The following SQL statement creates a database called "testDB":

CREATE DATABASE testDB;

The following SQL statement drops the existing database "testDB":

DROP DATABASE testDB;

Make sure you have admin privilege before dropping any database. Once a database is dropped, you can check it in the list of databases with the following SQL command: SHOW DATABASES;

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

BACKUP DATABASE databasename  
TO DISK = 'filepath';

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

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

The following SQL statement creates a full back up of the existing database "testDB" to the D disk:

BACKUP DATABASE testDB  
TO DISK = 'D:\backups\testDB.bak';

Always back up the database to a different drive than the actual database. Then, if you get a disk crash, you will not lose your backup file along with the database.

The following SQL statement creates a differential back up of the database "testDB":

BACKUP DATABASE testDB  
TO DISK = 'D:\backups\testDB.bak'  
WITH DIFFERENTIAL;

A differential back up reduces the back up time (since only the changes are backed up).

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

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 following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

CREATE TABLE Persons (  
    PersonID int,  
    LastName varchar(255),  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255)  
);

The PersonID column is of type int and will hold an integer.

The LastName, FirstName, Address, and City columns are of type varchar and will hold characters, and the maximum length for these fields is 255 characters.

A copy of an existing table can also be created using CREATE TABLE.

The new table gets the same column definitions. All columns or specific columns can be selected.

If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

CREATE TABLE new\_table\_name AS  
    SELECT column1, column2,...  
    FROM existing\_table\_name  
    WHERE ....;

The following SQL creates a new table called "TestTables" (which is a copy of the "Customers" table):

CREATE TABLE TestTable AS  
SELECT customername, contactname  
FROM customers;

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

DROP TABLE table\_name;

Be careful before dropping a table. Deleting a table will result in loss of complete information stored in the table!

The following SQL statement drops the existing table "Shippers":

DROP TABLE Shippers;

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

TRUNCATE TABLE table\_name;

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.

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

ALTER TABLE table\_name  
ADD column\_name datatype;

The following SQL adds an "Email" column to the "Customers" table:

ALTER TABLE Customers  
ADD Email varchar(255);

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;

The following SQL deletes the "Email" column from the "Customers" table:

ALTER TABLE Customers  
DROP COLUMN Email;

ALTER TABLE table\_name  
MODIFY column\_name datatype;

ALTER TABLE Persons  
ADD DateOfBirth date;

ALTER TABLE Persons  
ALTER COLUMN DateOfBirth year;

ALTER TABLE Persons  
DROP COLUMN DateOfBirth;

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

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**](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value
* [**UNIQUE**](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different
* [**PRIMARY KEY**](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [**FOREIGN KEY**](https://www.w3schools.com/sql/sql_foreignkey.asp) - Uniquely identifies a row/record in another table
* [**CHECK**](https://www.w3schools.com/sql/sql_check.asp) - Ensures that all values in a column satisfies a specific condition
* [**DEFAULT**](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column when no value is specified
* [**INDEX**](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly

## 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

The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

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

To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:

ALTER TABLE Persons  
MODIFY Age int NOT NULL;

## 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.

## SQL UNIQUE Constraint on CREATE TABLE

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

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

## SQL UNIQUE Constraint on ALTER TABLE

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

ALTER TABLE Persons  
ADD UNIQUE (ID);

To drop a UNIQUE constraint, use the following SQL:

ALTER TABLE Persons  
DROP CONSTRAINT UC\_Person;

## 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 PRIMARY KEY,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int  
);

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)  
);

In the example above there is only ONE PRIMARY KEY (PK\_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).

To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:

ALTER TABLE Persons  
ADD 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:

ALTER TABLE Persons  
ADD CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName);

If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created).

To drop a PRIMARY KEY constraint, use the following SQL:

ALTER TABLE Persons  
DROP CONSTRAINT PK\_Person;

## SQL FOREIGN KEY Constraint

A FOREIGN KEY is a key used to link two tables together.

