Department of MCA

DBMS Complex SQL Queries

Course Name : 23CAT603 - DATA BASE MANAGEMENT SYSTEM

Class : I Year / I Semester

Unit II – Complex SQL Queries

Agenda

- SQL GROUP BY Statement
- SQL HAVING Clause
- SQL EXISTS Operator
- SQL ANY and ALL Operators
- SQL SELECT INTO Statement
- SQL INSERT INTO SELECT Statement
- SQL CASE Expression
- SQL NULL Functions
- SQL Stored Procedures for SQL Server
- SQL Comments
- SQL Operators

SQL GROUP BY Statement

The SQL GROUP BY Statement

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.

GROUP BY Syntax

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

SQL GROUP BY Statement

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
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
8	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid	28023	Spain
9	Bon app'	Laurence Lebihans	12, rue des Bouchers	Marseille	13008	France
10	Bottom-Dollar Marketse	Elizabeth Lincoln	23 Tsawassen Blvd.	Tsawassen	T2F 8M4	Canada

SQL GROUP BY Statement

SQL GROUP BY Examples

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

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

Expr1000	Country
1	Canada
2	France
3	Germany
2	Mexico
1	Spain
1	Sweden
1	UK

SQL GROUP BY Statement

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;
```

Expr1000	Country
3	Germany
2	France
2	Mexico
1	Canada
1	Spain
1	Sweden
1	UK

SQL GROUP BY Statement

GROUP BY With JOIN Example

The following SQL statement lists the number of orders sent by each shipper:

Example

```
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

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

SQL GROUP BY Statement

And a selection from the "Shippers" table:

ShipperID	ShipperName
1	Speedy Express
2	United Package
3	Federal Shipping

GROUP BY With JOIN Example

The following SQL statement lists the number of orders sent by each shipper:

Example

```
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

ShipperName	NumberOfOrders
Federal Shipping	68
Speedy Express	54
United Package	74

SQL HAVING Clause

The SQL HAVING Clause

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

HAVING Syntax

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

SQL HAVING Examples

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

SQL HAVING Examples

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

Expr1000	**Country**
9	Brazil
11	France
11	Germany
7	UK
13	USA

SQL HAVING Examples

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;
```

Expr1000	Country
13	USA
11	Germany
11	France
9	Brazil
7	UK

The SQL EXISTS Operator

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);
```

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

The SQL EXISTS Operator

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

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

SupplierName
Exotic Liquid
New Orleans Cajun Delights
Tokyo Traders
Mayumi's
Pavlova, Ltd.

The SQL EXISTS Operator

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);
```

SupplierName
New Orleans Cajun Delights

SQL ANY and ALL Operators

<u>The SQL ANY Operator</u>

The ANY operator:

- returns a boolean value as a result
- returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

ANY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

Note: The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

<u>The SQL ALL Operator</u>

The ALL operator:

- returns a boolean value as a result
- returns TRUE if ALL of the subquery values meet the condition
- is used with SELECT, WHERE and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

ALL Syntax With SELECT

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

SQL ANY and ALL Operators

ALL Syntax With WHERE or HAVING

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

Note: The *operator* must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

SQL ANY Examples

The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity equal to 10 (this will return TRUE because the Quantity column has some values of 10):

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

ProductName
Chais
Chang
Chef Anton's Cajun Seasoning
Uncle Bob's Organic Dried Pears

SQL ANY and ALL Operators

The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity larger than 99 (this will return TRUE because the Quantity column has some values larger than 99):

Example

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity > 99);
```

ProductName
Steeleye Stout
Pâté chinois

SQL ANY and ALL Operators

SQL ALL Examples

The following SQL statement lists ALL the product names:

Example

```
SELECT ALL ProductName
FROM Products
WHERE TRUE;
```

ProductName
Chais
Chang
Aniseed Syrup
Chef Anton's Cajun Seasoning

The following SQL statement lists the ProductName if ALL the recor able has Quantity equal to 10. This will of course return FALSE because the Quantity column has many different values (not only the value of 10):

Example

```
SELECT ProductName
FROM Products
WHERE ProductID = ALL
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

Number of Records: 0

ProductName

SQL SELECT INTO Statement

SQL SELECT INTO Statement

The SQL SELECT INTO Statement

The SELECT INTO statement copies data from one table into a new table.

