1. SQL Server Administration

SQL Server: CREATE LOGIN statement

The CREATE LOGIN statement creates an identity used to connect to a SQL Server instance. The Login is then mapped to a database user (so before creating a user in SQL Server, you must first create a Login).

There are different types of Logins that you can create in SQL Server:

1. You can create a Login using Windows Authentication.
2. You can create a Login using SQL Server Authentication.

Syntax

The syntax for the CREATE LOGIN statement using Windows Authentication is:

```
CREATE LOGIN [domain_name\login_name]
FROM WINDOWS
[WITH DEFAULT_DATABASE = database_name
| DEFAULT_LANGUAGE = language_name];
```

OR

The syntax for the CREATE LOGIN statement using SQL Server Authentication is:

```
CREATE LOGIN login_name
WITH PASSWORD = {'password' | hashed_password HASHED} [MUST_CHANGE]
[, SID = sid_value
  | DEFAULT_DATABASE = database_name
  | DEFAULT_LANGUAGE = language_name
  | CHECK_EXPIRATION = {ON | OFF}
  | CHECK_POLICY = {ON | OFF}
  | CREDENTIAL = credential_name];
```

Example – Windows Authentication

```
CREATE LOGIN [Lijalem\Lijalem]
FROM WINDOWS;
```

This CREATE LOGIN example would create a new Login called [Lijalem\Lijalem] that uses Windows authentication.

Example - SQL Server Authentication

```
CREATE LOGIN Lijalem
WITH PASSWORD = 'pwd123';
```

This CREATE LOGIN example would create a new Login called Lijalem that uses SQL Server authentication and has a password of 'pwd123'.

If we want to force the password to be changed the first time that the Login is used, we could modify our example as follows:

```
CREATE LOGIN Lijalem
WITH PASSWORD = 'pwd123' MUST_CHANGE,
CHECK_EXPIRATION = ON;
```

This example uses the MUST_CHANGE option to force the password to be changed on the first login. It is important to note that the MUST_CHANGE option cannot be used when the CHECK_EXPIRATION is OFF.

Therefore, this example also specifies "CHECK_EXPIRATION = ON". Otherwise, the CREATE LOGIN statement would raise an error.

SQL Server: ALTER LOGIN statement

The ALTER LOGIN statement modifies an identity used to connect to a SQL Server instance. You can use the ALTER LOGIN statement to change a password, force a password change, disable a login, enable a login, unlock a login, rename a login, etc.

Syntax

The syntax for the ALTER LOGIN statement in SQL Server is:

```
ALTER LOGIN login_name
{ENABLE | DISABLE
| WITH PASSWORD = 'password' | hashed_password HASHED
         [OLD_PASSWORD = 'old_password’]
         | MUST_CHANGE
```

```
          | UNLOCK
        | DEFAULT_DATABASE = database_name
        | DEFAULT_LANGUAGE = language_name
        | NAME = new_login_name
        | CHECK_EXPIRATION = {ON | OFF}
        | CHECK_POLICY = {ON | OFF}
        | CREDENTIAL = credential_name
        | NO CREDENTIAL
| ADD CREDENTIAL new_credential_name
| DROP CREDENTIAL credential_name};
```

Example – Change Password

```
ALTER LOGIN Lijalem
WITH PASSWORD = 'passme123';
```

This ALTER LOGIN example would alter the Login called Lijalem and change the password of this login to ' passme123'.

Example – Change Password and Force Change

```
ALTER LOGIN Lijalem
WITH PASSWORD = 'passme123' MUST_CHANGE,
CHECK_EXPIRATION = ON;
```

This ALTER LOGIN example would alter the Login called Lijalem and change the password of this login to 'passme123'. But because we have specified the MUST CHANGE option and set the CHECK_EXPIRATION to ON, the password will have to be changed again in SQL Server after the first login (following the ALTER LOGIN statement). So in effect, it is like resetting a password to a temporary password for a Login.

SQL Server: DROP USER statement

The DROP USER statement is used to remove a user from the SQL Server database.

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

```
DROP USER user_name;
```

For example:

```
DROP USER Lijalem;
```

This DROP USER example would drop the user called Lijalem. This DROP USER statement will only run if Lijalem does not own any objects in the SQL Server database.

Example – Disable a Login

```
ALTER LOGIN Lijalem DISABLE;
```

This ALTER LOGIN example would disable the Login called Lijalem.

Example – Enable a Login

```
ALTER LOGIN Lijalem ENABLE;
```

This ALTER LOGIN example would enable the Login called Lijalem.

