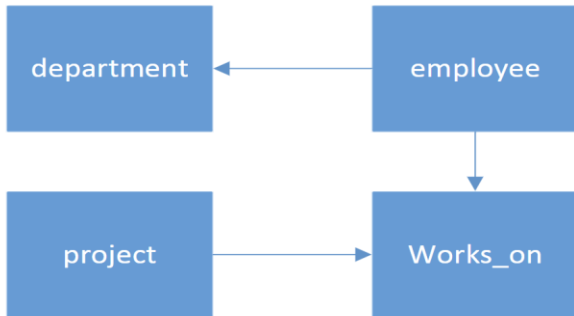


DATA MANAGEMENT AND DATABASE DESIGN
HOMEWORK: WEEK- 11

Use the sample database created in previous lecture to answer the following three questions.



1)

QUERY:

```
-- Creating a stored procedure named GetEmployeeInfo
CREATE PROCEDURE GetEmployeeInfo
@dept_no CHAR (4)
AS
BEGIN
-- Retrieving Employee Information based on Department
SELECT
e.emp_no,
e.emp_fname + ' ' + e.emp_lname AS Fullname,
d.dept_name AS DepartmentName
FROM
dbo. Employee e
JOIN
dbo. Department d ON e. dept_no=d. dept_no
WHERE
e. dept_no=@dept_no;
END;

EXEC GetEmployeeInfo @dept_no = 'd1';
```

SQLQuery1.sql - L...LKUMAR\laksh (54)*

```
USE
sample
GO

-- Creating a stored procedure named GetEmployeeInfo
CREATE PROCEDURE GetEmployeeInfo
    @dept_no CHAR(4)
AS
BEGIN
    -- Retrieving Employee Information based on Department
    SELECT
        e.emp_no,
        e.emp_fname + ' ' + e.emp_lname AS Fullname,
        d.dept_name AS DepartmentName
    FROM
        dbo.employee e
    JOIN
        dbo.department d ON e.dept_no=d.dept_no
    WHERE
        e.dept_no=@dept_no;
END;

EXEC GetEmployeeInfo @dept_no = 'd1';
```

84 %

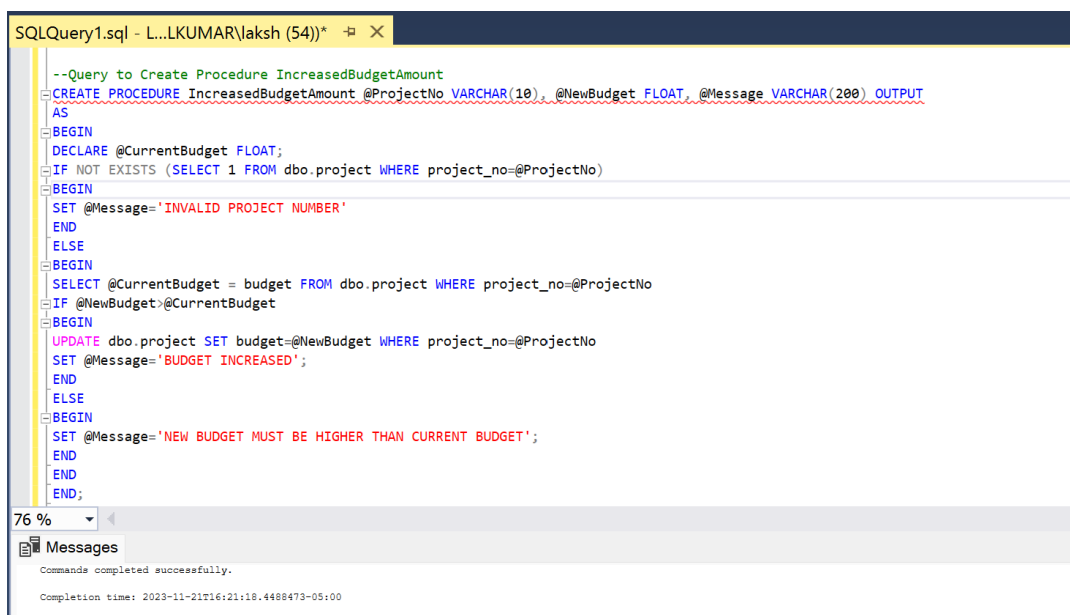
Results Messages

	emp_no	Fullname	DepartmentName
1	15000	John Smith	Accounting
2	28559	Matthew Hoyer	Accounting

