Engr. Shahid Ali Bhutta

UET TAXILA, DEPARTMENT OF COMPUTER ENGINEERING

DBMS LAB # 3

SQL Aggregate & Scalar Functions

**SQL Functions**

(Aggregate & Scalar functions)

# Lab Objective:

The aim of this lab is to understand the different functions used in the SQL.

# SQL Aggregate Functions

SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

* AVG() - Returns the average value
* COUNT() - Returns the number of rows
* MAX() - Returns the largest value
* MIN() - Returns the smallest value
* SUM() - Returns the sum

# SQL Scalar functions

SQL scalar functions return a single value, based on the input value.

Useful scalar functions:

* UCASE() - Converts a field to upper case
* LCASE() - Converts a field to lower case
* LEN() - Returns the length of a text field
* GETDATE() - Returns the current system date and time

**Tip:** The aggregate functions and the scalar functions will be explained in details in the next chapters.

# 1- The AVG() Function

The AVG() function returns the average value of a numeric column.

## SQL AVG() Syntax

SELECT AVG(column\_name) FROM table\_name

## Example

We have the following "Orders" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/11/12 | 1000 | Hansen |
| 2 | 2008/10/23 | 1600 | Nilsen |
| 3 | 2008/09/02 | 700 | Hansen |
| 4 | 2008/09/03 | 300 | Hansen |
| 5 | 2008/08/30 | 2000 | Jensen |
| 6 | 2008/10/04 | 100 | Nilsen |

Now we want to find the average value of the "OrderPrice" fields.

We use the following SQL statement:

SELECT AVG(OrderPrice) AS OrderAverage FROM Orders

The result-set will look like this:

|  |
| --- |
| **OrderAverage** |
| 950 |

Now we want to find the customers that have an OrderPrice value higher than the average OrderPrice value.

We use the following SQL statement:

SELECT Customer FROM Orders

WHERE OrderPrice>(SELECT AVG(OrderPrice) FROM Orders)

The result-set will look like this:

|  |
| --- |
| **Customer** |
| Hansen |
| Nilsen |
| Jensen |

# 2- SQL COUNT() Function

The COUNT() function returns the number of rows that matches a specified criteria.

Its syntax is:

The COUNT(column\_name) function returns the number of values (NULL values will not be counted) of the specified column:

SELECT COUNT(column\_name) FROM table\_name

## SQL COUNT(\*) Syntax

The COUNT(\*) function returns the number of records in a table:

SELECT COUNT(\*) FROM table\_name

## SQL COUNT (DISTINCT column\_name) Syntax

The COUNT (DISTINCT column\_name) function returns the number of distinct values of the specified column:

SELECT COUNT (DISTINCT column\_name) FROM table\_name

**Note:** COUNT(DISTINCT) works with ORACLE and Microsoft SQL Server, but not with Microsoft Access.

## Example

We have the following "Orders" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/11/12 | 1000 | Hansen |
| 2 | 2008/10/23 | 1600 | Nilsen |
| 3 | 2008/09/02 | 700 | Hansen |
| 4 | 2008/09/03 | 300 | Hansen |
| 5 | 2008/08/30 | 2000 | Jensen |
| 6 | 2008/10/04 | 100 | Nilsen |

Now we want to count the number of orders from "Customer Nilsen".

We use the following SQL statement:

SELECT COUNT(Customer) AS CustomerNilsen FROM Orders WHERE Customer='Nilsen'

The result of the SQL statement above will be 2, because the customer Nilsen has made 2 orders in total:

|  |
| --- |
| **CustomerNilsen** |
| 2 |

**SQL COUNT(\*) Example** If we omit the WHERE clause, like this:

SELECT COUNT(\*) AS NumberOfOrders FROM Orders

The result-set will look like this:

|  |
| --- |
| **NumberOfOrders** |
| 6 |

which is the total number of rows in the table.

**SQL COUNT(DISTINCT column\_name) Example** Now we want to count the number of unique customers in the "Orders" table.

We use the following SQL statement:

SELECT COUNT(DISTINCT Customer) AS NumberOfCustomers FROM Orders

The result-set will look like this:

|  |
| --- |
| **NumberOfCustomers** |
| 3 |

which is the number of unique customers (Hansen, Nilsen, and Jensen) in the "Orders" table

# 3- SQL MAX() Function

The MAX() function returns the largest value of the selected column.

## SQL MAX() Syntax

SELECT MAX(column\_name) FROM table\_name

**SQL MAX() Example** We have the following "Orders" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/11/12 | 1000 | Hansen |
| 2 | 2008/10/23 | 1600 | Nilsen |
| 3 | 2008/09/02 | 700 | Hansen |
| 4 | 2008/09/03 | 300 | Hansen |
| 5 | 2008/08/30 | 2000 | Jensen |
| 6 | 2008/10/04 | 100 | Nilsen |

Now we want to find the largest value of the "OrderPrice" column.

