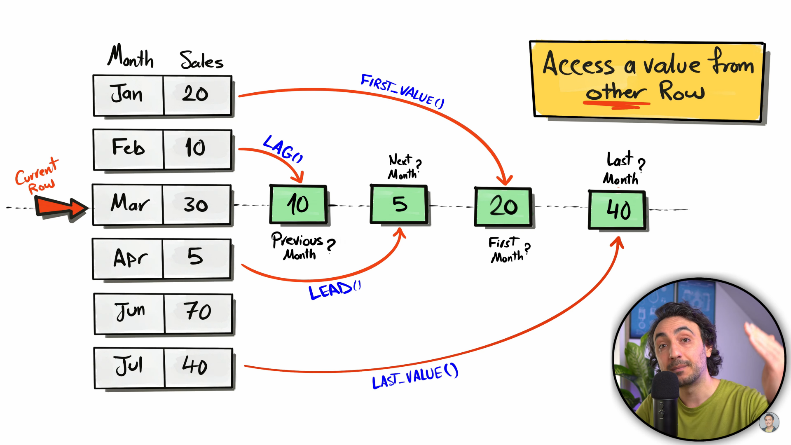
LAG(expr, offset, default)

FIRST\_VALUE(expr)

LAST\_VALUE(expr)

**Syntex :-**

**AVG(Sales) OVER (PARTITON BY category ORDER BY orderDate ROWS UNBOUNDED PRECEDING)**

1. **Lag()**
2. **Lead()**
3. **First\_value()**
4. **Last\_value()**
5. **LAG(expression, offset, default)**

