

Debre Markos University
Institute of Technology
Department of Software Engineering

Microsoft SQL Server 2012 Lab Manual-One

December 25, 2018

Debre Markos, Ethiopia

DDL (Data Definition Language)

Let's see creating, renaming database, creating, renaming, structure, altering, dropping and truncating tables using SQL server.

How to Create a database

To create a database in SQL, use the following formula:

Syntax: `CREATE DATABASE DatabaseName`

Example

```
create database RegistrarDB
use RegistrarDB
```

If you want the name of the database to be in different words, include them in square brackets. Here is an example:

```
CREATE DATABASE [Registrar DB];
```

Renaming a database

- 📄 To change the name of a database, Transact-SQL provides `sp_renamedb`.
- 📄 The formula used would be:

```
EXEC sp_renamedb 'ExistingName', 'NewName';
```

- 📄 The `EXEC sp_renamedb` expression is required.
- 📄 The ExistingName factor is the name of the database that you want to rename.
- 📄 The NewName factor is the name you want the database to have after renaming it. Here is an example of renaming a database:

```
Example: EXEC sp_renamedb 'MyDatabase', 'MyDb';
```

How to create table

To create a table, you can follow this formula/syntax:

```
CREATE TABLE TableName (Column1, Column2, Column3)
```

Or:

```
CREATE TABLE TableName (, Column2, Column3);
```

Example:

```
use RegistrarDB
create table Studenttbl
(Sid varchar(20) primary key, Studname varchar(50) not null, StudCollege
varchar(20) not null, StudAge int not null, StudGender varchar(6) not null);

create table Coursetbl
(Ccode varchar(20) primary key, CTitle varchar(50) not null,
CCredit int not null);

create table CourseRegistrationtbl
(RegID varchar(20) primary key,
Sid varchar(20) foreign key references Studenttbl(Sid) ,
CCode varchar(20) foreign key references Coursetbl(Ccode),
RegDate datetime not null, StudYear varchar(10) not null,
StudSemister varchar(10) not null);

create table Resulttbl(ResultID int IDENTITY(1,1) primary key,
Sid varchar(20) foreign key references Studenttbl(Sid),
CCode varchar(20) foreign key references Coursetbl(Ccode),
LetterGrade varchar(10) not null );

create table StudStatustbl(StatusID int IDENTITY(1,1) primary key, Sid
varchar(20) foreign key references Studenttbl(Sid) ,
SGPA float not null, CGPA float not null, StudStatus varchar(20));
```

SQL UNIQUE Constraint on CREATE TABLE

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

```
CREATE TABLE Persons
(
PId int NOT NULL UNIQUE,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
```

```
Address varchar(255),  
City varchar(255)
```

OR

```
CREATE TABLE Persons  
(  
  PId int NOT NULL,  
  LastName varchar(255) NOT NULL,  
  FirstName varchar(255),  
  Address varchar(255),  
  City varchar(255),  
  CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName)  
)
```

SQL UNIQUE Constraint on ALTER TABLE

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

```
ALTER TABLE Persons ADD UNIQUE (P_Id)
```

To DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

```
ALTER TABLE Persons DROP CONSTRAINT uc_PersonID
```

SQL DEFAULT Constraint

The DEFAULT constraint is used to insert a default value into a column.

The default value will be added to all new records, if no other value is specified.

SQL DEFAULT Constraint on CREATE TABLE

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

```
CREATE TABLE Persons  
(  
  PId int NOT NULL,  
  LastName varchar(255) NOT NULL,  
  FirstName varchar(255),  
  City varchar(255) DEFAULT 'Oslo'
```

```
Address varchar(255),  
City varchar(255) DEFAULT 'Debre Markos'  
)
```

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

```
CREATE TABLE Orders  
(  
OId int NOT NULL,  
OrderNo int NOT NULL,  
PId int,  
OrderDate date DEFAULT GETDATE()  
)
```

SQL DEFAULT Constraint 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 ALTER COLUMN City SET DEFAULT 'Debre Markos'
```

To DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:

```
ALTER TABLE Persons ALTER COLUMN City DROP DEFAULT;
```

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 Constraint on CREATE TABLE

The following SQL creates a CHECK constraint on the "PId" column when the "Persons" table is created. The CHECK constraint specifies that the column "PId" must only include integers greater than 10.

