**T-SQL PROGRAMMING**

**Structure of TSQL Program:**

**DECLARE**

**- - - -**

**BEGIN**

**-- we can also declare variables here**

**- - any sql and program statements**

**END**

**DECLARE**

**Declare block is used to declare the variable.**

**Variable is an Identifier which is used to identify the value.**

**Syntax:**

**DECLARE @var datatype(size);**

**DECLARE @var datatype(size), @var datatype(size);**

**BEGIN**

**Begin block is used to set the value for the variable.**

**Syntax:**

**Set @variablename= value**

**END**

**It is indicating end of your program.**

**Q) Write a sample program to display welcome message?**

DECLARE

@V1 VARCHAR(100)

BEGIN

SET @V1='WELCOME TO SQL SERVER PROGRAMMING';

PRINT @V1;

END;

Output: WELCOME TO SQL SERVER PROGRAMMING

DECLARE

@V1 VARCHAR(100)

BEGIN

SET @V1='WELCOME TO SQL SERVER PROGRAMMING'

PRINT 'Posted message is: '+@V1

END

Note:

# In the above program , + symbol is known as appending operator.

--Write a program to display product name, price, ---warranty of product id “p1” ?

DECLARE

@VPID VARCHAR(10)

BEGIN

SET @VPID='p1'

SELECT prodid,PNAME,price,warrenty FROM PRODUCTS WHERE PRODID=@VPID

end;

OTHER METHOD:

BEGIN

DECLARE @VPID VARCHAR(10), @V1 VARCHAR(10), @V2 MONEY, @V3 VARCHAR(10);

SET @VPID='ILTPQS'

SELECT @V1=PNAME,

@V2=price,

@V3=warrenty FROM PRODUCTS WHERE pid=@VPID

PRINT ' PID: '+@VPID

PRINT ' NAME: '+@V1

PRINT ' PRICE: '+CAST(@V2 AS VARCHAR(10))

PRINT ' WARRENTY: '+@V3

end;

**Q) Write a program to declare first name and last name and display full name?**

declare

@fname varchar(20),

@lname varchar(20),

@fullname varchar(20)

begin

set @fname='Sathya'

set @lname='Technlogies'

set @fullname=@fname++@lname

print 'full name is '+@fullname

end;

**Note:**

**IN the above program, ++ symbol is used to add 2 strings.**

**Q) Write a Program to perform addition of two numbers?**

declare

@fno int,

@sno int,

@add int

begin

set @fno=10

set @sno=20

set @add=@fno+@sno

print 'Sum is '+cast(@add as varchar(20))

end;

**Note:**

**In the above Program, CAST is a conversion function,**

**To convert numeric value into character value.**

**Q) Write a program to display employee name of employee id 1111?**

declare

@veid int ='1111',

@vename varchar(20)

begin

set @vename=(select ename from emps where eid=@veid)

print ' Given empid: '+cast(@veid as varchar(4))

print ' Employee Name: '+@vename

end

**Q) Write a procedure to display number of emps and total salary?**

DECLARE

@vcount int,

@vtotal numeric

BEGIN

Set @vcount=(select count(\*) from emps)

Set @vtotal=(select sum(sal) from emps)

Print 'Number of emps: '+cast(@vcount as varchar(10))

Print 'Total salary: '+cast(@vtotal as varchar(10))

end

Q) Write a Program to declare two numbers and check which number is greater?

declare

@a int,

@b int

begin

set @a=10

set @b=20

if(@a>@b)

print cast(@a as varchar(20))+' is big'

else

print cast(@b as varchar(20))+' is big'

end;

**STORED PROCEDURES**

**Stored Procedure is a program, that is saved under database server permanently.**

**We can use or execute the procedures repeatedly.**

**Syntax:**

**Create procedure <name>**

**As**

**---**

**---**

**Go**

**How to execute a procedure?**

**Exec <proc\_name>**

**Ex:**

**Write a procedure to display all company details?**

create procedure get\_all\_companies

as

select \* from company

go;

exec get\_all\_companies

**Q) Create Procedure to insert the record in Employee table**

create procedure emp\_reg

(

@veid numeric(4),

@vename varchar(20),

@vsal numeric(5),

@vjdate date,

@vdesg varchar(20),

@vegender char

)

as

insert into emps values(@veid,@vename,@vsal,@vjdate,@vdesg,@vegender)

go

exec emp\_reg 4545,'harini',34000,'2021-11-20','developer','F'