Expression = Any data type

Offset = 1

Default = NULL

Partition by = Optional

Order by = Required

**Example :- LAG(sales, 2, 10) OVER(PARTITION BY product\_id ORDER BY orderDate)**

1. **LEAD(expression, offset, default)**

Expression = Any data type

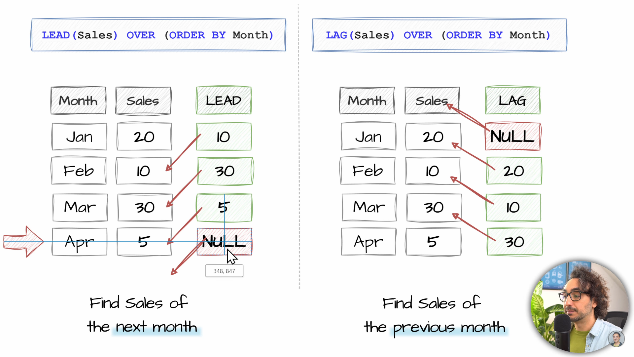
Offset = 1

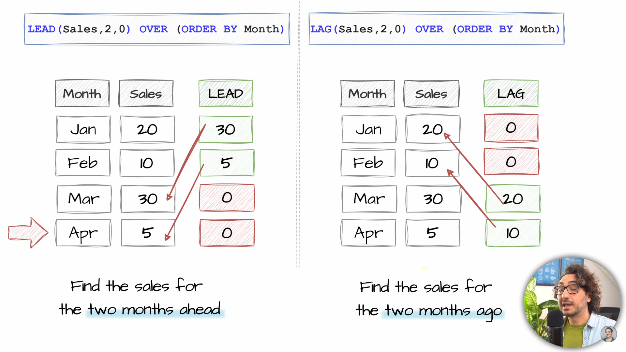
Default = NULL

Partition by = Optional

Order by = Required

**Example :- LEAD(sales, 2, 10) OVER(PARTITION BY product\_id ORDER BY orderDate)**

****

****

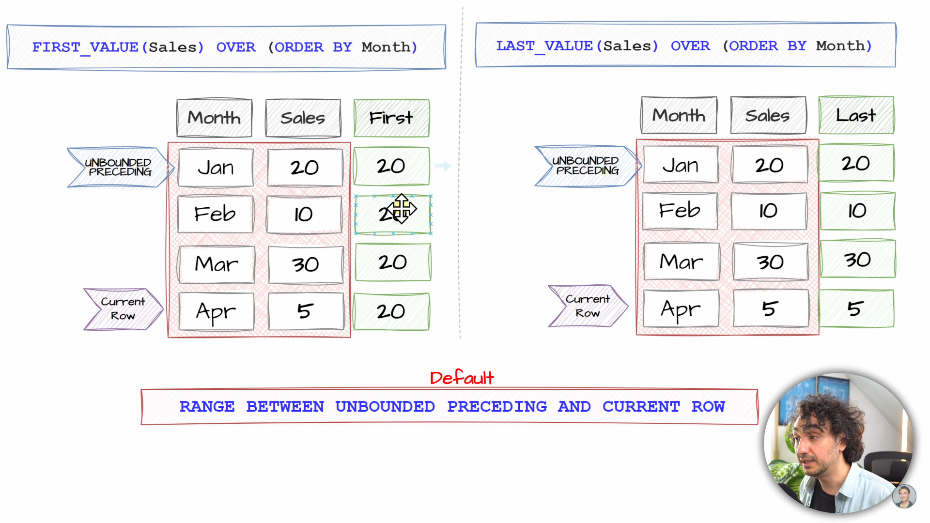
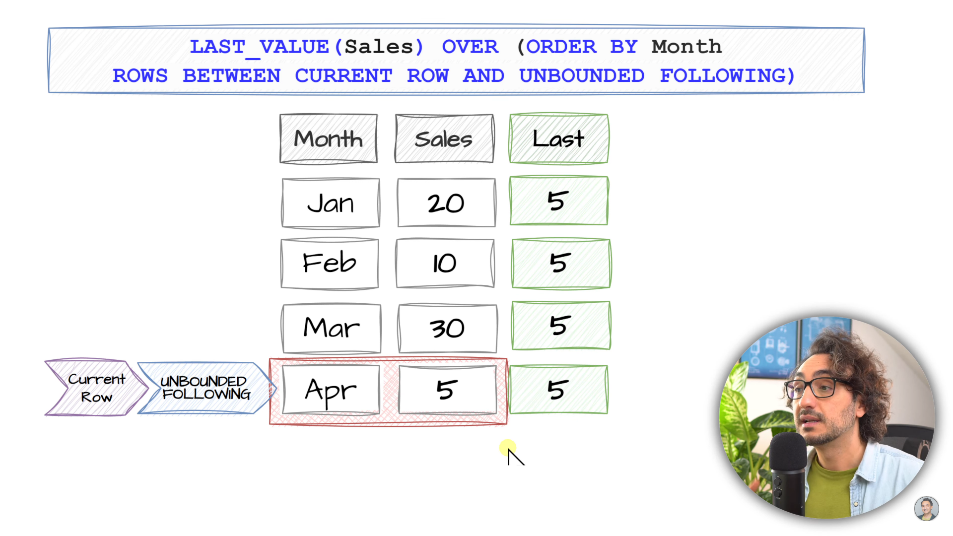
1. **FIRST\_VALUE(expression)**

Default :- RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

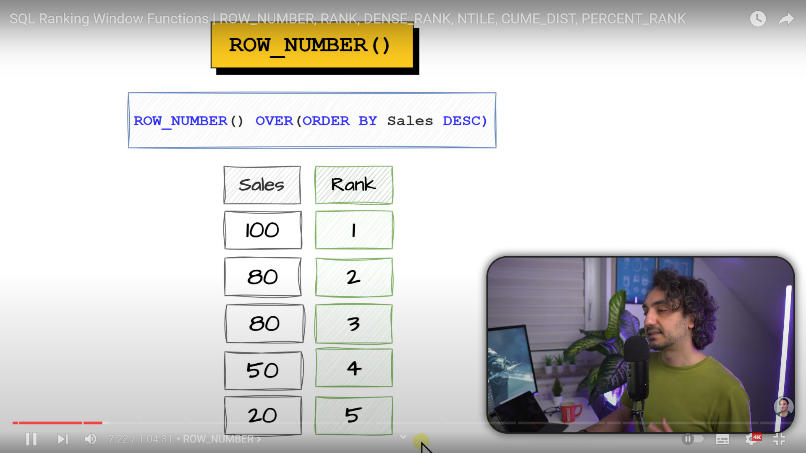
**Example :- FIRST\_VALUE(sales) OVER(ORDER BY month)**