Example – Unlock a Login

```
ALTER LOGIN Lijalem
WITH PASSWORD = 'passme123'
UNLOCK;
```

This ALTER LOGIN example would unlock the Login called Lijalem and set the password to 'passme123'.

Example – Rename a Login

```
ALTER LOGIN Lijalem
WITH NAME = MyAccount;
```

SQL Server: DROP LOGIN statement

The DROP LOGIN statement is used to remove an identity (ie: Login) used to connect to a SQL Server instance.

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

`DROP LOGIN login_name;`

You cannot drop a Login when it is currently logged into SQL Server.

If you drop a Login that has database users mapped to it, the users will be orphaned in SQL Server.

For example:

```
DROP LOGIN MyAccount;
```

This DROP LOGIN example would drop the Login called MyAccount. This DROP LOGIN statement will only run if techonthenet is not currently logged in. If MyAccount is currently logged into SQL Server, the DROP LOGIN statement will raise an error.

SQL Server: Find Logins in SQL Server

In SQL Server, there is a catalog view (i.e.: system view) called sys.sql_logins. You can run a query against this system view that returns all of the Logins that have been created in SQL Server as well as information about these Logins.

To retrieve all Logins in SQL Server, you can execute the following SQL statement:

```
SELECT * FROM master.sys.sql_logins;
```

SQL Server: CREATE USER statement

The CREATE USER statement creates a database user to log into SQL Server. A database user is mapped to a Login, which is an identity used to connect to a SQL Server instance.

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

```
CREATE USER user_name FOR LOGIN login_name;
```

For example:

```
CREATE USER Lijalem FOR LOGIN MyAccount;
```

This CREATE USER example would create a new database user called Lijalem in SQL Server. It would use the MyAccount Login as the identity to connect to the SQL Server instance.

SQL Server: DROP USER statement

The DROP USER statement is used to remove a user from the SQL Server database.

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

```
DROP USER user_name;
```

For example:

```
DROP USER Lijalem;
```

This DROP USER example would drop the user called Lijalem. This DROP USER statement will only run if Lijalem does not own any objects in the SQL Server database.

2. SQL Server Privileges

SQL GRANT REVOKE Commands

DCL commands are used to enforce database security in a multiple user database environment. Two types of DCL commands are GRANT and REVOKE. Only Database Administrator's or owners of the database object can provide/remove privileges on a database object.

SQL GRANT Command

SQL GRANT is a command used to provide access or privileges on the database objects to the users.

The Syntax for the GRANT command is:

```
GRANT privilege_name
ON object_name
TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];
```

For Example: GRANT SELECT ON employee TO user1; this command grants a SELECT permission on employee table to user1. You should use the WITH GRANT option carefully because for example if you GRANT SELECT privilege on employee table to user1 using the WITH GRANT option, then user1 can GRANT SELECT privilege on employee table to another user, such as user2 etc. Later, if you REVOKE the SELECT privilege on employee from user1, still user2 will have SELECT privilege on employee table.

SQL REVOKE Command:

The REVOKE command removes user access rights or privileges to the database objects.

The Syntax for the REVOKE command is:

```
REVOKE privilege_name
ON object_name
FROM {user_name |PUBLIC |role_name}
```

For Example: REVOKE SELECT ON employee FROM user1;This command will REVOKE a SELECT privilege on employee table from user1.When you REVOKE SELECT privilege on a table from a user, the user will not be able to SELECT data from that table anymore. However, if the user has received SELECT privileges on that table from more than one users, he/she can SELECT from that table until everyone who granted the permission revokes it. You cannot REVOKE privileges if they were not initially granted by you.

Privileges and Roles:

Privileges: Privileges defines the access rights provided to a user on a database object. There are two types of privileges.

- **System privileges**: This allows the user to CREATE, ALTER, or DROP database objects.
- **Object privileges**: This allows the user to EXECUTE, SELECT, INSERT, UPDATE, or DELETE data from database objects to which the privileges apply.

Few CREATE system privileges are:

System Privileges	Description
CREATE object	Allows users to create the specified object in their own schema.
CREATE ANY object	Allows users to create the specified object in any schema.

The above rules also apply for ALTER and DROP system privileges.

Few of the object privileges are:

Object Privileges	Description
INSERT	Allows users to insert rows into a table.
SELECT	Allows users to select data from a database object.
UPDATE	Allows user to update data in a table.
EXECUTE	Allows user to execute a stored procedure or a function.

