ANAND GANESH POLISETTI

CLASS XI: INFORMATICS PRACTICES : MYSQL NOTES

INTRODUCTION TO MYSQL

What is MySQL?

MySQL is a open source Relational Database Management System. MySQL is very fast reliable and flexible Database Management System. It provides a very high performance and it is multi threaded and multi user Relational Database management system.

MySQL Features

1. MySQL are very fast and much reliable for any type of application.
2. MySQL is very Lightweight application.
3. MySQL command line tool is very powerful and can be used to run SQL queries against database.
4. MySQL supports indexing and binary objects.
5. It is allow changes to structure of table while server is running.
6. MySQL has a wide user base.
7. It is a very fast thread-based memory allocation system.
8. MySQL Written in C and C++ language.
9. MySQL code is tested with different compilers.
10. MySQL is available as a separate program for use in a client/server network environment.

Advantages of MySql:

Reliability and Performance : MySQL is very reliable and high performance relational database management system. It can used to store many GB's of data into database.

Availability of Source: MySQL source code is available that's why now you can recompile the source code.

Cross-Platform support: MySQL supports more then twenty different platform including the major Linux distribution .Mac OS X, Unix and Microsoft windows.

Large pool of Trained and Certified Developers: MySQL is very popular and it is world most popular open source Database. So it is easy to find high quality staff around the world.

Powerful Uncomplicated software: The MySQL has most capabilities to handle most corporate database application and used to very easy and fast

Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

Below is an example of a table called "Persons":

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The table above contains three records (one for each person) and five columns (P_Id, LastName, FirstName, Address, and City).

RDBMS

RDBMS stands for Relational Database Management System. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access. The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

Pages

Home
Informatics Practices Class XI
NOTICES : INFORMATICS PRACTICES FOR CLASS XII & CLASS XI
Class XII : Informatics Practices : Chapter wise Assignments
Class XII : Study Material : Informatics Practices : Chapterwise
EBooks Collections
INFORMATICS PRACTICES CLASS XII : SAMPLE QUESTION
CLASS XI CHAPTER 3: Getting started with IDE
CLASS XI: CHAPTER 4:Programming Fundamentals
OLD CBSE Question Paper CLASS XII: INFORMATICS PRACTICES _2011
CLASS XI: PRACTICAL SOLUTION : INFORMATICS PRACTICES
Practical List for class XII :
CLASS XI: INFORMATICS PRACTICES : MYSQL NOTES
TIPS FOR SCORING GOOD MARKS IN EXAM
MY-SQL ASSIGNMENT FOR CLASS XII

Blog Archive

▼ 2013 (9)
▼ October (1)
Sample Paper –Class – XII: MYSQL
► September (1)
► June (2)
► May (1)
► April (1)
► January (3)
► 2012 (4)

About Me

aNAND gANESH

View my complete profile

Keep in Mind That...

- SQL is not case sensitive
- Semicolon after SQL Statements?

Classification of SQL statements

SQL provides many different types of commands used for different purposes. SQL can be divided into following parts:

1. Data Manipulation Language (DML) commands
2. Data Definition Language (DDL) commands.
3. Transaction Control Language (TCL) commands.

DML commands: A DML is a language that enables users to access on manipulates data as organized by the appropriate data model. The query and update commands form the DML part of SQL:

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database

DDL commands: The DDL commands, as the name suggests, allow you to perform tasks related to data definition. The DDL part of SQL permits database tables to be created or deleted. The most important DDL statements in SQL are:

- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database
- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table

TCL commands: The TCL commands used to manage and control the transactions of data in database. The most important TCL commands are:

- **COMMIT** – it make all the changes made by statement issued.
- **ROLLBACK** – it undoes all changes since the beginning of the transaction or since save point.
- **SAVEPOINT** – it marks a point upto successfully completed transaction.
- **SET TRANSACTION** – it establish properties for the current transaction.

SIMPLE QUERIES IN SQL

MySQL Data Types

In MySQL there are three main data types: text, number, and Date/Time.