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

The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Look at the following two tables:

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" 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 on CREATE TABLE

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

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

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

CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    CONSTRAINT FK\_PersonOrder FOREIGN KEY (PersonID)  
    REFERENCES Persons(PersonID)  
);

## SQL FOREIGN KEY on ALTER TABLE

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

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

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

ALTER TABLE Orders  
ADD CONSTRAINT FK\_PersonOrder  
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

To drop a FOREIGN KEY constraint, use the following SQL:

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 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 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)  
);

To allow naming of a CHECK constraint, and for defining a CHECK 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,  
    City varchar(255),  
    CONSTRAINT CHK\_Person CHECK (Age>=18 AND City='Sandnes')  
);

## 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');

To drop a CHECK constraint, use the following SQL:

ALTER TABLE Persons  
DROP CONSTRAINT CHK\_PersonAge;

## SQL DEFAULT Constraint

The DEFAULT constraint is used to provide 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'  
);

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

CREATE TABLE Orders (  
    ID int NOT NULL,  
    OrderNumber int NOT NULL,  
    OrderDate date DEFAULT GETDATE()  
);

## 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  
MODIFY City DEFAULT 'Sandnes';

## DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:

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.

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 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, ...);

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

## CREATE INDEX Example

The SQL statement below creates an index named "idx\_lastname" on the "LastName" column in the "Persons" table:

CREATE INDEX idx\_lastname  
ON Persons (LastName);

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

CREATE INDEX idx\_pname  
ON Persons (LastName, FirstName);

## DROP INDEX Statement

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

DROP INDEX index\_name;

## Syntax for Oracle

In Oracle the code is a little bit more tricky.

You will have to create an auto-increment field with the sequence object (this object generates a number sequence).

Use the following CREATE SEQUENCE syntax:

CREATE SEQUENCE seq\_person  
MINVALUE 1  
START WITH 1  
INCREMENT BY 1  
CACHE 10;

The code above creates a sequence object called seq\_person, that starts with 1 and will increment by 1. It will also cache up to 10 values for performance. The cache option specifies how many sequence values will be stored in memory for faster access.

To insert a new record into the "Persons" table, we will have to use the nextval function (this function retrieves the next value from seq\_person sequence):

INSERT INTO Persons (Personid,FirstName,LastName)  
VALUES (seq\_person.nextval,'Lars','Monsen');

The SQL statement above would insert a new record into the "Persons" table. The "Personid" column would be assigned the next number from the seq\_person sequence. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

## 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 functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

CREATE VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

## SQL CREATE VIEW Examples

The following SQL creates a view that shows all customers from Brazil:

CREATE VIEW [Brazil Customers] AS  
SELECT CustomerName, ContactName  
FROM Customers  
WHERE Country = 'Brazil';

We can query the view above as follows:

SELECT \* FROM [Brazil Customers];

The following SQL creates a view that selects every product in the "Products" table with a price higher than the average price:

CREATE VIEW [Products Above Average Price] AS  
SELECT ProductName, Price  
FROM Products  
WHERE Price > (SELECT AVG(Price) FROM Products);

We can query the view above as follows:

SELECT \* FROM [Products Above Average Price];

## SQL Updating a View

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

### **SQL CREATE OR REPLACE VIEW Syntax**

CREATE OR REPLACE VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

The following SQL adds the "City" column to the "Brazil Customers" view:

CREATE OR REPLACE VIEW [Brazil Customers] AS  
SELECT CustomerName, ContactName, City  
FROM Customers  
WHERE Country = 'Brazil';

## SQL Dropping a View

A view is deleted with the DROP VIEW command.

DROP VIEW view\_name;

The following SQL drops the "Brazil Customers" view:

DROP VIEW [Brazil Customers];

## SQL Injection

SQL injection is a code injection technique that might destroy your database.

SQL injection is one of the most common web hacking techniques.

SQL injection is the placement of malicious code in SQL statements, via web page input.