Roles: Roles are a collection of privileges or access rights. When there are many users in a database it becomes difficult to grant or revoke privileges to users. Therefore, if you define roles, you can grant or revoke privileges to users, thereby automatically granting or revoking privileges. You can either create Roles or use the system roles pre-defined by oracle.

Some of the privileges granted to the system roles are as given below:

System Role	Privileges Granted to the Role
CONNECT	CREATE TABLE, CREATE VIEW, CREATE SYNONYM, CREATE SEQUENCE, CREATE SESSION etc.
RESOURCE	CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER etc. The primary usage of the RESOURCE role is to restrict access to database objects.
DBA	ALL SYSTEM PRIVILEGES

Creating Roles: The Syntax to create a role is:

```
CREATE ROLE role_name
[IDENTIFIED BY password];
```

For Example: To create a role called "developer" with password as "pwd", the code will be as follows

```
CREATE ROLE developer
[IDENTIFIED BY pwd];
```

It's easier to GRANT or REVOKE privileges to the users through a role rather than assigning a privilege directly to every user. If a role is identified by a password, then, when you GRANT or REVOKE privileges to the role, you definitely have to identify it with the password.

We can GRANT or REVOKE privilege to a role as below.

For example: To grant CREATE TABLE privilege to a user by creating a developer role:

First, create a testing Role

```
Create role developer
```

Second, grant a CREATE TABLE privilege to the ROLE developer. You can add more privileges to the ROLE.

```
GRANT CREATE TABLE TO developer;
```

Third, grant the role to a user.

```
GRANT developer TO user1;
```

To revoke a CREATE TABLE privilege from developer ROLE, you can write:

```
REVOKE CREATE TABLE FROM developer;
```

The Syntax to drop a role from the database is:

```
DROP ROLE role_name;
```

For example: To drop a role called developer, you can write:

```
DROP ROLE developer;
```

3. SQL server programming

SQL Server: Comments within SQL

Did you know that you can place comments within your SQL statements in SQL Server (Transact-SQL)? These comments can appear on a single line or span across multiple lines. Let's look at how to do this.

Syntax: There are two syntaxes that you can use to create a comment within your SQL statement in SQL Server (Transact-SQL).

Syntax Using -- symbol

The syntax for creating a SQL comment using the -- symbol in SQL Server (Transact-SQL) is:

```
-- Comment goes here
```

In SQL Server, a comment started with -- symbol must be at the end of a line in your SQL statement with a line break after it. This method of commenting can only span a single line within your SQL and must be at the end of the line.

Syntax Using /* and */ symbols

The syntax for creating a SQL comment using /* and */ symbols in SQL Server (Transact-SQL) is:

```
/* comment goes here */
```

In SQL Server, a comment that starts with /* symbol and ends with */ and can be anywhere in your SQL statement. This method of commenting can span several lines within your SQL.

Example - Comment on a Single Line

```
SELECT employee_id, last_name
/* Author: 3rd Year IT Students */
FROM employees;
```

Here is a SQL comment that appears in the middle of the line:

```
SELECT /* Author: 3rd Year IT Students */ employee_id, last_name
FROM employees;
```

Here is a SQL comment that appears at the end of the line:

```
SELECT employee_id, last_name /* 3rd Year IT Students */
FROM employees;
```

Or

```
SELECT employee_id, last_name -- Author: 3rd Year IT Students
FROM employees;
```

Example – Comment on Multiple Lines

```
SELECT employee_id, last_name
/*
 * Author: 3rd Year IT Students
 * Purpose: To show a comment that spans multiple lines in your SQL
statement.
 */
FROM employees;
```

This SQL comment spans across multiple lines in SQL Server – in this example, it spans across 4 lines.

In SQL Server, you can also create a SQL comment that spans multiple lines using this syntax:

```
SELECT employee_id, last_name /* Author: 3rd Year IT Students
```

Purpose: To show a comment that spans multiple lines in your SQL statement. */

FROM employees;

SQL Server (Transact-SQL) will assume that everything after the /* symbol is a comment until it reaches the */ symbol, even if it spans multiple lines within the SQL statement. So in this example, the SQL comment will span across 2 lines.

SQL Server: Sequences (Autonumber)

Learn how to **create and drop sequences** in SQL Server (Transact-SQL) with syntax and examples.

Description

In SQL Server, you can create an autonumber field by using sequences. A sequence is an object in SQL Server (Transact-SQL) that is used to generate a number sequence. This can be useful when you need to create a unique number to act as a primary key.

