V-Ex Tech

V-EX TECH

Web Development

Java / Node.js / PHP / .Net / Python

Certification Course

Assured Placement Program

With International Certificate

About V-Ex Tech….

V-Ex Tech is an elevated education platform providing rigorous industry-relevant programs Designed and delivered on collaboration with industry professionals. It has been constantly Into process of creating an immersive learning experience binding latest technologies, pedagogy and services with enormous job placement opportunities too.

V-Ex Tech

SQL -STRUCTURE QUERY LANGUAGE

What is SQL?

- SQL stands for Structured Query Language
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database

QUERIES

`SELECT`, `UPDATE`, `DELETE`, `INSERT`, `WHERE`

DBMS AND RDBMS

SYNTEX

```
SELECT * FROM Customers;
```

- SQL keywords are NOT case sensitive: `select` is the same as `SELECT`

INSTALLATION OF SQL

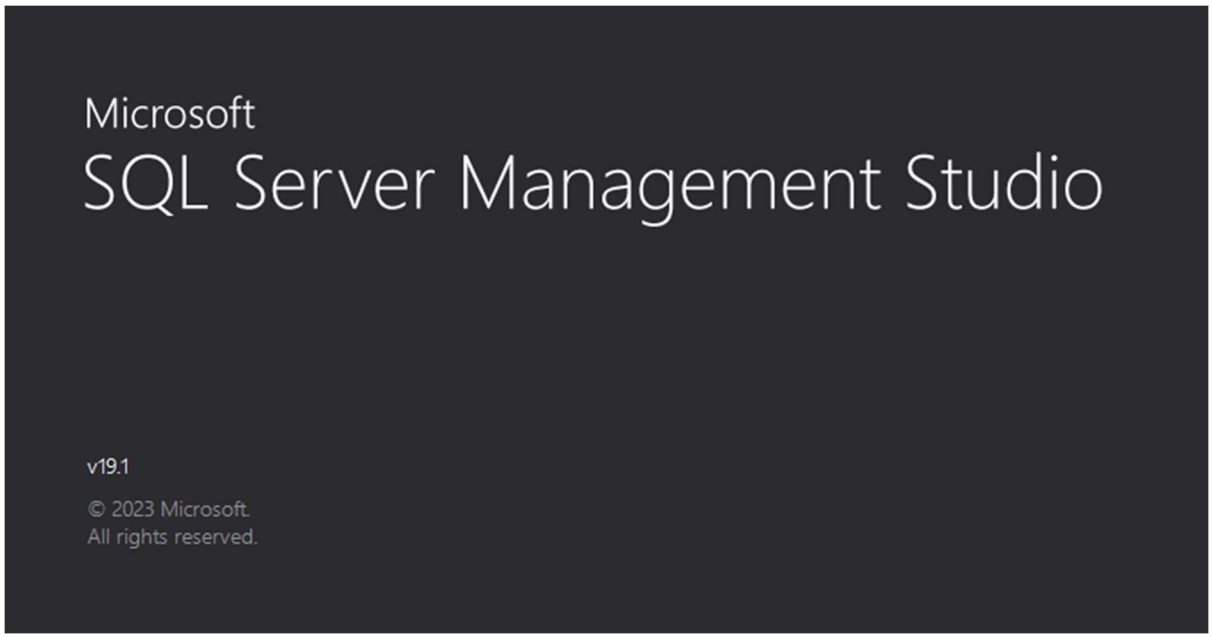
WATCH VIDEO:
https://www.youtube.com/watch?v=do6OOASiycM&list=PLfHFR4U0QuQKIVMPgf2FR8PxmMXk3WsCk

STEP-1:SEARCH MISCROSOFT SQL SERVER

CLICK ON EXPRESS MODE

STEP-2 -DOWNLOAD SSMS SERVER

V-Ex Tech



CREATE DATABASE

CREATE TABLE

PRIMARY KEY & FOREIGN KEY

FIND TABLE DATA:

SELECT * FROM TABLENAME

INSERT DATA

INSERT INTO TABLENAME (COLUMN) VALUES (‘ ‘)

SELECT Statement

```
SELECT column1, column2, ...
FROM table_name;
```

```
SELECT CustomerName, City FROM Customers;
```

SELECT DISTINCT Statement

SELECT DISTINCT Syntax

The `SELECT DISTINCT` statement is used to return only distinct (different) values.

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

INSERT INTO Statement

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

WHERE Clause

The `WHERE` clause is used to filter records.

Amazon filter

WHERE Syntax

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

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

AND, OR and NOT Operators

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

```
SELECT * FROM Customers
WHERE Country='Germany' AND City='Berlin';
```

OR Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

```
SELECT * FROM Customers
WHERE City='Berlin' OR City='München';
```

NOT Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

```
SELECT * FROM Customers
WHERE NOT Country='Germany';
```

ORDER BY Keyword

ORDER BY Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Example

```
SELECT * FROM Customers
ORDER BY Country;
```

DESC Example

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

```
SELECT * FROM Customers
ORDER BY Country, CustomerName;
```

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

NULL Values

We will have to use the `IS NULL` and `IS NOT NULL` operators instead.