## SQL in Web Pages

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will **unknowingly** run on your database.

Look at the following example which creates a SELECT statement by adding a variable (txtUserId) to a select string. The variable is fetched from user input (getRequestString):

txtUserId = getRequestString("UserId");  
txtSQL = "SELECT \* FROM Users WHERE UserId = " + txtUserId;

## 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.

The data type of a column defines what value the column can hold: integer, character, money, date and time, binary, and so on.

## SQL Data Types

Each column in a database table is required to have a name and a data type.

An SQL developer must decide what type of data that will be stored inside each column when creating a table. The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

Data types might have different names in different database. And even if the name is the same, the size and other details may be different! **Always check the documentation!**

SQL Server Data Types

String data types:

|  |  |  |  |
| --- | --- | --- | --- |
| Data type | Description | Max size | Storage |
| char(n) | Fixed width character string | 8,000 characters | Defined width |
| varchar(n) | Variable width character string | 8,000 characters | 2 bytes + number of chars |
| varchar(max) | Variable width character string | 1,073,741,824 characters | 2 bytes + number of chars |
| text | Variable width character string | 2GB of text data | 4 bytes + number of chars |
| nchar | Fixed width Unicode string | 4,000 characters | Defined width x 2 |
| nvarchar | Variable width Unicode string | 4,000 characters |  |
| nvarchar(max) | Variable width Unicode string | 536,870,912 characters |  |
| ntext | Variable width Unicode string | 2GB of text data |  |
| binary(n) | Fixed width binary string | 8,000 bytes |  |
| varbinary | Variable width binary string | 8,000 bytes |  |
| varbinary(max) | Variable width binary string | 2GB |  |
| nvarchar | Variable width Unicode string | 4,000 characters |  |
| image | Variable width binary string | 2GB |  |

Numeric data types:

|  |  |  |
| --- | --- | --- |
| Data type | Description | Storage |
| bit | Integer that can be 0, 1, or NULL |  |
| tinyint | Allows whole numbers from 0 to 255 | 1 byte |
| smallint | Allows whole numbers between -32,768 and 32,767 | 2 bytes |
| int | Allows whole numbers between -2,147,483,648 and 2,147,483,647 | 4 bytes |
| bigint | Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807 | 8 bytes |
| decimal(p,s) | Fixed precision and scale numbers.  Allows numbers from -10^38 +1 to 10^38 –1.  The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18.  The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0 | 5-17 bytes |
| numeric(p,s) | Fixed precision and scale numbers.  Allows numbers from -10^38 +1 to 10^38 –1.  The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18.  The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0 | 5-17 bytes |
| smallmoney | Monetary data from -214,748.3648 to 214,748.3647 | 4 bytes |
| money | Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807 |  |
| float(n) | Floating precision number data from -1.79E + 308 to 1.79E + 308.  The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53. | 4 or 8 bytes |
| real | Floating precision number data from -3.40E + 38 to 3.40E + 38 | 4 bytes |

Date and Time data types:

|  |  |  |
| --- | --- | --- |
| Data type | Description | Storage |
| datetime | From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds | 8 bytes |
| datetime2 | From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds | 6-8 bytes |
| smalldatetime | From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute | 4 bytes |
| date | Store a date only. From January 1, 0001 to December 31, 9999 | 3 bytes |
| time | Store a time only to an accuracy of 100 nanoseconds | 3-5 bytes |
| datetimeoffset | The same as datetime2 with the addition of a time zone offset | 8-10 bytes |
| timestamp | Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable |  |

Other data types:

|  |  |
| --- | --- |
| Data type | Description |
| sql\_variant | Stores up to 8,000 bytes of data of various data types, except text, ntext, and timestamp |
| uniqueidentifier | Stores a globally unique identifier (GUID) |
| xml | Stores XML formatted data. Maximum 2GB |
| cursor | Stores a reference to a cursor used for database operations |
| table | Stores a result-set for later processing |