**­­­­LECTURE 18:**

MySQL Tutorial

MySQL is a widely used relational database management system (RDBMS).

MySQL is free and open source.

MySQL is ideal for both small and large applications.

What is a Database Table?

A table is a collection of related data entries, and it consists of columns and rows.

A column holds specific information about every record in the table.

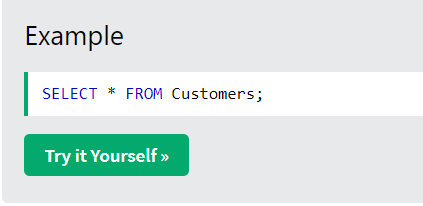
A record (or row) is each individual entry that exists in a table.

Following is a sample table with entries.



How to Use SQL

The following SQL statement selects all the records in the "Customers" table:



Keep in Mind That...

* SQL keywords are NOT case sensitive: select is the same as SELECT

In this tutorial we will write all SQL keywords in upper-case.

Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement.

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

In this tutorial, we will use semicolon at the end of each SQL statement.

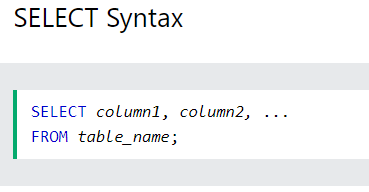
Some of The Most Important SQL Commands

* 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
* 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
* CREATE INDEX - creates an index (search key)
* DROP INDEX - deletes an index

The MySQL SELECT Statement

The SELECT statement is used to select data from a database.

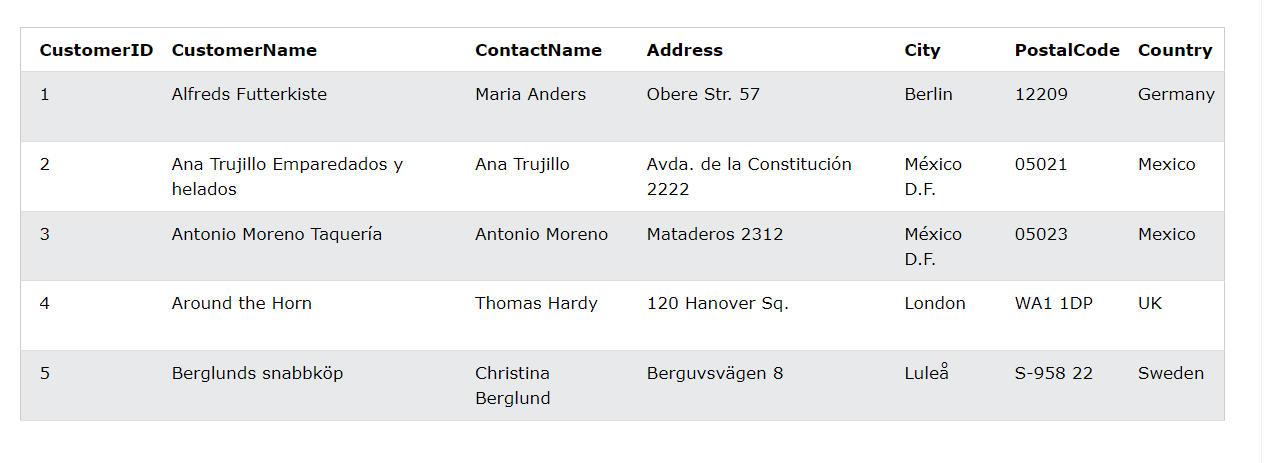
The data returned is stored in a result table, called the result-set



Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

SELECT \* FROM *table\_name*;

Demo Database



SELECT Columns Example

The following SQL statement selects the "CustomerName", "City", and "Country" columns from the "Customers" table:

Example

SELECT CustomerName, City, Country FROM Customers;

Example from class:

select customer\_name from sample.customer\_table;

SELECT \* Example

The following SQL statement selects ALL the columns from the "Customers" table:

Example

SELECT \* FROM Customers;

The MySQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT Syntax

SELECT DISTINCT *column1*,*column2, ...*  
FROM *table\_name*;

Example from class:

select distinct customer\_name from sample.customer\_table;

SELECT Example Without DISTINCT

The following SQL statement selects all (including the duplicates) values from the "Country" column in the "Customers" table:

Example

SELECT Country FROM Customers;

Now, let us use the SELECT DISTINCT statement and see the result.