```
CREATE TABLE Persons
(
  PId int NOT NULL CHECK (PId>10),
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255)
)
```

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

```
CREATE TABLE Persons
(
  PId int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255),
  CONSTRAINT chk_Person CHECK (PId>10 AND City='Debre Markos')
)
```

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

```
ALTER TABLE Persons ADD CHECK (PId>10)
```

To drop a CHECK constraint, use the following SQL:

```
ALTER TABLE Persons DROP CONSTRAINT chk_Person
```

Renaming a table

To rename a table using code, execute the **sp_rename** stored procedure using the following formula:

```
sp_rename ExistingTableName, TableNewName;
```

Here is an example:

```
sp_rename 'StaffMember', 'Employee';
```

In this case, the interpreter would look for a table named StaffMembers in the current or selected database. If it finds it, it would rename it Employees. If the table doesn't exist, you would receive an error.

To see the structure of tables

To see the structure of the table once created the formula is [Sp_help TableName](#)

Example:

```
use RegistrarDB  
sp_help Resulttbl;
```

How to alter / modify the structure of the table

To Drop a column

To delete a column using code, first open or access an empty query window, and use the following formula:

```
ALTER TABLE TableName DROP COLUMN ColumnName;
```

On the right side of the **ALTER TABLE** expression, type the name of the table. On the right side of the **DROP COLUMN** expression, enter the name of the undesired column. Here is an example:

```
use RegistrarDB  
alter table CourseRegistrationtbl drop RegID;
```

To Add a column

To add a new column to a table, follow this formula:

```
ALTER TABLE TableName  
ADD ColumnName Properties;  
use RegistrarDB
```

```
alter table CourseRegistrationtbl add RegID int IDENTITY(1,1) primary key;  
use RegistrarDB  
alter table Studenttbl add StudDepartment varchar(50);
```

How to modify the data type of the column?

To modify the data type for a column in a table we can use the following formula:

```
Alter table TableName Alter column ColumnName DataType;  
//Example:  
use RegistrarDB  
alter table Coursetbl alter column CTitle varchar(100);
```

To drop a table

To delete a table using SQL, use the following formula:

```
DROP TABLE TableName; or drop table databasename.dbo.tablename;
```

The DROP TABLE expression is required and it is followed by the name of the undesired table. (Since drop is DDL it cannot be restored if once dropped)

```
Example:  
use RegistrarDB  
drop table employeetbl; or drop table registrarDB.dbo.employeetbl;
```

To truncate table

Description: The keyword truncate removes all values from the table once and cannot be rolled back. The formula: `truncate table TableName;`

```
Example:  
use RegistrarDB  
truncate table StudStatustbl;
```

DML (Data Manipulation Language)

Insert values

- ❏ Destination columns should match.
- ❏ if no value is given for a column, null will be taken by default
- ❏ else if that column has “not null” constraint, it returns error

The syntax to insert values within a table

```
INSERT INTO TableName  
([column1],[column2],[column3],[column4],[column5],[column6])  
VALUES (<column1, DataType,>,< column1, DataType,>, < column1,  
DataType,> ,< column1, DataType,> ,< column1, DataType,> ,< column1, DataType,>)
```

Example:

```
use RegistrarDB  
  
insert into Coursetbl values('c005','internet programming', 3);  
  
Or  
  
use RegistrarDB  
INSERT INTO [Coursetbl]([Ccode],[Ctitle],[CCredit])  
VALUES('c005','internet programming','3');
```

Update values in a table

To update a single field with in a table we can use the following syntax:

```
Update TableName set ColumnName='TheNewValue'  
Where PrimaryKey='TheValueOfThePrimaryKeyForThatColumn';  
  
Example:  
  
use RegistrarDB  
update Coursetbl set Ctitle='computer organization and architecture'  
where Ccode='c001';
```

Delete a single row in a table

```
Example:  
  
