**Question & Answers**

**1Q.) What is user defined stored procedure and user defined function, explain and write the differences?**

**Ans:** The Stored Procedure is nothing but a name given to the code. Whenever we want that code, instead of writing the code again we will simply use the name. This stored procedure is mostly reduee the need of writing the same code again and again.

User Define Stored Procedure:

User-defined Stored Procedures are created by the database developers and administrators and stored in the current database. This type of stored procedure provides one or more SQL statements for retrieving, updating, and deleting values from database tables.

User Define Function:

A user-defined function (UDF) is a common fixture in programming languages, and the main tool of programmers for creating applications with reusable code. Since programs are mostly composed of code that comes from the programmer, or in this case the user, most of it is composed of user-defined functions occasionally punctuated by built-in functions.

Differences:

|  |  |
| --- | --- |
| User Defined Stored Procedures | User Define Function |
| 1. It supports select statements | 1. Both select statement and DML statements are accepted |
| 2. Only input values are allowed | 2. Both I/O are allowed |
| 3.Transactions are not allowed | 3. Transactions are allowed |
| 4. It cannot call stored procedures | 4. It can call functions |
| 5. Can be used in join clause | 5. Cannot be used in join clauses |

**2Q.) what are advantage of using stored procedure?**

**Ans:** The advantages of Stored Procedures are

1. Since stored procedures are compiled and stored, whenever you call a procedure the response is quick.

(It is faster)

2. you can group all the required SQL statements in a procedure and execute them at once.

(It is Pre-compliled)

3. Since procedures are stored on the database server which is faster than client. You can execute all the complicated quires using it, which will be faster.

(It reduces network traffic)

4. Access to the Oracle data can be restricted by allowing users to manipulate the data only through stored procedures that execute with their definer’s privileges.

(Its security is High)

5. Once you compile a stored procedure you can use it in any number of applications. If any changes are needed you can just change the procedures without touching the application code.

(It is Re-usable)

**Q3 :what is your understanding with complex stored procedure explain with example.**

**Ans:**A complex stored procedure is one that performs complex operations such as multiple queries, loops, conditional statements, etc.

Here’s an example of a complex stored procedure that retrieves data from multiple tables using joins and conditional statements.

create table Customers(

Customerid int,

CustomerName varchar(50),

ContactName varchar(50),

Address varchar(50),

City varchar(50))

insert into Customers values(1001,'bharat gas','Ramesh','kurnool','city')

insert into Customers values(1002,'bharat gas','radha','kadapa','city')

create table orders(

Orderid INT,

CustomerID INT,

EmployeeID INT,

OrderDate INT,

TotalAmount INT)

insert into orders values(4152,1001,5263,2023-03-29,526000)

insert into orders values(4153,1002,4563,2023-02-19,36000)

CREATE TABLE Employees(

EmployeeID INT,

FirstName VARCHAR(30),

LastName VARCHAR(30))

insert into Employees (EmployeeID,FirstName,LastName)

values(1001,'rajesh','reddy')

insert into Employees (EmployeeID,FirstName,LastName)

values(1002,'ramu','reddy')

DROP TABLE Employees

CREATE PROCEDURE dbo.GetOrdersByCustomerName

@CustomerName varchar(50)

AS

BEGIN

SET NOCOUNT ON;

SELECT o.OrderID, o.OrderDate, o.TotalAmount, e.EmployeeID,

ISNULL(e.LastName + ', ' + e.FirstName, 'N/A') as EmployeeName

FROM orders o

JOIN Customers c ON o.CustomerID = c.CustomerID

LEFT JOIN Employees e ON o.EmployeeID = e.EmployeeID

WHERE c.CustomerName = @CustomerName

SELECT COUNT(\*) as NumOrders, AVG(TotalAmount) as AvgTotalAmount

FROM orders o

JOIN Customers c ON o.CustomerID = c.CustomerID

WHERE c.CustomerName = @CustomerName

END

EXEC dbo.GetOrdersByCustomerName 'bharat gas';

**Q4: what are system defined stored procedure , list any five and explain.**

**Ans:** All system stored procedures are nothing but a name given to the code, which we can use again and again. So, without writing the actual code we can just call the name which we stored.