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

UPDATE Statement

UPDATE Syntax

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

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

SQL DELETE Statement

DELETE Syntax

```
DELETE FROM table_name WHERE condition;
```

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

Delete All Records

```
DELETE FROM table_name;
```

```
DELETE FROM Customers;
```

SELECT TOP Clause

The SELECT TOP clause is used to specify the number of records to return.

Syntax

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE condition;
```

```
SELECT TOP 3 * FROM Customers;
```

```
SELECT TOP 3 * FROM Customers
WHERE Country='Germany';
```

Aggrecate Function

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;
```

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

MAX() Syntax

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

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

COUNT(), AVG() and SUM() Functions

COUNT() Syntax

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

```
SELECT COUNT(ProductID)
FROM Products;
```

The `AVG()` function returns the average value of a numeric column.

AVG() Syntax

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

```
SELECT AVG(Price)
FROM Products;
```

The SUM() function returns the total sum of a numeric column.

SUM() Syntax

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

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

LIKE Operator

SQL LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

LIKE Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%or%';
```

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

```
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a__%';
```

```
SELECT * FROM Customers
WHERE ContactName LIKE 'a%o';
```

```
SELECT * FROM Customers
WHERE CustomerName NOT LIKE 'a%';
```

Wildcards

Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

Using the % Wildcard

```
SELECT * FROM Customers
WHERE City LIKE 'ber%';
```

```
SELECT * FROM Customers
WHERE City LIKE '[bsp]%';
```

The following SQL statement selects all customers with a City starting with "a", "b", or "c":

Example

```
SELECT * FROM Customers
WHERE City LIKE '[a-c]%';
```

```sql
SELECT * FROM Customers
WHERE City LIKE '[!bsp]%';
```

IN Operator

IN Syntax

```sql
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Example

```sql
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

```sql
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

```sql
SELECT * FROM Customers
WHERE Country IN (SELECT Country FROM Suppliers);
```

BETWEEN Operator

BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
```

```
SELECT * FROM Products
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

```
SELECT * FROM Products
WHERE ProductName NOT BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di
Giovanni'
ORDER BY ProductName;
```

Aliases

```
SELECT column_name AS alias_name
FROM table_name;
```

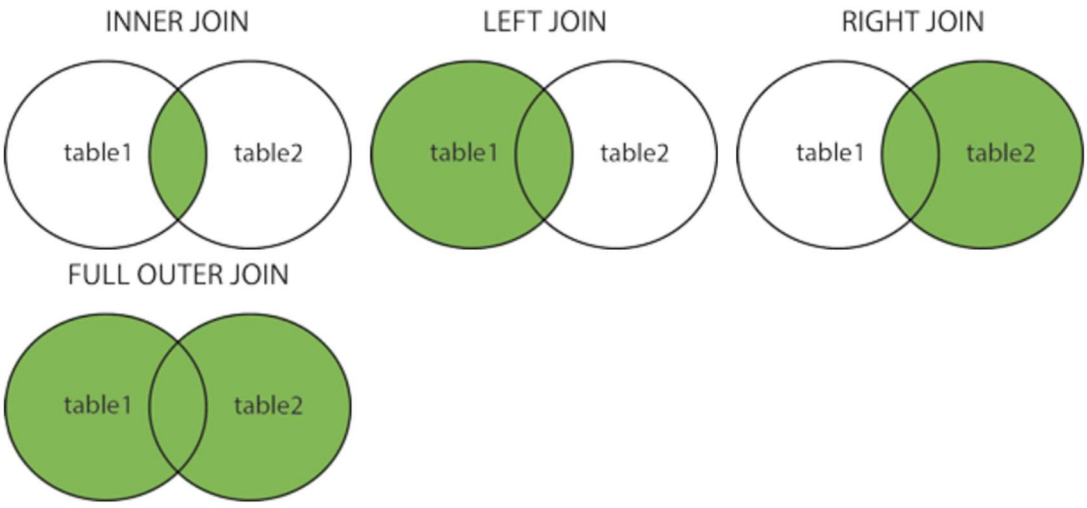
```
SELECT column_name(s)
FROM table_name AS alias_name;
```

```
SELECT CustomerID AS ID, CustomerName AS Customer
FROM Customers;
```

```
SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ', ' +
Country AS Address
FROM Customers;
```

Joins

- `(INNER) JOIN`: Returns records that have matching values in both tables
- `LEFT (OUTER) JOIN`: Returns all records from the left table, and the matched records from the right table
- `RIGHT (OUTER) JOIN`: Returns all records from the right table, and the matched records from the left table
- `FULL (OUTER) JOIN`: Returns all records when there is a match in either left or right table



