Database Management Systems

Associate Professor Dr. Raed Ibraheem Hamed

University of Human Development, College of Science and Technology
Computer Science Department

2015 – 2016

What is SQL?

SQL stands for Structured Query Language.

It is a standard language developed for accessing and modifying relational databases.

SQL in turn is used by a **database management system**. Some common database management systems are:

- MySQL
- SQLite
- PostgreSQL
- Oracle
- Microsoft SQL Server

Introduction to MySQL

MySQL, is an open source relational database management system. It is based on the **structure query language (SQL)**, which is used for adding, removing, and modifying information in the database. Standard SQL commands, such as CREATE , DROP, INSERT, and UPDATE can be used with MySQL.

- Relational databases
- Database design
- SQL
  - ✓ Creating databases
  - ✓ Creating tables
  - ✓ Selecting from, deleting, and updating tables

MySQL

- MySQL is free.
- Very widely used.
- Implements SQL database mangement.
- Linux already includes MySQL.
- Facebook uses MySQL.

At the MySQL syntax we can write in SQL instructions.

We will now learn more about the basic SQL commands.

SQL UPDATE Statement

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

SQL UPDATE Syntax

```
UPDATE table_name
SET column1=value1,column2=value2,...
WHERE some_column=some_value;
```

We use the following SQL statement:

Example

```
UPDATE Customers
SET ContactName='Alfred Schmidt', City='Hamburg'
WHERE CustomerName='Alfreds Futterkiste';
```

SQL UPDATE 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

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Hamburg	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

Update Warning!

Be careful when updating records. If we had omitted the WHERE clause, in the example above, like this:

UPDATE Customers
SET ContactName='Alfred Schmidt', City='Hamburg';

The "Customers" table would have looked like this:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Hamburg	12209	Germany
2	Ana Trujillo Emparedados y helados	Alfred Schmidt	Avda. de la Constitución 2222	Hamburg	05021	Mexico
3	Antonio Moreno Taquería	Alfred Schmidt	Mataderos 2312	Hamburg	05023	Mexico
4	Around the Horn	Alfred Schmidt	120 Hanover Sq.	Hamburg	WA1 1DP	UK
5	Berglunds snabbköp	Alfred Schmidt	Berguvsvägen 8	Hamburg	S-958 22	Sweden

INSERT INTO Example

Assume we wish to insert a new row in the "Customers" table.
We can use the following SQL statement:

Example

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

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
87	Wartian Herkku	Pirkko Koskitalo	Torikatu 38	Oulu	90110	Finland
88	Wellington Importadora	Paula Parente	Rua do Mercado, 12	Resende	08737-363	Brazil
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
87	Wartian Herkku	Pirkko Koskitalo	Torikatu 38	Oulu	90110	Finland
88	Wellington Importadora	Paula Parente	Rua do Mercado, 12	Resende	08737-363	Brazil
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland
92	Cardinal	Tom B. Erichsen	Skagen 21	Stavanger	4006	Norway

SQL DELETE Statement

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

SQL DELETE Syntax

DELETE FROM *table_name*
WHERE *some_column=some_value*;

Example

```
DELETE FROM Customers
WHERE CustomerName='Alfreds Futterkiste' AND
ContactName='Maria Anders';
```

SQL DELETE Statement

"Customers" table:

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

The "Customers" table will now look like this:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
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

Delete All Data

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 sound :

DELETE FROM *table_name*;

or

DELETE * FROM *table_name*;

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

IN Operator

The following SQL statement selects all customers with a City of "Paris" or "London":

Example

```
SELECT * FROM Customers
WHERE City IN ('Paris', 'London');
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
11	B's Beverages	Victoria Ashworth	Fauntleroy Circus	London	EC2 5NT	UK
16	Consolidated Holdings	Elizabeth Brown	Berkeley Gardens 12 Brewery	London	WX1 6LT	UK
19	Eastern Connection	Ann Devon	35 King George	London	WX3 6FW	UK
53	North/South	Simon Crowther	South House 300 Queensbridge	London	SW7 1RZ	UK
57	Paris spécialités	Marie Bertrand	265, boulevard Charonne	Paris	75012	France

SQL CREATE DATABASE and Tables

The following SQL statement creates a **database** called "my_db":

```
CREATE DATABASE my_db;
```

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

```
CREATE TABLE  Persons
(
PersonID int,
LastName varchar(55),
FirstName varchar(55),
Address varchar(55),
City varchar(55));
```

SQL Constraints

SQL constraints are used to specify rules for the data in a table.

If there is any violation between the constraint and the data action, the action is aborted by the constraint.

Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

SQL CREATE TABLE + CONSTRAINT Syntax

```
CREATE TABLE table_name
(
column_name1 data_type(size) constraint_name,
column_name2 data_type(size) constraint_name,
....        );
```

SQL Constraints

In SQL, we have the following constraints:

1) **NOT NULL** - Indicates that a column cannot store NULL value
2) **UNIQUE** - Ensures that each row for a column must have a unique value
3) **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have a unique identity which helps to find a particular record in a table more easily and quickly
4) **FOREIGN KEY** - Ensure the referential integrity of the data in one table to match values in another table
5) **CHECK** - Ensures that the value in a column meets a specific condition
6) **DEFAULT** - Specifies a default value for a column

SQL NOT NULL Constraint

The NOT NULL constraint enforces a column to **NOT accept NULL values**.

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

Example:-

```
CREATE TABLE PersonsNotNull
(
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 following SQL creates a UNIQUE constraint on the "P_Id" column when the "Persons" table is created:

Example:-

```
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 UNIQUE Constraint

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

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

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:

```
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.

Note that the "P_Id" column in the "Orders" table points to the "P_Id" column in the "Persons" table.

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

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.

SQL FOREIGN KEY Constraint

- 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 links between tables.
- The FOREIGN KEY constraint also prevents invalid data from being 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.

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)
)
```

SQL DEFAULT Constraint

The DEFAULT constraint is used to insert a default value into a column.

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 'London '
)
```

The default value will be added to all new records, if no other value is specified.

Thank you

???