**These are the 5 system stored procedures:**

**1.sp\_help:**

This will display the Stored procedure Name, Schema Name, created date, and Time or if there are any parameters, then Parameter Name, Data Type, Length, Precision, Scale, Collation, etc. as result.

**2.sp\_helptext:**

This will display the content of the stored procedure as result.

**3.sp\_depends:**

This will show where the procedure is dependent like name of tables, functions, etc.

**4.Repeat():**

This function in MySQL is used to repeat a string a specified number of times.

Syntax: Repeat(str, count)

**5.sp\_Rename:**

Changes the name of a user-created object in the current database. This object can be a table, index, column, alias data type, or Microsoft

**Q5: what type of triggers, and on what event you can pass trigger.**

**Ans:**

**Triggers:**

Triggers are database object. Basically, these are a special type of stored procedure that is automatically fired/executed when a DDL or DML command statement related to the trigger is executed.

There are four types of Triggers in sql:

1.DDL Triggers

DDL (Data Definition Language) statements are like CREATE, ALTER, and DROP and certain system-defined stored procedures that perform DDL-like operations.

2.DML Triggers

DML (Data Manipulation Language) statements are like INSERT, UPDATE, and DELETE and stored procedures that perform DML-like operations.

3.CLR triggers

CLR (Common Language Runtime) CLR triggers are used to execute managed code in response to a variety of Transact-SQL statements.

4.LOGON Triggers

LOGON triggers do not fire if authentication fails. We can use these triggers to audit and control server sessions. Such as to track login activity or limit the number of sessions for a specific login. triggers fire stored procedures in response to a LOGON event

**Q6: write about recursive trigger.**

**Ans:** When a trigger on one table invokes another trigger on a different or related table, and so on, that trigger is said to be recursive or nested. There is a **cap of 32** on the total number of recursive calls that can be done.

There are two types of recursion:

1.Direct Recursion:

When a trigger fires and then execute an action that sets off another trigger, this recursion normally occurs.

2.In-Direct Recursion:

When a trigger activates and completes an action that prompts another trigger of the same type **(AFTER or INSTEAD OF)** to activate, a recursive cycle is created. So, when an **INSTEAD OF** trigger is triggered a second time, indirect recursion can happen,

**Q7: explain importance of indexes, on OLTP system and OLAP system.**

**Ans:**

OLTP:

OLTP stands for Online Transaction Processing. OLTP has the work to administer day-to-day transactions in any organization. The main goal of OLTP is data processing not data analysis.

OLAP:

Online Analytical Processing (OLAP) consists of a type of software tool that is used for data analysis for business decisions.

Importance of Indexes on OLTP System:

1. Emphasis on Rapid Processing.

2. Processing a large number of simple transactions.

3. Multi-User access while ensuring data integrity.

4. Availability of Indexed Datasets.

5. Data backup available at all times.

Importance of Indexes on OLAP System:

1.OLAP is ideal for data mining, business intelligence and complex analytical calculations, as well as business reporting functions like financial analysis, budgeting and sales forecasting.

2.The core of most OLAP databases is the OLAP cube, which allows you to quickly query, report on and analyze multidimensional data.

3. The OLAP cube extends the row-by-column format of a traditional relational database schema and adds layers for other data dimensions.

4. This historical, aggregated data for OLAP is usually stored in a star schema or snowflake schema.

**Q7: what is difference between rowstore and columnstore indexes.**

**Ans:**

**Columnstore:**

A columnstore is data that's logically organized as a table with rows and columns, and physically stored in a column-wise data format.

