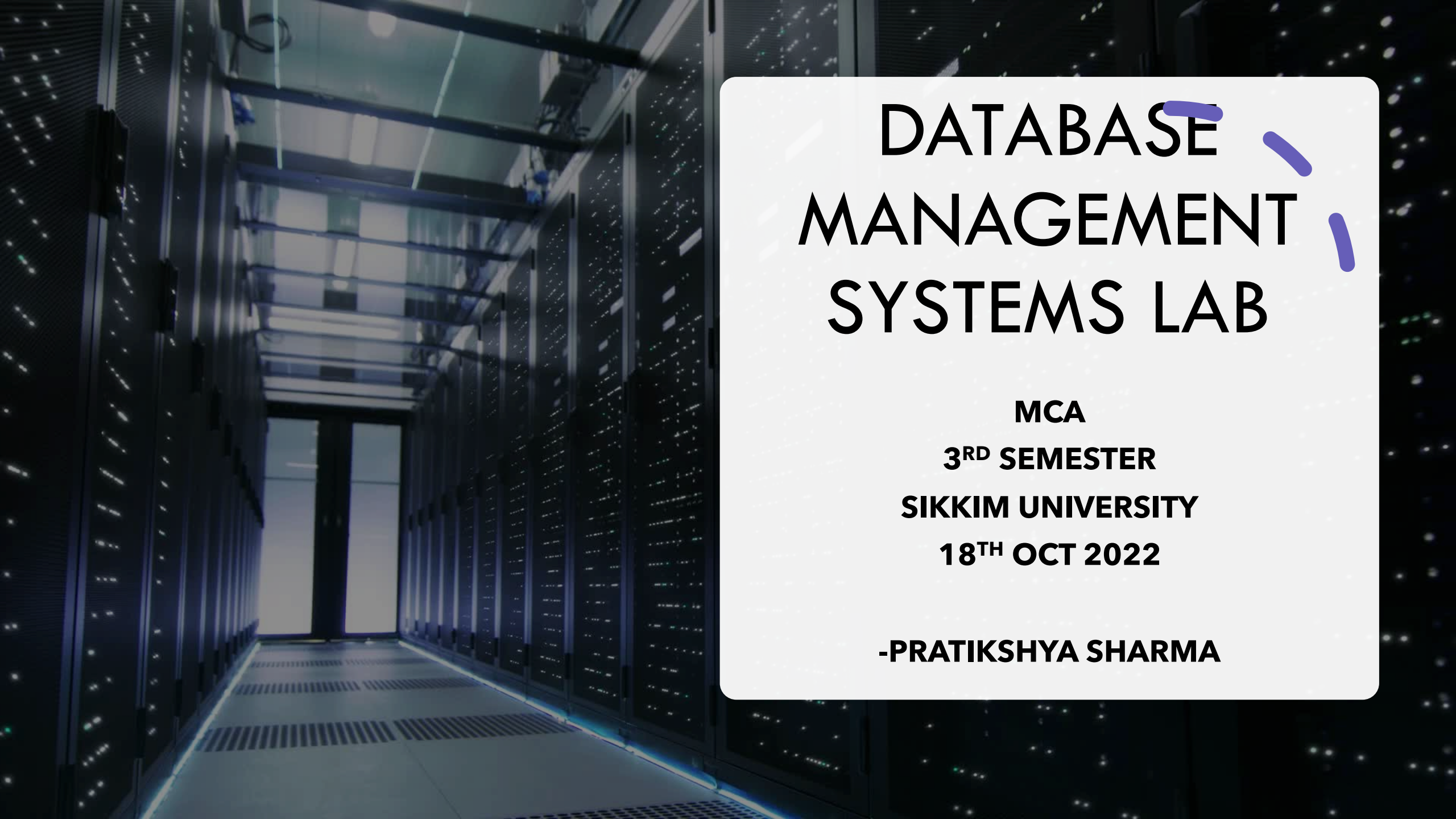

DATABASE MANAGEMENT SYSTEMS LAB

MCA

3RD SEMESTER

SIKKIM UNIVERSITY

18TH OCT 2022

-PRATIKSHYA SHARMA

IN Operator With a SubQuery

- The IN Operator in SQL allows to specifies multiple values in WHERE clause, it can help you to easily test if an expression matches any value in the list of values.
- The use of IN Operator reduces the need for multiple OR conditions in statements like SELECT, INSERT, UPDATE, and DELETE.

