<>-not equal to sign

**Types of Sql Statements**

SQL commands are mainly categorized into four categories as:

1. DDL – Data Definition Language(create,drop,alter,truncate,rename,comment)
2. DQl – Data Query Language
3. DML – Data Manipulation Language
4. DCL – Data Control Language



**TRUNCATE**

TRUNCATE: This is used to remove all records from a table, including all spaces allocated for the records are removed.

truncate table GraduatedStudents

Now all the records are removed. Only the structure of the tables is preserved.

**Creating a table**

Create database databasename;

create table students(

student\_id int primary key,

name varchar(20),

major varchar(20)

)

Here we gotta set primary keys manually which is not right:

insert into students values(1,'Vako',20);

insert into students values(2,'Vako',20);

But with identity keyword it sets that automatically and starts from zero.

create table people(

person\_id int primary key identity,

name varchar(20),

age int

)

So we would just do:

insert into people values('Vako',20);

insert into people values('Lala',25);

That’s it!

**Comments**

--our comment or for multiline comments we use /\* \*/

**Alter a table(adding a new column)**

if we want to change a table we use alter keyword!

If we forgot to add a column then we can use ‘alter’ keyword:

Alter table tableName add columnName varchar(16)

alter table students add gpa int

then we use ‘update’ keyword to set the values because they are null.

Update tableName set column1=value1, column2=value2

where condition

update students set gpa=7 where student\_id=3

**SQL constraints (NOT NULL, UNIQUE)**

If we wanna add a required column meaning that it cannot be null then we use **not null** attribute:

create table people(

id int primary key identity,

[name] varchar(16) not null,

age int unique

)

So now there can’t be equal ages, and name cannot be omitted however it can be an empty string.

insert into people values('Vahid',20)

insert into people values('Hey',20)

Violation of UNIQUE KEY constraint

We can also add contsraints as shown below

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')  
);

**Check constraint**

The CHECK constraint is used to limit the value range that can be placed in a column.

create table Students(

Id int primary key identity,

Name varchar(16) check(name like '%a%')

)

insert into Students values('Hey')

The INSERT statement conflicted with the CHECK constraint "CK\_\_Students\_\_Name\_\_3A81B327".

It has to contain an a in it!

create table Students(

Id int primary key identity,

Name varchar(16) check(name like '%a%'),

Age int check(Age>1 and Age<10)

)

insert into Students values('Hay',9)

**Naming Convetion for Constraints**

PK\_TableName\_ColumnName -- primary key constraint FK\_TableName\_ColumnName - foreign key constraint CK\_TableName\_ColumnName - check constraint

UQ - for unique constraint

DF - for default constraint

**Deleting a record(row) from a table**

We can use just **delete** keyword here:

delete from people where [name]=''

**Foreign key**

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

create table pets(

pet\_id int primary key identity,

name varchar(16) not null

)

alter table people add pet\_id int foreign key references pets(pet\_id)

**Horizontal Joins**

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Inner join -  selects 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 (if left table has more fields then right table’s fields will be set to null)

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



select \*

from people

inner join pets

on people.pet\_id=pets.pet\_id

Self Join-A self join is a regular join, but the table is joined with itself. it is only used in self-referencing relationships. We specify it by using simple inner, right or left joins.

create table Students(

Id int primary key identity,

Name varchar(16)

)

alter table Students add Dependency\_id int references Students(id)

select d.Name,d.Dependency\_id,s.Name as 'Depends on' from Students as s inner join Students d on s.Id=d.Dependency\_id



Non-equal join-When you join two tables using other conditional operators, beyond the equal sign, non-equil JOINs come into play. Comparison operators, like <, >, <=, >=, !=, and <> and the BETWEEN operator work perfectly for joining tables in SQL.

create table Students(

Id int primary key identity,

Name varchar(16),

Age int

)

create table GraduatedStudents(

Id int primary key identity,

Name varchar(16),

Age int

)

insert into Students values('Vahid',20),('Nihad',25),('Azer',18),('Lala',35)

insert into GraduatedStudents values('Zakir',50),('Ferid',15),('Leyla',28),('Vaqif',45)

select \* from Students inner join GraduatedStudents on Students.Id=GraduatedStudents.Id and GraduatedStudents.Age>30