**RowStore:**

A rowstore is data that's logically organized as a table with rows and columns, and physically stored in a row-wise data format. This format is the traditional way to store relational table data.

|  |  |
| --- | --- |
| RowStore | Columnstore |
| Rowstore indexes perform best on queries  that seek into the data, when searching for  a particular value, or for queries on a  small range of values. | Columnstore indexes give high performance gains  for analytic queries that scan large amounts  of data, especially on large tables. |
| Use rowstore indexes with transactional  workloads because they tend to require  mostly table seeks instead of table scans. | Use columnstore indexes on data warehousing  and analytics workloads, especially on fact  tables, because they tend to require full table  scans rather than table seeks. |

**Q8: write script of merge join and write the importance.**

**Ans:** The **Merge Join** operator is one of four operators that join data from two input streams into a single combined output stream. Merge Join is the most effective of all join operators. However, it requires all input data to be sorted by the join columns. Often this means that a Merge Join can’t be used without adding extra Sort operators.

1.The importance of Merge Join is that it is the most effective of all join operators

2.However, it requires all input data to be sorted by the join columns. Often this means that a Merge Join can’t be used without adding extra Sort operators.

3.These extra sorts increase the total plan cost. In such cases, the optimizer tends to choose other join operators instead.

**Q9: Write the script for following:**

**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)**

**Ans:**

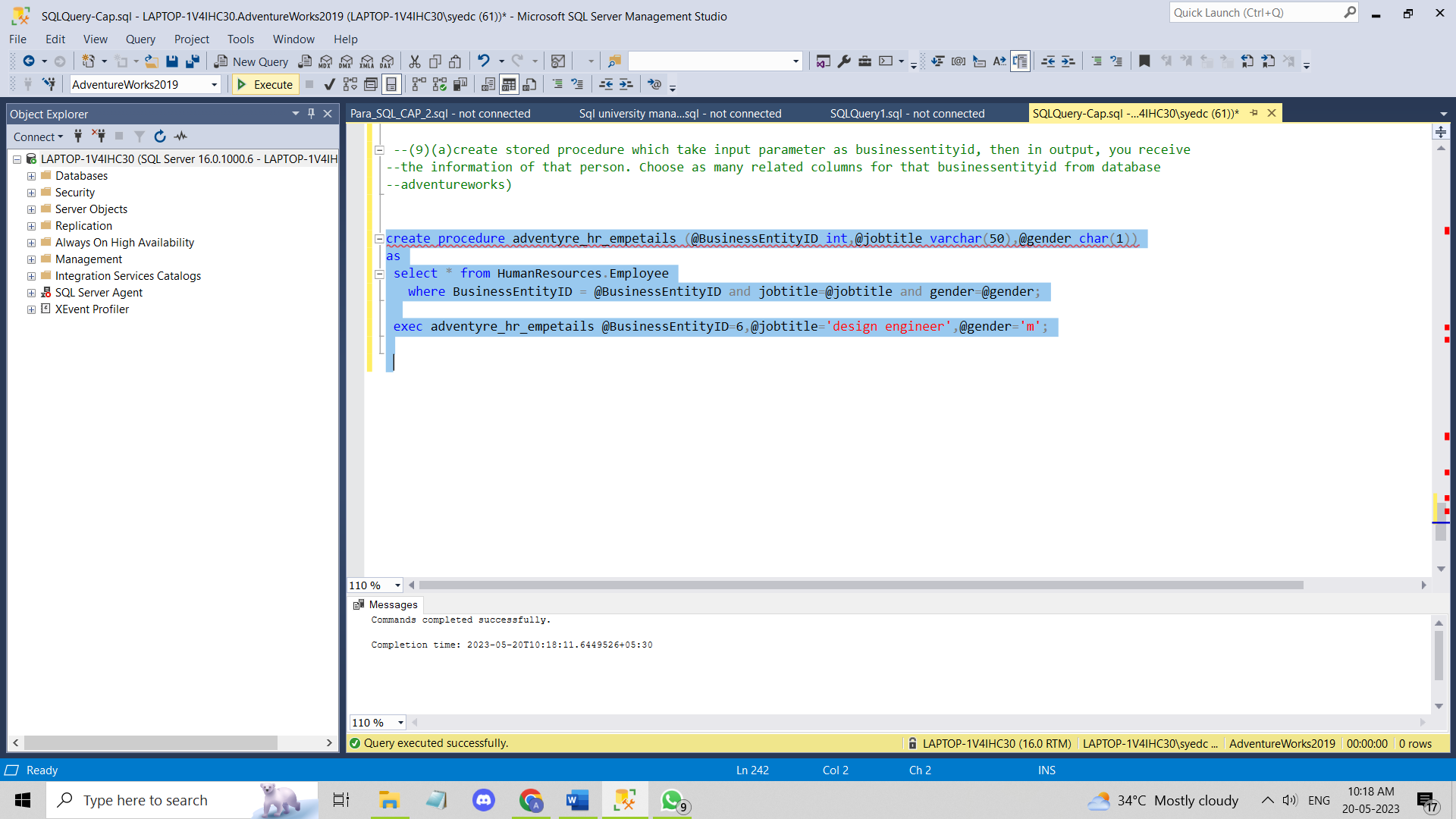
create procedure adventyre\_hr\_empetails (@BusinessEntityID int,@jobtitle varchar(50),@gender char(1))