We use the following SQL statement:

SELECT MAX(OrderPrice) AS LargestOrderPrice FROM Orders

The result-set will look like this:

|  |
| --- |
| **LargestOrderPrice** |
| 2000 |

# The MIN() Function

The MIN() function returns the smallest value of the selected column.

# SQL MIN() Syntax

SELECT MIN(column\_name) FROM table\_name

# SQL MIN() Example

We have the following "Orders" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/11/12 | 1000 | Hansen |
| 2 | 2008/10/23 | 1600 | Nilsen |
| 3 | 2008/09/02 | 700 | Hansen |
| 4 | 2008/09/03 | 300 | Hansen |
| 5 | 2008/08/30 | 2000 | Jensen |
| 6 | 2008/10/04 | 100 | Nilsen |

Now we want to find the smallest value of the "OrderPrice" column.

We use the following SQL statement:

SELECT MIN(OrderPrice) AS SmallestOrderPrice FROM Orders

The result-set will look like this:

|  |
| --- |
| **SmallestOrderPrice** |
| 100 |

# SQL SUM() Function

The SUM() function returns the total sum of a numeric column.

## SQL SUM() Syntax

SELECT SUM(column\_name) FROM table\_name SQL SUM() Example

We have the following "Orders" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/11/12 | 1000 | Hansen |
| 2 | 2008/10/23 | 1600 | Nilsen |
| 3 | 2008/09/02 | 700 | Hansen |
| 4 | 2008/09/03 | 300 | Hansen |
| 5 | 2008/08/30 | 2000 | Jensen |
| 6 | 2008/10/04 | 100 | Nilsen |

Now we want to find the sum of all "OrderPrice" fields".

We use the following SQL statement:

SELECT SUM(OrderPrice) AS OrderTotal FROM Orders

The result-set will look like this:

|  |
| --- |
| **OrderTotal** |
| 5700 |

# SQL GROUP BY Statement

Aggregate functions often need an added GROUP BY statement.

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

## SQL GROUP BY Syntax

SELECT column\_name, aggregate\_function(column\_name)

FROM table\_name

WHERE column\_name operator value GROUP BY column\_name

## SQL GROUP BY Example

We have the following "Orders" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/11/12 | 1000 | Hansen |
| 2 | 2008/10/23 | 1600 | Nilsen |
| 3 | 2008/09/02 | 700 | Hansen |
| 4 | 2008/09/03 | 300 | Hansen |
| 5 | 2008/08/30 | 2000 | Jensen |
| 6 | 2008/10/04 | 100 | Nilsen |

Now we want to find the total sum (total order) of each customer.

We will have to use the GROUP BY statement to group the customers.

We use the following SQL statement:

SELECT Customer,SUM(OrderPrice) FROM Orders GROUP BY Customer

The result-set will look like this:

|  |  |
| --- | --- |
| **Customer** | **SUM(OrderPrice)** |
| Hansen | 2000 |
| Nilsen | 1700 |
| Jensen | 2000 |

Nice! Isn't it? :)

Let's see what happens if we omit the GROUP BY statement:

SELECT Customer,SUM(OrderPrice) FROM Orders

The result-set will look like this:

|  |  |
| --- | --- |
| **Customer** | **SUM(OrderPrice)** |
| Hansen | 5700 |
| Nilsen | 5700 |
| Hansen | 5700 |
| Hansen | 5700 |
| Jensen | 5700 |
| Nilsen | 5700 |

The result-set above is not what we wanted.

**Explanation of why the above SELECT statement cannot be used:** The SELECT statement above has two columns specified (Customer and SUM(OrderPrice). The "SUM(OrderPrice)" returns a single value (that is the total sum of the "OrderPrice" column), while "Customer" returns 6 values (one value for each row in the "Orders" table). This will therefore not give us the correct result. However, you have seen that the GROUP BY statement solves this problem.

## GROUP BY More Than One Column

We can also use the GROUP BY statement on more than one column, like this:

SELECT Customer,OrderDate,SUM(OrderPrice) FROM Orders

GROUP BY Customer,OrderDate

# The HAVING Clause

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

## SQL HAVING Syntax

SELECT column\_name, aggregate\_function(column\_name)

FROM table\_name

WHERE column\_name operator value

GROUP BY column\_name

HAVING aggregate\_function(column\_name) operator value

**SQL HAVING Example** We have the following "Orders" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/11/12 | 1000 | Hansen |
| 2 | 2008/10/23 | 1600 | Nilsen |
| 3 | 2008/09/02 | 700 | Hansen |
| 4 | 2008/09/03 | 300 | Hansen |
| 5 | 2008/08/30 | 2000 | Jensen |
| 6 | 2008/10/04 | 100 | Nilsen |

Now we want to find if any of the customers have a total order of less than 200.

We use the following SQL statement:

SELECT Customer,SUM(OrderPrice) FROM Orders

GROUP BY Customer

HAVING SUM(OrderPrice)<200

The result-set will look like this:

|  |  |
| --- | --- |
| **Customer** | **SUM(OrderPrice)** |
| Nilsen | 100 |

Now we want to find if the customers "Hansen" or "Jensen" have a total order of more than 1500.

