Lab Manual for Database Management System

Lab No. 2

Ordering, Grouping & Aggregate Functions in SQL

LAB 2: ORDERING, GROUPING & AGGREGATE FUNCTIONS IN SQL

1. INTRODUCTION:

Till now, we have been discussing filtering in the data: that is, defining the conditions that determine which rows will be included in the final set of rows to be fetched and returned from the database. Once we have determined which columns and rows will be included in the results of our **SELECT** query, we may want to control the order in which the rows appear—sorting the data.

2. OBJECTIVE:

After completing this lab the student should be able to:

1. *Understand sort data in SQL Server clause*
2. *Understand the purpose of using Built In Functions*
3. *Arrange data in groups*
4. *Understand the difference between having and where.*

3. THEORY

i. The ORDER BY Clause

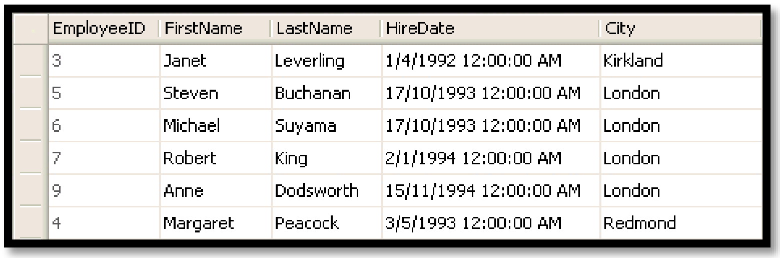
To sort the data rows, we include the **ORDER BY** clause. The **ORDER BY** clause includes one or more column names that specify the sort order.

Syntax →

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Example

If we return to one of our first **SELECT** statements, we can sort its results by City with the following statement: ***SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees ORDER BY City.*** By default, the sort order for a column is ascending (from lowest value to highest value), as shown below for the previous query:



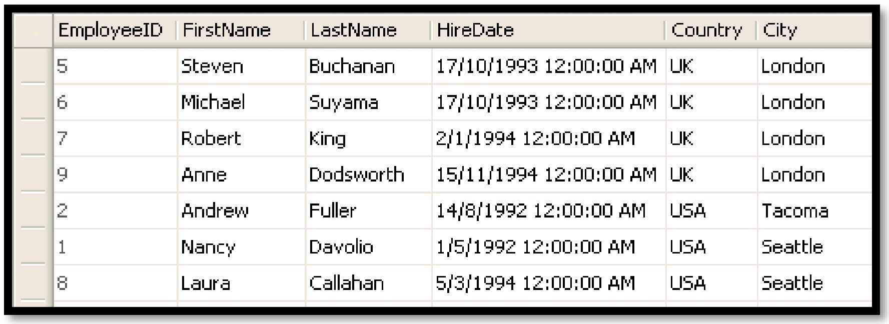
EmployeeID	FirstName	LastName	HireDate	City
3	Janet	Leverling	1/4/1992 12:00:00 AM	Kirkland
5	Steven	Buchanan	17/10/1993 12:00:00 AM	London
6	Michael	Suyama	17/10/1993 12:00:00 AM	London
7	Robert	King	2/1/1994 12:00:00 AM	London
9	Anne	Dodsworth	15/11/1994 12:00:00 AM	London
4	Margaret	Peacock	3/5/1993 12:00:00 AM	Redmond

If we want the sort order for a column to be descending, we can include the **DESC** keyword after the column name.