SELECT DISTINCT Examples

The following SQL statement selects only the DISTINCT values from the "Country" column in the "Customers" table:

Example

SELECT DISTINCT Country FROM Customers;

The following SQL statement counts and returns the number of different (distinct) countries in the "Customers" table:

Example

SELECT COUNT(DISTINCT Country) FROM Customers;

MySQL WHERE Clause

The MySQL WHERE Clause

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

WHERE Syntax

SELECT *column1*,*column2, ...*  
FROM *table\_name*  
WHERE *condition*;

WHERE Clause Example

The following SQL statement selects all the customers from "Mexico":

Example

SELECT \* FROM Customers  
WHERE Country = 'Mexico';

Example from class:

select\* from sample.customer\_table

where customer\_name='Cardinal';

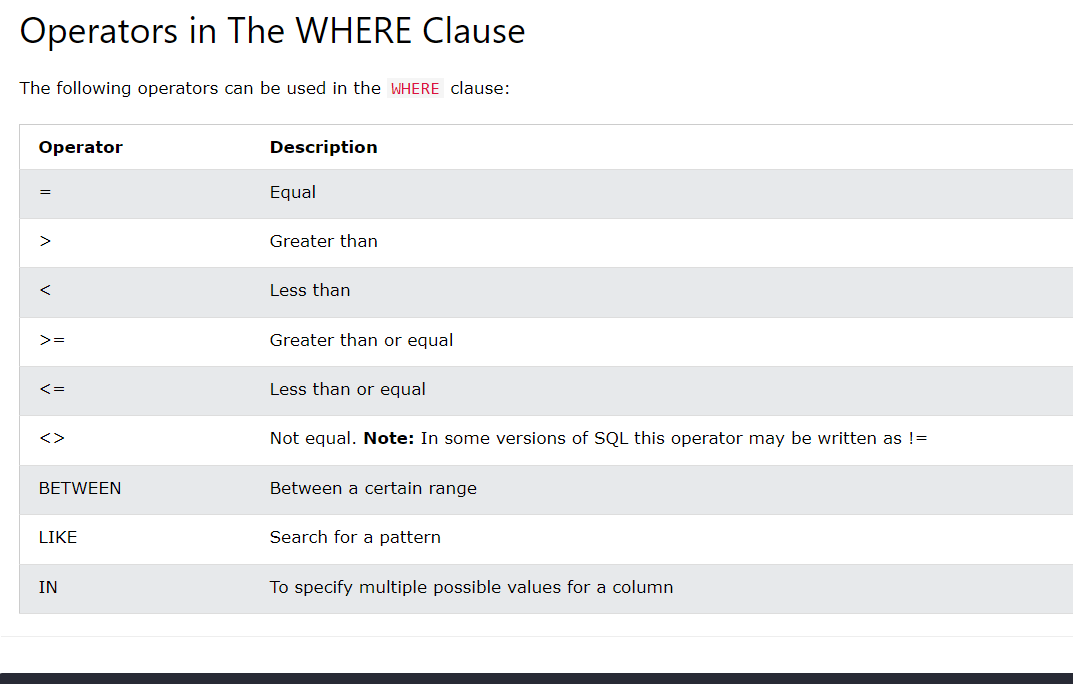
Text Fields vs. Numeric Fields

SQL requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes:

Example

SELECT \* FROM Customers  
WHERE CustomerID = 1;



MySQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax

SELECT *column1*,*column2, ...*  
FROM *table\_name*  
ORDER BY *column1, column2, ...*ASC|DESC;

ORDER BY Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" column:

Example

SELECT \* FROM Customers  
ORDER BY Country;