2)

QUERY:

```
--Query to Create Procedure IncreasedBudgetAmount
CREATE PROCEDURE IncreasedBudgetAmount @ProjectNo VARCHAR (10),
@NewBudget FLOAT, @Message VARCHAR (200) OUTPUT
AS
BEGIN
DECLARE @CurrentBudget FLOAT;
IF NOT EXISTS (SELECT 1 FROM dbo. Project WHERE project_no=@ProjectNo)
BEGIN
SET @Message='INVALID PROJECT NUMBER'
END
ELSE
BEGIN
SELECT @CurrentBudget = budget FROM dbo. Project WHERE
project_no=@ProjectNo
IF @NewBudget>@CurrentBudget
BEGIN
UPDATE dbo. Project SET budget=@NewBudget WHERE project_no=@ProjectNo
SET @Message='BUDGET INCREASED';
END
ELSE
BEGIN
SET @Message='NEW BUDGET MUST BE HIGHER THAN CURRENT BUDGET';
END
END
END;
```

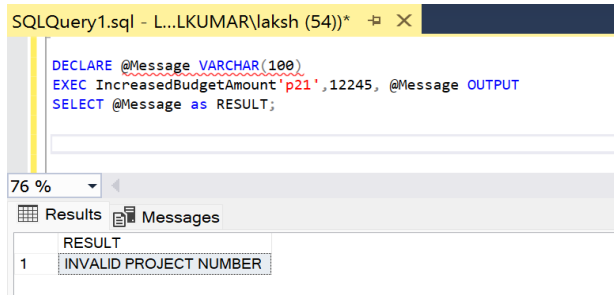


The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a query window titled 'SQLQuery1.sql - L...LKUMAR\laksh (54))' containing the same T-SQL code as shown in the previous block. The bottom pane, titled 'Messages', shows the execution results: 'Commands completed successfully.' and 'Completion time: 2023-11-21T16:21:18.4488473-05:00'. The zoom level is set to 76%.

Q1)

QUERY:

```
DECLARE @Message VARCHAR (100)
EXEC IncreasedBudgetAmount 'p21',12245, @Message OUTPUT
SELECT @Message as RESULT;
```



The screenshot shows a SQL query window titled 'SQLQuery1.sql - L...LKUMAR\laksh (54))' with the following code:

```
DECLARE @Message VARCHAR(100)
EXEC IncreasedBudgetAmount 'p21',12245, @Message OUTPUT
SELECT @Message as RESULT;
```

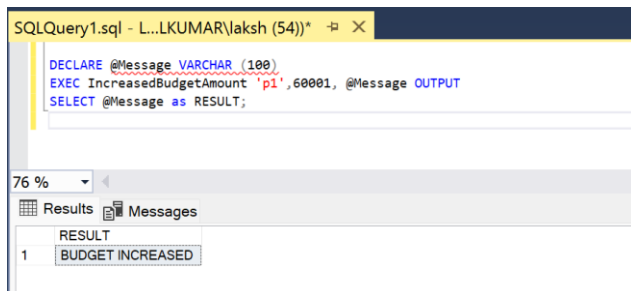
Below the code, the 'Results' tab is selected, showing a single row with the message 'INVALID PROJECT NUMBER'.