Text types:

Data type	Description
CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. **Note:** If you put a greater value than 255 it will be converted to a TEXT type
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 characters
BLOB	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(x,y,z,etc.)	Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted.

	Note: The values are sorted in the order you enter them. You enter the possible values in this format: ENUM('X','Y','Z')
SET	Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice

Number types:

Data type	Description
TINYINT(size)	-128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis
SMALLINT(size)	-32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis
MEDIUMINT(size)	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis
INT(size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
BIGINT(size)	-9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED*. The maximum number of digits may be specified in parenthesis
FLOAT(size,d)	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DOUBLE(size,d)	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DECIMAL(size,d)	A DOUBLE stored as a string , allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

*The integer types have an extra option called UNSIGNED. Normally, the integer goes from an negative to positive value. Adding the UNSIGNED attribute will move that range up so it starts at zero instead of a negative number.

Date types:

Data type	Description
DATE()	A date. Format: YYYY-MM-DD **Note:** The supported range is from '1000-01-01' to '9999-12-31'
DATETIME()	*A date and time combination. Format: YYYY-MM-DD HH:MM:SS **Note:** The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP()	*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MM:SS **Note:** The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC
TIME()	A time. Format: HH:MM:SS **Note:** The supported range is from '-838:59:59' to '838:59:59'
YEAR()	A year in two-digit or four-digit format. **Note:** Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

*Even if DATETIME and TIMESTAMP return the same format, they work very differently. In an INSERT or UPDATE query, the TIMESTAMP automatically set itself to the current date and time. TIMESTAMP also accepts various formats, like YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD, or YYMMDD.

-

SQL SELECT Statement

The SELECT statement is used to select data from a database. The result is stored in a result table, called the result-set. **Syntax**

SELECT column_name(s)
FROM table_name
and
SELECT * FROM table_name

An SQL SELECT Example

The "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 columns named "LastName" and "FirstName" from the table above.
We use the following SELECT statement:
SELECT LastName, FirstName FROM Persons
The result-set will look like this:

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

SELECT * Example

Now we want to select all the columns from the "Persons" table. We use the following SELECT statement:
SELECT * FROM Persons
Tip: The asterisk (*) is a quick way of selecting all columns!
The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The SQL SELECT DISTINCT Statement

In a table, some of the columns may contain duplicate values. This is not a problem, however, sometimes you will want to list only the different (distinct) values in a table.
The DISTINCT keyword can be used to return only distinct (different) values.

SQL SELECT DISTINCT Syntax

SELECT DISTINCT column_name(s)
FROM table_name

SELECT DISTINCT Example

The "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 only the distinct values from the column named "City" from the table above.
We use the following SELECT statement:
SELECT DISTINCT City FROM Persons
The result-set will look like this:

City
Sandnes
Stavanger

The WHERE Clause

The WHERE clause is used to extract only those records that fulfill a specified criterion.

SQL WHERE Syntax

SELECT column_name(s)
FROM table_name
WHERE column_name operator value

WHERE Clause Example

The "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 only the persons living in the city "Sandnes" from the table above.We use the following SELECT statement:
SELECT * FROM Persons
WHERE City='Sandnes'
The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Quotes Around Text Fields

SQL uses single quotes around text values (most database systems will also accept double quotes). Although, numeric values should not be enclosed in quotes. For text values:
This is correct:
SELECT * FROM Persons WHERE FirstName='Tove'
This is wrong:
SELECT * FROM Persons WHERE FirstName=Tove
For numeric values:
This is correct:
SELECT * FROM Persons WHERE Year=1965
This is wrong:
SELECT * FROM Persons WHERE Year='1965'

Operators Allowed in the WHERE Clause

With the WHERE clause, the following operators can be used:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	If you know the exact value you want to return for at least one of the columns

Note: In some versions of SQL the <> operator may be written as !=

The AND & OR Operators

The AND & OR operators are used to filter records based on more than one condition. The AND operator displays a record if both the first condition and the second condition is true. And OR operator displays a record if either the first condition or the second condition is true.