as

select \* from HumanResources.Employee

where BusinessEntityID = @BusinessEntityID and jobtitle=@jobtitle and gender=@gender;

exec adventyre\_hr\_empetails @BusinessEntityID=6,@jobtitle='design engineer',@gender='m';



**9.(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.**

**Ans:**

create database capstone;

use capstone

CREATE TABLE Employee

(

EmployeeID int PRIMARY KEY IDENTITY(1,1),

EmployeeName varchar(50) NOT NULL,

EmployeeSalary decimal(18,2) NOT NULL

);

INSERT INTO Employee (EmployeeName, EmployeeSalary)

VALUES ('RAmesh', 60000.00);

INSERT INTO Employee (EmployeeName, EmployeeSalary)

VALUES ('ramu', 40000.00);

INSERT INTO Employee (EmployeeName, EmployeeSalary)

VALUES ('radha', 20000.00);

CREATE TABLE AuditEmployee

(

AuditEmployeeID int PRIMARY KEY IDENTITY(1,1),

UserID int NOT NULL,

UserName varchar(50) NOT NULL,

AuditDateTime datetime NOT NULL,

SessionID varchar(50) NOT NULL,

ActionType varchar(50) NOT NULL,

EmployeeID int NULL,

EmployeeName varchar(50) NULL,

EmployeeSalary decimal(18,2) NULL

);

INSERT INTO AuditEmployee (UserID, UserName, AuditDateTime, SessionID, ActionType, EmployeeID, EmployeeName, EmployeeSalary)

VALUES (1001, 'Ramesh', GETDATE(), 'RH123', 'INSERT', 1, 'Ramesh', 60000.00);

---create procedure -------------------

CREATE PROCEDURE AuditEmployee\_SP

@UserID int,

@UserName varchar(50),

@SessionID varchar(50),

@ActionType varchar(50),

@EmployeeID int = NULL,

@EmployeeName varchar(50) = NULL,

@EmployeeSalary decimal(18,2) = NULL

AS

BEGIN

SET NOCOUNT ON;

IF @ActionType = 'INSERT'

BEGIN

INSERT INTO AuditEmployee (UserID, UserName, AuditDateTime, SessionID, ActionType, EmployeeID, EmployeeName, EmployeeSalary)

VALUES (@UserID, @UserName, GETDATE(), @SessionID, @ActionType, @EmployeeID, @EmployeeName, @EmployeeSalary)

END

ELSE IF @ActionType = 'UPDATE'

BEGIN

DECLARE @OldEmployeeName varchar(50), @OldEmployeeSalary decimal(18,2)

SELECT @OldEmployeeName = EmployeeName, @OldEmployeeSalary = EmployeeSalary FROM Employee WHERE EmployeeID = @EmployeeID

UPDATE Employee SET EmployeeName = @EmployeeName, EmployeeSalary = @EmployeeSalary WHERE EmployeeID = @EmployeeID

INSERT INTO AuditEmployee (UserID, UserName, AuditDateTime, SessionID, ActionType, EmployeeID, EmployeeName, EmployeeSalary)

VALUES (@UserID, @UserName, GETDATE(), @SessionID, @ActionType, @EmployeeID, @OldEmployeeName, @OldEmployeeSalary)