RESULT
1 INVALID PROJECT NUMBER

Q2)

QUERY:

```
DECLARE @Message VARCHAR (100)
EXEC IncreasedBudgetAmount 'p1',60001, @Message OUTPUT
SELECT @Message as RESULT;
```



The screenshot shows a SQL query window titled 'SQLQuery1.sql - L...LKUMAR\laksh (54))' with the following code:

```
DECLARE @Message VARCHAR(100)
EXEC IncreasedBudgetAmount 'p1',60001, @Message OUTPUT
SELECT @Message as RESULT;
```

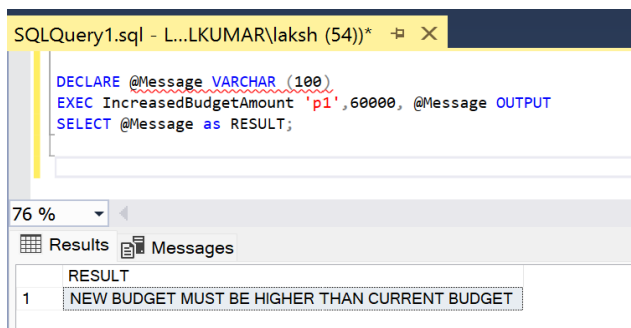
Below the code, the 'Results' tab is selected, showing a single row with the message 'BUDGET INCREASED'.

RESULT
1 BUDGET INCREASED

Q3)

QUERY:

```
DECLARE @Message VARCHAR (100)
EXEC IncreasedBudgetAmount 'p1',60000, @Message OUTPUT
SELECT @Message as RESULT;
```



The screenshot shows a SQL query window titled 'SQLQuery1.sql - L...LKUMAR\laksh (54))' with the following code:

```
DECLARE @Message VARCHAR(100)
EXEC IncreasedBudgetAmount 'p1',60000, @Message OUTPUT
SELECT @Message as RESULT;
```

Below the code, the 'Results' tab is selected, showing a single row with the message 'NEW BUDGET MUST BE HIGHER THAN CURRENT BUDGET'.