AND Operator Example

The "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 only the persons with the first name equal to "Tove" AND the last name equal to "Svendson":
We use the following SELECT statement:
SELECT * FROM Persons
WHERE FirstName='Tove'
AND LastName='Svendson'
The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes

OR Operator Example

Now we want to select only the persons with the first name equal to "Tove" OR the first name equal to "Ola":
We use the following SELECT statement:

SELECT * FROM Persons
WHERE FirstName='Tove'
OR FirstName='Ola'
The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Combining AND & OR

You can also combine AND and OR (use parenthesis to form complex expressions).
Now we want to select only the persons with the last name equal to "Svendson" AND the first name equal to "Tove" OR to "Ola":
We use the following SELECT statement:
SELECT * FROM Persons WHERE
LastName='Svendson'
AND (FirstName='Tove' OR FirstName='Ola')
The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes

The ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set by a specified column.The ORDER BY keyword sort the records in ascending order by default. If you want to sort the records in a descending order, you can use the DESC keyword.

SQL ORDER BY Syntax

SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC

ORDER BY Example

The "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
4	Nilsen	Tom	Vingvn 23	Stavanger

Now we want to select all the persons from the table above, however, we want to sort the persons by their last name.
We use the following SELECT statement:
SELECT * FROM Persons
ORDER BY LastName
The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
4	Nilsen	Tom	Vingvn 23	Stavanger
3	Pettersen	Kari	Storgt 20	Stavanger
2	Svendson	Tove	Borgvn 23	Sandnes

ORDER BY DESC Example

Now we want to select all the persons from the table above, however, we want to sort the persons descending by their last name.
We use the following SELECT statement:
SELECT * FROM Persons
ORDER BY LastName DESC
The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger
1	Hansen	Ola	Timoteivn 10	Sandnes

The INSERT INTO Statement

The INSERT INTO statement is used to insert a new row in a table.

SQL INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two forms.

The first form doesn't specify the column names where the data will be inserted, only their values:

INSERT INTO table_name
VALUES (value1, value2, value3,...)

The second form specifies both the column names and the values to be inserted:

INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)

SQL INSERT INTO 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 insert a new row in the "Persons" table.

We use the following SQL statement:

INSERT INTO Persons
VALUES (4,'Nilsen', 'Johan', 'Bakken 2', 'Stavanger')

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger

Insert Data Only in Specified Columns

It is also possible to only add data in specific columns.

The following SQL statement will add a new row, but only add data in the "P_Id", "LastName" and the "FirstName" columns:

INSERT INTO Persons (P_Id, LastName, FirstName)
VALUES (5, 'Tjessem', 'Jakob')

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob		

The UPDATE Statement

The UPDATE statement is used to update existing records in a table.

SQL UPDATE Syntax

UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value

Note: Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

SQL UPDATE Example

The "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
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob		

Now we want to update the person "Tjessem, Jakob" in the "Persons" table.

We use the following SQL statement:

UPDATE Persons
SET Address='Nissestien 67', City='Sandnes'
WHERE LastName='Tjessem' AND FirstName='Jakob'

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	Nissestien 67	Sandnes

The DELETE Statement

The DELETE statement is used to delete rows in a table.

SQL DELETE Syntax

DELETE FROM table_name
WHERE some_column=some_value

Note: Notice the WHERE clause in the DELETE syntax. The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

SQL DELETE Example

The "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
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	Nissestien 67	Sandnes

Now we want to delete the person "Tjessem, Jakob" in the "Persons" table.
We use the following SQL statement:

DELETE FROM Persons
WHERE LastName='Tjessem' AND FirstName='Jakob'

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger

Delete All Rows

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

DELETE FROM table_name
or
DELETE * FROM table_name

Note: Be very careful when deleting records. You cannot undo this statement!