SELECT INTO Syntax

Copy all columns into a new table:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

Copy only some columns into a new table:

```
SELECT column1, column2, column3, ...
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

The new table will be created with the column-names and types as defined in the old table. You can create new column names using the AS clause.

SQL SELECT INTO Examples

The following SQL statement creates a backup copy of Customers:

```
SELECT * INTO CustomersBackup2017
FROM Customers;
```

The following SQL statement uses the IN clause to copy the table into a new table in another database:

```
SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'
FROM Customers;
```

The following SQL statement copies only a few columns into a new table:

```
SELECT CustomerName, ContactName INTO CustomersBackup2017
FROM Customers;
```

The following SQL statement copies only the German customers into a new table:

```
SELECT * INTO CustomersGermany
FROM Customers
WHERE Country = 'Germany';
```

The following SQL statement copies data from more than one table into a new table:

```
SELECT Customers.CustomerName, Orders.OrderID
INTO CustomersOrderBackup2017
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

Tip: SELECT INTO can also be used to create a new, empty table using the schema of another. Just add a WHERE clause that causes the query to return no data:

```
SELECT * INTO newtable
FROM oldtable
WHERE 1 = 0;
```

SQL INSERT INTO SELECT Statement

SQL INSERT INTO SELECT Statement

The SQL INSERT INTO SELECT Statement

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.

The INSERT INTO SELECT statement requires that the data types in source and target tables match.

Note: The existing records in the target table are unaffected.

INSERT INTO SELECT Syntax

Copy all columns from one table to another table:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

Copy only some columns from one table into another table:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

SQL INSERT INTO SELECT Statement

SQL INSERT INTO SELECT Examples

Copy "Suppliers" into "Customers" (the columns that are not filled with data, will contain NULL):

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers;
```

Example

Copy "Suppliers" into "Customers" (fill all columns):

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
SELECT SupplierName, ContactName, Address, City, PostalCode, Country FROM Suppliers;
```

Example

Copy only the German suppliers into "Customers":

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers
WHERE Country='Germany';
```

SQL CASE Expression

The SQL CASE Expression

The CASE expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. 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.

CASE Syntax

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

OrderID	Quantity	QuantityText
10248	12	The quantity is under 30
10248	10	The quantity is under 30
10248	5	The quantity is under 30

SQL CASE Examples

The following SQL goes through conditions and returns a value when the first condition is met:

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN 'The quantity is greater than 30'
    WHEN Quantity = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

SQL CASE Expression

The following SQL will order the customers by City. However, if City is NULL, then order by Country:

Example

```
SELECT CustomerName, City, Country
FROM Customers
ORDER BY
(CASE
   WHEN City IS NULL THEN Country
   ELSE City
END);
```

CustomerName	City	Country
Drachenblut Delikatessend	Aachen	Germany
Rattlesnake Canyon Grocery	Albuquerque	USA
Old World Delicatessen	Anchorage	USA
Vaffeljernet	Århus	Denmark

SQL Stored Procedures for SQL Server

What is a Stored Procedure?

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

Stored Procedure Syntax

```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;
```

Execute a Stored Procedure

```
EXEC procedure_name;
```

SQL Stored Procedures for SQL Server

Stored Procedure Example

The following SQL statement creates a stored procedure named "SelectAllCustomers" that selects all records from the "Customers" table:

```
CREATE PROCEDURE SelectAllCustomers
AS
SELECT * FROM Customers
GO;
```

Execute the stored procedure above as follows:

Example

```
EXEC SelectAllCustomers;
```

SQL Stored Procedures for SQL Server

Stored Procedure With One Parameter

The following SQL statement creates a stored procedure that selects Customers from a particular City from the "Customers" table:

Example

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)
AS
SELECT * FROM Customers WHERE City = @City
GO;
```

Execute the stored procedure above as follows:

Example

```
EXEC SelectAllCustomers @City = 'London';
```

Stored Procedure With Multiple Parameters

Setting up multiple parameters is very easy. Just list each parameter and the data type separated by a comma as shown below.

The following SQL statement creates a stored procedure that selects Customers from a particular City with a particular PostalCode from the "Customers" table:

Example

```sql
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode
nvarchar(10)
AS
SELECT * FROM Customers WHERE City = @City AND PostalCode = @PostalCode
GO;
```

Execute the stored procedure above as follows:

Example

```sql
EXEC SelectAllCustomers @City = 'London', @PostalCode = 'WA1 1DP';
```

SQL Comments

SQL Comments

Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.

Note: Comments are not supported in Microsoft Access databases!

Single Line Comments

Single line comments start with --.

Any text between -- and the end of the line will be ignored (will not be executed).

The following example uses a single-line comment as an explanation:

```
-- Select all:
SELECT * FROM Customers;
```

The following example uses a single-line comment to ignore the end of a line:

Example

```
SELECT * FROM Customers -- WHERE City='Berlin';
```

The following example uses a single-line comment to ignore a statement:

Example

```
-- SELECT * FROM Customers;
SELECT * FROM Products;
```

References

1. https://www.w3schools.com/sql/sql_insert_into_select.asp
2. https://www.geeksforgeeks.org/having-clause-in-ms-sql-server/?ref=ml_lbp