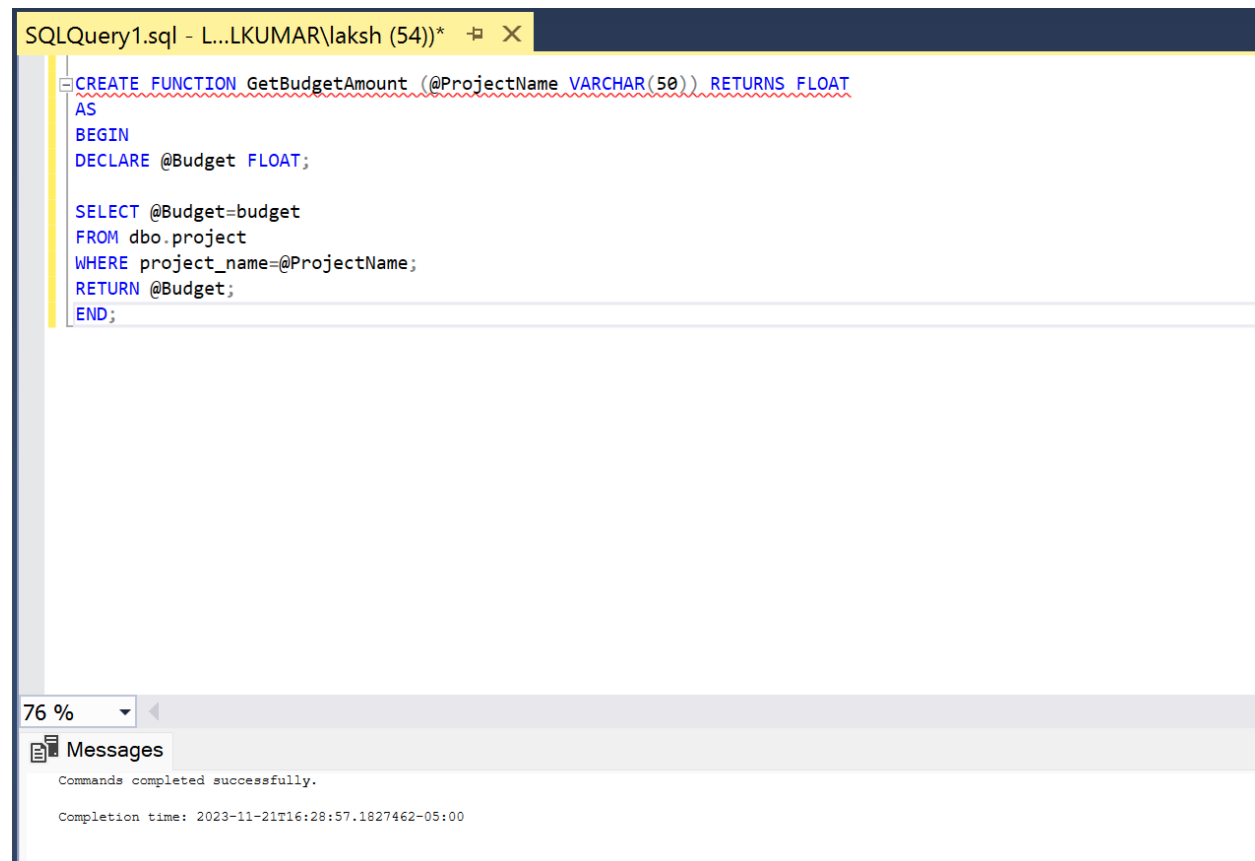
RESULT
1 NEW BUDGET MUST BE HIGHER THAN CURRENT BUDGET

3)

QUERY:

--Query to Create Procedure GetBudgetAmount

```
CREATE FUNCTION GetBudgetAmount (@ProjectName VARCHAR(50)) RETURNS  
FLOAT  
AS  
BEGIN  
DECLARE @Budget FLOAT;  
  
SELECT @Budget=budget  
FROM dbo.project  
WHERE project_name=@ProjectName;  
RETURN @Budget;  
END;
```




The screenshot displays the SQL Server Enterprise Manager interface. The top pane shows the execution of the SQL query to create the 'GetBudgetAmount' function. The bottom pane shows the 'Messages' window with the message 'Commands completed successfully.' and the completion time '2023-11-21T16:28:57.1827462-05:00'.

```
SQLQuery1.sql - L...LKUMAR\laksh (54)*  X
```

```
CREATE FUNCTION GetBudgetAmount (@ProjectName VARCHAR(50)) RETURNS FLOAT  
AS  
BEGIN  
DECLARE @Budget FLOAT;  
  
SELECT @Budget=budget  
FROM dbo.project  
WHERE project_name=@ProjectName;  
RETURN @Budget;  
END;
```

76 %

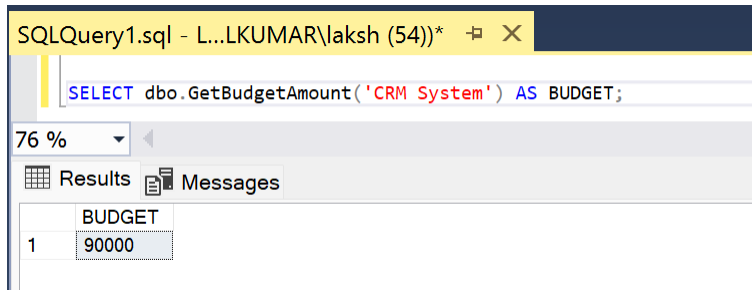
 Messages

Commands completed successfully.

Completion time: 2023-11-21T16:28:57.1827462-05:00

Q1)

QUERY:

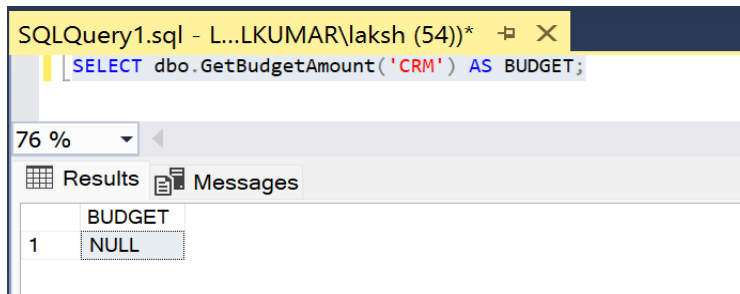


The screenshot shows a SQL query window titled 'SQLQuery1.sql - L...LKUMAR\laksh (54))*'. The query is `SELECT dbo.GetBudgetAmount('CRM System') AS BUDGET;`. The results pane shows a single row with the value 90000 for the column BUDGET.