Try it Yourself

Test your SQL Skills

On this page you can test your SQL skills.
We will use the **Customers** table in the Northwind database:

CompanyName	ContactName	Address	City
Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin
Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå
Centro comercial Moctezuma	Francisco Chang	Sierras de Granada 9993	México D.F.
Ernst Handel	Roland Mendel	Kirchgasse 6	Graz
FISSA Fabrica Inter. Salchichas S.A.	Diego Roel	C/ Moralzarzal, 86	Madrid
Galería del gastrónomo	Eduardo Saavedra	Rambla de Cataluña, 23	Barcelona
Island Trading	Helen Bennett	Garden House Crowther Way	Cowes
Königlich Essen	Philip Cramer	Maubelstr. 90	Brandenburg
Laughing Bacchus Wine Cellars	Yoshi Tannamuri	1900 Oak St.	Vancouver
Magazzini Alimentari Riuniti	Giovanni Rovelli	Via Ludovico il Moro 22	Bergamo

North/South	Simon Crowther	South House 300 Queensbridge	London
Paris spécialités	Marie Bertrand	265, boulevard Charonne	Paris
Rattlesnake Canyon Grocery	Paula Wilson	2817 Milton Dr.	Albuquerque
Simons bistro	Jytte Petersen	Vinbæltet 34	København
The Big Cheese	Liz Nixon	89 Jefferson Way Suite 2	Portland
Vaffeljernet	Palle Ibsen	Smagsløget 45	Århus
Wolski Zajazd	Zbyszek Piestrzeniewicz	ul. Filtrowa 68	Warszawa

To preserve space, the table above is a subset of the Customers table used in the example below.

To see how SQL works, you can copy the SQL statements below and paste them into the textarea, or you can make your own SQL statements.

```
SELECT * FROM customers

SELECT CompanyName, ContactName FROM customers

SELECT * FROM customers WHERE companyname LIKE 'a%'

SELECT CompanyName, ContactName
FROM customers
WHERE CompanyName > 'a'
```

When using SQL on text data, "alfred" is greater than "a" (like in a dictionary).

```
SELECT CompanyName, ContactName
FROM customers
WHERE CompanyName > 'g'
AND ContactName > 'g'
```

SQL LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

The LIKE Operator

The LIKE operator is used to search for a specified pattern in a column.

SQL LIKE Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern
```

LIKE Operator Example

The "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 persons living in a city that starts with "s" from the table above.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE 's%'
```

The "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Next, we want to select the persons living in a city that ends with an "s" from the "Persons" table.

We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE '%s'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Next, we want to select the persons living in a city that contains the pattern "tav" from the "Persons" table.
We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE '%tav%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
3	Pettersen	Kari	Storgt 20	Stavanger

It is also possible to select the persons living in a city that NOT contains the pattern "tav" from the "Persons" table, by using the NOT keyword.
We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City NOT LIKE '%tav%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

SQL Wildcards

SQL wildcards can substitute for one or more characters when searching for data in a database.
SQL wildcards must be used with the SQL LIKE operator.
With SQL, the following wildcards can be used:

Wildcard	Description
%	A substitute for zero or more characters
_	A substitute for exactly one character
[charlist]	Any single character in charlist
[^charlist] or [!charlist]	Any single character not in charlist

SQL Wildcard Examples

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

Using the % Wildcard

Now we want to select the persons living in a city that starts with "sa" from the "Persons" table. We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE 'sa%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Next, we want to select the persons living in a city that contains the pattern "nes" from the "Persons" table. We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE City LIKE '%nes%'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Using the _ Wildcard

Now we want to select the persons with a first name that starts with any character, followed by "la" from the "Persons" table.
We use the following SELECT statement:

SELECT * FROM Persons
WHERE FirstName LIKE '_la'
The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes

Next, we want to select the persons with a last name that starts with "S", followed by any character, followed by "end", followed by any character, followed by "on" from the "Persons" table.
We use the following SELECT statement:
SELECT * FROM Persons
WHERE LastName LIKE 'S_end_on'
The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes

Using the [charlist] Wildcard

Now we want to select the persons with a last name that starts with "b" or "s" or "p" from the "Persons" table.
We use the following SELECT statement:
SELECT * FROM Persons
WHERE LastName LIKE '[bsp]%'
The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Next, we want to select the persons with a last name that do not start with "b" or "s" or "p" from the "Persons" table.
We use the following SELECT statement:
SELECT * FROM Persons
WHERE LastName LIKE '[!bsp]%'
The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes

The IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

SQL IN Syntax

SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...)

