CAPSTONE-2

Q1: What is user defined stored procedure and user defined function, explain and write the differences? (5)

**Stored Procedure: - Stored procedures are Transact SQL codes that are stored in databases of SQL server and can be used whenever they are needed. They are stored in the databases with a specific name accordingly either by the user or the Pre-defined by the SQL server and they perform various tasks accordingly. They are basically created in order to not to rewrite the codes again and again and can be helpful in saving time. Stored procedures are created once and can be called multiple times whenever required by the user.**

**Stored procedures are also of two types according to the parameters they take from the user or not. So, this classify them into two types that are Stored Procedures with parameters and Stored Procedure without parameters. Stored Procedures without parameters are just SQL queries which results the desired output and will keep providing the similar output according to the procedure, whereas Stored procedures with parameters will provide results according to the provided input from the user and then will give the output according to the input. The output from the stored procedure will be shown once RETURN is used**

**Stored Procedure is basically divided into 2 types: -**

**1)User Defined Stored Procedure: -These are the Stored Procedures that are created by the user according to their need and can have desired outcomes from it according the user requirement. They can also be called multiple times by the user and they are also used to maintain efficiency by not re-writing the code multiple times and just calling the procedure by its SP name.**

**2)System Stored Procedure: -These are pre-Defined stored procedure by the system of SQL server database and each of them have a functionality according to its name. They always use a database that is named as Master or MSDB Database to be stored in. They always manage SQL server through administrative task.**

**They always use a prefix to be called into execution and that is SP\_.**

**Difference in UDF & SSP: -**

**UDF: -**

* **They are generally programmed to return a result or an outcome by the user.**
* **Only allow Select statements and DML commands are not allowed.**
* **UDF doesn’t allow try and catch for exception handling.**
* **UDF doesn’t allow table variable and temporary tables.**
* **UDF will only allow input of parameter and doesn’t allow output from the same parameter as input.**
* **UDF can be used in creating a join clause.**

**SSP: -**

* **SSP does have some allowance on whether or not to return a value and it depends on SP if it has a value to return or not.**
* **SSP does allow the use of select statement as well as DML commands.**
* **SSP allows table variables as well as temporary tables.**
* **SSP allow the use of try and catch for exception handling.**
* **SSP allows both input and output parameters.**
* **SSP can’t be used in creating a join clause.**

Q2: what are advantage of using stored procedure? (5)

**Stored Procedure: - Stored procedures are Transact SQL codes that are stored in databases of SQL server and can be used whenever they are needed. They are stored in the databases with a specific name accordingly either by the user or the Pre-defined by the SQL server and they perform various tasks accordingly. They are basically created in order to not to rewrite the codes again and again and can be helpful in saving time. Stored procedures are created once and can be called multiple times whenever required by the user.**

**Stored procedures are also of two types according to the parameters they take from the user or not. So, this classify them into two types that are Stored Procedures with parameters and Stored Procedure without parameters. Stored Procedures without parameters are just SQL queries which results the desired output and will keep providing the similar output according to the procedure, whereas Stored procedures with parameters will provide results according to the provided input from the user and then will give the output according to the input. The output from the stored procedure will be shown once RETURN is used**

**There are multiple advantages of using a stored procedure: -**

* **Stored procedures are only compiled once and after that they just need to recalled whenever needed.**
* **They are compiled in a single execution plan.**
* **Stored procedures allow user to give desired input and get the output needed.**
* **Procedures can easily be transferred by just copying the set of queries or just calling the stored procedures wherever needed so they have better mobility.**
* **Stored Procedures are generally efficient for the user as there is no use of re-writing the code multiple times and just calling the function.**
* **Stored procedures take multiple parameter options and makes it dynamic SQL coding for the user.**
* **On the basis of security procedures can be restricted by the administrative rights to be manipulated by admin privileges.**
* **Because of just a single compilation process it is always way quicker than writing the code and compiling it again and again.**

Q3: what is your understanding with complex stored procedure explain with example. (5)

**Stored Procedures** :- **Stored procedures are Transact SQL codes that are stored in databases of SQL server and can be used whenever they are needed. They are stored in the databases with a specific name accordingly either by the user or the Pre-defined by the SQL server and they perform various tasks accordingly. They are basically created in order to not to rewrite the codes again and again and can be helpful in saving time. Stored procedures are created once and can be called multiple times whenever required by the user.**