INNER JOIN

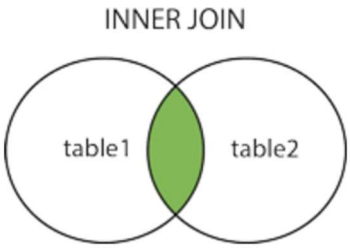
INNER JOIN Syntax

```
SELECT column_name
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

JOIN Three Tables

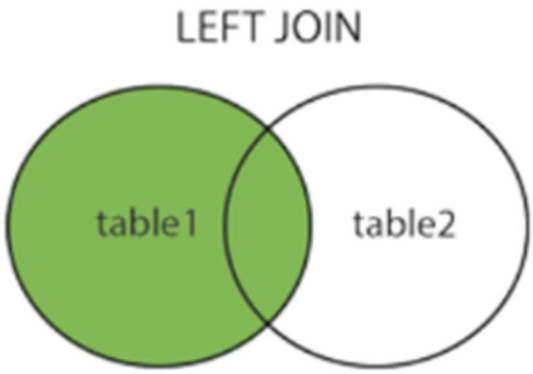
```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```



LEFT JOIN

LEFT JOIN Syntax

```
SELECT column_name
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

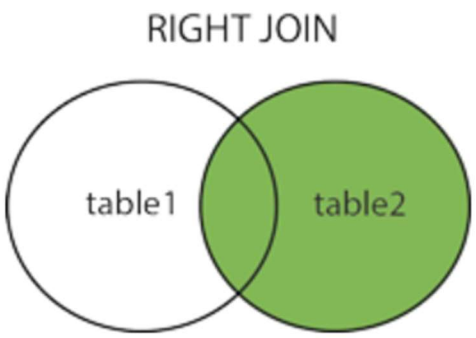



```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

RIGHT JOIN

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```



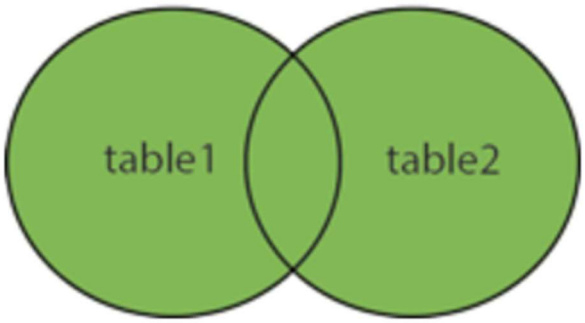

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

FULL OUTER JOIN

FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```



Self Join

Self Join Syntax

```
SELECT column_name
FROM table1 T1, table1 T2
WHERE condition;
```

```
SELECT A.CustomerName AS CustomerName1,
B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

UNION Operator

When two column name same

If some customers or suppliers have the same city, each city will only be listed once, because UNION selects only distinct values.

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

UNION Syntax

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

UNION ALL Syntax

The `UNION` operator selects only distinct values by default. To allow duplicate values, use `UNION ALL`:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

```sql
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

GROUP BY Statement

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

GROUP BY Syntax

```sql
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

```sql
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

```sql
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

```sql
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM
Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

HAVING Clause

HAVING Syntax

```sql
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

```sql
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

SQL EXISTS Operator

The EXISTS operator returns TRUE if the subquery returns one or more records.

EXISTS Syntax

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID =
Suppliers.supplierID AND Price < 20);
```

ANY and ALL Operators

The ANY operator:

- returns a boolean value as a result
- returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

The *operator* must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

SQL ALL Operator

The `ALL` operator:

- returns a boolean value as a result
- returns TRUE if ALL of the subquery values meet the condition
- is used with `SELECT`, `WHERE` and `HAVING` statements

```
SELECT ALL column_name(s)
FROM table_name
WHERE condition;
```

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity > 99);
```

```
SELECT ALL ProductName
FROM Products
WHERE TRUE;
```

SELECT INTO Statement

SELECT INTO Syntax

Copy all columns into a new table:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

```
SELECT * INTO CustomersBackup2017
FROM Customers;
```

The following SQL statement uses the IN clause to copy the table into a new table in another database:

```
SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'
FROM Customers;
```

Copy only some columns into a new table:

```
SELECT column1, column2, column3, ...
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

The following SQL statement copies only a few columns into a new table:

```
SELECT CustomerName, ContactName INTO CustomersBackup2017
FROM Customers;
```