Sub Queries:

- The SQL queries where one or more SELECT statements are nested with the WHERE clause of another SELECT statement are called subquery.
- The first statement of such type of query is called **outer query** where as the inside one is called an **inner query**.
- In the execution of such queries, the inner query will be evaluated first, and the outer query receives the value of the inner query.

IN Operator With a SubQuery

- A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.
- A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

IN Operator With a SubQuery

- There are a few rules that subqueries must follow –
- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

Subqueries with the SELECT Statement

- Subqueries are most frequently used with the SELECT statement.

Syntax

SELECT column_name [, column_name] FROM table1 [, table2] WHERE column_name OPERATOR (SELECT column_name [, column_name] FROM table1 [, table2] [WHERE])

Example

Consider the CUSTOMERS table having the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Now, let us check the following subquery with a SELECT statement.

SELECT * FROM CUSTOMERS WHERE ID IN (SELECT ID FROM CUSTOMERS WHERE SALARY > 4500) ;This would produce the following result.

This would produce the following result.

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

Subqueries with the INSERT Statement

- Subqueries also can be used with INSERT statements.
- The INSERT statement uses the data returned from the subquery to insert into another table.
- The selected data in the subquery can be modified with any of the character, date or number functions.

Syntax

INSERT INTO table_name [(column1 [, column2])] SELECT [*|column1 [, column2] FROM table1 [, table2] [WHERE VALUE OPERATOR]

Example

Consider a table CUSTOMERS_BKP with similar structure as CUSTOMERS table. Now to copy the complete CUSTOMERS table into the CUSTOMERS_BKP table, you can use the following syntax.

```
INSERT INTO CUSTOMERS_BKP SELECT * FROM CUSTOMERS WHERE ID IN (SELECT ID FROM CUSTOMERS) ;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Subqueries with the UPDATE Statement

- The subquery can be used in conjunction with the UPDATE statement.
- Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

SYNTAX

```
UPDATE table SET column_name = new_value [ WHERE OPERATOR [ VALUE ] (SELECT COLUMN_NAME FROM TABLE_NAME) [ WHERE) ]
```

Example

Assuming, we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table. The following example updates SALARY by 0.25 times in the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
UPDATE CUSTOMERS SET SALARY = SALARY * 0.25 WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
WHERE AGE >= 27 );
```

This would impact two rows and finally CUSTOMERS table would have the following records.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |   125.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  2125.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Subqueries with the DELETE Statement

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned in previous slides.

Syntax

```
DELETE FROM TABLE_NAME [ WHERE OPERATOR [ VALUE ] (SELECT COLUMN_NAME FROM TABLE_NAME) [ WHERE) ]
```

Example

Assuming, we have a CUSTOMERS_BKP table available which is a backup of the CUSTOMERS table. The following example deletes the records from the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
DELETE FROM CUSTOMERS WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP WHERE AGE >= 27 );
```

This would impact two rows and finally the CUSTOMERS table would have the following records.

```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  2 | Khilan   |  25 | Delhi   |  1500.00 |
|  3 | kaushik  |  23 | Kota    |  2000.00 |
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  6 | Komal    |  22 | MP      |  4500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

EXISTS Operator

- The SQL The EXISTS operator is used to test for the existence of any record in a subquery.
- The EXISTS operator returns TRUE if the subquery returns one or more records.

EXISTS Syntax

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

Demo Database

Below is a selection from the "Products" table in the Northwind sample database:

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

And a selection from the "Suppliers" table:

SupplierID	SupplierName	ContactName	Address	City	PostalCode	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	London	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA
4	Tokyo Traders	Yoshi Nagase	9-8 Sekimai Musashino-shi	Tokyo	100	Japan

SQL EXISTS Examples

The following SQL statement returns TRUE and lists the suppliers with a product price less than 20:

Example

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products
WHERE Products.SupplierID =
Suppliers.supplierID AND Price < 20);
```

Result:

Number of Records: 24

SupplierName
Exotic Liquid
New Orleans Cajun Delights
Tokyo Traders
Mayumi's
Pavlova, Ltd.
Specialty Biscuits, Ltd.
PB Knäckebröd AB
Refrescos Americanas LTDA
Heli Süßwaren GmbH & Co. KG
Plutzer Lebensmittelgroßmärkte AG
Formaggi Fortini s.r.l.
Norske Meierier
Bigfoot Breweries
Svensk Sjöföda AB
Aux joyeux ecclésiastiques
New England Seafood Cannery
Leka Trading
Lyngbysild
Zaanse Snoepfabriek
Karkki Oy
G'day, Mate
Ma Maison
Pasta Buttini s.r.l.
Escargots Nouveaux

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

Example

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

GROUP BY

- The SQL The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".
- The GROUP BY statement 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
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Demo Database

Below is a selection from the "Customers" table in the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

SQL GROUP BY Examples

The following SQL statement lists the number of customers in each country:

Example

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

Result:

Number of Records: 21

COUNT(CustomerID)	Country
3	Argentina
2	Austria
2	Belgium
9	Brazil
3	Canada
2	Denmark
2	Finland
11	France
11	Germany
1	Ireland
3	Italy
5	Mexico
1	Norway
1	Poland
2	Portugal
5	Spain
2	Sweden
2	Switzerland
7	UK
13	USA
4	Venezuela

The following SQL statement lists the number of customers in each country, sorted high to low.

Example

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

Number of Records: 21

COUNT(CustomerID)	Country
13	USA
11	Germany
11	France
9	Brazil
7	UK
5	Spain
5	Mexico
4	Venezuela
3	Italy
3	Canada
3	Argentina
2	Switzerland
2	Sweden
2	Portugal
2	Finland
2	Denmark
2	Belgium
2	Austria
1	Poland
1	Norway
1	Ireland

SQL HAVING Clause

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

Syntax

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

Demo Database

Below is a selection from the "Customers" table in the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

SQL HAVING Examples

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers.

Example

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

Result:

Number of Records: 5

COUNT(CustomerID)	Country
9	Brazil
11	France
11	Germany
7	UK
13	USA

The following SQL statement lists the number of customers in each country, sorted high to low (Only include countries with more than 5 customers).

Example

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

Result:

Number of Records: 5

COUNT(CustomerID)	Country
13	USA
11	Germany
11	France
9	Brazil
7	UK

Demo Database

Below is a selection from the "Orders" table in the Northwind sample database:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2

And a selection from the "Employees" table:

EmployeeID	LastName	FirstName	BirthDate	Photo	Notes
1	Davolio	Nancy	1968-12-08	EmpID1.pic	Education includes a BA....
2	Fuller	Andrew	1952-02-19	EmpID2.pic	Andrew received his BTS....
3	Leverling	Janet	1963-08-30	EmpID3.pic	Janet has a BS degree....

The following SQL statement lists the employees that have registered more than 10 orders.

Example

```
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;
```

Result:

Number of Records: 8

LastName	NumberOfOrders
Buchanan	11
Callahan	27
Davolio	29
Fuller	20
King	14
Leverling	31
Peacock	40
Suyama	18

The following SQL statement lists if the employees "Davolio" or "Fuller" have registered more than 25 orders:

Example

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
WHERE LastName = 'Davolio' OR LastName = 'Fuller'
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

Result:

Number of Records: 1

LastName	NumberOfOrders
Davolio	29

SQL STRING FUNCTIONS

REVERSE() Function

- The REVERSE() function reverses a string and returns the result.

Syntax

- `REVERSE(string)`

Example

- Reverse the text in CustomerName:

```
SELECT REVERSE(CustomerName)
FROM Customers;
```

SQL | Views

- Views in SQL are kind of virtual tables.
- A view also has rows and columns as they are in a real table in the database.
- We can create a view by selecting fields from one or more tables present in the database.
- A View can either have all the rows of a table or specific rows based on certain condition.

Syntax

CREATE VIEW view_name AS SELECT column1, column2..... FROM table_name WHERE condition; **view_name**: Name for the View **table_name**: Name of the table **condition**: Condition to select rows

StudentDetails

S_ID	NAME	ADDRESS
1	Harsh	Kolkata
2	Ashish	Durgapur
3	Pratik	Delhi
4	Dhanraj	Bihar
5	Ram	Rajasthan

StudentMarks

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

In this example we will create a View named DetailsView from the table StudentDetails.

Query:

```
CREATE VIEW DetailsView AS SELECT NAME,
ADDRESS FROM StudentDetails WHERE S_ID < 5;
```

To see the data in the View, we can query the view in the same manner as we query a table.

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

- In this example, we will create a view named StudentNames from the table StudentDetails.

CREATE VIEW StudentNames AS SELECT S_ID, NAME FROM StudentDetails ORDER BY NAME;

If we now query the view as,

SELECT * FROM StudentNames;

Output

S_ID	NAMES
2	Ashish
4	Dhanraj
1	Harsh
3	Pratik
5	Ram

Creating View from multiple tables

- In this example we will create a View named MarksView from two tables StudentDetails and StudentMarks.
- To create a View from multiple tables we can simply include multiple tables in the SELECT statement.

Query:

```
CREATE VIEW MarksView AS SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS FROM StudentDetails, StudentMarks WHERE StudentDetails.NAME = StudentMarks.NAME;
```

To display data of View MarksView:

```
SELECT * FROM MarksView;
```

Output:

NAME	ADDRESS	MARKS
Harsh	Kolkata	90
Pratik	Delhi	80
Dhanraj	Bihar	95
Ram	Rajasthan	85

DELETING VIEWS

- SQL allows us to delete an existing View.
- We can delete or drop a View using the DROP statement.

Syntax:

```
DROP VIEW view_name;
```

view_name: Name of the View which we want to delete.

For example, if we want to delete the View **MarksView**, we can do this as:

```
DROP VIEW MarksView;
```

UPDATING VIEWS

There are certain conditions needed to be satisfied to update a view.

If any one of these conditions is **not** met, then we will not be allowed to update the view.

1. The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.
2. The SELECT statement should not have the DISTINCT keyword.
3. The View should have all NOT NULL values.
4. The view should not be created using nested queries or complex queries.
5. The view should be created from a single table. If the view is created using multiple tables then we will not be allowed to update the view.

•

UPDATING VIEWS

We can use the **CREATE OR REPLACE VIEW** statement to add or remove fields from a view.

Syntax:

CREATE OR REPLACE VIEW view_name AS SELECT column1,coulmn2,.. FROM table_name WHERE condition;

Example

- If we want to update the view **MarksView** and add the field AGE to this View from **StudentMarks** Table, we can do this as:

CREATE OR REPLACE VIEW MarksView AS SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS, StudentMarks.AGE FROM StudentDetails, StudentMarks WHERE StudentDetails.NAME = StudentMarks.NAME;

If we fetch all the data from MarksView now as:

- SELECT * FROM MarksView;

Output:

NAME	ADDRESS	MARKS	AGE
Harsh	Kolkata	90	19
Pratik	Delhi	80	19
Dhanraj	Bihar	95	21
Ram	Rajasthan	85	18

Inserting a row in a view:

- We can insert a row in a View in a same way as we do in a table.
- We can use the INSERT INTO statement of SQL to insert a row in a View.

Syntax

INSERT INTO view_name(column1, column2 , column3,..)
VALUES(value1, value2, value3..); **view_name**: Name of the View

Example:

- In the below example we will insert a new row in the View DetailsView which we have created above in the example of “creating views from a single table”.

```
INSERT INTO DetailsView(NAME, ADDRESS) VALUES("Suresh","Gurgaon");
```

If we fetch all the data from DetailsView now as,

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar
Suresh	Gurgaon

Deleting a row from a View

- Deleting rows from a view is also as simple as deleting rows from a table.
- We can use the DELETE statement of SQL to delete rows from a view.
- Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.

Syntax:

DELETE FROM view_name WHERE condition;

view_name:Name of view from where we want to delete rows

condition: Condition to select rows

Example:

- In this example we will delete the last row from the view DetailsView which we just added in the above example of inserting rows.

```
DELETE FROM DetailsView WHERE NAME="Suresh";
```

If we fetch all the data from DetailsView now as,

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

VIEWS WITH CHECKPOINT

- The WITH CHECK OPTION clause in SQL is a very useful clause for views.
- It is applicable to a updatable view.
- If the view is not updatable, then there is no meaning of including this clause in the CREATE VIEW statement.
- The WITH CHECK OPTION clause is used to prevent the insertion of rows in the view where the condition in the WHERE clause in CREATE VIEW statement is not satisfied.
- If we have used the WITH CHECK OPTION clause in the CREATE VIEW statement, and if the UPDATE or INSERT clause does not satisfy the conditions then they will return an error.

Example:

In the below example we are creating a View SampleView from StudentDetails Table with WITH CHECK OPTION clause.

```
CREATE VIEW SampleView AS SELECT S_ID, NAME FROM StudentDetails
WHERE NAME IS NOT NULL WITH CHECK OPTION;
```

- In this View if we now try to insert a new row with null value in the NAME column then it will give an error because the view is created with the condition for NAME column as NOT NULL.
- For example,though the View is updatable but then also the below query for this View is not valid:

```
INSERT INTO SampleView(S_ID) VALUES(6);
```

NOTE: The default value of NAME column is *null*.