**How to display list of procedure names?**

SELECT ROUTINE\_NAME

FROM mydb1.INFORMATION\_SCHEMA.ROUTINES

WHERE ROUTINE\_TYPE = 'PROCEDURE'

**3) create a stored procedure to validate username and password**

CREATE PROCEDURE Verify @username nchar(20), @password nchar(20)

AS

IF EXISTS(SELECT \* FROM Login where Username=@username and Password=@password)

PRINT('User Exists')

ELSE

PRINT('User Does not Exist')

GO

**EXEC Verify @username='David', @password='David123'**

**GO**

**Ex:**

**-- write a procedure to display emp information based on given empid?**

create procedure get\_empinfo @veid int

as

begin

declare @vename varchar(10),

@vsal money,

@vjoindate date,

@vdesg varchar(10),

@vgender char

select @vename=ename,

@vsal=sal,

@vjoindate=jdate,

@vdesg=desg,

@vgender=egender

from emps

where eid=@veid;

Print ' Given empid: '+ cast(@veid as varchar(10))

print ' ------------------------------------------'

print 'Name: '+@vename

print 'Salary: '+cast(@vsal as varchar(10))

print 'Working as: '+@vdesg

print 'Join dt: '+cast(@vjoindate as varchar(10))

print 'gender: '+@vgender

print '--------------------------------------------'

end;

Go

Sample Output:

Given empid: 1121

------------------------------------------

Name: haritha

Salary: 60920.00

Working as: developer

Join dt: 2018-06-20

gender: F

--------------------------------------------

Completion time: 2022-07-11T19:26:08.8634697+05:30

EX:

write a procedure to display number of emps and total salary under given designation?

create procedure get\_emp\_count\_total\_salary @vdesg varchar(10)

as

begin

declare @vcnt int, @vtotal money

set @vcnt=(select count(\*) from emps)

set @vtotal=(select sum(sal) from emps)

print 'Given designition: '+@vdesg

Print 'Number of emps: '+ cast(@vcnt as varchar(10))

Print 'Total salary: '+cast(@vtotal as varchar(10))

Print ' End of OutPut '

END;

SAMPLE OUTPUT:

Given designition: DEVELOPER

Number of emps: 10

Total salary: 361900.00

End of OutPut

Completion time: 2022-07-11T19:33:44.2816619+05:30

--WRITE A PROCEDURE TO DISPLAY PRODUCT INFORMATION LIKE PRODUCT ID, PRODUCT NAME, PRICE, manufacturing date, warranty and COMPANY NAME of given product id?

create procedure proc\_get\_product\_details @vpid varchar(10)

as

begin

declare @vpname varchar(10),

@vprice numeric(10,2),

@vmfgdt date,

@vwarrenty varchar(10),

@vcmpname varchar(10)

select

@vpname=p.pname,

@vprice=p.price,

@vmfgdt=p.mdgdt,

@vwarrenty=p.warrenty,

@vcmpname=c.cmpname

from products p, company c

where p.prodid=@vpid and p.cmpid=c.cmpid;

print ' Given product id: '+@vpid

print ' --------------------------'

print ' Product Name: '+@vpname

print ' Price: '+cast(@vprice as varchar(10))

print ' Mfg Date: '+cast(@vmfgdt as varchar(10))

print ' warrenty: '+@vwarrenty

print ' Company Name: '+@vcmpname

print '------------------------------------'

end;

Sample Output:

**exec proc\_get\_product\_details p1**

**Given product id: p1**

**--------------------------**

**Product Name: tv**

**Price: 45000.00**

**Mfg Date: 2020-11-23**

**warrenty: 5 years**

**Company Name: sony**

**----------------------------------------**

**Completion time: 2022-07-11T19:48:30.4971435+05:30**

**CONDITIONAL STATEMENTS**

We can write conditions in a program under conditional statements. Based on condition result the set of statements will be executed.

**IF Boolean\_expression**

**BEGIN**

**{ statement\_block }**

**END**

In this syntax, if the Boolean expression evaluates to TRUE then the statement block in the [BEGIN...END](https://www.sqlservertutorial.net/sql-server-stored-procedures/sql-server-begin-end/) block is executed. Otherwise, the statement block is skipped and the control of the program is passed to the statement after the END keyword.

