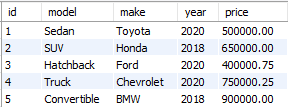
**VAISHALI BOKADIYA**

**DAY 6 ASSESSMENT**

**ADVANCE SQL TOPICS**

**DATABASE:**



**PARTITION BY AND OVER:**

Partitioning in MySQL is done by PARTION BY. It is used to split or partition the rows of a table into separate tables in different locations, but still, it is treated as a single table.

EXAMPLE:

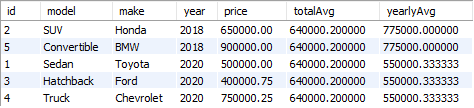
SELECT \*,

AVG(price) OVER () AS totalAvg,

AVG(price) OVER (PARTITION BY year ORDER BY year) as yearlyAvg

FROM CARS;

OUTPUT:



**REGEX:**

Regular expressions allow you to perform complex pattern matching and manipulation of text fields in your SQL queries.

EXAMPLE:

SELECT \* FROM CARS

WHERE make REGEXP 'o[rnt]';

OUTPUT:



SELECT \* FROM CARS

WHERE make REGEXP '^Toy';

OUTPUT:



SELECT \* FROM CARS

WHERE year REGEXP '^201';

OUTPUT:



**MATERIALIZED VIEWS**

Materialized views are the pre-calculated results of a query that is generally stored in a table.

EXAMPLE:

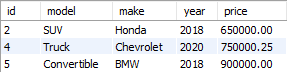
CREATE VIEW expensiveCars AS

SELECT \* FROM CARS

WHERE price>(SELECT AVG(price) FROM CARS);

SELECT \* FROM expensiveCars;

OUTPUT:



**Total aggregation using OVER and PARTITION BY in SQL queries:**

Partitioning in MySQL is done by PARTION BY. It is used to split or partition the rows of a table into separate tables in different locations, but still, it is treated as a single table.

EXAMPLE:

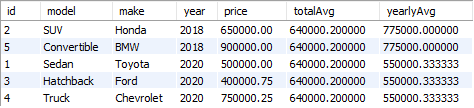
SELECT \*,

AVG(price) OVER () AS totalAvg,

AVG(price) OVER (PARTITION BY year ORDER BY year) as yearlyAvg

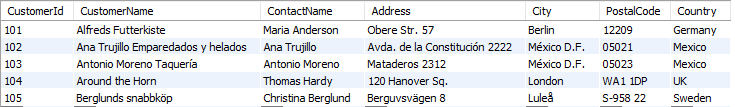
FROM CARS;

OUTPUT:

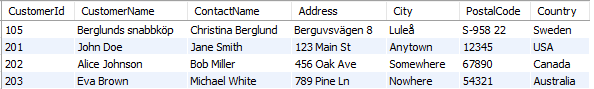


**DATABASE:**

CUSTOMERS1 TABLE:



CUSTOMERS2 TABLE:



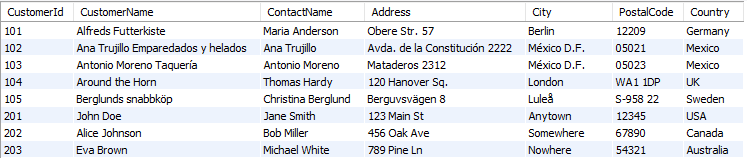
**Difference between UNION, EXCEPT and INTERSECT operators in SQL:**

**UNION:**

EXAMPLE:

UNION in SQL fetches all the rows of both tables but it does not includes duplicate rows.

SELECT \* FROM CUSTOMERS1 UNION SELECT \* FROM CUSTOMERS2;

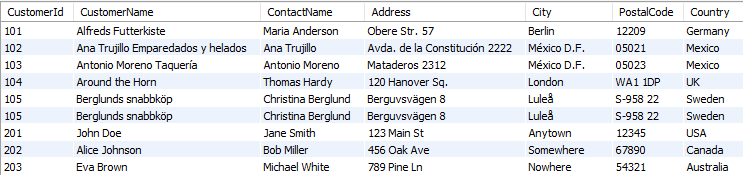


**UNION ALL:**

UNION ALL in SQL works similar to UNION. It fetches all the rows of both tables and also includes duplicate rows.