	BUDGET
1	90000

Q2)

QUERY:



The screenshot shows a SQL query window titled 'SQLQuery1.sql - L...LKUMAR\laksh (54))*'. The query is `SELECT dbo.GetBudgetAmount('CRM') AS BUDGET;`. The results pane shows a single row with the value NULL for the column BUDGET.

	BUDGET
1	NULL

4)

“**Transaction**” is the single logical operation on the data to satisfy ACID (Atomicity, Consistency, Isolation and Durability) properties. A transaction is the process of gathering SQL statements that interact with a database.

5)

“**Atomicity**”. Atomicity is the property that ensures a transaction remains completed or remains not completed, nothing between. In the case of a CREATE TABLE command, either the table is created successfully or not created at all. It cannot be in a state where the table is partially created.

6)

“**Isolation**”. Isolation is the property which ensures that a particular transaction is isolated from the remaining transactions. It makes sure that one or more transactions can run concurrently only if they do not interfere with each other.

7)

Violates the “**Durability**” property. Durability refers to the property where a transaction guarantees to remain committed, that is all changes should be permanent and should not be lost under any event.

8)

Concurrency control is needed for the following reasons:

- **To ensure isolation of transactions:** Isolation is the property which ensures that a particular transaction is isolated from the remaining transactions. It makes sure that one or more transactions can run concurrently only if they do not interfere with each other.
- **To maintain atomicity:** A transaction a single atomic unit. Atomicity ensures that at the steps in a transaction must succeed else the whole transaction rollback. If the transaction succeeds, the changes are permanently committed.
- **To maintain consistency of database:** Consistency is the property which ensures that a transaction makes no changes is made so as to violate the rules or constraints placed on the particular data.
- **To prevent data inconsistency:** Data inconsistency is where same data or redundant data is present in the same database.
- **To avoid deadlocks:** Deadlock is the process or transaction that block one another from proceeding further since each of process has locked the resorceses each other need.

9)

The major difference between local and distributed transactions is that they differ in the scope of data they affect.

- **Local transaction** is confined to only one database instance. All of the data of the transaction is managed by the local database engine. They are faster and easier to manage when compared to distributed transactions.
- **Distributed transactions** have multiple database instances. The data is stored in multiples databases. They are more complex to handle than local transactions. It is necessary to access data that is stored in multiple databases.

10)

When a save point has to be created within a transaction, the **save transaction** statement is used. It is used to roll back to a specific point in the transaction. It is necessary for situations where a single portion of the transaction has to be changed without affecting the whole transaction.

11) Discuss the difference between row-level and page-level locking.

Difference between row-level locking and page-level locking are used in database systems to prevent access to the same data by the use of multiple transactions, concurrently.

Row-level locking is a type of locking, where only specific rows of data which are being accessed is locked. This results in high levels of concurrency since it can still access the rows that are not locked.

Page-level locking is a type of locking that locks an entire page of data. It keeps more data than necessary under lock, which in turn impacts the performance of other transactions.

12)

Yes, user can explicitly influence the locking behavior of the system. Below are some ways in which it influences the behavior:

By using locks: Locks are used in SQL in order to prevent the use of resources concurrently by two or more transactions. If one lock is held within a transaction, no other transaction can not modify until the lock is released.

By using clauses: Clause is a function that fetches all the records that are required from a database table.

By using concurrency control: Concurrency control is the process that ensures the integrity of the database is preserved, when multiple users update the same rows at the same time. Optimistic concurrency control allows conflicts to happen whereas pessimistic concurrency prevents the conflicts before happening.