IN Operator Example

The "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 persons with a last name equal to "Hansen" or "Pettersen" from the table above.
We use the following SELECT statement:
SELECT * FROM Persons
WHERE LastName IN ('Hansen','Pettersen')
The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The BETWEEN Operator

The BETWEEN operator selects a range of data between two values. The values can be numbers, text, or dates.

SQL BETWEEN Syntax

SELECT column_name(s)
FROM table_name
WHERE column_name
BETWEEN value1 AND value2

BETWEEN Operator Example

The "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 persons with a last name alphabetically between "Hansen" and "Pettersen" from the table above.
We use the following SELECT statement:

```
SELECT * FROM Persons
WHERE LastName
BETWEEN 'Hansen' AND 'Pettersen'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes

Note: The BETWEEN operator is treated differently in different databases.
In some databases, persons with the LastName of "Hansen" or "Pettersen" will not be listed, because the BETWEEN operator only selects fields that are between and excluding the test values).
In other databases, persons with the LastName of "Hansen" or "Pettersen" will be listed, because the BETWEEN operator selects fields that are between and including the test values).
And in other databases, persons with the LastName of "Hansen" will be listed, but "Pettersen" will not be listed (like the example above), because the BETWEEN operator selects fields between the test values, including the first test value and excluding the last test value.
Therefore: Check how your database treats the BETWEEN operator.

Example 2

To display the persons outside the range in the previous example, use NOT BETWEEN:

```
SELECT * FROM Persons
WHERE LastName
NOT BETWEEN 'Hansen' AND 'Pettersen'
```

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

SQL Alias

You can give a table or a column another name by using an alias. This can be a good thing to do if you have very long or complex table names or column names.
An alias name could be anything, but usually it is short.

SQL Alias Syntax for Tables

```
SELECT column_name(s)
FROM table_name
AS alias_name
```

SQL Alias Syntax for Columns

```
SELECT column_name AS alias_name
FROM table_name
```

Alias Example

Assume we have a table called "Persons" and another table called "Product_Orders". We will give the table aliases of "p" an "po" respectively. Now we want to list all the orders that "Ola Hansen" is responsible for.
We use the following SELECT statement:

```
SELECT po.OrderID, p.LastName, p.FirstName
FROM Persons AS p,
Product_Orders AS po
WHERE p.LastName='Hansen' AND p.FirstName='Ola'
```

The same SELECT statement without aliases:

```
SELECT Product_Orders.OrderID, Persons.LastName, Persons.FirstName
FROM Persons,
Product_Orders
WHERE Persons.LastName='Hansen' AND Persons.FirstName='Ola'
```

As you'll see from the two SELECT statements above; aliases can make queries easier to both write and to read.

The CREATE TABLE Statement

The CREATE TABLE statement is used to create a table in a database.

SQL CREATE TABLE Syntax

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
....
)
```

The data type specifies what type of data the column can hold. For a complete reference of all the data types available in MS Access, MySQL, and SQL Server, go to our complete Data Types reference.

CREATE TABLE Example

Now we want to create a table called "Persons" that contains five columns: P_Id, LastName, FirstName, Address, and City.