Sql starts checking like this-> first it starts checkhing the first record of GraduatedStudents id if it is equal to the first Students id and GraduatedStudents Age is greater than 30 then it goes to the second record and checks the same GraduatedStudents id with the Students id and the age if it returns true for all then it selects the same GraduatedStudents again

create table Students(

Id int primary key identity,

Name varchar(16),

Grade int

)

create table Marks(

Id int primary key identity,

Letter varchar(1),

MinGrade int,

MaxGrade int,

)

insert into Marks values('A',91,100),('B',81,90),('C',71,80),('D',61,70)

insert into Students values('Zakir',70),('Ferid',95),('Leyla',78),('Vaqif',85),('Vahid',100)

select \* from Students join Marks on Students.Grade between Marks.MinGrade and Marks.MaxGrade



Cross join-It takes all the records from the table on the left and pairs every single record of it with the table on the right. It is useful when we want to compare all the records on the left table with every single row on the right table.

create table A(

Id int primary key identity,

Name varchar(16)

)

create table B(

Id int primary key identity,

Name varchar(16)

)

insert into A values('A1')

insert into B values('B1'),('B2'),('B3')

select A.Name A ,B.Name B from A cross join B



**Multi Joins**

create table Students(

Id int primary key identity,

Name varchar(16),

Grade int

)

create table Groups(

Id int primary key identity,

Name varchar(16),

)

create table GroupStudents(

Id int primary key identity,

Group\_id int foreign key references [Groups](Id),

Student\_id int foreign key references Students(Id)

)

insert into Students values('Zakir',70),('Ferid',95),('Leyla',78),('Vaqif',85),('Vahid',100)

insert into Groups values('P319'),('P320')

insert into GroupStudents values (1,1), (1,2), (1,3), (2,4), (2,5)

select s.name,g.Name as Groups from Students s join GroupStudents on s.Id=GroupStudents.Student\_id join Groups g on g.Id=GroupStudents.Group\_id



**Deleting a field (column)**

In order to delete an entire column, we use “drop” keyword.

alter table people add [name] nvarchar(20)

alter table people drop column [name]

**Group by (clause)**

For instance, we have got people from the same country and now we can use group by clause to select all the people from the same country.

If there is a cloumn and aggregate function up in our select statement then we can’t select like this->

select name, count(\*) from Students

Column 'Students.name' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

So here we gotta use group by clause.

select country, count(\*) from Students group by country

select country, name,count(\*) from Students group by country,name

Here it first groups them by country and when there are two identical country names it groups them by their names

America Jeck 1

Cacusus Lala 1

Azerbaijan Nihad 1

Azerbaijan Vahid 1

After group by clause, if we wanna filter it then we have to use having clause.

select country,name,count(\*) from Students group by country,name having country='Azerbaijan'

We can use where clause only before group by clause!

select country,name,count(\*) from Students where country='Azerbaijan' group by country,name

select CustomerName, SUM(Amount) as Expenditure from Orders group by CustomerName order by CustomerName

**Copying values from another table**

insert into GraduatedStudents select name,surname,phoneNumber,score,comment from Students where isGraduated=1

**Vertical Joins:** **Union, Union All, Intersect, Except**

1. **UNION**: Combine two or more result sets into a single set, without duplicates.
2. **UNION ALL**: Combine two or more result sets into a single set, including all duplicates.
3. **INTERSECT**: Takes the data that is in both tables
4. **EXCEPT**: Takes the data that is in the first table but not in the second

select \* from UnGraduatedStudents

select \* from GraduatedStudents



select \* from UnGraduatedStudents

union

select \* from GraduatedStudents



select \* from UnGraduatedStudents

union all

select \* from GraduatedStudents



select \* from UnGraduatedStudents

intersect

select \* from GraduatedStudents



select \* from UnGraduatedStudents

Except

select \* from GraduatedStudents



**Create Index**

The CREATE INDEX statement is used to create indexes in tables for columns. 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.

The search is implemented by binary search method.