**Complexity in Stored Procedures arises with the stored procedures containing multiple clauses, SET operations, Joins as well as multiple subqueries which increases the vast reach due to which multiple data sets can be captured can multiple values can be fetched from various data sets.**

**Example For Complex Stored Procedure :-**

create procedure Forlength(@EmplID int)

as

begin

declare @lengthOfName int;

set @lengthofname = (select len(employee\_name) from employee where employee\_ID=@EmplID)

select employee\_ID, employee\_name,gender,salary,@lengthOfName as 'Length of Characters' from Employee where Employee\_ID=@EmplID;

end

Syntax for Execution :-

exec Forlength @EmplID = 2;

Q4: what are system defined stored procedure, list any five and explain. (5)

**These are pre-Defined stored procedure by the system of SQL server database and each of them have a functionality according to its name. They always use a database that is named as Master or MSDB Database to be stored in. They always manage SQL server through administrative task.**

**They always use a prefix to be called into execution and that is SP\_.**

**There are multiple System Stored Procedures in the database and all of them have different functionality which are generally related to manipulation of the queries and databases and tables to get the desired output by the user**

**Some of the System Stored Procedures are with the parameters they take are as follows: -**

1. **SP\_RENAME: -This SP is used to rename database objects like tables , views or procedure names and takes multiple parameters.**

**sp\_rename @objectname= ‘objectname, @newname= ‘newname’;**

**Example :- exec sp\_rename ‘Department.city’,’Location’,’Column**

1. **SP\_CHANGEOWNER :- This Stored Procedure is used to change the owner of a database object.**

**Sp\_changeobjectowner[@objname=]’object’,[@newowner=]’owner’**

**Example :- exec sp\_changeobjectowner = ‘author’,’corporate\GeorgeW’;**

1. **SP\_HELP :-This SP is used to give the information about a database object , a data type or a user defined data type.**

**SP\_HELP [[ObjName]=`Name’]**

**Example :- use ‘Master’(Database Name)`**

**Go**

**Exec sp\_help**

**Go**

1. **SP\_TABLES:- This SP is used to return a list of objects in an environment that can be used in a query or queried , this is applicable for any table or view.**

**Sp\_tables[[@tablename=]’name’]**

**[,[@tableowner=]’owner’]**

**[,[@table\_qualifier=]’qualifier’]**

**[,[@table\_type=]’type’];**

**Example :-**

**Column name Data type Description**

**TABLE\_QUALIFIER sysname Table qualifier name. In SQL Server, this column represents the database name. This field can be NULL.**

**TABLE\_OWNER sysname Table owner name. In SQL Server, this column represents the name of the database user who created the table. This field always returns a value.**

**TABLE\_NAME sysname Table name. This field always returns a value.**

**TABLE\_TYPE varchar(32) Table, system table, or view.**

**5 .SP\_DATABASES :-This SP returns the list of databases that are either present in an instance of the SQL server or are accessible through the gateway**

**SP\_EXEC DATABASES;**

**EX:- USE MASTER**

**GO**

**EXEC SP\_DATABASES**

Q5: what type of triggers, and on what event you can pass trigger.(5)

**Triggers :- Triggers are special Stored Procedures which are used perform a certain set of operations using queries during or after certain DDL or DML commands. Triggers are fired automatically whenever a user invokes a certain DDL or DML command on a specific entity or object that is linked with the trigger during the creation of this stored procedure .These linked entities or objects can be 1 or more than one. Triggers can also be of two types that are with parameters and without parameters.**

**The execution of a trigger happens only when the event with which the trigger is associated is activated or executed.**

**Syntax:-**

**CREATE TRIGGER trigger\_name**

**ON {table|view}**

**[WITH ENCRYPTION|EXECUTE AS]**

**{FOR|AFTER|INSTEAD OF} {[CREATE|ALTER|DROP|INSERT|UPDATE|DELETE ]}**

**AS**

**BEGIN**

**sql\_statements{}**

**END**

**GO**

**There are 4 types of triggers that can be created by the user and all of them have different working accordingly, these 4 types of triggers are:-**

**1.Data Definition Language Triggers(DDL Triggers):-DDL commands triggers can be created in the SQL environment and stored procedures that perform DDL operations like**

* **Create**
* **Alter**
* **Drop**

**During the use of DDL commands in triggers only FOR/AFTER clause can be used that means only AFTER clause can be used in DDL statements.**

