SQL

The SQL CREATE DATABASE Statement:

The CREATE DATABASE statement is used to create a new SQL database.

Syntax

```
CREATE DATABASE databasename;
```

The SQL DROP DATABASE Statement:

The DROP DATABASE statement is used to drop an existing SQL database.

Syntax

```
DROP DATABASE databasename;
```

The SQL BACKUP DATABASE Statement:

The BACKUP DATABASE statement is used in SQL Server to create a full back up of an existing SQL database.

Syntax

```
BACKUP DATABASE databasename
TO DISK = 'filepath';
```

The SQL BACKUP WITH DIFFERENTIAL Statement:

A differential back up only backs up the parts of the database that have changed since the last full database backup.

Syntax

```
BACKUP DATABASE databasename
TO DISK = 'filepath'
WITH DIFFERENTIAL;
```

The SQL CREATE TABLE Statement:

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

Syntax

```
CREATE TABLE table_name (
   column1 datatype,
   column2 datatype,
   column3 datatype,
  ....
);
```

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

The SQL DROP TABLE Statement:

The DROP TABLE statement is used to drop an existing table in a database.

Syntax

```
DROP TABLE table_name;
```

SQL ALTER TABLE Statement:

- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
- The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

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

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

SQL Create Constraints:

- Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.
- SQL constraints are used to specify rules for data in a table.

Syntax:

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

SQL Constraints:

- SQL constraints are used to specify rules for the data in a table.
- Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.
- Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Prevents actions that would destroy links between tables
- CHECK - Ensures that the values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column if no value is specified
- CREATE INDEX - Used to create and retrieve data from the database very quickly

SQL UNIQUE Constraint:

- The UNIQUE constraint ensures that all values in a column are different.

- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

- A PRIMARY KEY constraint automatically has a UNIQUE constraint.

- However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    UNIQUE (ID)
);
```

SQL NOT NULL Constraint:

- By default, a column can hold NULL values.

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

- This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field

SQL NOT NULL on CREATE TABLE:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

SQL NOT NULL on ALTER TABLE:

```
ALTER TABLE Persons
MODIFY COLUMN Age int NOT NULL;
```

SQL PRIMARY KEY Constraint:

- The PRIMARY KEY constraint uniquely identifies each record in a table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

SQL PRIMARY KEY on CREATE TABLE

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

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

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

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

SQL PRIMARY KEY on ALTER TABLE

```
ALTER TABLE Persons
ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);
```

DROP a PRIMARY KEY Constraint

```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

SQL FOREIGN KEY Constraint:

- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

- A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

- The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

SQL FOREIGN KEY on CREATE TABLE

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

SQL FOREIGN KEY on ALTER TABLE

```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

DROP a FOREIGN KEY Constraint

```
ALTER TABLE Orders
DROP FOREIGN KEY FK_PersonOrder;
```

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 column it will allow 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 on CREATE TABLE

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

SQL CHECK on ALTER TABLE

To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

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

```
ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes'
);
```

DROP a CHECK Constraint

```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

SQL DEFAULT Constraint:

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

SQL DEFAULT on CREATE TABLE

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);
```

SQL DEFAULT on ALTER TABLE

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:

```
ALTER TABLE Persons
ALTER City DROP DEFAULT;
```

SQL CREATE INDEX Statement:

- The CREATE INDEX statement is used to create indexes in tables.
- Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

CREATE UNIQUE INDEX Syntax

Creates a unique index on a table. Duplicate values are not allowed:

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

Note: The syntax for creating indexes varies among different databases. Therefore: Check the syntax for creating indexes in your database.

DROP INDEX Statement

The DROP INDEX statement is used to delete an index in a table

```
ALTER TABLE table_name
DROP INDEX index_name;
```

AUTO INCREMENT Field:

- Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.
- Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

Syntax :

The following SQL statement defines the "Personid" column to be an auto-increment primary key field in the "Persons" table:

```
CREATE TABLE Persons (
    Personid int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (Personid)
);
```

- ✓ MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature.
- ✓ By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.
- ✓ To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:

  ```
  ALTER TABLE Persons AUTO_INCREMENT=100;
  ```
- ✓ To insert a new record into the "Persons" table, we will NOT have to specify a value for the "Personid" column (a unique value will be added automatically):
  - ```
    INSERT INTO Persons (FirstName,LastName)
    VALUES ('Lars','Monsen');
    ```
- ✓ The SQL statement above would insert a new record into the "Persons" table. The "Personid" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".
    ```

SQL Date Data Types:

MySQL comes with the following data types for storing a date or a date/time value in the database:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY

Syntax

```
SELECT * FROM Orders WHERE OrderDate='2008-11-11'
```

SQL CREATE VIEW Statement:

- In SQL, a view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.
- A view is created with the CREATE VIEW statement.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Note: A view always shows up-to-date data! The database engine recreates the view, every time a user queries it

SQL Updating a View

A view can be updated with the CREATE OR REPLACE VIEW statement.

SQL CREATE OR REPLACE VIEW Syntax

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

SQL Dropping a View

A view is deleted with the DROP VIEW statement.

SQL DROP VIEW Syntax

```
DROP VIEW view_name;
```

What is a Stored Procedure?

- A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.
- So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.
- You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

Stored Procedure Syntax

```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;
```

Execute a Stored Procedure

```
EXEC procedure_name;
```

Stored Procedure With One Parameter

The following SQL statement creates a stored procedure that selects Customers from a particular City from the "Customers" table:

EXAMPLE:

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)
AS
SELECT * FROM Customers WHERE City = @City
GO;
```

Execute the stored procedure above as follows:

```
EXEC SelectAllCustomers @City = 'London';
```

Stored Procedure With Multiple Parameters

- Setting up multiple parameters is very easy. Just list each parameter and the data type separated by a comma as shown below.
- The following SQL statement creates a stored procedure that selects Customers from a particular City with a particular PostalCode from the "Customers" table:

EXAMPLE:

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30),
@PostalCode nvarchar(10)
AS
SELECT * FROM Customers WHERE City = @City AND PostalCode =
@PostalCode
GO;
```

Execute the stored procedure above as follows:

```
EXEC SelectAllCustomers @City = 'London', @PostalCode = 'WA1 1DP';
```

SQL Hosting

If you want your web site to be able to store and retrieve data from a database, your web server should have access to a database-system that uses the SQL language.

If your web server is hosted by an Internet Service Provider (ISP), you will have to look for SQL hosting plans.

The most common SQL hosting databases are MS SQL Server, Oracle, MySQL, and MS Access

MySQL

MySQL is also a popular database software for web sites.

MySQL is a very powerful, robust and full featured SQL database system.

MySQL is an inexpensive alternative to the expensive Microsoft and Oracle solutions