We add an ordinary WHERE clause to the SQL statement:

SELECT Customer,SUM(OrderPrice) FROM Orders

WHERE Customer='Hansen' OR Customer='Jensen'

GROUP BY Customer

HAVING SUM(OrderPrice)>1500

The result-set will look like this:

|  |  |
| --- | --- |
| **Customer** | **SUM(OrderPrice)** |
| Hansen | 2000 |
| Jensen | 2000 |

# SQL UCASE() Function

## The UCASE() Function

The UCASE() function converts the value of a field to uppercase.

SQL UCASE() Syntax

SELECT UCASE(column\_name) FROM table\_name

Syntax for SQL Server

SELECT UPPER(column\_name) FROM table\_name

## SQL UCASE() Example

We have the following "Persons" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Now we want to select the content of the "LastName" and "FirstName" columns above, and convert the "LastName" column to uppercase.

We use the following SELECT statement:

SELECT UCASE(LastName) as LastName,FirstName FROM Persons

The result-set will look like this:

|  |  |
| --- | --- |
| **LastName** | **FirstName** |
| HANSEN | Ola |
| SVENDSON | Tove |
| PETTERSEN | Kari |

# SQL LCASE() Function

The LCASE() function converts the value of a field to lowercase.

## SQL LCASE() Syntax

SELECT LCASE(column\_name) FROM table\_name

## Syntax for SQL Server

SELECT LOWER(column\_name) FROM table\_name

## SQL LCASE() Example

We have the following "Persons" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Now we want to select the content of the "LastName" and "FirstName" columns above, and convert the "LastName" column to lowercase.

We use the following SELECT statement:

SELECT LCASE(LastName) as LastName,FirstName FROM Persons

The result-set will look like this:

|  |  |
| --- | --- |
| **LastName** | **FirstName** |
| hansen | Ola |
| svendson | Tove |
| pettersen | Kari |

# SQL LEN() Function

The LEN() function returns the length of the value in a text field.

## SQL LEN() Syntax

SELECT LEN(column\_name) FROM table\_name

**SQL LEN() Example** We have the following "Persons" table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P\_Id** | **LastName** | **FirstName** | **Address** | **City** |
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

Now we want to select the length of the values in the "Address" column above.

We use the following SELECT statement:

SELECT LEN(Address) as LengthOfAddress FROM Persons

The result-set will look like this:

|  |
| --- |
| **LengthOfAddress** |
| 12 |
| 9 |
| 9 |

# SQL GETDATE() Function

The GETDATE() function returns the current system date and time.

SQL GETDATE() Syntax

SELECT GETDATE();

**TASKS:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Reg\_No** | **Courses** | **Course\_Code** | **Offered\_By** |
| Ali | 01 | DIP | 1001 | Mr. A |
| Basit | 02 | DBMS | 1002 | Mr. X |
| Akram | 03 | OS | 1003 | Mr. Y |
| Asad | 04 | DBMS | 1002 | Mr. X |
| Zeeshan | 05 | DIP | 1001 | Mr. A |
| Muneer | 06 | OS |  | Mr. Y |
| Shafqat | 07 | NM | 1004 | Mr. H |
| Ahsan | 08 | OS | 1003 | Mr. Y |
| Ikram | 09 | DIP |  | Mr. A |
| Hassan | 10 | DSP |  |  |

For the above table perform the following tasks:-

## TASK 1:

* Calculate the number of records for the 3rd, 4th and 5th column.
* Find distinct number of records for the Course Code=1002 as Total.
* Find number of students registered for the course DIP as Total Courses.

## TASK 2:

Convert the text valued fields in the above table to the lower case and upper case alphabets.

## TASK 3:

Using GROUP BY statement, group the courses for the above table.

## TASK 4:

Select maximum of the Reg no and smallest valued course code for the above given table.

## TASK 5:

Find the length of each record for the first column in the above table as MAXIMUM LENGTH.

## TASK 6:

|  |  |  |  |
| --- | --- | --- | --- |
| **O\_Id** | **OrderDate** | **OrderPrice** | **Customer** |
| 1 | 2008/11/12 | 1000 | Hansen |
| 2 | 2008/10/23 | 1600 | Nilsen |
| 3 | 2008/09/02 | 700 | Hansen |
| 4 | 2008/09/03 | 300 | Hansen |
| 5 | 2008/08/30 | 2000 | Jensen |
| 6 | 2008/10/04 | 100 | Nilsen |

Find the average value for the 3rd column.

## TASK 7:

Find if the customers "Hansen" or "Nilsen" have a total order of less than 2100 for the following table:

|  |  |  |
| --- | --- | --- |
| **O\_Id** | **OrderPrice** | **Customer** |
| 1 | 1000 | Hansen |
| 2 | 1600 | Nilsen |
| 3 | 700 | Hansen |
| 4 | 300 | Hansen |
| 5 | 2000 | Jensen |
| 6 | 100 | Nilsen |

Also find if any customer have order of more than 1800.

## TASK 8:

Find the total sum (total order) of each customer.