The **ORDER BY** clause is not limited to a single column. You can include a comma- delimited list of columns to sort by—the rows will all be sorted by the first column specified and then by the next column specified. If we add the Country field to the **SELECT** clause and want to sort by Country and City, we would write: ***SELECT EmployeeID, FirstName, LastName, HireDate, Country, City FROM Employees ORDER BY Country, City DESC***

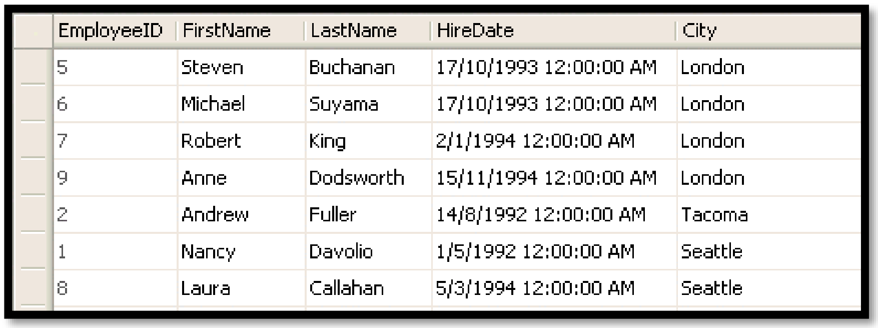
Note that to make it interesting, we have specified the sort order for the City column to be descending (from highest to lowest value). The sort order for the Country column is still ascending. We could be more explicit about this by writing: ***SELECT EmployeeID, FirstName, LastName, HireDate, Country, City FROM Employees ORDER BY Country ASC, City DESC***

But this is not necessary and is rarely done. The results returned by this query are:



EmployeeID	FirstName	LastName	HireDate	Country	City
5	Steven	Buchanan	17/10/1993 12:00:00 AM	UK	London
6	Michael	Suyama	17/10/1993 12:00:00 AM	UK	London
7	Robert	King	2/1/1994 12:00:00 AM	UK	London
9	Anne	Dodsworth	15/11/1994 12:00:00 AM	UK	London
2	Andrew	Fuller	14/8/1992 12:00:00 AM	USA	Tacoma
1	Nancy	Davolio	1/5/1992 12:00:00 AM	USA	Seattle
8	Laura	Callahan	5/3/1994 12:00:00 AM	USA	Seattle

It is important to note that a column does not need to be included in the list of selected (returned) columns in order to be used in the **ORDER BY** clause. If we don't need to see/use the Country values, but are only interested in them as the primary sorting field we could write the query as: ***SELECT EmployeeID, FirstName, LastName, HireDate, City FROM Employees ORDER BY Country ASC, City DESC*** with the results being sorted in the same order as before:



EmployeeID	FirstName	LastName	HireDate	City
5	Steven	Buchanan	17/10/1993 12:00:00 AM	London
6	Michael	Suyama	17/10/1993 12:00:00 AM	London
7	Robert	King	2/1/1994 12:00:00 AM	London
9	Anne	Dodsworth	15/11/1994 12:00:00 AM	London
2	Andrew	Fuller	14/8/1992 12:00:00 AM	Tacoma
1	Nancy	Davolio	1/5/1992 12:00:00 AM	Seattle
8	Laura	Callahan	5/3/1994 12:00:00 AM	Seattle

ii. GROUPING (Transact-SQL)

GROUPING Indicates whether a specified column expression in a GROUP BY list is aggregated or not. GROUPING returns 1 for aggregated or 0 for not aggregated in the result set. GROUPING can be used only in the **SELECT <select>** list, **HAVING**, and **ORDER BY** clauses when GROUP BY is specified.

Syntax →
```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
```

Example

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

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

iii. Aggregate Functions

Aggregate functions perform a calculation on a set of values and return a single value. Except for COUNT, aggregate functions ignore null values. Aggregate functions are frequently used with the GROUP BY clause of the SELECT statement.

All aggregate functions are deterministic. This means aggregate functions return the same value any time that they are called by using a specific set of input values Transact-SQL provides the following aggregate functions:

AVG	MIN	SUM	STDEVP
CHECKSUM_AGG	OVER Clause	VAR	MAX
COUNT	ROWCOUNT_BIG	VARP	GROUPING
COUNT_BIG	STDEV	GROUPING_ID	

Some of the **functions** are explained in detail:

1) ***AVG (Transact-SQL)*** returns the average of the values in a group. Null values are ignored.