ORDER BY DESC Example

The following SQL statement selects all customers from the "Customers" table, sorted DESCENDING by the "Country" column:

Example

SELECT \* FROM Customers  
ORDER BY Country DESC;

ORDER BY Several Columns Example

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

Example

SELECT \* FROM Customers  
ORDER BY Country, CustomerName;

MySQL INSERT INTO Statement

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

INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

INSERT INTO *table\_name* (*column1*,*column2*,*column3*, ...)  
VALUES (*value1*,*value2*,*value3*, ...);

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

INSERT INTO *table\_name*  
VALUES (*value1*,*value2*,*value3*, ...);

INSERT INTO Example

The following SQL statement inserts a new record in the "Customers" table:

Example

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)  
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

­

Insert Data Only in Specified Columns

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

The following SQL statement will insert a new record, but only insert data in the "CustomerName", "City", and "Country" columns (CustomerID will be updated automatically):

Example

INSERT INTO Customers (CustomerName, City, Country)  
VALUES ('Cardinal', 'Stavanger', 'Norway');

MySQL UPDATE Statement

The MySQL UPDATE Statement

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

UPDATE Syntax

UPDATE *table\_name*  
SET *column1*=*value1*,*column2*=*value2*, ...  
WHERE *condition*;

UPDATE Table

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person *and* a new city.

Example

UPDATE Customers  
SET ContactName = 'Alfred Schmidt', City = 'Frankfurt'  
WHERE CustomerID = 1;

UPDATE Multiple Records

It is the WHERE clause that determines how many records will be updated.

The following SQL statement will update the PostalCode to 00000 for all records where country is "Mexico":

Example

UPDATE Customers  
SET PostalCode = 00000  
WHERE Country = 'Mexico';

MySQL DELETE Statement

The MySQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

DELETE Syntax

DELETE FROM *table\_name*WHERE *condition*;

SQL DELETE Example

The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

Example

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

Delete All Records

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*;

The following SQL statement deletes all rows in the "Customers" table, without deleting the table:

Example

DELETE FROM Customers;

Constraints in MySql

* Primary key
* Not null
* Unique
* Binary
* Unsigned
* Zero fill

**1. Primary Key:**

In MySQL, a primary key is a column or a set of columns that uniquely identifies each row in a table. Here are some key points about primary keys:

1. Uniqueness: Every value in the primary key column(s) must be unique across all rows in the table. This uniqueness constraint ensures that each row can be uniquely identified by its primary key.
2. Non-nullability: A primary key column cannot contain null values. This ensures that every row in the table has a valid primary key value.
3. Single or Composite: A primary key can be composed of a single column or multiple columns. When multiple columns are used, their combination must be unique.
4. Indexed: MySQL automatically creates an index on the primary key column(s). This index allows for fast lookups and efficient enforcement of the uniqueness constraint.

**2. Not null:**

In MySQL, NOT NULL is a constraint that can be applied to a column in a table. It indicates that the column must have a value, and NULL values are not allowed.

**3. Unique**

In MySQL, the UNIQUE constraint is used to ensure that all values in a column are unique, meaning no two rows can have the same value in that column. If you try to insert or update a row with a value that already exists in the column with a UNIQUE constraint, MySQL will generate an error.

**4. Binary**

In MySQL, the BINARY data type is used to store binary data, such as byte strings or binary-encoded values. Here are some key points about the BINARY data type:

1. Fixed-Length Binary Data: The BINARY data type is used to store binary strings of fixed length. It is similar to CHAR, but it stores binary data rather than character data.
2. Storage Size: The storage size of a BINARY column is fixed and determined by the length specified when defining the column. Each character in the binary string occupies one byte of storage.

**5. Unsigned**

In MySQL, the UNSIGNED attribute can be applied to numeric data types to restrict the column to accept only non-negative values.

**6. Zero fill**

In MySQL, the ZEROFILL attribute can be applied to numeric data types to instruct MySQL to pad the displayed values with leading zeros up to the specified width. This attribute is primarily used for formatting purposes when presenting numeric values to users or applications.