**DDL commands are used to manage administrative tasks like database operations.**

**Syntax For DDL Triggers :-**

**CREATE TRIGGER trigger\_name**

**ON {table|view}**

**[WITH ENCRYPTION|EXECUTE AS]**

**{FOR|AFTER } {[CREATE|ALTER|DROP]}**

**AS**

**BEGIN**

**sql\_statements{}**

**END**

**GO**

**2.Data Manipulation Language Triggers(DML Triggers):-DML commands triggers can be created in the SQL environment and stored procedures that perform DML operations like**

* **Insert**
* **Update**
* **Delete**

**DML commands triggers are also of two types:-**

1. **After Trigger :-These are the types of triggers that are fired after the execution of statement or action that triggered it.**
2. **Instead of Trigger :- These are the type of triggers that are fired before the execution of the statement or action that triggered it.**

**CREATE TRIGGER trigger\_name**

**ON {table|view}**

**[WITH ENCRYPTION|EXECUTE AS]**

**{FOR|AFTER|INSTEAD OF} {[ INSERT|UPDATE|DELETE ]}**

**AS**

**BEGIN**

**sql\_statements{}**

**END**

**GO**

**3.CLR Triggers:- CLR triggers are a special type of stored procedure that automatically runs when a language event executes. This works with the .NET Framework common language after the integration of Microsoft SQL Server integration with .NET Framework Common Language Runtime(CLR).**

**These triggers are allowed to be coded in .NET languages like C# ,Visual Basics & F#. We can write both DDL & DML trigger queries using supported CLR language like C#, Visual Basics, F#.**

**4.Logon Triggers:-Logon Triggers are the type of triggers that are fired or executed during a LOGON even in a SQL server. LOGON Trigger is activated on the event when the user session is being established and that is after the authentication process finishes but before the actual connection is established and during this connection all the error message that are occurred during this trigger will be saved in the error log and during this trigger event if any authentication fails during the LOGON then the trigger fails.**

**Syntax For LOGON Triggers:-**

**CREATE TRIGGER trigger\_name**

**ON ALL SERVER**

**[WITH ENCRYPTION]**

**{FOR|AFTER} LOGON**

**AS**

**sql\_statement [1...n ]**

Q6: write about recursive trigger.(5)

**Recursive Triggers :- Recursive Triggers are the types of triggers which calls itself repeatedly and due this trigger calling itself again and again leads to an infinite loop of a trigger calling itself again and again and due to recursion or nesting of this trigger there are chances that this Recursive trigger will hit the governor limit.**

**These nested triggers in SQL Server are automatically executed on action when a certain database is created. They execute as a result of DDL commands(Create, Alter, Drop) or DML commands(Insert, Update, Delete).**

**Nested Triggers are classified into two types :-**

* **After Triggers :- These triggers are executed after a DDL or DML command is executed is performed.**
* **Instead Of Triggers:- These triggers are executed in place of DDL or DML commands.**

**So the triggers that are triggered by another triggers is when nesting of trigger comes into the play and this cause nesting of Triggers or recursion of triggers that is NESTED TRIGGERS OR RECURSIVE TRIGGERS.**

**Example for a Recursive Trigger :-**

**CREATE DATABASE Showroom**

**GO**

**Use Showroom**

**CREATE TABLE Car**

**(**

**CarId int identity(1,1) primary key,**

**Name varchar(100),**

**Make varchar(100),**

**Model int ,**

**Price int ,**

**Type varchar(20)**

**)**

**insert into Car( Name, Make, Model , Price, Type)**

**VALUES ('Corrolla','Toyota',2015, 20000,'Sedan'),**

**('Civic','Honda',2018, 25000,'Sedan'),**

**('Passo','Toyota',2012, 18000,'Hatchback'),**

**('Land Cruiser','Toyota',2017, 40000,'SUV'),**

**('Corrolla','Toyota',2011, 17000,'Sedan')**

**CREATE TABLE CarLog**

**(**

**LogId int identity (1,1) primary key,**

**CarId int,**

**CarName varchar (100),**

**)**

**CREATE TRIGGER [dbo].[CarLOG\_INSERT]**

**ON [dbo].[CarLog]**

**INSTEAD OF INSERT**

**AS**

**BEGIN**

**PRINT('DATA CANNOT BE INSERTED DIRECTLY IN CarLog TABLE')**

**END**

**CREATE TABLE CarLog**

**(**

**LogId int identity (1,1) primary key,**

**CarId int,**

**CarName varchar (100),**

**)**

**CREATE TRIGGER [dbo]. [CAR\_INSERT]**

**ON [dbo]. [Car]**

**AFTER INSERT**

**AS**

**BEGIN**

**SET NOCOUNT ON;**

**DECLARE @car\_id INT, @car\_name VARCHAR (50)**

**SELECT @car\_id = INSERTED.CarId, @car\_name = INSERTED.name**

**FROM INSERTED**

**INSERT INTO CarLog**

**VALUES (@car\_id, @car\_name)**

**END**

Q7: explain importance of indexes, on OLTP system and OLAP system. (5)