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, ...);

DROP INDEX index\_name ON table\_name;

create table Students(

Id int primary key identity,

Name varchar(16)

)

create index idx\_Student\_Names on Students(Name)

select Name from Students

**Procedures**

There are 2 types of stored procedures: system procedures and user defined procedures.

create procedure GetStudentsInfo

As

select s.name,g.Name as Groups from Students s join GroupStudents gs on s.Id=gs.Student\_id join Groups g on g.Id=gs.Group\_id

create procedure GetStudentsById @Id int

As

begin

select \* from Students where Id=@Id

end

exec GetStudentsById 2

or we can execute it by not typing the exec keyword GetStudentsById 2

drop procedure GetStudentsById

If we want to pass a table name then it is going to be a string and note that we can’t use strings in select statements so we gotta use the entire select statement as a string then we need to parse it by executing sp\_executesql procedure.

Remember that sys.sp\_executesql only takes nvarchar!

create Procedure GetStudents @Table varchar(100),@GivenAge int

As

begin

declare @sqlquery nvarchar(200)

set @sqlquery='select \* from '+@Table+' where Age>'+str(@GivenAge)+' or Age<'+str(@GivenAge)

exec sys.sp\_executesql @sqlquery

end

exec GetStudents 'People',10

**Views**

In SQL, a view is a virtual table. If we have got an enormous select statement then we use vews.

create View v\_GetStudentsInfo

As

select s.name,g.Name as Groups from Students s join GroupStudents gs on s.Id=gs.Student\_id join Groups g on g.Id=gs.Group\_id

select \* from v\_GetStudentsInfo

If we don’t need to pass an argument or anything wide like this then we can just use views instead of procedures.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

drop view v\_GetStudentsInfo

The naming convention for view is as follows 🡪 v\_viewName

As keyword is used in sql to show the body of the view.

**Functions**

There are 2 types of user defined functions: Table valued functions and scalar valued functions. Scalar valued functions return either int, varchar(n), bit. Table valued functions return a table.

The main differences between a function and procedure are that in functions you always have to return a value however in procedures you can both return or not return a value. Also you can’t execute a procedure in a function however you can call a function in a procedure. In functions, you can only use select statement but in procedures you can use insert, update and etc. One advantage of using functions is that you can call a function in a select statement but you cannot use procedures in a select statement.

create Function GetPerson(@Name varchar(10))

returns int

as

begin

declare @Count int

select @count=Sum(Age) from People where Name=@Name

return @count

end

select dbo.GetPerson('Vahid')

drop function GetPerson

**Triggers**

A SQL trigger is a database object which fires when an event occurs in a database.

create trigger ShowAfterInsert

on Products

after insert

as

begin

select \* from Products

end

insert into Products values('Solfetka'), ('Rucka')

drop trigger ShowAfterInsert

DML trigger statements use two special tables: the deleted table and the inserted tables. SQL Server automatically creates and manages these tables. You can use these temporary, memory-resident tables to test the effects of certain data modifications and to set conditions for DML trigger actions.

create trigger AfterDelete on Products

after delete

as

begin

insert into DelProducts(Name) select Name from deleted where Id in (select Id from deleted)

end

Now when we delete something it is first gonna insert the name of the deleted item in the DelProducts table and then it will delete it.

We can also use “instead of delete”

create trigger InsertInsteadOfDelete on Products

instead of delete

as

begin

insert into DelProducts(Name) select Name from deleted where Id in (select Id from deleted)

end

**Postgres**

In Postgress shell, some common commands:

\l to list all the databases

\c or \connect to connect to a database

\dt to list all the tables

\d “Stories” to list all the info (column and etc) of a table

Select \* from “Stories”;

When we are in the postgres container we need to get into the postgres database🡺 psql –U user-name

We must use semicolon at the end of our query to tell the shell that it is the end of our query and also double quotes are required if we include capital letters in our table name in postgres.

**Run psql**

When we run postgres we need to specify the password🡺

docker run –name mypsql –e POSTGRES\_PASSWORD=password postgres

**Restore psql database**

First we need to be in D:\PostgreSQL\14\bin folder to use this command 🡺 pg\_dump -U user -d db\_name -h 127.0.0.1 > dump.sql

dump.sql is created by the psql shell. We can also exclude host part.

Then we can seed our new database from this dump.sql file 🡺

cat dump.sql | docker exec -i <container-name> psql -U <user> -d <database> Here everything that was in the dump.sql file was copied to the database.