END

ELSE IF @ActionType = 'DELETE'

BEGIN

DECLARE @OldEmployeeName2 varchar(50), @OldEmployeeSalary2 decimal(18,2)

SELECT @OldEmployeeName2 = EmployeeName, @OldEmployeeSalary2 = EmployeeSalary FROM Employee WHERE EmployeeID = @EmployeeID

DELETE FROM Employee WHERE EmployeeID = @EmployeeID

INSERT INTO AuditEmployee (UserID, UserName, AuditDateTime, SessionID, ActionType, EmployeeID, EmployeeName, EmployeeSalary)

VALUES (@UserID, @UserName, GETDATE(), @SessionID, @ActionType, @EmployeeID, @OldEmployeeName2, @OldEmployeeSalary2)

END

END

EXEC AuditEmployee\_SP

@UserID = 1001,

@UserName = 'Ramesh',

@SessionID = 'RH123',

@ActionType = 'update',

@EmployeeName = 'Ramesh',

@EmployeeSalary = 20000.00

INSERT INTO AuditEmployee (UserID, UserName, AuditDateTime, SessionID, ActionType, EmployeeID, EmployeeName, EmployeeSalary)

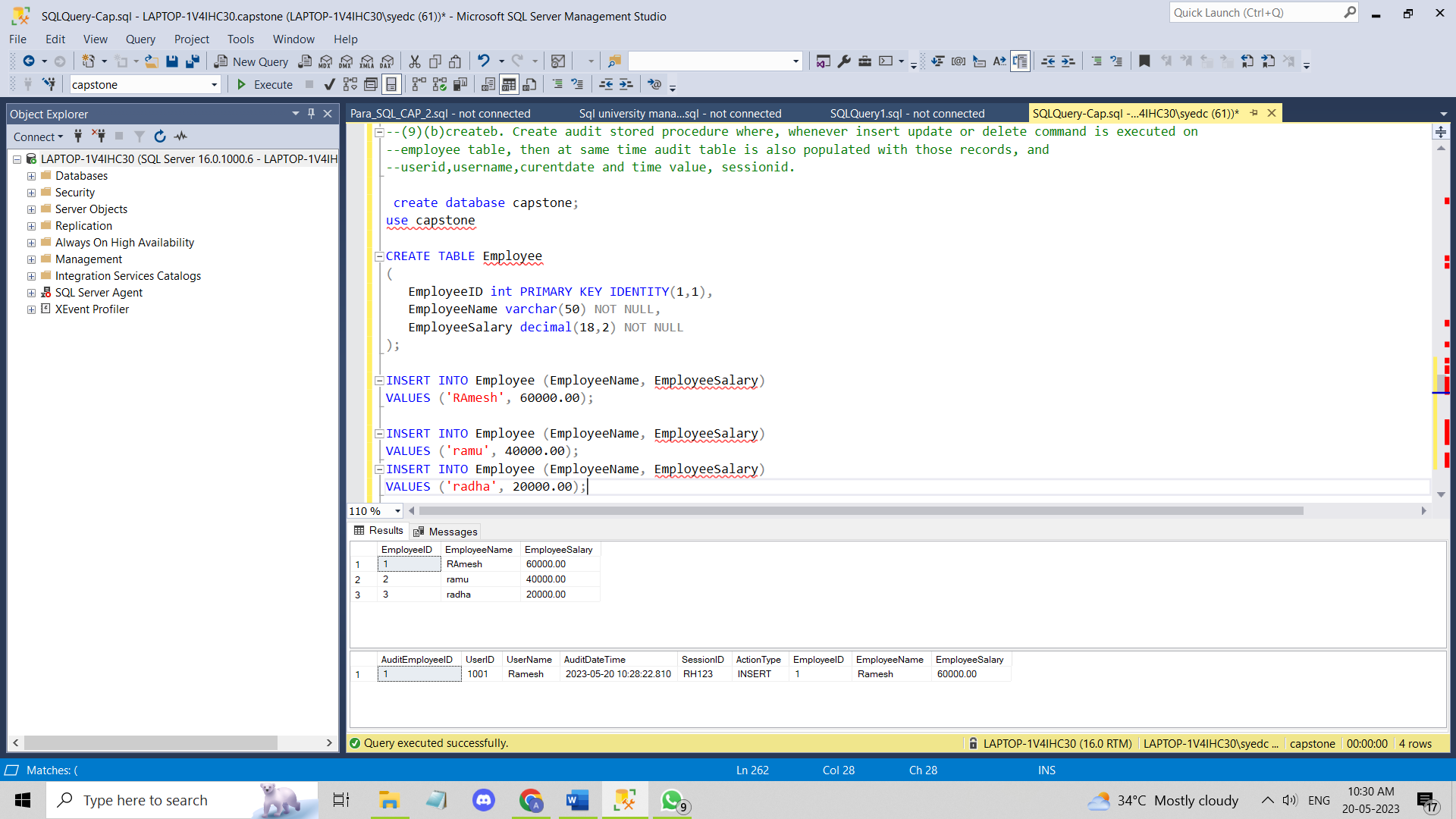
VALUES (1003, 'radha', GETDATE(), 'rds123', 'insert', 3, 'radha123', 70000.00);