EXAMPLE:

SELECT \* FROM CUSTOMERS1 UNION ALL SELECT \* FROM CUSTOMERS2;



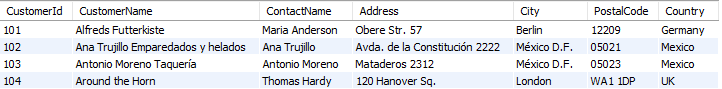
**EXCEPT:**

EXCEPT in SQL fetches all the rows of first table except for the common rows with second table.

EXAMPLE:

SELECT \* FROM CUSTOMERS1 EXCEPT SELECT \* FROM CUSTOMERS2;

OUTPUT:



**INTERSECT**

INTERSECT in SQL fetches only the rows which are common in both the tables.

EXAMPLE:

SELECT \* FROM CUSTOMERS1 INTERSECT SELECT \* FROM CUSTOMERS2;

OUTPUT:



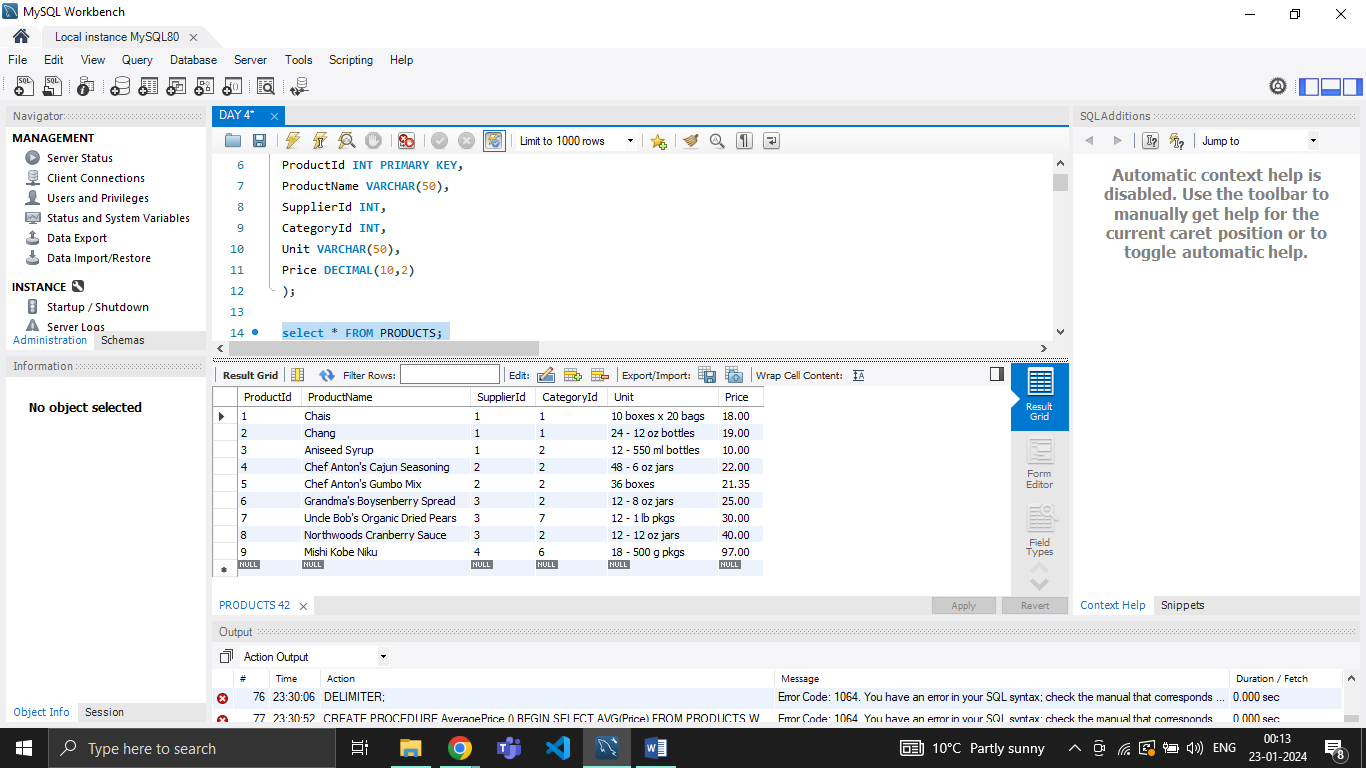
**Order of execution of SQL queries:**

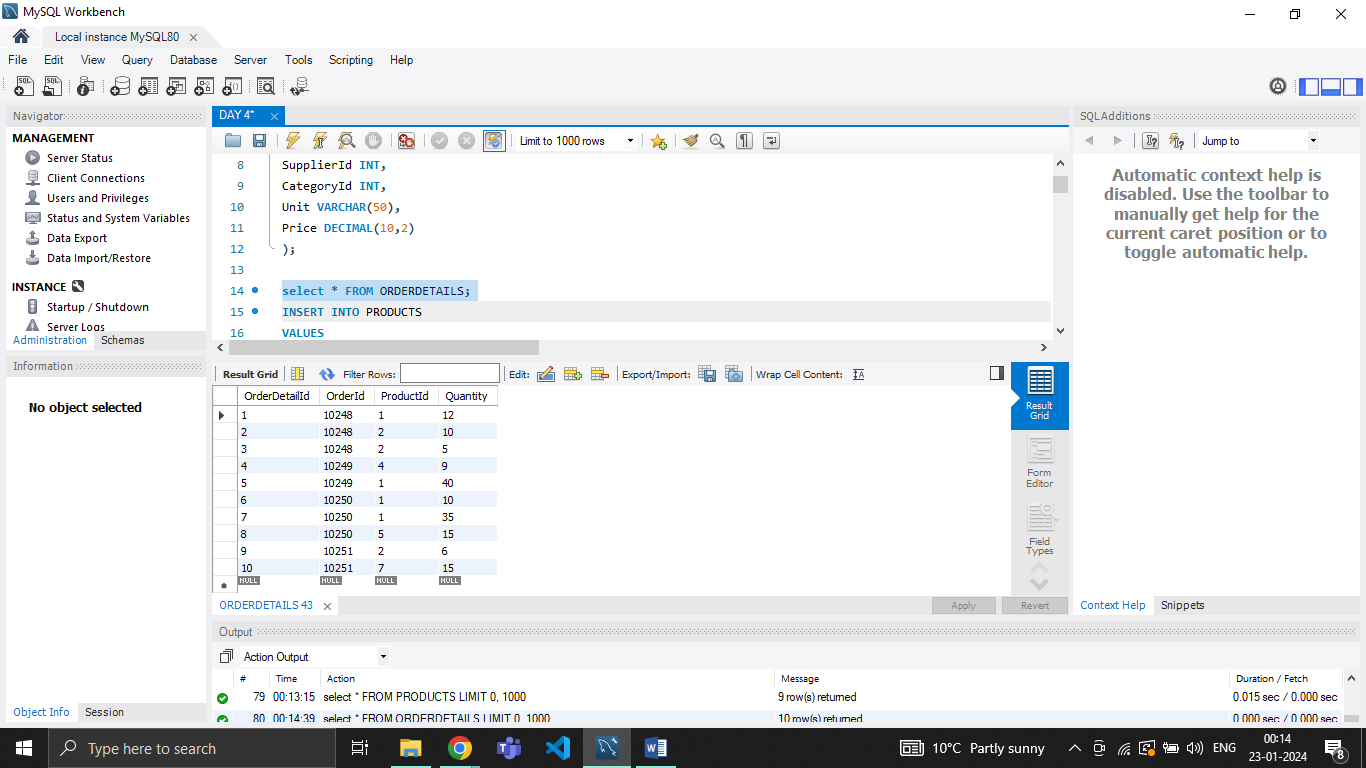
Order of execution defines the order in which the SQL server will execute the commands of the given SQL statement.