**Indexing: - Indexes are an object in the environment with which the server gets boost in retrieval of data and it retrieves data faster. The data is retrieved row by row with the help of a pointer. It creates a rapid path access to locate data faster and reduces disk I/O.**

**Indexes speed up select queries and where clauses to execute faster but it slows down input with update and insert statement. Indexes can be created with no effect on data or tables.**

**Syntax: -**

**CREATE INDEX index**

**ON TABLE column;**

**OLAP & OLTP System**

**OLTP (Online Transaction Processing System) :-It is an online processing system that is used for online database modifying system. It is focused on Insertion, Updating & Deletion. It is traditional DBMS data that is always detailed and organized. OLTP systems manages transaction-oriented applications on the internet like ATM, Online Banking, Sending a text, Entry Operations.**

**OLAP (Online Analytical Processing System):-It is an online processing system that is used for extraction of data from large databases and analyzing it for decision making. OLAP helps in problem solving , decision support , planning. OLAP is preferable for reading operation than writing for select queries. Transactions in OLAP are less frequent as compared to OLTP.OLAP is preferred in financial reporting, forecasting.**

**OLAP uses Column Stored Indexing and they use more read operations, Column stored indexes are better for performing sequential**

**Difference in OLAP vs OLTP according to indexes: -**

**1.Time: -Processing time in OLAP systems will be higher as OLAP fetches data from different OLTPs and all the OLTPs with indexing will retrieve data or execute queries from its row-by-row selection much faster than OLAP systems which will retrieve its data from different indexes of OLTP systems.**

**2.Queries: - Queries in OLTP will be executed faster as the OLTP is focused on updating, deletion & Insertion & if there is Indexing in the database object then it will execute queries faster where as in OLAP systems it is preferable to opt for reading only so Select queries will be executed faster in OLAP systems with indexing.**

**3.Normalization: - OLTP system always a normalization form of 3NF which means all the tables will be properly normalization i.e., all there won’t be any data redundancy that means no duplication of data that will help the indexing to result in precise data rather than resulting duplicate data where as in OLAP there won’t be any normalization and the tables and entities will be Denormalized so even with proper indexing there is a chance of resulting of duplicate data.**

**4.Data: - OLTP system indexing will have the source data so indexing in source data will be faster than OLAP that will be the online access to the OLTP system with which even with indexing the time frame will be higher but the approach from OLAP system for data with indexing will be much higher and a vast set of data can be accessed with the help of OLAP systems.**

Q7: what is difference between rowstore and columnstore indexes. (5)

**Rowstore Indexing: -**

* **Rowstore indexing is the basic way of storing the index of entities in a database.**
* **This way of storing indexes were designed in order to retrieve data faster by enabling queries to locate data faster by indexing rather than doing a scan for the whole table.**
* **The physical storage is row oriented for rowstore data and is organized by rows and columns.**
* **Rowstore indexes in SQL Server uses B-Tree Structure for data pages.**
* **Rowstore indexes perform best for queries that search for particular values or a small range of values.**
* **Rowstore indexes tend to be better for online transaction processing workload and it uses update and seek operations.**
* **Rowstore indexing are better at performing random read and writes.**

**Columnstore Indexing: -**

* **Column Indexing also similarly has organized data in the row and column format but in this the data is physically stored in a column format.**
* **These column stored indexes are efficient with read only queries with large data sets. Columns stored Indexes don’t work the best with seeking individual values.**
* **Columns contain data that enables the data to be highly compressed, improving memory utilization & reduces disk usage.**
* **Clustered and Non-Clustered Indexes work the same just the clustered columnstore index provides primary storage for the table whereas rowstore index provides secondary index.**
* **A clustered columnstore index can have one or more non clustered B tree index. Columnstore Indexing are better for Online Analytical Processing (OLAP).**
* **Columnstore indexes are better for performing sequential read and writes.**