INSERT INTO AuditEmployee (UserID, UserName, AuditDateTime, SessionID, ActionType, EmployeeID, EmployeeName, EmployeeSalary)

VALUES (1003, 'radha', GETDATE(), 'rds123', 'delete', 3, 'radha123', 70000.00);

select \* from Employee

select \* from AuditEmployee



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

**Ans:**

create table emp\_detail

(Id int,

salary int,

name char(30),

depart varchar(30)

)

insert into emp\_detail (id,salary,name,depart)

values (1001,25000,'nanda','sales');

insert into emp\_detail (id,salary,name,depart)

values (1002,28000,'Ramesh','accounts');

CREATE PROCEDURE get\_employee\_info

@employee\_id INT,

@salary INT OUTPUT,

@name CHAR(30) OUTPUT,

@department VARCHAR(30),

@default\_salary INT = 0

AS

BEGIN

SET NOCOUNT ON;

IF @department IS NOT NULL

BEGIN

SELECT @salary = salary, @name = name FROM emp\_detail

WHERE Id = @employee\_id AND depart = @department;

END

ELSE

BEGIN

SELECT @salary = salary, @name = name FROM emp\_detail

WHERE Id = @employee\_id;

END

IF @salary IS NULL

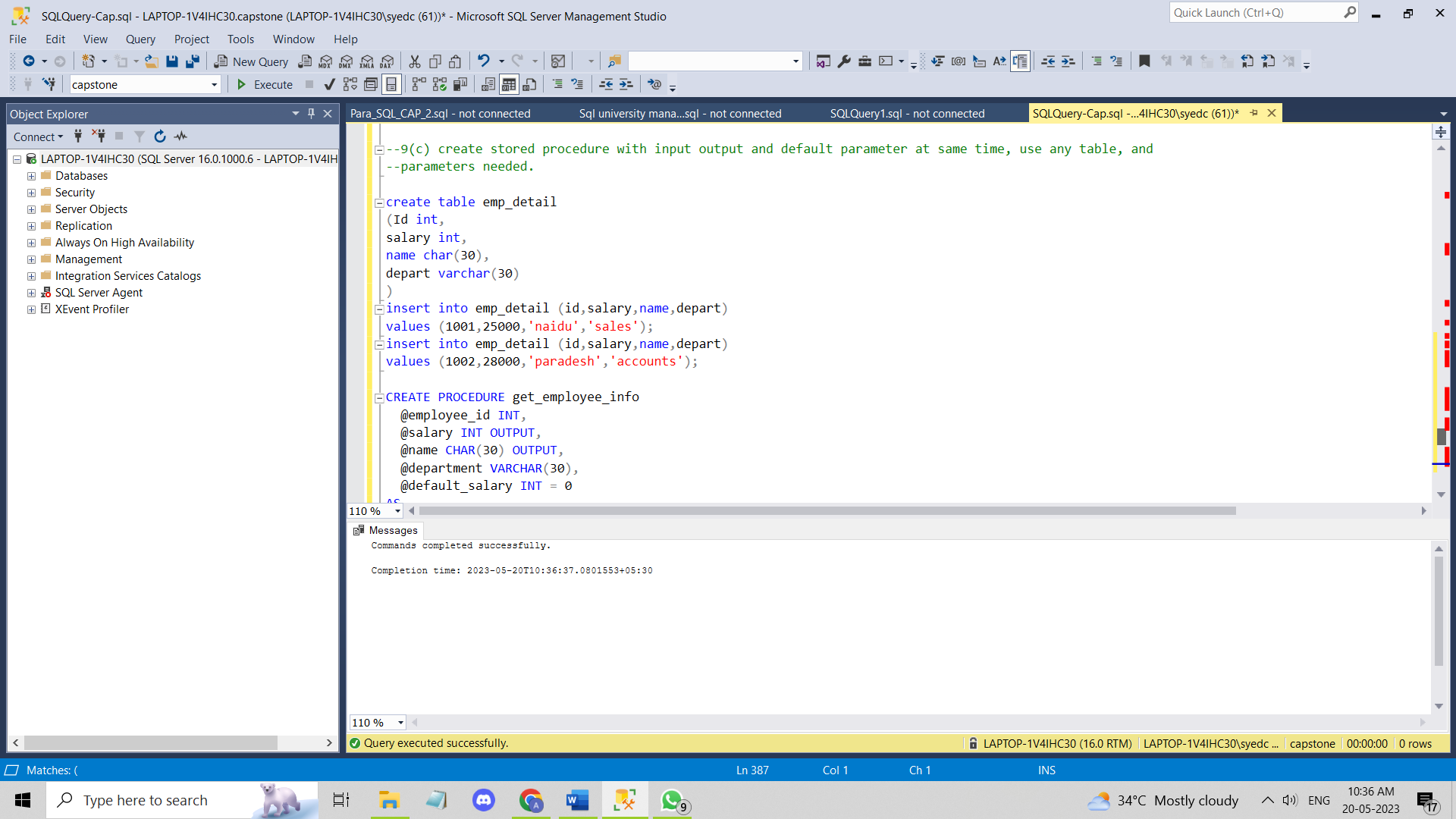
BEGIN

SET @salary = @default\_salary;

END

END

DECLARE @salary INT = 0, @name CHAR(30) = '';



**9(d) create deterministic and non-deterministic user defined function, take any example.**

**Ans:**

CREATE FUNCTION dbo.Addition\_Deterministic (@num1 INT, @num2 INT)

RETURNS INT

AS

BEGIN

DECLARE @result INT

SET @result = @num1 + @num2

RETURN @result

END

GO

SELECT dbo.Addition\_Deterministic(5, 10) AS 'Result'

CREATE FUNCTION GetNextID()

RETURNS INT

AS

BEGIN

DECLARE @result INT;

