**=>CREATE DATABASE** databasename;

**=>DROP DATABASE** databasename;

**=>BACKUP DATABASE** databasename

TO DISK = 'filepath';

**=>CREATE TABLE** Persons (

PersonID int,

LastName varchar(255),FirstName varchar(255),Address varchar(255), City varchar(255)

);

**=>DROP TABLE** table\_name;

**=>TRUNCATE TABLE** table\_name;

**ALTER TABLE** Customers

**ADD** Email varchar(255);

**ALTER TABLE** Customers

**DROP COLUMN** Email;

**ALTER TABLE** table\_name

**ALTER COLUMN** 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.

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 UNIQUE,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int

);

Alter table CustomersCopy

**ADD CONSTRAINT** UC\_CustomerName **UNIQUE** (CustomerName)

**=>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).

Alter table CustomersCopy

**ADD CONSTRAINT** PK\_ID **PRIMARY KEY** (Id)

Alter table CustomersCopy

**drop CONSTRAINT** PK\_ID

**=>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.

The **FOREIGN KEY** constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

CREATE TABLE Orders (

OrderID int NOT NULL **PRIMARY KEY**,

OrderNumber int NOT NULL,

PersonID int **FOREIGN KEY REFERENCES** Persons(PersonID)

);

ALTER TABLE Orders

ADD **FOREIGN KEY** (PersonID) **REFERENCES** Persons(PersonID);

ALTER TABLE Orders

**ADD CONSTRAINT** FK\_PersonOrder

**FOREIGN KEY** (PersonID) **REFERENCES** Persons(PersonID);

ALTER TABLE Orders

**DROP CONSTRAINT** 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.

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int **CHECK** (Age>=18)

);

ALTER TABLE Persons

ADD **CHECK** (Age>=18);

ALTER TABLE Persons

**ADD CONSTRAINT** CHK\_PersonAge **CHECK** (Age>=18 AND City='Sandnes');

ALTER TABLE Persons

**DROP CONSTRAINT** 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.

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

City varchar(255) **DEFAULT** 'Sandnes'

);

ALTER TABLE Persons

ADD CONSTRAINT df\_City

**DEFAULT** 'Sandnes' FOR City;

ALTER TABLE Persons

ALTER COLUMN 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** index\_name

ON table\_name (column1, column2, ...);

**CREATE INDEX** idx\_lastname

ON Persons (LastName);

**CREATE INDEX** idx\_pname

ON Persons (LastName, FirstName);

**DROP INDEX** table\_name.index\_name;

**=>SQL AUTO INCREMENT Field**

The MS SQL Server uses the **IDENTITY** keyword to perform an auto-increment feature.

In the example above, the starting value for IDENTITY is 1, and it will increment by 1 for each new record.

Tip: To specify that the "Personid" column should start at value 10 and increment by 5, change it to **IDENTITY(10,5).**

CREATE TABLE Persons (

Personid int **IDENTITY(1,1)** PRIMARY KEY,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int

);

**=>SQL Date Data Types**

SQL Server 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

**SMALLDATETIME** - format: YYYY-MM-DD HH:MI:SS

**TIMESTAMP** - format: a unique number

**Note**: The date types are chosen for a column when you create a new table in your database!

**=>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** [Brazil Customers] AS

SELECT CustomerName, ContactName

FROM Customers

WHERE Country = 'Brazil';

SELECT \* FROM **[Brazil Customers];**

**CREATE VIEW [Products Above Average Price]** AS

SELECT ProductName, Price

FROM Products

WHERE Price > (SELECT AVG(Price) FROM Products);

SELECT \* FROM **[Products Above Average Price**];

**CREATE OR REPLACE VIEW** [Brazil Customers] AS

SELECT CustomerName, ContactName, City

FROM Customers

WHERE Country = 'Brazil';

**DROP VIEW** view\_name;

SQL Server Data Types

String Data Types

Numeric Data Types

Date and Time Data Types

Other Data Types