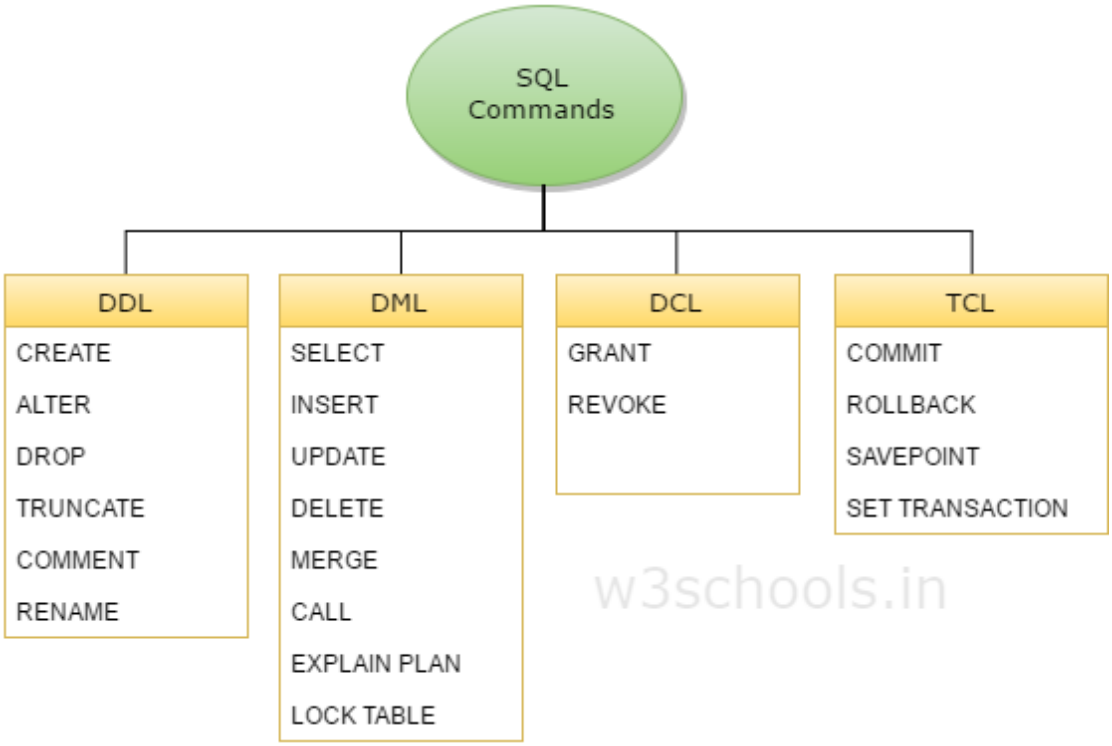
All SQL Server Commands PART ONE



DDL

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

Syntax:

CREATE DATABASE databasename;

Example

CREATE DATABASE testDB;

2- ALTER : 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.

A. ALTER TABLE - ADD Column

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

Syntax:

ALTER TABLE table_name

ADD column_name datatype;

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

Example

ALTER TABLE Customers

ADD Email varchar(255);

B. ALTER TABLE - DROP Column

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

Syntax:

ALTER TABLE *table_name*
DROP COLUMN *column_name*;

Example:

ALTER TABLE Customers
DROP COLUMN Email;

C. ALTER TABLE - RENAME Column

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

ALTER TABLE *table_name*
RENAME COLUMN *old_name* to *new_name*;

D. ALTER TABLE – ALTER/MODIFY DATATYPE

To change the data type of a column in a table, use the following syntax:

ALTER TABLE *table_name*
ALTER COLUMN *column_name datatype*;

3- DROP : is used to drop an existing table in a database.

Syntax
DROP TABLE table_name;

Example
DROP TABLE Shippers;

DML

1. **SELECT : used to select data from a database.**

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

Syntax

SELECT column1, column2, ...
FROM table_name;

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;
Example
SELECT CustomerName, City FROM Customers;

2. **INSERT :used to insert new records in a table.**

INSERT INTO Syntax

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

The first way specifies both the column names and the values to be inserted:

INSERT INTO table_name (column1, column2, column3(... ,

VALUES (value1, value2, value3;(... ,

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. 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:

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country(

VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

3. UPDATE : used to modify the existing records in a table.

UPDATE Syntax

UPDATE table_name SET column1 = value1, column2 = value2, ...
WHERE condition;

Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

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;

4. DELETE : used to delete existing records in a table.

DELETE Syntax

DELETE FROM table_name WHERE condition;

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

DELETE Example

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

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

The SQL WHERE Clause

The WHERE clause is used to filter records.
It is used to extract only those records that fulfill a specified condition.

Syntax

SELECT *column1, column2, ...*
FROM *table_name*
WHERE *condition*;

Example

Select all customers from Mexico:

SELECT * FROM Customers
WHERE Country='Mexico';

SQL MIN() and MAX() Functions

The SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

MIN() Syntax

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

MAX() Syntax

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

SQL NOT NULL Constraint

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

EXAMPLE: 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
ALTER 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 PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

SQL PRIMARY KEY on ALTER TABLE

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);
```

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.

Look at the following two tables:

Persons Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

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 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.

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)
);
```