**IF ELSE statement:**

IF Boolean\_expression

BEGIN

-- Statement block executes when the Boolean expression is TRUE

END

ELSE

BEGIN

-- Statement block executes when the Boolean expression is FALSE

END

**Ex:**

**Write a program to find the biggest of 2 numbers?**

declare

@x int,

@y int

begin

set @x=100

set @y=200

if @x > @y

print 'x is greater than y'

else

print 'y is greater than x'

end;

**Ex:**

**Write a procedure to find which employee has less salary from given 2 employee numbers?**

select\* from emps

Create procedure proc\_find\_high\_salary

(

@vempid1 decimal(10),

@vempid2 decimal(10,2)

)

as

declare

@s1 decimal(10,2),

@s2 decimal(10,2)

begin

set @s1 = (select sal from emps where [eid]=@vempid1);

set @s2 = (select sal from emps where [eid]=@vempid2);

print ' empid-1: '+cast(@vempid1 as varchar(10))

print ' empid-2: '+cast(@vempid2 as varchar(10))

print ' salary-1: '+cast(@s1 as varchar(10))

print ' salary-2: '+cast(@s2 as varchar(10))

if @s1 > @s2

print ' employee 1 has higher salary'

else if @s1 = @s2

print ' Both emploees are having equal salary'

else

print 'Employee 2 has higher salary'

end;

exec proc\_find\_high\_salary 1191,1120

OUTPUT:

empid-1: 1191

empid-2: 1120

salary-1: 10000.00

salary-2: 10000.00

Both emploees are having equal salary

**LOOPS**

**WHILE {condition holds}  
BEGIN  
{…do something…}  
END;**

**EX:**

**WRITE A PROGRAM TO PRINT 1 TO 10 NUMBERS?**

begin

declare @v int=1

while @v<=10

begin

print @v

set @v=@v+1

end;

end;

use mydb1

select table\_name from information\_schema.tables

Ex:

Write a procedure to display employee names from the id 1110 to the id 1120

select \* from emps

alter procedure proc\_names\_while

as

begin

declare @vempid numeric(5)=1110

declare @vename varchar(20)

while @vempid<=1120

begin

set @vename=(select ename from emps where eid=@vempid)

print 'empid: '+cast(@vempid as varchar(10))+' empname='+@vename

set @vempid=@vempid+1

end

print ' Task completed'

END

GO

EXEC proc\_names\_while

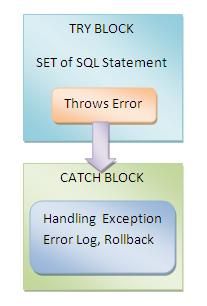
**EXCEPTIONS**

An error condition during a program execution is called an exception and the mechanism for resolving such an exception is known as exception handling.

SQL Server provides **TRY**, **CATCH** blocks for exception handling.

We can put all **T-SQL statements into a TRY BLOCK** and the code for **exception handling can be put into a CATCH block.**

We can also generate user-defined errors using a THROW block.



BEGIN TRY

     --code to try

END TRY

BEGIN CATCH

     --code to run if an error occurs

--is generated in try

END CATCH

Anything between the BEGIN TRY and END TRY is the code that we want to monitor for an error. So, if an error occurred inside this TRY statement, then the control immediately transferred to the CATCH

statement and then it would have started executing code line by line.

BEGIN TRY

-- Generate a divide-by-zero error

  SELECT

    1 / 0 AS Error;

END TRY

BEGIN CATCH

  SELECT

    ERROR\_NUMBER() AS ErrorNumber,

    ERROR\_STATE() AS ErrorState,

    ERROR\_SEVERITY() AS ErrorSeverity,

    ERROR\_PROCEDURE() AS ErrorProcedure,

    ERROR\_LINE() AS ErrorLine,

    ERROR\_MESSAGE() AS ErrorMessage;

END CATCH;

GO

Ex:

BEGIN TRY

-- Generate a divide-by-zero error

  SELECT

    1 / 0 AS Error;

END TRY

BEGIN CATCH

  SELECT

    ERROR\_NUMBER() AS ErrorNumber,

    ERROR\_LINE() AS ErrorLine,

    ERROR\_MESSAGE() AS ErrorMessage;

END CATCH;

GO

use mydb1

select table\_name from information\_schema.tables

select \* from emps

BEGIN

DECLARE @vename varchar(20),@vsal numeric(5),@veid numeric(4)

begin try

set @veid=1111

set @vename=(select ename from emps where [eid]=@veid)

set @vsal=(select sal from emps where eid=@veid)

print ' empid '+@veid

print ' empname '+@vename

print ' salary= '+@vsal

end try

begin catch

SELECT

ERROR\_NUMBER() AS ErrorNumber,

ERROR\_LINE() AS ErrorLine,

ERROR\_MESSAGE() AS ErrorMessage;

END CATCH;

End;

**Note:**

**In the above program, we have some conversion errors. We can see that error information after execution of above program.**

**USER DEFINED FUNCTIONS**

 The main idea behind functions is to store them in the database and avoid writing the same code over and over again. Also, you can control what is the input and define the structure/type of output.