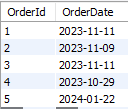
The order in which the SQL commands are executed are:

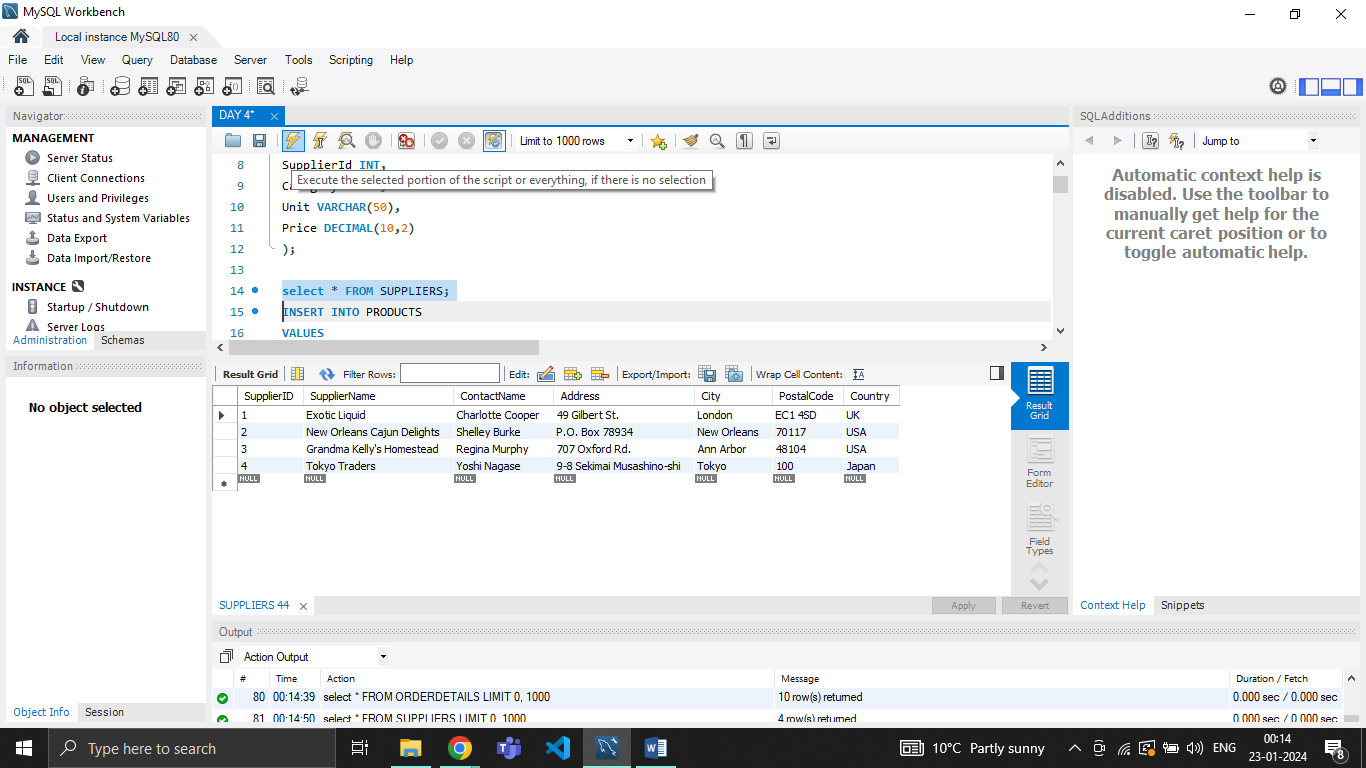
1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY
7. LIMIT/OFFSET.