We use the following CREATE TABLE statement:

```
CREATE TABLE Persons
(
P_Id int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

The P_Id column is of type int and will hold a number. The LastName, FirstName, Address, and City columns are of type varchar with a maximum length of 255 characters.

The empty "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City

The empty table can be filled with data with the INSERT INTO statement.

SQL Constraints

Constraints are used to limit the type of data that can go into a table. Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement). We will focus on the following constraints:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

The next chapters will describe each constraint in details.

SQL NOT NULL Constraint

The NOT NULL constraint enforces a column to NOT accept NULL values. The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.

The following SQL enforces the "P_Id" column and the "LastName" column to not accept NULL values:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

SQL UNIQUE Constraint

The UNIQUE constraint uniquely identifies each record in a database table. The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns. A PRIMARY KEY constraint automatically has a UNIQUE

constraint defined on it. Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint on the "P_Id" column when the "Persons" table is created:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
UNIQUE (P_Id)
)
```

SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a database table.
Primary keys must contain unique values.
A primary key column cannot contain NULL values.
Each table should have a primary key, and each table can have only one primary key.

SQL PRIMARY KEY Constraint on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "P_Id" column when the "Persons" table is created:

MySQL:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
PRIMARY KEY (P_Id)
)
```

SQL FOREIGN KEY Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.
Let's illustrate the foreign key with an example. Look at the following two tables:
The "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

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Note that the "P_Id" column in the "Orders" table points to the "P_Id" column in the "Persons" table.
The "P_Id" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.
The "P_Id" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.
The FOREIGN KEY constraint is used to prevent actions that would destroy link between tables.
The FOREIGN KEY constraint also prevents that invalid data is inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

SQL FOREIGN KEY Constraint on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "P_Id" column when the "Orders" table is created:

```
CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
PRIMARY KEY (O_Id),
FOREIGN KEY (P_Id) REFERENCES Persons(P_Id)
)
```

SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.
If you define a CHECK constraint on a single column it allows only certain values for this column.
If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SQL CHECK Constraint on CREATE TABLE

The following SQL creates a CHECK constraint on the "P_Id" column when the "Persons" table is created. The CHECK constraint specifies that the column "P_Id" must only include integers greater than 0.

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CHECK (P_Id>0)
)
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

SQL DEFAULT Constraint

The DEFAULT constraint is used to insert a default value into a column.
The default value will be added to all new records, if no other value is specified.

SQL DEFAULT Constraint on CREATE TABLE

The following SQL creates a DEFAULT constraint on the "City" column when the "Persons" table is created:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255) DEFAULT 'Sandnes'
)
```

The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE():

```
CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
OrderDate date DEFAULT GETDATE()
)
```

The ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

SQL ALTER TABLE Syntax

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype
```

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name
DROP COLUMN column_name
```