Create Sequence

You may wish to create a sequence in SQL Server to handle an autonumber field.

Syntax: The syntax to create a sequence in SQL Server (Transact-SQL) is:

```
CREATE SEQUENCE [schema.]sequence_name
  [AS datatype]
  [START WITH value]
  [INCREMENT BY value]
  [MINVALUE value | NO MINVALUE]
  [MAXVALUE value | NO MAXVALUE]
  [CYCLE | NO CYCLE]
  [CACHE value | NO CACHE];
```

For example:

```
CREATE SEQUENCE contacts_seq
  AS BIGINT
  START WITH 1
  INCREMENT BY 1
  MINVALUE 1
  MAXVALUE 99999
  NO CYCLE
  CACHE 10;
```

This would create a sequence object called contacts_seq. The first sequence number that it would use is 1 and each subsequent number would increment by 1 (i.e.: 2, 3, 4...}. It will cache up to 10 values for performance. The maximum value that the sequence number can be is 99999 and the sequence will not cycle once that maximum is reached. So you can simplify your CREATE SEQUENCE statement as follows:

```
CREATE SEQUENCE contacts_seq
  START WITH 1
  INCREMENT BY 1;
```

Now that you've created a sequence object to simulate an autonumber field, we'll cover how to retrieve a value from this sequence object. To retrieve the next value in the sequence order, you need to use the NEXT VALUE FOR command.

For example:

```
SELECT NEXT VALUE FOR contacts_seq;
```

This would retrieve the next value from contacts_seq. The next value statement needs to be used in a SQL statement. For example:

```
INSERT INTO contacts
(contact_id, last_name)
VALUES
(NEXT VALUE FOR contacts_seq, 'Smith');
```

This INSERT statement would insert a new record into the contacts table. The contact_id field would be assigned the next number from the contacts_seq sequence. The last_name field would be set to 'Smith'.

Drop Sequence

Once you have created your sequence in SQL Server (Transact-SQL), you might find that you need to remove it from the database.

Syntax: The syntax to a drop a sequence in SQL Server (Transact-SQL) is:

```
DROP SEQUENCE sequence_name;
```

sequence_name: The name of the sequence that you wish to drop.

For example:

```
DROP SEQUENCE contacts_seq;
```

This example would drop the sequence called contacts_seq.

Properties of Sequence

Once you have created your sequence in SQL Server (Transact-SQL), you might want to view the properties of the sequence.

Syntax: The syntax to a view the properties of a sequence in SQL Server (Transact-SQL) is:

```
SELECT *
FROM sys.sequences
WHERE name = 'sequence_name';
```

sequence_name: The name of the sequence that you wish to view the properties for.

For example:

```
SELECT *
FROM sys.sequences
WHERE name = 'contacts_seq';
```

This example would query the sys.sequences system view and retrieve the information for the sequence called contacts_seq.

The sys.sequences view contains the following columns:

Column	Explanation
name	Sequence name that was assigned in CREATE SEQUENCE statement
object_id	Object ID

Column	Explanation
principal_id	Owner of the sequence
schema_id	Schema ID where the sequence was created
parent_object_id	ID of the parent object
type	SO
type_desc	SEQUENCE_OBJECT
create_date	Date/time when the sequence was created
modify_date	Date/time when the sequence was last modified
is_ms_shipped	0 or 1
is_published	0 or 1
is_schema_published	0 or 1
start_value	Starting value for sequence
increment	Value used to increment sequence
minimum_value	Minimum value allowed for sequence
maximum_value	Maximum value allowed for sequence
is_cycling	0 or 1. 0=NO CYCLE, 1=CYCLE
is_cached	0 or 1, 0=NO CACHE, 1=CACHE
cache_size	Cache size if is_cached = 1
system_type_id	System type ID for sequence
user_type_id	User type ID for sequence
precision	Maximum precision for sequence's datatype

Column	Explanation
scale	Maximum scale for sequence's datatype
current_value	Last value returned by the sequence
is_exhausted	0 or 1. 0=More values available in sequence. 1=No values available in sequence

SQL Server: Procedures

What is a procedure in SQL Server?

In SQL Server, a procedure is a stored program that you can pass parameters into. It does not return a value like a function does. However, it can return a success/failure status to the procedure that called it.

Create Procedure

You can create your own stored procedures in SQL Server (Transact-SQL).