**DATABASE:**

****

****

****

****

**Total aggregation using SQL queries:**

**SUM():**

SELECT SUM(Quantity) FROM OrderDetails JOIN Products ON Products.ProductId=OrderDetails.ProductId WHERE Price>20;

OUTPUT:



**COUNT():**

SELECT COUNT(OrderDetailId) FROM OrderDetails JOIN Products ON Products.ProductId=OrderDetails.ProductId WHERE Price>20;

OUTPUT:



**AVG():**

SELECT AVG(Quantity) FROM OrderDetails JOIN Products ON Products.ProductId=OrderDetails.ProductId WHERE Price>20;

OUTPUT:



**MIN():**

SELECT MIN(Price) FROM PRODUCTS;

OUTPUT:



**MAX():**

SELECT MAX(Price) FROM PRODUCTS;

OUTPUT:



**Rules and restrictions to group and filter data in SQL queries:**

GROUP BY:

All the columns in the select statement that aren’t aggregated should be specified in a GROUP BY clause in the query.  It is advisable to include all non-aggregated columns from your SELECT clause in your GROUP BY clause. If you don’t, there are cases where the query will return the desired results, there are also instances where a random value from the non-aggregated row will be used as the representative for all the values returned by the query.

FILTER:

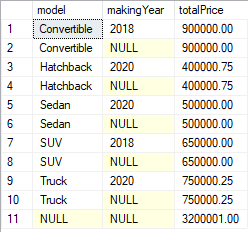
WHERE also limits the values in a query against which an aggregate function is run. FILTER is more flexible than WHERE because you can use more than one FILTER modifier in an aggregate query while you can only use only one WHERE clause.

**How to calculate subtotal in SQL queries:**

**ROLL UP:**

SELECT model, makingYear, SUM(price) AS totalPrice FROM CARS

GROUP BY ROLLUP(model, makingYear);



**GROUPING:**

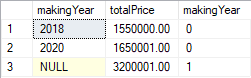
SELECT makingYear,

SUM(price) AS totalPrice ,

GROUPING(make) AS make

FROM CARS

GROUP BY ROLLUP(makingYear);



**CTE:**

WITH CTE AS (

SELECT model, price ,

ROW\_NUMBER() OVER(ORDER BY NEWID())

AS RowNumber FROM cars

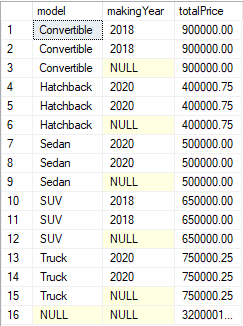
)

SELECT model, makingYear, SUM(price) AS totalPrice

FROM CTE

GROUP BY ROLLUP(model, RowNumber)

GROUPING SETS:



**GROUPING SET:**

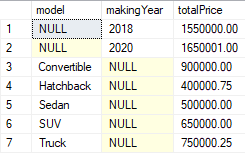
SELECT

-- model,makingYear,

-- SUM(price) AS totalPrice

-- FROM CARS

-- GROUP BY GROUPING SETS(model,makingYear)



**Star schema:**

A star schema consists of a central fact table that references multiple dimension tables. Each dimension table is denormalized ("flattened") to avoid the query overhead that comes with a highly normalized schema, which can require a large number of joins to retrieve the necessary data.

**Snow flacking schema:**

Snowflaking is a form of dimensional modeling in which dimensions are stored in multiple related dimension tables. A snowflake schema is a variation of the star schema that normalizes the dimension tables to increase data integrity, simplify data maintenance and reduce the amount of disk space. In certain situations, it can also improve query performance.