Syntax →
```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

Example

The following example calculates the average price of products.

SELECT AVG (Price) FROM Products;

2) ***MIN (Transact-SQL)*** returns the minimum value in the expression.

Syntax → ***SELECT MIN(column_name)***
FROM table_name
WHERE condition;

Example

The following example returns the lowest (minimum) price of products.

SELECT MIN(Price) FROM Products;

3) ***MAX (Transact-SQL)*** returns the maximum value in the expression.

Syntax → ***SELECT MAX(column_name)***
FROM table_name
WHERE condition;

MAX can be used with numeric, character, and datetime columns, but not with bit columns. Aggregate functions and subqueries are not permitted.

Example

The following example returns the lowest (maximum) price of products.

SELECT MAX(Price) FROM Products;

4) ***COUNT (Transact-SQL)*** returns the number of items in a group. COUNT works like the COUNT_BIG function. The only difference between the two functions is their return values. COUNT always returns an int data type value. COUNT_BIG always returns a bigint data type value.

Syntax → ***expression SELECT COUNT(column_name)***
FROM table_name
WHERE condition;

Example

The following example counts the total number of Products IDs present in Product tabel.

```
SELECT COUNT(ProductID) FROM Products;
```

5) ***SUM (Transact-SQL)*** returns the sum of all the values, or only the DISTINCT values, in the expression. SUM can be used with numeric columns only. Null values are ignored.

Syntax →

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

Example

The following example returns the summation of quantity of all orders in tabel of order details.

```
SELECT SUM(Quantity) FROM OrderDetails;
```

iv. 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);
```

Example

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

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

4. ACTIVITY TIME BOXING

Activity Name	Activity Time	Total Time
Instruments Allocation + Setting up Lab	10 mints	10 mints
Walk through Theory & Tasks (Lecture)	60 mints	60 mints
Implementation & Practice time	90 mints	80 mints
Evaluation Time	20 mints	20 mints
	Total Duration	180 nts

5. EXERCISE:

PERFORMANCE TASKS:

1. ***All Example Tasks.***
2. *Write a query to order employee first name in Descending Order.*
3. *Display the highest, lowest, sum and average UnitPrice of each Category. Label column as CategoryId, Maximum, Minimum, Sum and Average, respectively. (Table: Products)*
4. *List only those cities in which more than or equals to 2 employees are living.*
5. *From the [Order Details] table, select the Product's id, maximum price and minimum price for each specific product in the table, sort the list by product id in ascending order.*
6. *From customers table, Count all customers in each region whose contactTitle contains manager and region is not null. (Table: Customers)*

LAB FILE TASKS:

1. *Retrieve the product names and their corresponding quantities in stock for products with UnitsInStock below 50. Sort the list by UnitsInStock in ascending order. (Table: Products)*
2. *List the employee names and their hire dates for employees hired after the year 1992. Order the results by hire date in descending order. (Table: Employees)*
3. *Display the highest, lowest, sum and average UnitPrice of each Category, where highest UnitPrice lies in the range of 50$ to 100$. Label column as CategoryId, Maximum, Minimum, Sum and Average, respectively. (Table: Products)*
4. *From customers table, Count all customers in each region where region is not null. (Table: Customers).*
5. *Write a query to display the number of ContactName with same ContactTitle. Sort contact title in descending order. (Table: Customers)*
6. *Write a query that count all orders against each product id. No of orders should be greater than 50. (Table: [Order Details]).*

7. *How many people are in each unique city in the employee table that have more than one person in the city? Select the city and display the number of how many people are in each if it's greater than 1.(Table: Employees)*
8. *Find the product name, maximum price and minimum price of each product having maximum price greater than 20.00 $. Order by maximum price. (Tabel: Products)*
9. *Write a query to list no of customers with same ContactTitle if No of customers is greater than 5. However their ContactTitle does not contain Manager. Order by contact title in Descending order(Table: Customers)*
10. *Retrieve the count of products in each category where the unit price is less than 30 dollars. Label the columns as CategoryID and ProductCount. (Tables: Products)*