Q8: write script of merge join and write the importance. (5)

**Physical Joins :- Physical Joins are the type of joins in which in the SQL queries the user don’t write the user don’t use/write the and rather than use/write these Joins are executed inside SQL engine as algorithms or operators for the implementation of logical joins.Physical joins can be viewed under execution plan tab is SSMS  
These physical joins are further divided into 3 different categories :-**

**1.Nested Loop**

**2.Hash Join**

**3. Merge Join**

**Merge Joins: - Merge Joins are the types of joins where in the join the output that is provided by joining 2 sorted data sets in FULL, INNER or LEFT join. During Merge joins both the input data sets should be sorted and the columns from both the data set should have matching meta-data, that means user cannot join two columns in which one column has a numeric data type whereas the other has string data type plus if the first column is a string set, then the column that is being merged with the first one should not have the length of the string set greater than the length string set of the first data.**

**Syntax :-**

**USE pubs**

**GO**

**SELECT a.au\_id**

**FROM authors a**

**JOIN titleauthor b ON a.au\_id = b.au\_id**

**OPTION (MERGE JOIN)**

**GO**

**Merge Joins are very fast with the queries but the cost of performance for completing merge joins with sort operations are really high. If the data is already fetched with the B-Tree indexes then the merge joins are the fastest available algorithm in all the join algorithms as the query optimizer selects the best execution plan for a given select statement but it is not necessary that it will select the desirable join type from the user end and this join type can be changed by usage of the OPTION clause during the join to select the user desirable join type.**

Q9: Write the script for following: (10\*5=50)

a. create stored procedure which take input parameter as businessentityid, then in output, you receive the information of that person. Choose as many related columns for that businessentityid from database adventureworks)

**Creating Procedure** :-

create procedure EMP\_DETAILS(@BID varchar(max) )

as

select perper.BusinessEntityID,

perper.FirstName,

perper.MiddleName,

perper.LastName,

hre.LoginID,

hre.JobTitle,

hrd.[Name] as Dept\_Name,

hrd.GroupName,

hre.BirthDate,

hre.MaritalStatus,

hre.Gender,

hre.HireDate,

hre.ModifiedDate,

pbidcontact.PersonID,

Phonenum.PhoneNumber,

phonenumtype.[Name] as Contact\_Type,

peaddress.EmailAddress,

PCT.ContactTypeID,

PCT.[Name] as Roll,HRED.ShiftID,

HRED.StartDate,HRED.EndDate,

Hred.ModifiedDate,

HRE.VacationHours,

HRE.SickLeaveHours

from

person.Person PerPer join Person.BusinessEntity perBID

on perper.BusinessEntityID=perBID.BusinessEntityID

full join person.BusinessEntityAddress PBIDAddress

on perper.BusinessEntityID=PBIDAddress.BusinessEntityID

full join person.BusinessEntityContact PBIDContact

on perper.BusinessEntityID=PBIDContact.BusinessEntityID

full join person.ContactType PCT

on PBIDContact.ContactTypeID=PCT.ContactTypeID

full join person.EmailAddress PEAddress

on perper.BusinessEntityID=PEAddress.BusinessEntityID

full join Person.Password PPassword

on PerPer.BusinessEntityID=PPassword.BusinessEntityID

full join person.PersonPhone Phonenum

on perper.BusinessEntityID=Phonenum.BusinessEntityID

full join person.PhoneNumberType phonenumtype

on Phonenum.PhoneNumberTypeID = phonenumtype.PhoneNumberTypeID

full join HumanResources.Employee as HRE

on HRE.BusinessEntityID = PerPer.BusinessEntityID

full join HumanResources.EmployeeDepartmentHistory as HRED

on HRE.BusinessEntityID = HRED.BusinessEntityID

full join HumanResources.Department as HRD

on HRD.DepartmentID = HRED.DepartmentID

full join HumanResources.Shift as HRS

on HRED.ShiftID = HRS.ShiftID

where PerPer.BusinessEntityID = @BID

go

**Execution Of Procedure** :-

b. Create audit stored procedure where, whenever insert update or delete command is executed on employee table, then at same time audit table is also populated with those records, and userid,username,curentdate and time value, sessionid.