CREATE FUNCTION [database\_name.]function\_name (parameters)

RETURNS data\_type

AS

BEGIN

    SQL statements

    RETURN value

END;

ALTER FUNCTION [database\_name.]function\_name (parameters)

RETURNS data\_type

AS

BEGIN

    SQL statements

    RETURN value

END;

DROP FUNCTION [database\_name.]function\_name;

Ex:

Write a function to get salary of employee from given emp id?

create function func\_salary(@veid numeric)

returns numeric

as

begin

declare @vsal numeric;

select @vsal=sal from emps where eid = @veid;

return(@vsal);

end;

How to execute the above function?

select dbo.func\_salary(1110) as sal;

**CURSORS**

**Cursor** is a Temporary Memory or Temporary Work Station.

It is Allocated by Database Server at the Time of Performing DML(Data Manipulation Language) operations on Table by User.

There are 2 types of Cursors:

Implicit Cursors

Explicit Cursors.

**Implicit Cursors:**  
Implicit Cursors are also known as Default Cursors of SQL SERVER.

These Cursors are allocated by SQL SERVER when the user performs DML operations.

**Explicit Cursors:**  
Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner.

**How to create Explicit Cursor:**

1. **Declare Cursor Object.**  
   **Syntax :**

DECLARE cursor\_name CURSOR FOR SELECT \* FROM table\_name

DECLARE s1 CURSOR FOR SELECT \* FROM studDetails

1. **Open Cursor Connection.**  
   **Syntax :** OPEN cursor\_connection

OPEN s1

1. **Fetch Data from cursor.**  
   There are total 6 methods to access data from cursor. They are as follows :  
   **FIRST** is used to fetch only the first row from cursor table.  
   **LAST** is used to fetch only last row from cursor table.  
   **NEXT** is used to fetch data in forward direction from cursor table.  
   **PRIOR** is used to fetch data in backward direction from cursor table.  
   **ABSOLUTE n** is used to fetch the exact nth row from cursor table.  
   **RELATIVE n** is used to fetch the data in incremental way as well as decremental way.  
   **Syntax :** FETCH NEXT/FIRST/LAST/PRIOR/ABSOLUTE n/RELATIVE n FROM cursor\_name
2. FETCH FIRST FROM s1
3. FETCH LAST FROM s1
4. FETCH NEXT FROM s1
5. FETCH PRIOR FROM s1
6. FETCH ABSOLUTE 7 FROM s1
7. FETCH RELATIVE -2 FROM s1
8. **Close cursor connection.**  
   **Syntax :** CLOSE cursor\_name

CLOSE s1

1. **Deallocate cursor memory.**  
   **Syntax :** DEALLOCATE cursor\_name

DEALLOCATE s1

Ex:

Write a procedure to display all employee names?

Create procedure proc\_enames\_cursors

As

Begin

Declare @vename varchar(10)

Declare ecur CURSOR FOR select ename from emps;

Open ecur;

Fetch next from ecur into @vename;

While @@FETCH\_STATUS = 0

BEGIN

Print @vename

Fetch next from ecur into @vename;

End

Close ecur;

Deallocate ecur;

END;

Exec proc\_enames\_cursors;

**TRIGGERS**

The CREATE TRIGGER statement allows you to create a new trigger that is fired automatically whenever an event such as [INSERT](https://www.sqlservertutorial.net/sql-server-basics/sql-server-insert/), [DELETE](https://www.sqlservertutorial.net/sql-server-basics/sql-server-delete/), or [UPDATE](https://www.sqlservertutorial.net/sql-server-basics/sql-server-update/) occurs against a table.

CREATE TRIGGER [schema\_name.]trigger\_name

ON table\_name

AFTER {[INSERT],[UPDATE],[DELETE]}

[NOT FOR REPLICATION]

AS

{sql\_statements}

## **“Virtual” tables for triggers: INSERTED and DELETED**

SQL Server provides two virtual tables that are available specifically for triggers called INSERTED and DELETED tables. SQL Server uses these tables to capture the data of the modified row before and after the event occurs.

There are three main characteristics that make triggers different than stored procedures:

* Triggers cannot be manually executed by the user.
* There is no chance for triggers to receive parameters.
* You cannot commit or rollback a transaction inside a trigger.