NIBM WORLD WIDE

SOFTWARE ENGINEERING

DATABASE MANAGEMENT SYSTEMS

STORED PROCEDURES AND SQL TRIGGERS

National Institute of Business Management

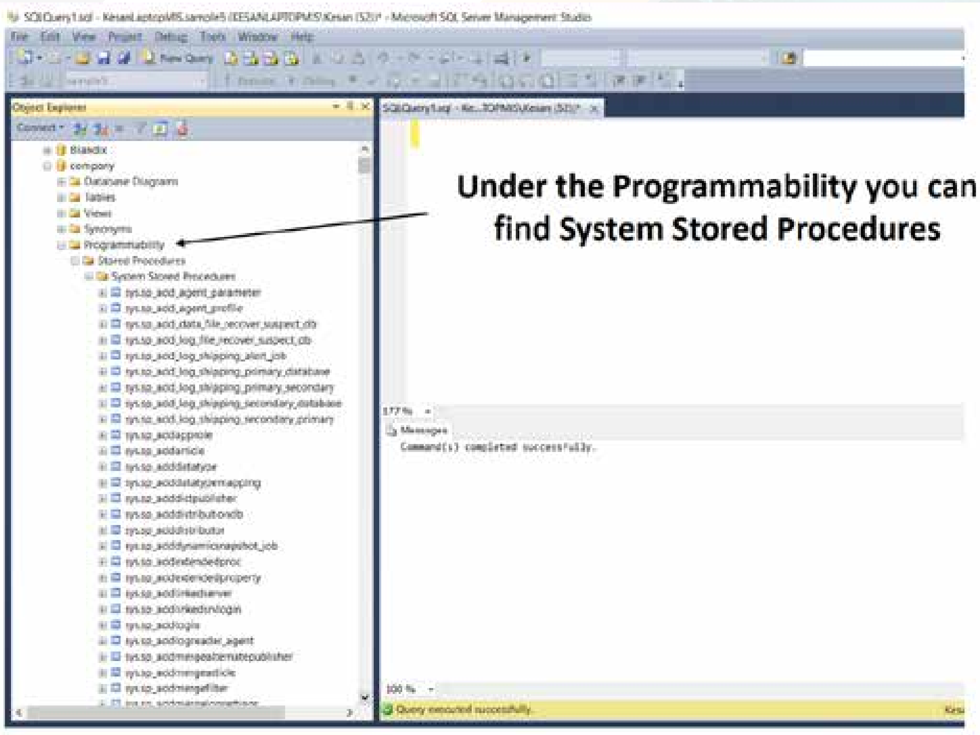
Lesson 12 – Stored Procedures and SQL Triggers

Stored Procedure

- A SQL Server stored procedure groups one or more Transact-SQL statements into a logical unit and is stored as an object in the Database Server.

- When a stored procedure is called at the first time, **SQL Server creates an execution plan and stores it in the plan cache.** In the subsequent executions of the stored procedure, **SQL Server reuses the plan so that the stored procedure can execute very fast with reliable performance.**

There are many system Stored Procedures in SQL Server. **If you can remember we used some of them.**



```
Command(s) completed successfully.
```

Query executed successfully.

Figure 12.0.1 System Stored Procedures

Syntax:

```
CREATE PROCEDURE procedure_name
AS BEGIN
sql_statement
END;
```

Execute a Stored Procedure:

```
EXEC procedure_name;
```

Example 1:

The following query returns a Employee Name from the Employee table

```
Use Company;
Select
Ename, Salary
from Employee order by Salary;
```

You can create a stored procedure that wraps this query using the CREATE PROCEDURE statement:

```
CREATE PROCEDURE FindEmployee
AS BEGIN
    SELECT
        Ename, Salary
    FROM
        Employee
ORDER BY
       Salary;
END;


EXEC FindEmployee;
```

Create a Procedure

Begin the Procedure

SQL Statement

End the Procedure

After you create the Procedure this is how you Execute.

Creating a Stored Procedure with One Parameter

Add a parameter to the stored procedure to find the Employees whose Salary are greater than an input Salary:

```
CREATE PROCEDURE Find_Employee_sal (@min_sal AS INT)
AS
BEGIN
    SELECT
        Ename, Salary
    FROM
        Employee
    WHERE
        Salary >= @min_sal
ORDER BY
        Salary;
END;
```

Create a Procedure with the Parameter (Same as passing parameter in the Function)

Where condition with the parameter. The value for this parameter we can give when we execute the Stored Procedure