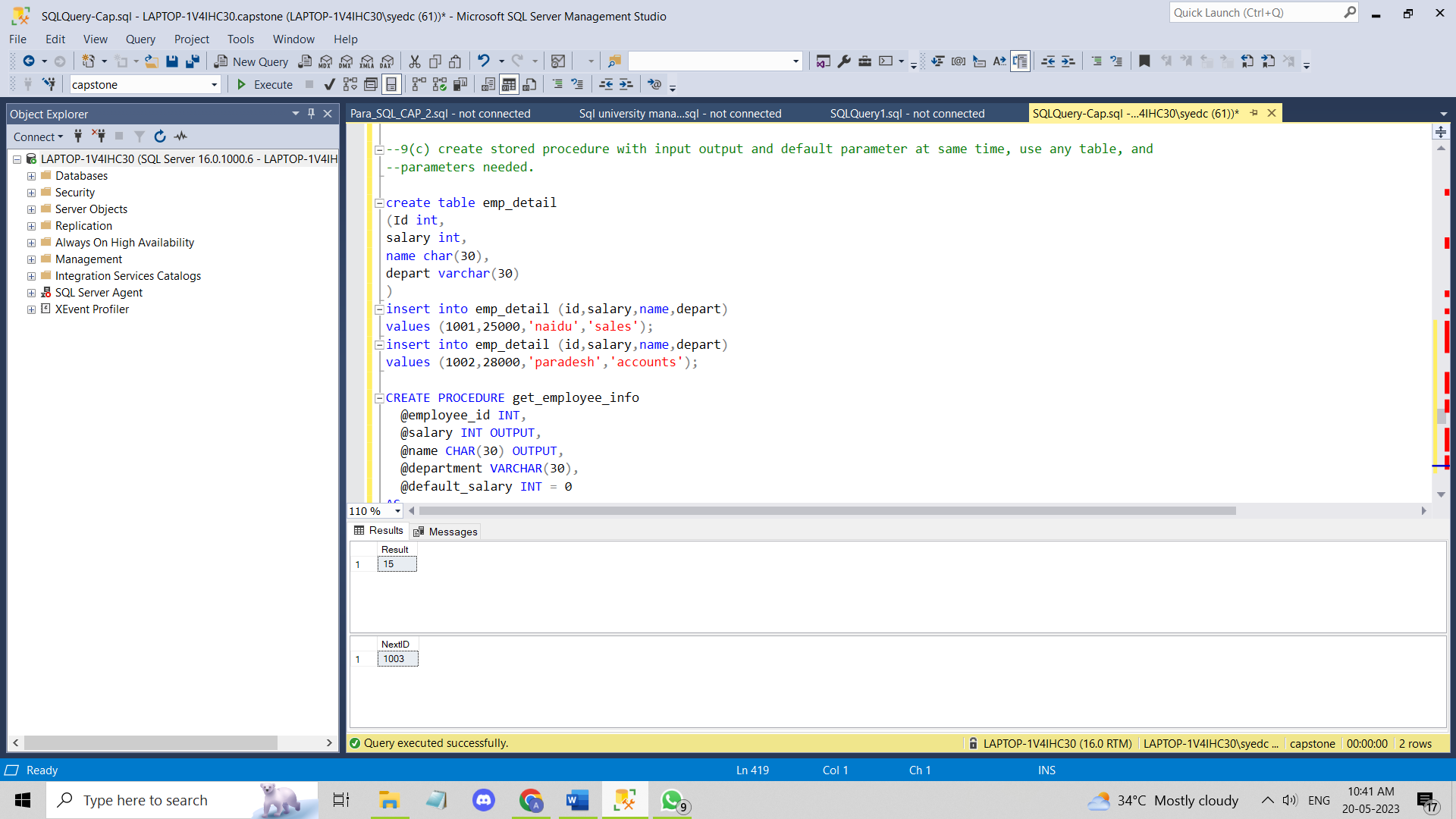
SET @result = (SELECT MAX(Id) FROM emp\_detail) + 1;

RETURN @result;

END

SELECT dbo.Addition\_Deterministic(5, 10) AS 'Result'

SELECT dbo.GetNextID() AS NextID;



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

**Ans:** create table customer

(Id int,

name char(30),

item int,

)

insert into customer values(1001,'ramu',20);

insert into customer values(1002,'rajubabu',50);

insert into customer values(1003,'rajesh',50);

create table AuditTable

(Id int ,

tablename varchar(30),Modifiediby varchar(30),

ModifiedDate varchar(30),Auditaction varchar(30))

create trigger Audit

on customer

after update,insert

as

insert into AuditTable(Id,tablename,Modifiediby,ModifiedDate,Auditaction)

select i.Id,i.name,i.item,suser\_sname(),getdate()

from customer c

inner join inserted i on c.Id = i.Id

go

--//insert in values ---//--Insert DDL opertions--

insert into customer values(1001,'ramu',20);

insert into customer values(1002,'rajubabu',50);

insert into customer values(1003,'rajesh',50);

---//update oprations--///

update customer

set name='Ramesh'

where id=1001

---////Delete operations--////

delete from customer

where id=1001

select \* from customer

select \* from AuditTable