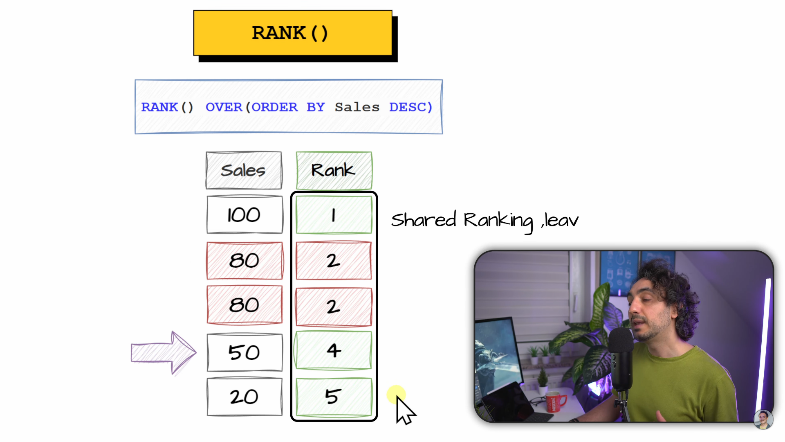
1. **LAST\_VALUE(expression)**

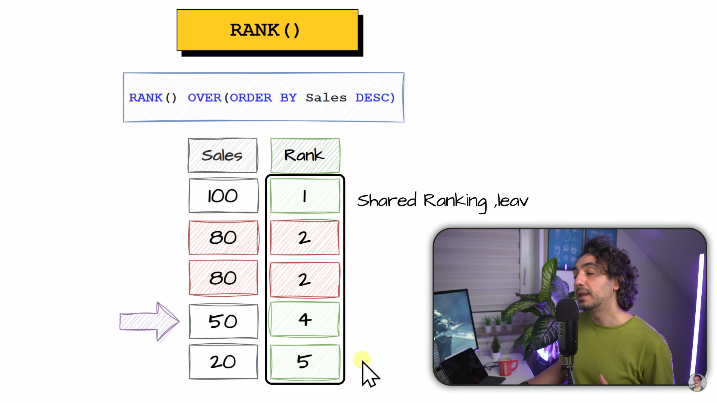
**Example :- LAST\_VALUE(sales) OVER(ORDER BY month ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)**