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype
```

SQL ALTER TABLE Example

Look at the "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 add a column named "DateOfBirth" in the "Persons" table.
We use the following SQL statement:
ALTER TABLE Persons
ADD DateOfBirth date
Notice that the new column, "DateOfBirth", is of type date and is going to hold a date. The data type specifies what type of data the column can hold. For a complete reference of all the data types available in MS Access, MySQL, and SQL Server, go to our complete Data Types reference.
The "Persons" table will now like this:

P_Id	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

Change Data Type Example

Now we want to change the data type of the column named "DateOfBirth" in the "Persons" table.
We use the following SQL statement:
ALTER TABLE Persons
ALTER COLUMN DateOfBirth year
Notice that the "DateOfBirth" column is now of type year and is going to hold a year in a two-digit or four-digit format.

—

—

SQL Functions

SQL has many built-in functions for performing calculations on data.

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
- FIRST() - Returns the first value
- LAST() - Returns the last value
- 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
- MID() - Extract characters from a text field
- LEN() - Returns the length of a text field
- ROUND() - Rounds a numeric field to the number of decimals specified
- NOW() - Returns the current system date and time
- FORMAT() - Formats how a field is to be displayed

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

The AVG() Function

The AVG() function returns the average value of a numeric column.
SQL AVG() Syntax
SELECT AVG(column_name) FROM table_name

SQL AVG() 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

SQL COUNT() Function

« PreviousNext Chapter »

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

SQL COUNT(column_name) Syntax
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.

SQL COUNT(column_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 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.

The 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

The 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

The 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 2000.
We use the following SQL statement:
SELECT Customer,SUM(OrderPrice) FROM Orders
GROUP BY Customer
HAVING SUM(OrderPrice)<2000
The result-set will look like this:

Customer	SUM(OrderPrice)
Nilsen	1700

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

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

The 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

The MID() Function

The MID() function is used to extract characters from a text field.

SQL MID() Syntax

SELECT MID(column_name,start[,length]) FROM table_name

Parameter	Description
column_name	Required. The field to extract characters from
start	Required. Specifies the starting position (starts at 1)
length	Optional. The number of characters to return. If omitted, the MID() function returns the rest of the text

SQL MID() 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 extract the first four characters of the "City" column above.

We use the following SELECT statement:

SELECT MID(City,1,4) as SmallCity FROM Persons

The result-set will look like this:

SmallCity
Sand
Sand
Stav

The 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

The ROUND() Function

The ROUND() function is used to round a numeric field to the number of decimals specified.

SQL ROUND() Syntax

SELECT ROUND(column_name,decimals) FROM table_name

Parameter	Description
column_name	Required. The field to round.
decimals	Required. Specifies the number of decimals to be returned.

SQL ROUND() Example

We have the following "Products" table:

Prod_Id	ProductName	Unit	UnitPrice
1	Jarlsberg	1000 g	10.45
2	Mascarpone	1000 g	32.56
3	Gorgonzola	1000 g	15.67

Now we want to display the product name and the price rounded to the nearest integer.

We use the following SELECT statement:

SELECT ProductName, ROUND(UnitPrice,0) as UnitPrice FROM Products

The result-set will look like this:

ProductName	UnitPrice
Jarlsberg	10
Mascarpone	33
Gorgonzola	16

The NOW() Function

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

SQL NOW() Syntax

SELECT NOW() FROM table_name

SQL NOW() Example

We have the following "Products" table:

Prod_Id	ProductName	Unit	UnitPrice
1	Jarlsberg	1000 g	10.45
2	Mascarpone	1000 g	32.56
3	Gorgonzola	1000 g	15.67

Now we want to display the products and prices per today's date.

We use the following SELECT statement:

SELECT ProductName, UnitPrice, Now() as PerDate FROM Products

The result-set will look like this:

ProductName	UnitPrice	PerDate
Jarlsberg	10.45	10/7/2008 11:25:02 AM
Mascarpone	32.56	10/7/2008 11:25:02 AM
Gorgonzola	15.67	10/7/2008 11:25:02 AM

---

13 comments:

Michelle Zody 6 May 2015 at 08:11

great job

Reply

Michelle Zody 6 May 2015 at 08:12

great job

Reply

Anonymous 2 November 2015 at 05:19

This comment has been removed by the author.

Reply

Unknown 28 February 2016 at 03:49

GREAT JOB

Reply

Unknown 16 December 2016 at 17:26

great job bro you are easing the way of students to get marks easily and complete their notes copy fastly and with ease .....(:- lol

Reply

Unknown 22 March 2017 at 10:09

Really fantastic . These notes are in very simple language and very attractive which helps in better understanding . Thanks.....

Reply

Unknown 22 March 2017 at 10:11

Please tell how foreign key works .....

Reply

Unknown 22 March 2017 at 10:11

Please tell how foreign key works .....

Reply

Shubham Ranjan 23 October 2017 at 07:26

Thanks a lot this helped me a LOT ,
🤘🤘🤘🤘

Reply

lipi 23 July 2020 at 02:49

helpful

Reply

harika 12 March 2022 at 01:28

how to download this notes.its very helpful

Reply

Shubham 23 December 2022 at 06:20

Marvelous work. Thank you a lot!!!!!

Reply

renuka 1 April 2024 at 21:47

Very good article . Thanks for sharing.
Snowflake Training
Snowflake Training in Hyderabad
Snowflake Online Training
Snowflake Online Training Hyderabad
Snowflake Training Online
Snowflake Training in Ameerpet
Snowflake Training Institute in Hyderabad

Reply

Enter comment

Home

Subscribe to: Posts (Atom)