**Creating Audit Table :-**

create table audit

(

Employee\_ID int,

Employee\_Name varchar(50),

Gender varchar(2),

M\_ID int,

Salary int,

Email varchar(50),

[User\_ID] int,

[Username] varchar(50),

[Time] datetime,

[Session\_ID] int

);

select \* from audit;

go

**Syntax for Insertion:-**

create trigger A\_F\_I\_E

on Employee

after insert

as

begin

declare @EID int;

declare @Ename varchar(50);

declare @Gen varchar(2);

declare @MID int;

declare @Sal int;

declare @EM varchar(50);

declare @U\_ID varchar(50);

declare @UName varchar(50);

declare @timedate datetime;

declare @S\_ID int;

set @EID=(select Employee\_ID from inserted);

set @Ename=(select Employee\_Name from inserted);

set @Gen=(select Gender from inserted);

set @MID=(select M\_ID from inserted);

set @Sal=(select Salary from inserted);

set @EM=(select EMAIL from inserted);

set @U\_ID =(select SUSER\_ID());

set @UName= (select SUSER\_NAME());

set @timedate =(select GETDATE());

set @S\_ID=(select @@SPID);

insert into [audit] values

(@EID,@Ename,@Gen,@MID,@Sal,@EM,@U\_ID,@UName,@timedate,@S\_ID);

End

Inserting into Employee table to check:-

insert into Employee

(Employee\_ID,Employee\_Name,Gender,M\_ID,Salary,Deptno,EMAIL)

values

(12,'JONAH','M',3,34000,20,'jonah@gmail.com');



**Syntax for Updation:-**

create trigger Up\_F\_I\_E

on Employee

for update

as

begin

declare @EID int;

declare @Ename varchar(50);

declare @Gen varchar(2);

declare @MID int;

declare @Sal int;

declare @EM varchar(50);

declare @U\_ID varchar(50);

declare @UName varchar(50);

declare @timedate datetime;

declare @S\_ID int;

set @EID=(select Employee\_ID from inserted);

set @Ename=(select Employee\_Name from inserted);

set @Gen=(select Gender from inserted);

set @MID=(select M\_ID from inserted);

set @Sal=(select Salary from inserted);

set @EM=(select EMAIL from inserted);

set @U\_ID =(select SUSER\_ID());

set @UName= (select SUSER\_NAME());

set @timedate =(select GETDATE());

set @S\_ID=(select @@SPID);

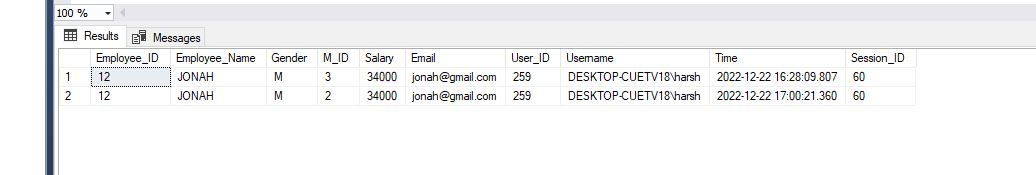
insert into [audit] values

(@EID,@Ename,@Gen,@MID,@Sal,@EM,@U\_ID,@UName,@timedate,@S\_ID);

End

Updating into Employee Table to check Trigger :-

update Employee set M\_ID= 2 where Employee\_ID=12;



**Syntax for Delete Trigger** :-

create trigger D\_F\_I\_E

on Employee

for delete

as

begin

declare @EID int;

declare @Ename varchar(50);

declare @Gen varchar(2);

declare @MID int;

declare @Sal int;

declare @EM varchar(50);

declare @U\_ID varchar(50);

declare @UName varchar(50);

declare @timedate datetime;

declare @S\_ID int;

set @EID=(select Employee\_ID from deleted);

set @Ename=(select Employee\_Name from deleted);

set @Gen=(select Gender from deleted);

set @MID=(select M\_ID from deleted);

set @Sal=(select Salary from deleted);

set @EM=(select EMAIL from deleted);

set @U\_ID =(select SUSER\_ID());

set @UName= (select SUSER\_NAME());

set @timedate =(select GETDATE());

set @S\_ID=(select @@SPID);

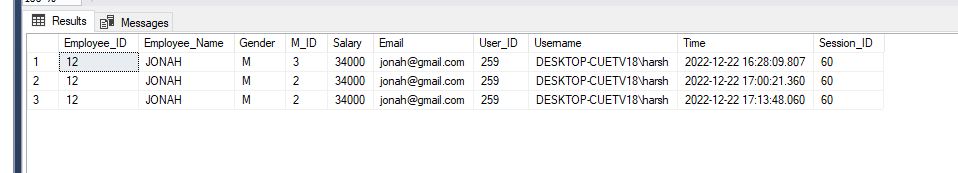
insert into [audit] values

(@EID,@Ename,@Gen,@MID,@Sal,@EM,@U\_ID,@UName,@timedate,@S\_ID);

End

**Deleting from Employee to check** :-

delete from Employee where Employee\_ID=12;



c. create stored procedure with input output and default parameter at same time, use any table, and parameters needed.