Syntax: The syntax to create a stored procedure in SQL Server (Transact-SQL) is:

```
CREATE { PROCEDURE | PROC } [schema_name.]procedure_name
   [ @parameter [type_schema_name.] datatype
     [ VARYING ] [ = default ] [ OUT | OUTPUT | READONLY ]
   , @parameter [type_schema_name.] datatype
     [ VARYING ] [ = default ] [ OUT | OUTPUT | READONLY ] ]
[ WITH { ENCRYPTION | RECOMPILE | EXECUTE AS Clause } ]
[ FOR REPLICATION ]
AS
BEGIN
   [declaration_section]
   executable_section
END;
```

The following is a simple example of a procedure:

```
CREATE PROCEDURE FindSite
  @site_name VARCHAR(50) OUT
AS
BEGIN
   DECLARE @site_id INT;
   SET @site_id = 8;
   IF @site_id < 10
      SET @site_name = 'TechOnTheNet.com';
   ELSE
      SET @site_name = 'CheckYourMath.com';
END;
```

This procedure is called FindSite. It has one parameter called @site_name which is an output parameter that gets updated based on the variable @site_id.

You could then reference the new stored procedure called FindSite as follows:

```
USE [test]
GO
DECLARE @site_name varchar(50);
EXEC FindSite @site_name OUT;
PRINT @site_name;
GO
```

Drop Procedure

Once you have created your procedure in SQL Server (Transact-SQL), you might find that you need to remove it from the database.

Syntax: The syntax to a drop a stored procedure in SQL Server (Transact-SQL) is:

```
DROP PROCEDURE procedure_name;
```

For example:

```
DROP PROCEDURE FindSite;
```

This DROP PROCEDURE example would drop the stored procedure called FindSite.

SQL Server: Declare Variables

What is a variable in SQL Server?

In SQL Server (Transact-SQL), a variable allows a programmer to store data temporarily during the execution of code.

Syntax: The syntax to declare variables in SQL Server using the DECLARE statement is:

```
DECLARE @variable_name datatype [= initial_value ],
        @variable_name datatype [= initial_value ],...;
```

Example – Declare a variable

```
DECLARE @StudentName VARCHAR (50);
```

This DECLARE statement example would declare a variable called @StudentName that is a VARCHAR datatype, with a length of 50 characters.

You then change the value of the @StudentName variable using the SET statement, as follows:

```
SET @StudentName = 'Abebe Alemu';
```

Next, let's also look at how to declare an INT variable in SQL Server.

For example:

```
DECLARE @studentAge INT;
```

To assign a value to the @studentAge variable, you can use the SET statement, as follows:

```
SET @studentAge = 25;
```

This SET statement would assign the variable @studentAge to the integer 25.

Example – Declare more than one variable

```
DECLARE @StudentName VARCHAR (50),
        @studentAge INT;
```

In this example, we are declaring two variables. The first is the variable called @StudentName which is defined as a VARCHAR (50) and the second is a variable called @studentAge which is declared as an INT.

Example – Declare a variable with an initial value

```
DECLARE @StudentName VARCHAR (50) = 'Abebe Alemu';
```

This DECLARE statement example would declare a variable called @StudentName that is a VARCHAR datatype with a length of 50 characters. It would then set the variable of the @StudentName variable to 'Abebe Alemu'.

Finally, let's look at how to declare an INT variable in SQL Server and assign an initial value. For example:

```
DECLARE @studentAge INT = 25;
```

This variable declaration example would declare a variable called @studentAge that is an INT datatype. It would then set the value of the @studentAge variable to the integer value of 25.

Example - Declare more than one variable with an initial value

```
DECLARE @StudentName VARCHAR (50) = 'Abebe Alemu',
@studentAge INT = 25;
```

In this example, we are declaring two variables and both of these variables are assigned initial values in their declaration.

SQL Server: IF...ELSE Statement

In SQL Server, the IF...ELSE statement is used to execute code when a condition is TRUE, or execute different code if the condition evaluates to FALSE.

Syntax: The syntax is for the IF...ELSE statement in SQL Server (Transact-SQL) is:

```
IF condition
   {...statements to execute when condition is TRUE...}
[ELSE
   {...statements to execute when condition is FALSE...} ]
```

ELSE

Optional. You would use the ELSE condition when you want to execute a set of statements when the IF condition evaluated to FALSE (i.e.: the condition was not met).

Note

- Within the IF...ELSE statement, the ELSE condition is optional.
- There is no ELSE IF condition in the IF...ELSE statement. Instead, you will have to nest multiple IF...ELSE statements to achieve the desired effect.