EXAMPLE:

create database snowflake;

use snowflake;

create table salestable(

product\_id int not null primary key,

order\_id int not null,

customer\_id int not null,

employeer\_id int not null,

total int not null,

Quantity int not null,

discount int

);

INSERT INTO salestable (product\_id, order\_id, customer\_id, employeer\_id, total, Quantity, discount)

VALUES

(1, 1001, 101, 201, 500, 2, 10),

(2, 1002, 102, 202, 750, 3, 15),

(3, 1003, 103, 203, 1000, 1, 5);

create table time\_dimension(

order\_id int not null primary key,

order\_date date not null

);

INSERT INTO time\_dimension (order\_id, order\_date)

VALUES

(1001, '2024-01-26'),

(1002, '2024-01-27'),

(1003, '2024-01-28');

create table customer\_dimension(

customer\_id int not null primary key,

city\_id int not null,

customer\_name char(30) not null,

address varchar(50) not null,

city char(25) not null,

zip int not null

);

INSERT INTO customer\_dimension (customer\_id, city\_id, customer\_name, address, city, zip)

VALUES

(101, 1, 'John Doe', '123 Main St', 'Anytown', 12345),

(102, 2, 'Jane Smith', '456 Oak St', 'Othercity', 67890),

(103, 1, 'Bob Johnson', '789 Maple St', 'Anytown', 54321);

create table product\_dimension(

product\_id int not null primary key,

Product\_name varchar(50) not null,

product\_prize decimal not null

);

INSERT INTO product\_dimension (product\_id, Product\_name, product\_prize)

VALUES

(1, 'Widget A', 19.99),

(2, 'Gadget B', 29.99),

(3, 'Thingamajig C', 39.99);

create table emp\_dimension(

employeer\_id int not null primary key,

emp\_name varchar(30) not null,

department varchar(25) not null,

department\_id int not null

);

INSERT INTO emp\_dimension (employeer\_id, emp\_name, department, department\_id)

VALUES

(201, 'John Manager', 'Sales', 1),

(202, 'Jane Developer', 'Engineering', 2),

(203, 'Bob Analyst', 'Finance', 3);

create table city\_dimension(

city\_id int not null primary key,

city\_name char(30) not null,

state char(25),

country char(20)

);

INSERT INTO city\_dimension (city\_id, city\_name, state, country)

VALUES

(1, 'Anytown', 'California', 'USA'),

(2, 'Othercity', 'New York', 'USA'),

(3, 'Someplace', 'Texas', 'USA');

create table Product\_category\_dimension(

product\_id int not null primary key,

name varchar(50) not null,

pro\_description varchar(50) not null,

unit\_prize int not null ,

FOREIGN KEY (product\_id) REFERENCES product\_dimension(product\_id)

);

CREATE TABLE Product\_category\_dimension (

product\_id INT NOT NULL PRIMARY KEY,

name VARCHAR(50) NOT NULL,

pro\_description VARCHAR(50) NOT NULL,

unit\_prize INT NOT NULL,

FOREIGN KEY (product\_id) REFERENCES product\_dimension(product\_id)

);

create table department\_dimension(department\_id int,

department varchar(25) not null,

location varchar(25) not null

);

INSERT INTO department\_dimension (department\_id, department, location)

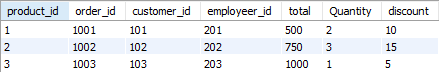
VALUES

(1, 'Sales', 'New York'),

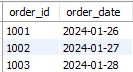
(2, 'Engineering', 'San Francisco'),

(3, 'Finance', 'Chicago');

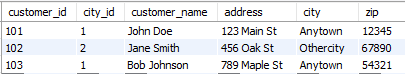
select \* from salestable;



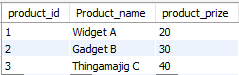
select \* from time\_dimension;



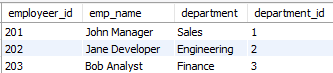
select \* from customer\_dimension;



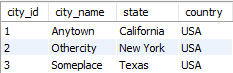
select \* from product\_dimension;



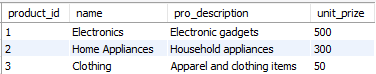
select \* from emp\_dimension;



select \* from city\_dimension;



select \* from Product\_category\_dimension;



select \* from department\_dimension;