use RegistrarDB  
delete from Coursetbl where Ccode='c001';
```

DRL (Data Retrieval Language)

Select statement

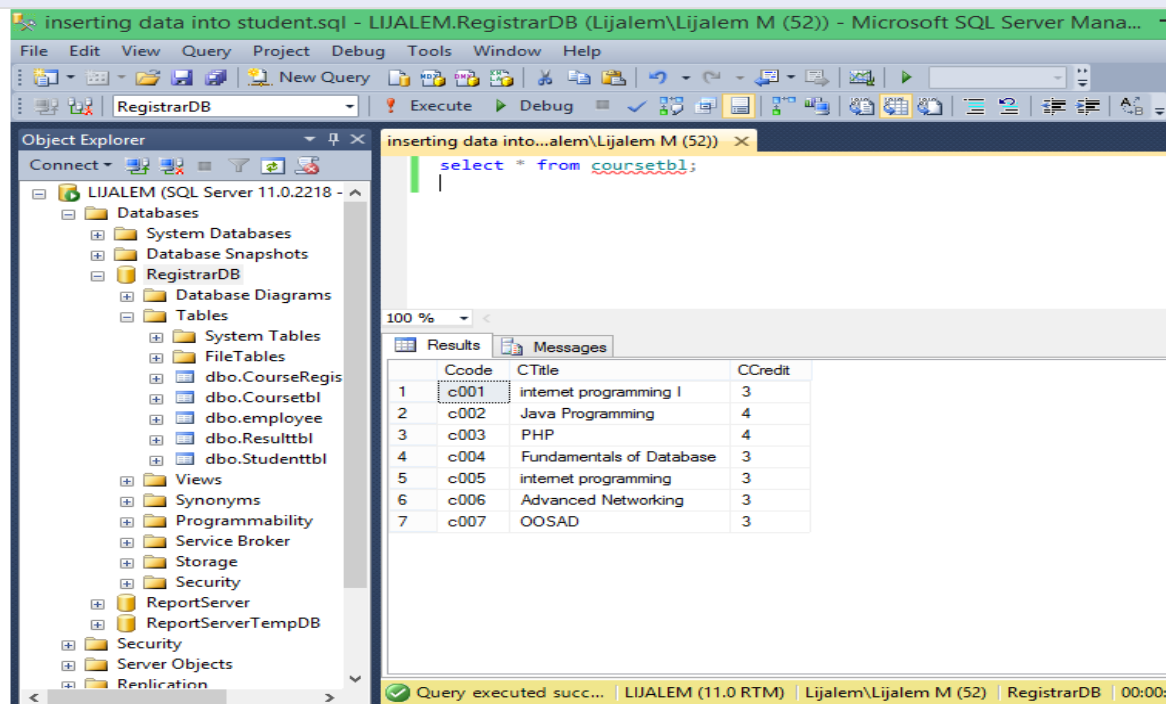
☐ The SELECT operator can be used, among other things, to display a value. The SELECT keyword uses the following syntax:

```
SELECT What;
```

- Based on this, to use it, where it is needed, type SELECT followed by a number, a word, a string, or an expression.
- To display a sentence using SELECT, type it in single-quotes on the right side of this operator. Here is an executed example

Use RegistrarDB

```
select * from Statussampleview;  
select * from Resulttbl;  
select * from CourseRegistrationtbl;  
select * from Student_Backup;  
select * from Coursetbl;  
select * from StudStatustbl;  
select Studenttbl.Studname, Studenttbl.StudDepartment, Resulttbl.LetterGrade from  
Studenttbl, Resulttbl  
where Resulttbl.Sid=Studenttbl.Sid;
```



SQL Server: Comparison Operators

This SQL Server tutorial explores all of the comparison operators used to test for equality and inequality, as well as the more advanced operators in SQL Server (Transact-SQL).

Description

Comparison operators are used in the WHERE clause to determine which records to select.

Here is a list

of the comparison operators that you can use in SQL Server (Transact-SQL):

Comparison Operator	Description
=	Equal
<>	Not Equal
!=	Not Equal
>	Greater Than
>=	Greater Than or Equal
<	Less Than
<=	Less Than or Equal
!>	Not Greater Than
!<	Not Less Than
IN ()	Matches a value in a list
NOT	Negates a condition
BETWEEN	Within a range (inclusive)
IS NULL	NULL value
IS NOT NULL	Non-NULL value
LIKE	Pattern matching with % and _
EXISTS	Condition is met if subquery returns at least one row

There are many comparison operators in SQL Server and Transact-SQL. Let's explore how to use the more common operators.

Example - Equality Operator

In SQL Server, you can use the = operator to test for equality in a query.

For example:

```
SELECT *  
FROM employees  
WHERE first_name = 'Jane';
```

In this example, the SELECT statement above would return all rows from the employees table where the first_name is equal to Jane.

Example - Inequality Operator

In SQL Server, you can use the <> or != operators to test for inequality in a query.

For example, we could test for inequality using the <> operator, as follows:

```
SELECT *  
FROM employees  
WHERE first_name <> 'Jane';
```

In this example, the SELECT statement would return all rows from the employees table where the first_name is not equal to Jane.

Or you could also write this query using the != operator, as follows:

```
SELECT *  
FROM employees  
WHERE first_name != 'Jane';
```

Both of these queries would return the same results.

Example - Greater Than Operator

You can use the > operator in SQL Server to test for an expression greater than.

```
SELECT *  
FROM employees  
WHERE employee_id > 3000;
```

In this example, the SELECT statement would return all rows from the employees table where the employee_id is greater than 3000. An employee_id equal to 3000 would not be included in the result set.

Example - Greater Than or Equal Operator

In SQL Server, you can use the >= operator to test for an expression greater than or equal to.

```
SELECT *  
FROM employees
```

```
WHERE employee_id >= 3000;
```

In this example, the SELECT statement would return all rows from the employees table where the employee_id is greater than or equal to 3000. In this case, n employee_id equal to 3000 would be included in the result set.

Example - Less Than Operator

You can use the < operator in SQL Server to test for an expression less than.