**Creation of Procedure** :-

create Procedure Employee\_Fetch(@EMP\_IDin int,@MANIDout int output )

as

begin

declare @manager int;

select a.employee\_ID,a.Employee\_Name, b.DeptName,a.EMAIL

from employee a inner join department b

on a.deptno=b.deptno

where Employee\_ID = @EMP\_IDin

select @manager=a.M\_ID from employee a where Employee\_ID =@EMP\_IDin;

set @MANIDout=@manager

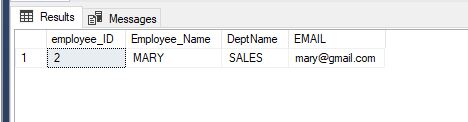
end

**Execution Of Procedure** :-

declare @manager1 int

exec Employee\_Fetch @emp\_IDin = 2,@MANIDout=@manager1 out

print @manager1;



d. create deterministic and non-deterministic user defined function, take any example.

**Deterministic Functions: - Deterministic Functions are the type of Functions that will results as the fixed output value for a fixed input value and it will not change whenever it is used. Any determinism cannot be changed for any built-in SQL function and this applicable for all the deterministic functions that are implemented in the SQL.**

**There are many Deterministic Functions that are built in SQL :-**

1. **Power**
2. **Round**
3. **Exp**
4. **Square**
5. **DateAdd**
6. **ISnull**
7. **DataLength**

**Deterministic Function :-**

**Syntax for Deterministic Function Creation :-**

CREATE FUNCTION dt

( @in INT,@valu int,@pow int )

RETURNS VARCHAR(50)

AS

BEGIN

DECLARE @inp int;

declare @val int;

declare @po int;

set @inp=@in;

set @po=@pow;

set @val = (select POWER(@inp,@po));

RETURN @val;

END;

**Execution of Deterministic Function :-**

use Capstone\_1a;

go

select dbo.dt(2,1,3);

go

**Non-Deterministic Functions: -Non-Deterministic Functions are the type of functions will result in different output values with a fixed set of input values provided for them, this is also going to take place even if the database is intact and not changed at all. Any determinism cannot be changed for any built in SQL function and this is applicable for any non-deterministic functions implemented in the SQL.**

**There are many Non-Deterministic Functions that are built in SQL:-**

* 1. **Getdate**
  2. **RAND**
  3. **FORMAT**
  4. **DENSERANK**
  5. **NTILE**
  6. **PERCENTRANK**
  7. **ROWNUMBER**

**Creating Non Deterministic Function**

**Syntax**

CREATE FUNCTION ndt

( @in INT )

RETURNS VARCHAR(50)

AS

BEGIN

declare @val int;

set @val = cast((select getdate())as int);

RETURN @val;

END;

**Syntax for Execution :-**

use Capstone\_1a;

go

select dbo.ndt(1);

go

e. create user defined function so that if any column character length is checked then, user defined function gives output as length of characters for each row.

Syntax for Procedure :

create procedure Forlength(@EmplID int)

as

begin

declare @lengthOfName int;

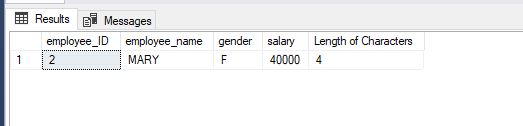
set @lengthofname = (select len(employee\_name) from employee where employee\_ID=@EmplID)

select employee\_ID, employee\_name,gender,salary,@lengthOfName as 'Length of Characters' from Employee where Employee\_ID=@EmplID;

end

Syntax for Execution :-

exec Forlength @EmplID = 2;



Q10: create triggers, to make audit table on DML operation. (use any table)(10)