```
SELECT * INTO CustomersGermany
FROM Customers
WHERE Country = 'Germany';
```

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.

The INSERT INTO SELECT statement requires that the data types in source and target tables match.

INSERT INTO SELECT Syntax

Copy all columns from one table to another table:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

INSERT INTO SELECT Statement

Copy only some columns from one table into another table:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers;
```

```
INSERT INTO Customers (CustomerName, ContactName, Address, City,
PostalCode, Country)
SELECT SupplierName, ContactName, Address, City,
PostalCode, Country FROM Suppliers;
```

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers
WHERE Country='Germany';
```

CASE Expression

CASE Syntax

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

Below is a selection from the "OrderDetails" table in the Northwind sample database:

OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN 'The quantity is greater than 30'
    WHEN Quantity = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

Example

```
SELECT CustomerName, City, Country
FROM Customers
ORDER BY
(CASE
    WHEN City IS NULL THEN Country
    ELSE City
END);
```

ISNULL()Functions

P_Id	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder
1	Jarlsberg	10.45	16	15
2	Mascarpone	32.56	23	
3	Gorgonzola	15.67	9	20

```
select isnull(CreateBy,0) from PersonalDetails
```

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.

Stored Procedure Syntax

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

Execute a Stored Procedure

```
EXEC procedure_name;
```

Example

```
CREATE PROCEDURE SelectAllCustomers
AS
SELECT * FROM Customers
GO;


EXEC SelectAllCustomers;
```

Stored Procedure With One Parameter

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


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

Stored Procedure With Multiple Parameters

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

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

Comments

Single Line Comments

```
--Select all:
SELECT * FROM Customers;
```

Multi-line Comments

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```

```
SELECT CustomerName, /*City,*/ Country FROM Customers;
```

Operators

Operator	Description	Example
+	Add	Try it
-	Subtract	Try it
*	Multiply	Try it
/	Divide	Try it
%	Modulo	Try it

Bitwise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

SQL Comparison Operators

Operator	Description	Example
=	Equal to	Try it

>	Greater than	Try it
<	Less than	Try it
>=	Greater than or equal to	Try it
<=	Less than or equal to	Try it
<>	Not equal to	Try it

CREATE DATABASE Statement

```
CREATE DATABASE databasename;
```

```
CREATE DATABASE testDB;
```

BACKUP DATABASE for SQL Server

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';
```

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

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,
   ....
);
```

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

Create Table Using Another Table

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

Syntax

```sql
CREATE TABLE new_table_name AS
    SELECT column1, column2,...
    FROM existing_table_name
    WHERE ....;
```

```sql
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;
```

DROP TABLE Statement

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

```
DROP TABLE table_name;
```

```
DROP TABLE Shippers;
```

DROP DATABASE Statement

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

```
DROP DATABASE databasename;
```

```
DROP DATABASE testDB;
```

TRUNCATE TABLE

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

```
TRUNCATE TABLE table_name;
```

ALTER TABLE Statement

The `ALTER TABLE` statement is used to add, delete, or modify columns in an existing table.

ALTER TABLE - ADD Column

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

```
ALTER TABLE Customers
ADD Email varchar(255);
```

```
ALTER TABLE Persons
ADD DateOfBirth date;
```

ALTER TABLE - DROP COLUMN

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

```
ALTER TABLE Customers
DROP COLUMN Email
```

ALTER TABLE - RENAME COLUMN

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

ALTER TABLE - ALTER/MODIFY DATATYPE

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```

```
ALTER TABLE Persons
ALTER COLUMN DateOfBirth year;
```

SQL Constraints

- 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

V-Ex Tech

NOT NULL on CREATE TABLE

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

NOT NULL on ALTER TABLE

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

SQL UNIQUE

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

PRIMARY KEY

The PRIMARY KEY constraint uniquely identifies each record in a table.

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

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

DROP a PRIMARY KEY Constraint

```
ALTER TABLE Persons
DROP CONSTRAINT PK_Person;
```

FOREIGN KEY Constraint

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

SQL FOREIGN KEY on CREATE 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
DROP CONSTRAINT FK_PersonOrder;
```

CHECK on CREATE TABLE

```
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);
```

DEFAULT on CREATE TABLE

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

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

AUTO INCREMENT Field

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

```sql
CREATE TABLE Persons (
    Personid int IDENTITY(1,1) PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

Date Data Types

- 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

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

CREATE VIEW Statement

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.

CREATE VIEW Syntax

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

```
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 * FROM [Products Above Average Price];
```

SQL Updating a View

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

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

```
DROP VIEW view_name;
```

```
DROP VIEW [Brazil Customers];
```

MySQL Functions

https://www.w3schools.com/sql/sql_ref_mysql.asp

How to Calculate Age From Date of Birth in SQL?

```
Create function fn_AgeCalc(@birthDate date)
returns int
as
Begin
    Declare @age int
    Set @age = datediff(yy,@birthDate,getdate())
    Return @age
End
```

```
Select dbo.fn_AgeCalc('12-05-1990')
```

Result: 20

V-Ex Tech