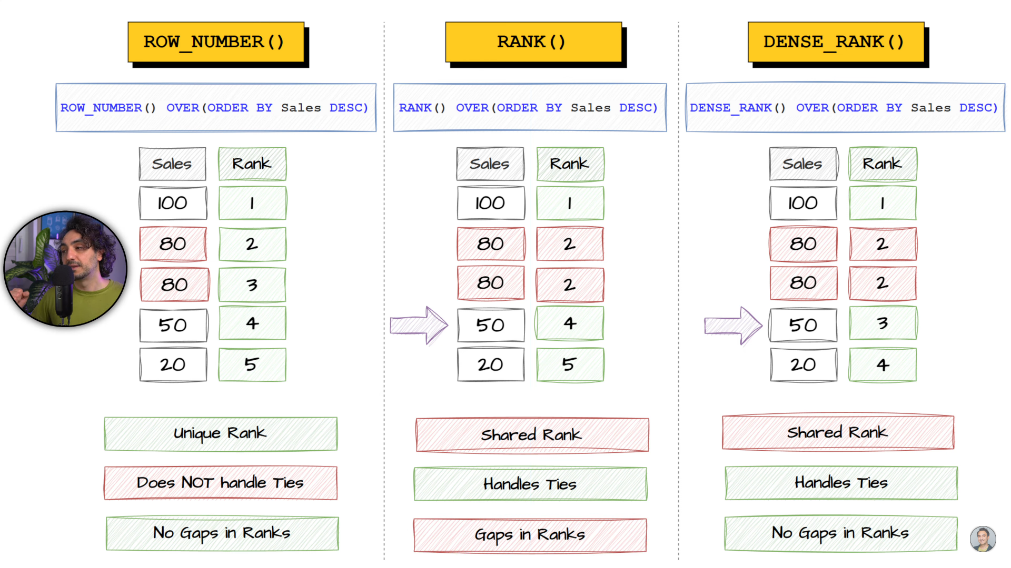
****

1. **ROW\_NUMBER()**
2. **RANK()**
3. **DENSE\_RANK()**

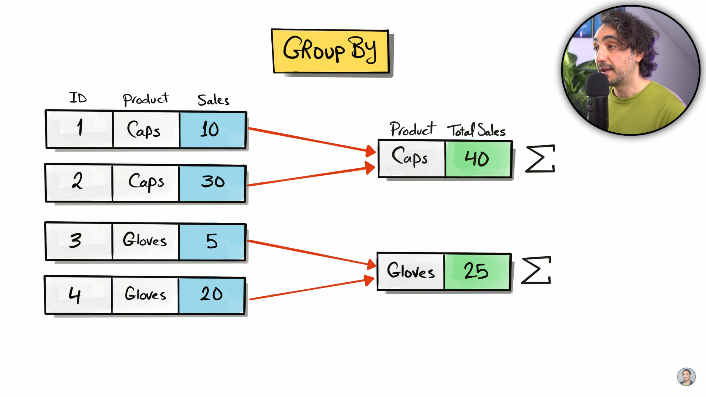
****

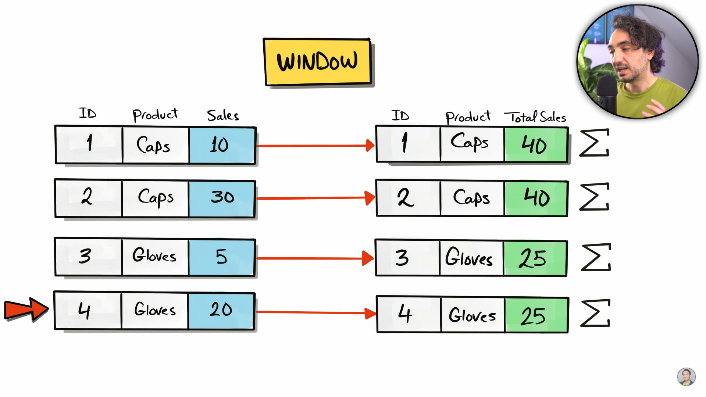
****

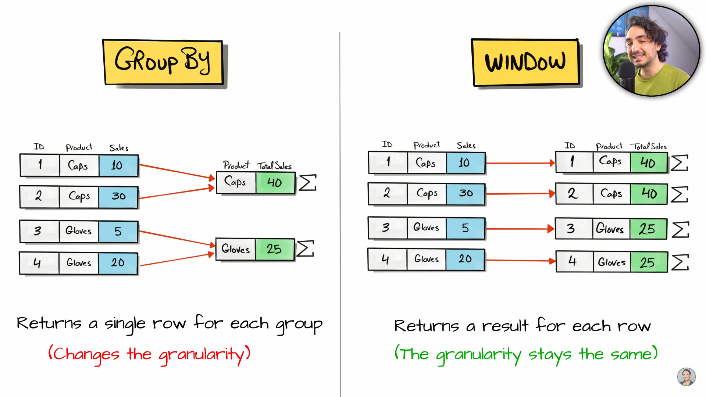
****

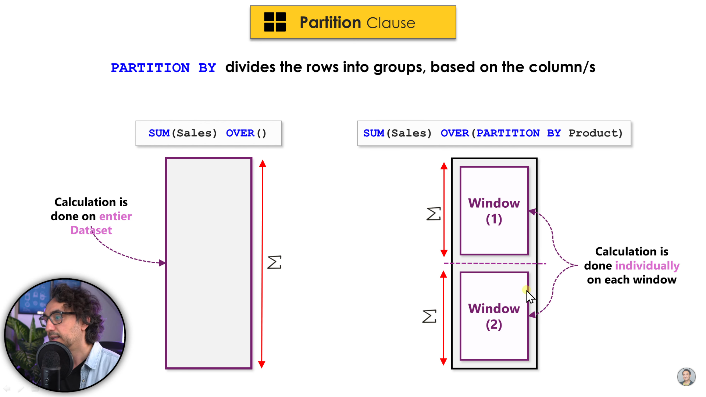
****

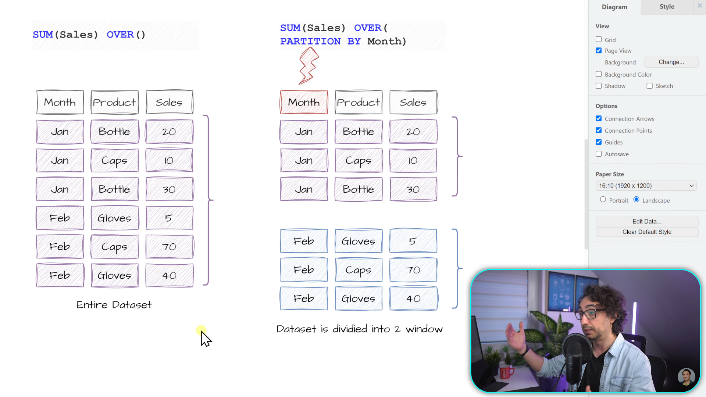
**PARTITION BY :-**

****

****

****

****

****

**Window Frame :-**

1. ROWS BETWEEN CURRENT ROW AND 2 FOLLOWING
2. ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
3. ROWS BETWEEN 1 PRECEDING AND CURRENT ROW
4. ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
5. ROWS BETEWEN 1 PRECEDING AND 1 FOLLOWING
6. ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING

**Types of Subquery :-**

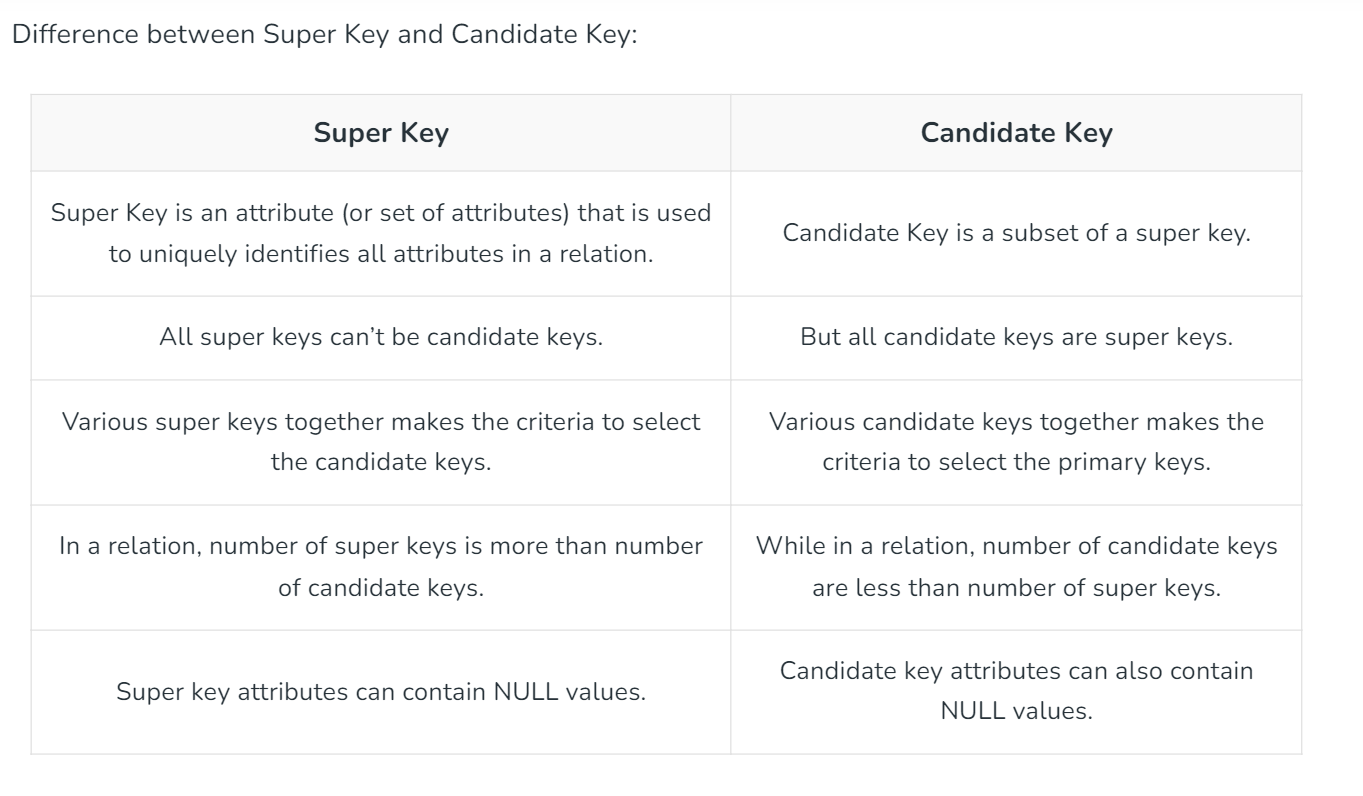
* **The result it returns**
* **Based on working.**