DML Triggers :-

1Insert

2Update

3Delete

**Creating Audit Table :-**

create table audit

(

Employee\_ID int,

Employee\_Name varchar(50),

Gender varchar(2),

M\_ID int,

Salary int,

Email varchar(50),

[User\_ID] int,

[Username] varchar(50),

[Time] datetime,

[Session\_ID] int

);

select \* from audit;

go

**Insertion Trigger:-**

**Syntax for Insertion:-**

create trigger A\_F\_I\_E

on Employee

after insert

as

begin

declare @EID int;

declare @Ename varchar(50);

declare @Gen varchar(2);

declare @MID int;

declare @Sal int;

declare @EM varchar(50);

declare @U\_ID varchar(50);

declare @UName varchar(50);

declare @timedate datetime;

declare @S\_ID int;

set @EID=(select Employee\_ID from inserted);

set @Ename=(select Employee\_Name from inserted);

set @Gen=(select Gender from inserted);

set @MID=(select M\_ID from inserted);

set @Sal=(select Salary from inserted);

set @EM=(select EMAIL from inserted);

set @U\_ID =(select SUSER\_ID());

set @UName= (select SUSER\_NAME());

set @timedate =(select GETDATE());

set @S\_ID=(select @@SPID);

insert into [audit] values

(@EID,@Ename,@Gen,@MID,@Sal,@EM,@U\_ID,@UName,@timedate,@S\_ID);

End

Inserting into Employee table to check:-

insert into Employee

(Employee\_ID,Employee\_Name,Gender,M\_ID,Salary,Deptno,EMAIL)

values

(12,'JONAH','M',3,34000,20,'jonah@gmail.com');



**Updation Trigger :-**

**Syntax for Updation:-**

create trigger Up\_F\_I\_E

on Employee

for update

as

begin

declare @EID int;

declare @Ename varchar(50);

declare @Gen varchar(2);

declare @MID int;

declare @Sal int;

declare @EM varchar(50);

declare @U\_ID varchar(50);

declare @UName varchar(50);

declare @timedate datetime;

declare @S\_ID int;

set @EID=(select Employee\_ID from inserted);

set @Ename=(select Employee\_Name from inserted);

set @Gen=(select Gender from inserted);

set @MID=(select M\_ID from inserted);

set @Sal=(select Salary from inserted);

set @EM=(select EMAIL from inserted);

set @U\_ID =(select SUSER\_ID());

set @UName= (select SUSER\_NAME());

set @timedate =(select GETDATE());

set @S\_ID=(select @@SPID);

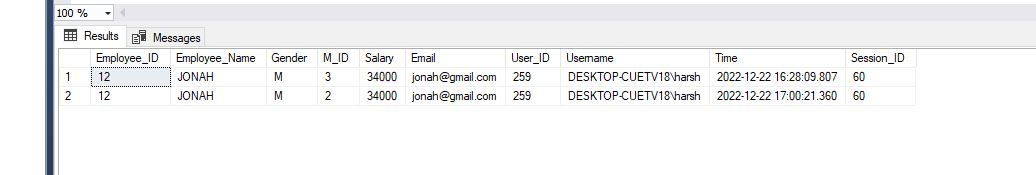
insert into [audit] values

(@EID,@Ename,@Gen,@MID,@Sal,@EM,@U\_ID,@UName,@timedate,@S\_ID);

End

Updating into Employee Table to check Trigger :-

update Employee set M\_ID= 2 where Employee\_ID=12;



**Syntax for Delete Trigger** :-

create trigger D\_F\_I\_E

on Employee

for delete

as

begin

declare @EID int;

declare @Ename varchar(50);

declare @Gen varchar(2);

declare @MID int;

declare @Sal int;

declare @EM varchar(50);

declare @U\_ID varchar(50);

declare @UName varchar(50);

declare @timedate datetime;

declare @S\_ID int;

set @EID=(select Employee\_ID from deleted);

set @Ename=(select Employee\_Name from deleted);

set @Gen=(select Gender from deleted);

set @MID=(select M\_ID from deleted);

set @Sal=(select Salary from deleted);

set @EM=(select EMAIL from deleted);

set @U\_ID =(select SUSER\_ID());

set @UName= (select SUSER\_NAME());

set @timedate =(select GETDATE());

set @S\_ID=(select @@SPID);

insert into [audit] values

(@EID,@Ename,@Gen,@MID,@Sal,@EM,@U\_ID,@UName,@timedate,@S\_ID);

End

**Deleting from Employee to check** :-

delete from Employee where Employee\_ID=12;