```
SELECT *  
FROM employees  
WHERE employee_id < 500;
```

In this example, the SELECT statement would return all rows from the employees table where the employee_id is less than 500. An employee_id equal to 500 would not be included in the result set.

Example - Less Than or Equal Operator

In SQL Server, you can use the <= operator to test for an expression less than or equal to.

```
SELECT * FROM employees  
WHERE employee_id <= 500;
```

In this example, the SELECT statement would return all rows from the employees table where the employee_id is less than or equal to 500. In this case, n employee_id equal to 500 would be included in the result set.

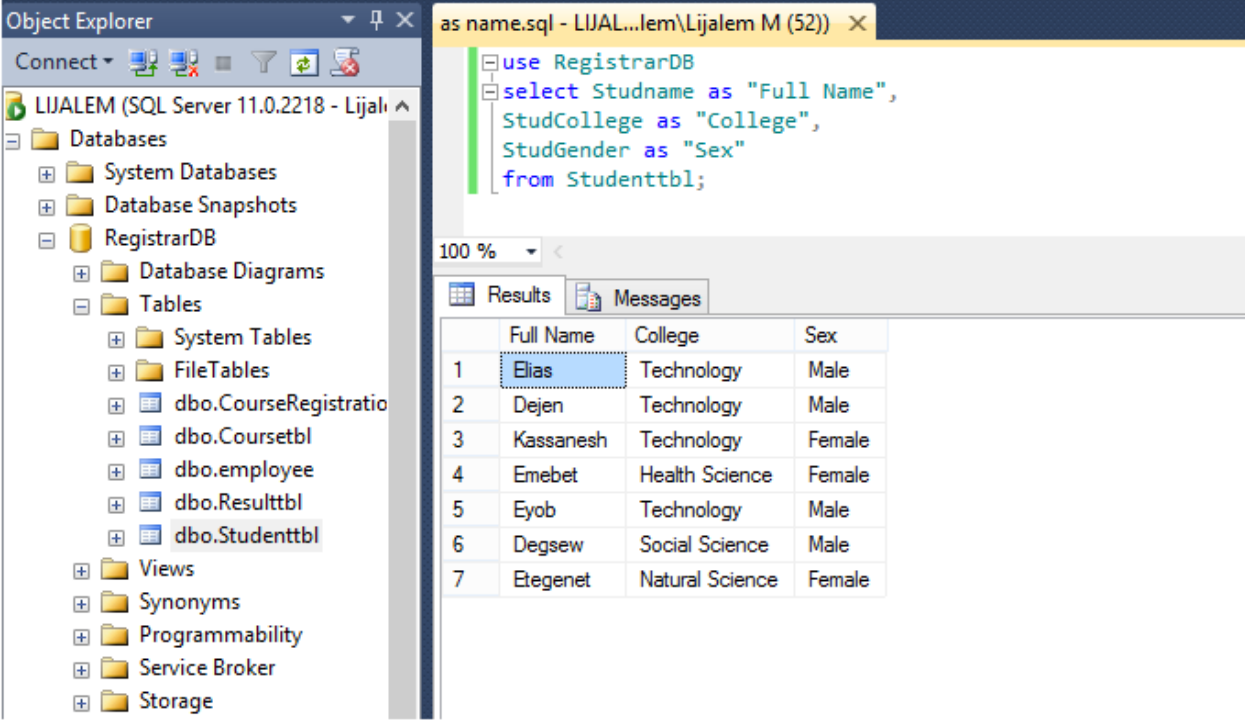
How to select last inserted row

```
use RegistrarDB  
SELECT * from Studenttbl where [SID] = ALL (SELECT MAX (SID)  
from Studenttbl)
```

How to assign our own name for the columns in the database?

Example:

```
use RegistrarDB
select Studname as "Full Name", StudCollege as "College", StudGender as "Sex"
from Studenttbl;
```



The screenshot shows the Microsoft SQL Server Enterprise Manager interface. On the left, the 'Object Explorer' pane displays the 'RegistrarDB' database structure, including tables like 'Studenttbl'. The main window shows the SQL query: `use RegistrarDB; select Studname as "Full Name", StudCollege as "College", StudGender as "Sex" from Studenttbl;`. Below the query, the 'Results' tab displays the data from the 'Studenttbl' table.

	Full Name	College	Sex
1	Elias	Technology	Male
2	Dejen	Technology	Male
3	Kassanesh	Technology	Female
4	Emebet	Health Science	Female
5	Eyob	Technology	Male
6	Degsew	Social Science	Male
7	Etegenet	Natural Science	Female

Transact-SQL: LIKE

The LIKE operator of Transact-SQL is used to with a wildcard (missing letters in the pattern) to specify a pattern to select one or more records from a table or a view. If you are visually creating the statement, in the Criteria section, in the box corresponding to Filter for the column on which the condition would be applied, type the **LIKE** condition. In Transact-SQL, the **LIKE** operator is used in a formula as follows:

```
Expression LIKE pattern;
```

The Expression factor is the expression that will be evaluated. This must be a clear and valid expression.

The pattern factor can be a value to be found in Expression. For example, it can be the same type of value used in a **WHERE** statement. In this case, the equality operator would be the same as **LIKE**. If you want to match any character, in any combination, for any length, use the

% wildcard. If you precede it with a letter, as in **S%**, the condition would consist of finding any string that starts with S.

Imagine that you want to get a list of students whose last names start with S. You would type the condition as **LIKE 'S%'**.

If you set the letter in between (**% letter %**) with a letter, as in **%S%**, the condition would consist of finding any string that contains S in the middle.

Example:

```
select StudName, StudAge from Studenttbl where Studname like 'm%';  
// display student name from student table where student name starts with 'm'
```

```
select StudName, StudAge from Studenttbl where Studname like '%m'; // display  
student name from student table where student name end with 'm'
```

```
select StudName, StudAge from Studenttbl where Studname like '%m%';  
// display student name from student table where student name has 'm' letter in  
between.
```

```
select StudName, StudAge from Studenttbl where Studname like '[G or g]'; //  
displays student name from student table where name strats with upper case or  
small case g.
```

```
select StudName, StudAge from Studenttbl where Studname like '_t%'; // displays  
student name from student table where second letter name is t.
```

```
select StudName, StudAge from Studenttbl where Studname not like '_t%';
```

```
use RegistrarDB  
select studenttbl.sid from Studenttbl  
where Studenttbl. StudCollege like '[T or t]%'  
Intersect  
(select StudStatustbl.Sid from StudStatustbl)
```

Example:

```
Select studCollege, count(*) as NumOfStudInEachCollege From Studenttbl  
Group by StudCollege;
```

```
use RegistrarDB  
select studenttbl.sid from Studenttbl  
where Studenttbl.StudCollege like '[T or t]%'  
union (select StudStatustbl.Sid from StudStatustbl)
```

Example:

```
Select * from Studenttbl order by studAge desc;
```

SQL Server: WHERE Clause

This SQL Server tutorial explains how to use the WHERE clause in SQL Server (Transact-SQL) with syntax and examples.

Description: The SQL Server (Transact-SQL) WHERE clause is used to filter the results from a SELECT, INSERT, UPDATE, or DELETE statement.

Syntax: The syntax for the WHERE clause in SQL Server (Transact-SQL) is:

```
WHERE conditions;
```

Conditions: The conditions that must be met for records to be selected.

Example - With Single condition

It is difficult to explain the syntax for the SQL Server WHERE clause, so let's look at some examples. We'll start by looking at how to use the WHERE clause with only a single condition. For example:

```
SELECT *  
FROM employees  
WHERE first_name = 'abebe';
```


In this SQL Server WHERE clause example, we've used the WHERE clause to filter our results from the employees table. The SELECT statement above would return all rows from the employees table where the first_name is 'abebe'. Because the * is used in the SELECT, all fields from the employees table would appear in the result set.

Example - Using AND condition

Let's look at how to use the WHERE clause with the AND condition.

For example:

```
SELECT *  
FROM employees  
WHERE last_name = 'Elias'  
AND employee_id >= 3000;
```

This SQL Server WHERE clause example uses the WHERE clause to define multiple conditions. In this case, this SELECT statement uses the AND condition to return all employees that have a last_name of 'Elias' and the employee_id is greater than or equal to 3000.

Example - Using OR condition

Let's look at how to use the WHERE clause with the OR condition.

For example:

```
SELECT employee_id, last_name, first_name  
FROM employees  
WHERE last_name = 'alemu'  
OR first_name = 'elias';
```

This SQL Server WHERE clause example uses the WHERE clause to define multiple conditions, but instead of using the AND condition, it uses the OR condition. In this case, this SELECT statement would return all employee_id, last_name, and first_name values from the employees table where the last_name is 'alemu' or the first_name is 'Elias'.

Example - Combining AND & OR conditions

Let's look at how to use the WHERE clause when we combine the AND & OR conditions in a single SQL statement.

For example:

```
SELECT *  
FROM employees  
WHERE (city = 'Debre Markos' AND last_name = 'Samuel')  
OR (employee_id = 82);
```

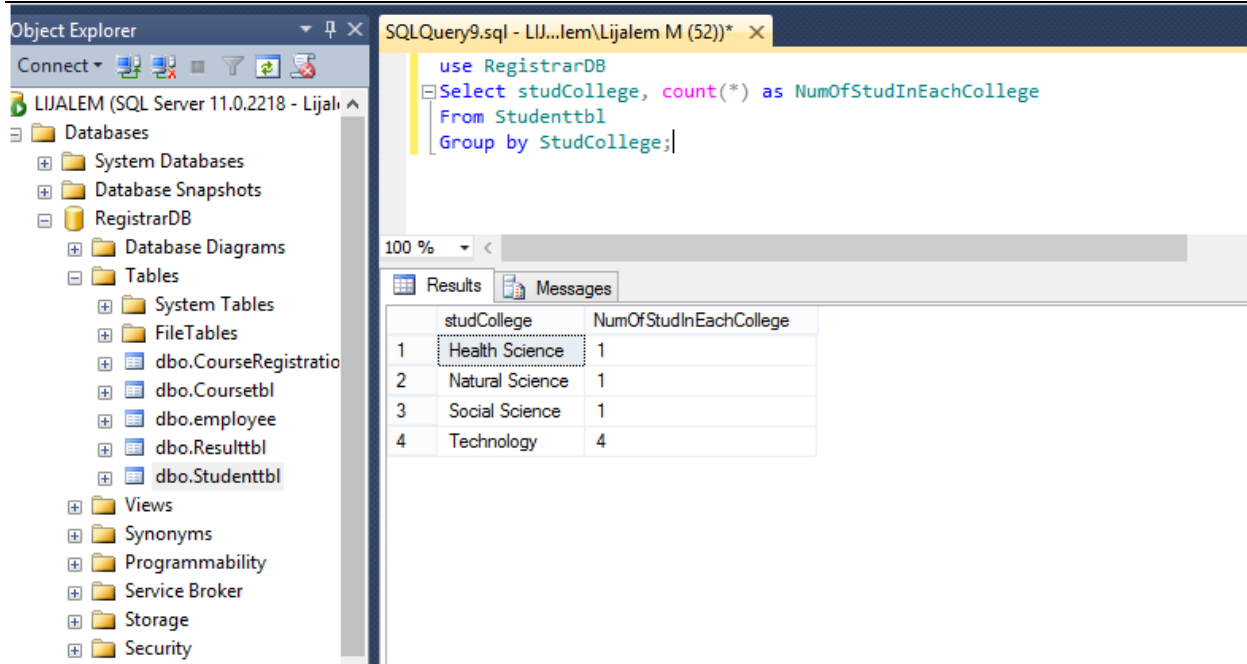
This SQL Server WHERE clause example uses the WHERE clause to define multiple conditions, but it combines the AND condition and the OR condition. This example would return all employees that reside in the city of 'Debre Markos' and whose last_name is 'Samuel' as well as all employees whose employee_id is equal to 82.

The parentheses determine the order that the AND and OR conditions are evaluated. Just like you learned in the order of operations in Math class!

Example:

```
use RegistrarDB  
select StudCollege, count(*) from Studenttbl group by StudCollege;
```

```
use RegistrarDB  
select Studenttbl.Studname, Studenttbl.StudCollege,  
StudStatustbl.CGPA, StudStatustbl.StudStatus from Studenttbl, StudStatustbl where  
Studenttbl.Sid=StudStatustbl.Sid;
```



SQL Server: ORDER BY Clause

This SQL Server tutorial explains how to use the ORDER BY clause in SQL Server (Transact-SQL) with syntax and examples.

Description: The SQL Server (Transact-SQL) ORDER BY clause is used to sort the records in your result set. The ORDER BY clause can only be used in SELECT statements.

Syntax: The syntax for the ORDER BY clause in SQL Server (Transact-SQL) is:

```
SELECT expressions
FROM tables
[WHERE conditions]
ORDER BY expression [ ASC | DESC ];
```

The SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements. Notice that each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types. Also, the columns in each SELECT statement must be in the same order.

SQL UNION Syntax

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

Note: The UNION operator selects only distinct values by default. To allow duplicate values, use the ALL keyword with UNION.

SQL UNION ALL Syntax

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

Example:

```
SELECT CID FROM Customer UNION SELECT City FROM Supplier ORDER BY City;
SELECT City FROM Customer UNION ALL SELECT City FROM Supplier ORDER BY City;
```

SQL FULL OUTER JOIN Keyword

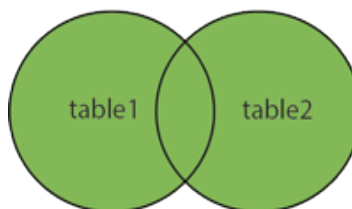
The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).

The FULL OUTER JOIN keyword combines the result of both LEFT and RIGHT joins.

SQL FULL OUTER JOIN Syntax

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

FULL OUTER JOIN



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

SQL LEFT JOIN Keyword

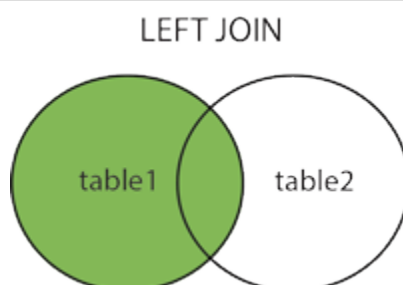
The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.

SQL LEFT JOIN Syntax

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

or:

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



SQL INNER JOIN Keyword

The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns in both tables.

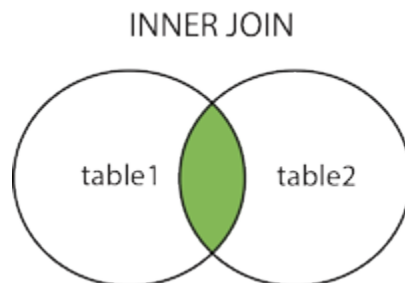
SQL INNER JOIN Syntax

```
SELECT column_name(s)
FROM table1
```

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

or:

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



Example:

```
Select studenttbl.studname, studStatustbl.studStatus  
From studenttbl left join studStatustbl ON  
Studenttbl.Sid=studstatustbl.sid;
```

The right join returns all the rows from the second table (student Status table). Even if there are no matches in the first table (student table)

Example:

```
use RegistrarDB  
select Studenttbl.Studname,StudStatustbl.StudStatus  
from Studenttbl left join StudStatustbl ON  
Studenttbl.Sid=StudStatustbl.Sid
```

Example:

```
use RegistrarDB
select Studenttbl.Studname,StudStatustbl.StudStatus
from StudStatustbl right join Studenttbl ON
Studenttbl.Sid=StudStatustbl.Sid
```

How to create view

Example:

```
CREATE VIEW statusreport
AS
SELECT Sid AS Expr1 FROM Studenttbl;
```

Average, Maximum and Minimum

Example:

```
Select avg(studstatustbl.SGPA) as averageGrade,
MAX(studstatustbl.SPGA) as SemesterTop
Max(studstatus.CGPA) as YearTOP
From studstatustbl;
```

DISTINCT

Example

```
Select COUNT(DISTINCT studenttbl.studname) as 'NO OF Students'
From studenttbl
Where studGender like '[m or M] %';
```

Select into

```
use RegistrarDB
SELECT Studenttbl.Studname,
Studenttbl.StudCollege,
StudStatustbl.CGPA INTO Student_Backup
FROM StudStatustbl , Studenttbl where Studenttbl.Sid=StudStatustbl.Sid
and Studenttbl.StudCollege like '[T or t ]%';
```

To verify

```
select * from Student_Backup;
```

Top

Example:

```
Select top 5 studname  
From studenttbl;
```

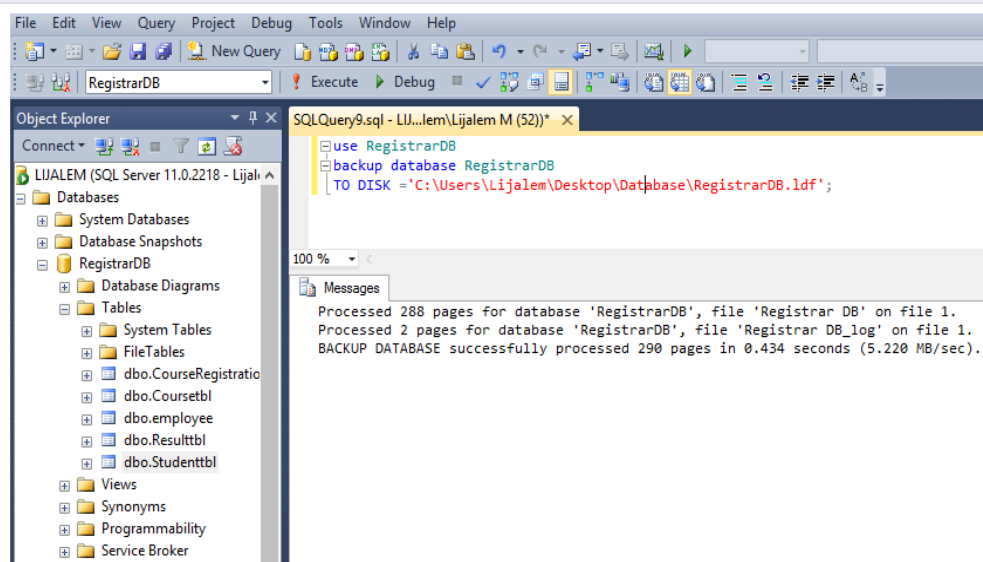
getdate()

Example:

```
use RegistrarDB  
select getdate() as Today;
```

How to back up a database

```
use RegistrarDB  
backup database RegistrarDB  
TO DISK ='C:\Users\Lijalem\Desktop\Database\RegistrarDB.ldf';
```

**SQL Server: VIEW**

Learn how to create, update, and drop VIEWS in SQL Server (Transact-SQL) with syntax and examples.

What is a VIEW in SQL Server?

A VIEW, in essence, is a virtual table that does not physically exist in SQL Server. Rather, it is created by a query joining one or more tables.

Create VIEW

Syntax: The syntax for the CREATE VIEW statement in SQL Server (Transact-SQL) is:

```
CREATE VIEW [schema_name.]view_name AS
[ WITH { ENCRYPTION | SCHEMABINDING | VIEW_METADATA }
SELECT expressions
FROM tables
[WHERE conditions];
```

For example:

```
CREATE VIEW prod_inv AS
SELECT products.product_id, products.product_name, inventory.quantity
FROM products
INNER JOIN inventory
ON products.product_id = inventory.product_id
WHERE products.product_id >= 1000;
```

This SQL Server CREATE VIEW example would create a virtual table based on the result set of the SELECT statement. The view would be called prod_inv.

You can now query the SQL Server VIEW as follows:

```
SELECT * FROM prod_inv;
```

Update VIEW

You can modify the definition of a VIEW in SQL Server without dropping it by using the ALTER VIEW Statement.

Syntax: The syntax for the ALTER VIEW statement in SQL Server (Transact-SQL) is:

```
ALTER VIEW [schema_name.]view_name AS
[ WITH { ENCRYPTION | SCHEMABINDING | VIEW_METADATA }
SELECT expressions
FROM tables
WHERE conditions;
```

Example

Here is an example of how you would use the ALTER VIEW Statement in SQL Server (Transact-SQL):

```
ALTER VIEW prod_inv AS
SELECT products.product_name, inventory.quantity
FROM products
INNER JOIN inventory
ON products.product_id = inventory.product_id
WHERE products.product_id >= 500
AND products.product_id <= 1000;
```

This ALTER VIEW example would update the definition of the VIEW called prod_inv without dropping it in SQL Server. The VIEW must exist for you to be able to execute an ALTER VIEW command.

Drop VIEW

Once a VIEW has been created in SQL Server, you can drop it with the DROP VIEW Statement.

Syntax: The syntax for the DROP VIEW statement in SQL Server (Transact-SQL) is:

```
DROP VIEW view_name;
```

View_name: The name of the view that you wish to drop.

Example: Here is an example of how to use the DROP VIEW Statement in SQL Server (Transact-SQL):

```
DROP VIEW prod_inv;
```

This DROP VIEW example would drop/delete the VIEW called prod_inv in SQL Server (Transact-SQL).

Creating save point

A user can set a save point, or marker, within a transaction. The save point defines a location to which transaction can return if part of the transaction is conditionally canceled. If a transaction is rolled back to a save point, it must proceed to completion with more Transact-SQL statements if needed and a COMMIT TRANSACTION statement, or it must be canceled altogether by rolling the transaction back to its beginning. To cancel an entire transaction, use the form ROLLBACK TRANSACTION transaction_name. All the statements or procedures of the transaction are undone. Duplicate save point names are allowed in a transaction, but a ROLLBACK TRANSACTION statement that specifies the save point name will only roll the transaction back to the most recent SAVE TRANSACTION using that name.

Example:

```
use RegistrarDB
begin transaction
save transaction s1
delete from Coursetbl
where Ccode='c002';
```

Rollback / Restoring

ROLLBACK TRANSACTION without a savepoint_name or transaction_name rolls back to the beginning of the transaction. When nesting transactions, this same statement rolls back all inner transactions to the outermost BEGIN TRANSACTION statement.

In both cases, ROLLBACK TRANSACTION decrements the @@TRANCOUNT system function to 0. ROLLBACK TRANSACTION savepoint_name does not decrement @@TRANCOUNT. A transaction cannot be rolled back after a COMMIT TRANSACTION statement is executed, except when the COMMIT TRANSACTION is associated with a nested transaction that is contained within the transaction being rolled back. In this instance, the nested transaction

will also be rolled back, even if you have issued a COMMIT TRANSACTION for it. Within a transaction, duplicate save point names are allowed, but a ROLLBACK TRANSACTION using the duplicate save point name rolls back only to the most recent SAVE TRANSACTION using that save point name.

Example:

```
use RegistrarDB
begin transaction s1
rollback transaction
```

To Be Continued.....