1. **The result it returns or Returned Data :-**

* **Scaler subquery.**
* **It returns single value.**
* **Row subquery.**
* **It returns multiple rows with single column.**
* **Table subquery.**
* **It returns multiple rows with multiple columns.**

1. **Based on working.**

* **Independent subquery.**
* **The inner query is not related to outer query.**
* **Correlated subquery.**
* **The inner query is related to outer query.**

**Difference Between super key and candidate key.**

**ROW\_NUMBER() :- Assign a unique number to each row.**

**RANK() :- Assign a rank to each row. It handles ties. It leaves gaps in ranking.**

**DENSE\_RANK() :- Assign rank to each row. It handles ties. It doesn’t leaves gaps in ranking.**

**NTILE() :- Divides the row into a specific number of approximately equal groups (buckets).**

**CUME\_DIST() :- Cumulative distribution calculates the distribution of data points within a window.**

**PERCENT\_RANK() :- Calculate the relative position of each row.**

**LEAD() :- Access a values from the next row within a window.**

**LAG() :- Access a values from the previous row within a window.**

**FIRST\_VALUE() :- Access a value from the first row within a window.**

**LAST\_VALUE() :- Access a value from the last row within a window.**

1) Primary Key :-

CREATE TABLE employees (

employee\_id INT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

email VARCHAR(100)

);

2) Foreign Key :-

CREATE TABLE departments (

department\_id INT PRIMARY KEY,

department\_name VARCHAR(100)

);

CREATE TABLE employees (

employee\_id INT PRIMARY KEY,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

email VARCHAR(100),

department\_id INT,

FOREIGN KEY (department\_id) REFERENCES departments(department\_id)

);

3) Unique Key :-

CREATE TABLE users (

user\_id INT PRIMARY KEY,

username VARCHAR(50) UNIQUE,

email VARCHAR(100) UNIQUE

);

4) Super key :-

CREATE TABLE products (

product\_id INT,

product\_code VARCHAR(50),

product\_name VARCHAR(100),

PRIMARY KEY (product\_id)

);

-- Here, (product\_id), (product\_code), and (product\_id, product\_code) are all super keys

5) Candidate Key :-

CREATE TABLE students (

student\_id INT PRIMARY KEY,

registration\_number VARCHAR(50) UNIQUE,

email VARCHAR(100) UNIQUE

);

6) Composite Key :-

CREATE TABLE order\_items (

order\_id INT,

product\_id INT,

quantity INT,

PRIMARY KEY (order\_id, product\_id)

);

Union :-

It is used to combine the rows and remove any duplicates .

Union all :-

It is used to combine the rows without removing duplicates.

**CTE – Common Table Expression :-**

A Common Table Expression (CTE) is a powerful SQL feature that allows you to create a temporary, named result set within a query. This result set can then be referenced within the same query, making complex SQL statements easier to read and maintain.