Example – IF...ELSE Statement

```
DECLARE @studentAge INT;
SET @studentAge = 15;
IF @studentAge < 25
   PRINT 'correct';
ELSE
   PRINT 'incorrect';
GO
```

Example – No ELSE condition

```
DECLARE @studentAge INT;
SET @studentAge = 15;
IF @studentAge < 25
   PRINT 'correct';
GO
```

Example – Nested IF...ELSE Statements

```
DECLARE @studentAge INT;
SET @studentAge = 15;
IF @studentAge < 25
   PRINT 'correct;
ELSE
BEGIN
   IF @studentAge < 50
      PRINT 'min;
   ELSE
      PRINT 'max';
END;
GO
```

SQL Server: WHILE LOOP

Description

In SQL Server, you use a WHILE LOOP when you are not sure how many times you will execute the loop body and the loop body may not execute even once.

Syntax: The syntax for the WHILE LOOP in SQL Server (Transact-SQL) is:

```
WHILE condition
BEGIN
   {...statements...}
END;
```

For example:

```
DECLARE @studentAge INT;
SET @studentAge = 0;
WHILE @studentAge <= 10
BEGIN
   PRINT 'Inside WHILE LOOP';
   SET @studentAge = @studentAge + 1;
END;
PRINT 'Done WHILE LOOP';
GO
```

In this WHILE LOOP example, the loop would terminate once the @studentAge exceeded 10 as specified by:

```
WHILE @studentAge <= 10
```

SQL Server: FOR LOOP

In SQL Server, there is no FOR LOOP. However, you simulate the FOR LOOP using the WHILE LOOP.

Syntax: The syntax to simulate the FOR Loop in SQL Server (Transact-SQL) is:

```
DECLARE @a INT = 0;
WHILE @a < a_total
BEGIN
```

```
   {...statements...}
   SET @a = @a + 1;
END;
```

For example:

```
DECLARE @a INT = 0;

WHILE @a < 10
BEGIN
   PRINT 'Inside simulated FOR LOOP';
   SET @a = @a + 1;
END;
PRINT 'Done simulated FOR LOOP';
GO
```

SQL Server: BREAK Statement

In SQL Server, the BREAK statement is used when you want to exit from a WHILE LOOP and execute the next statements after the loop's END statement.

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

```
BREAK;
```

- You use the BREAK statement to terminate a WHILE LOOP early.
- If there are nested WHILE LOOPs, the BREAK statement will terminate the innermost WHILE LOOP.
- See also the WHILE LOOP and CONTINUE statement.

For example:

```
DECLARE @a INT;
SET @a = 0;
WHILE @a <= 10
BEGIN
   IF @a = 2
      BREAK;
   ELSE
      PRINT 'Inside WHILE LOOP';
   SET @a = @a+ 1;
END;
PRINT 'Done WHILE LOOP';
GO
```

SQL Server: CONTINUE Statement

In SQL Server, the CONTINUE statement is used when you are want a WHILE LOOP to execute again. It will ignore any statements after the CONTINUE statement

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

```
CONTINUE;
```

You use the CONTINUE statement to restart a WHILE LOOP and execute the WHILE LOOP body again from the start.

For example:

```
DECLARE @a INT;
SET @a = 0;
WHILE @a <= 10
BEGIN
   IF @a = 2
      BREAK;
   ELSE
   BEGIN
      SET @a = @a + 1;
      PRINT 'Inside WHILE LOOP';
      CONTINUE;
   END;
END;
PRINT 'Done WHILE LOOP';
GO
```

SQL Server: GOTO Statement

The GOTO statement causes the code to branch to the label after the GOTO statement.

Syntax: The syntax for the GOTO statement in SQL Server (Transact-SQL) consists of two parts - the GOTO statement and the Label Declaration:

GOTO statement

The GOTO statement consists of the GOTO keyword, followed by a label_name.

```
GOTO label_name;
```

Label Declaration

The Label Declaration consists of the label_name, followed by at least one statement to execute.

```
label_name:
 {...statements...}
```

Note

- Label_name must be unique within the scope of the code.
- There must be at least one statement to execute after the Label Declaration.

For example:

```
DECLARE @a INT;
SET @a = 0;
WHILE @a <= 10
BEGIN
   IF @a = 2
      GOTO MyPoint;
   SET @a = @a + 1;
END;
MyPoint:
   PRINT 'You Got It';
GO
```

To be continued...