This is how you execute the parametrized Stored Procedure

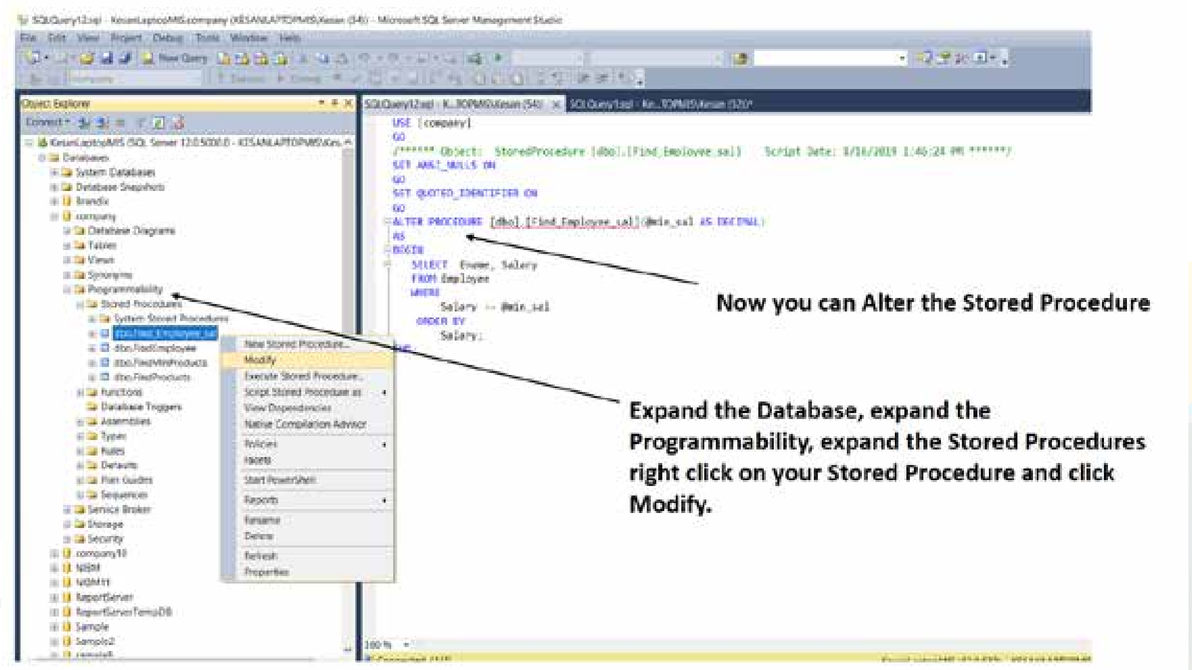
```
EXEC Find_Employee_sal 16000;
```

This is @min_sal parameter.

Now Find_Employee_sal Stored Procedure need a parameter to check the salaries greater than the given value.

- Here we added a parameter named **@min_sal** to the stored procedure. Every parameter must start with the **@** sign. The **AS INT** keywords specify the data type of the @min_sal parameter. The parameter must be surrounded by the opening and closing brackets.
- Then we used @min_sal parameter in the WHERE clause of the SELECT statement to filter only the Employees whose Salary are greater than or equal to the **@min_sal**.

Alter the Stored Procedure



```
USE [company]
GO
/****** Object:  StoredProcedure [dbo].[Find_Employee_sal]    Script Date: 3/16/2019 1:46:24 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[Find_Employee_sal](@min_sal AS DECIMAL)
AS
BEGIN
   SELECT  Ename, Salary
   FROM Employee
   WHERE
      Salary >= @min_sal
    ORDER BY
      Salary;
```

Now you can Alter the Stored Procedure

Expand the Database, expand the Programmability, expand the Stored Procedures right click on your Stored Procedure and click Modify.

Figure 12.0.2 Alter the Stored Procedure

Creating a Stored Procedure with Multiple Parameters

Stored procedures can take one or more parameters. The parameters are separated by commas. The following stored procedure has two parameters @min_sal and @max_sal for find salaries between those two parameters.

```
CREATE PROCEDURE Find_Employee_max_min_sal (@min_sal AS INT, @max_sal AS
INT)
AS
BEGIN
   SELECT
        Ename, Salary
   FROM
        Employee
   WHERE
       Salary >= @min_sal AND
       Salary <= @max_sal
ORDER BY
       Salary;
END;
```

Create a Procedure with the Parameters (Same as passing parameter in the Function)

Where condition with the parameter. The value for this parameter we can give when we execute the Stored Procedure

This is how you execute the parametrized Stored Procedure

```
EXEC Find_Employee_max_min_sal 18000, 45000;
```

This is for @min_sal parameter

This is for @max_sal parameter

Using named parameters

In case stored procedures have multiple parameters, it is better and more clear to execute the stored procedures using named parameters.

For example, the following statement executes the Find_Employee_max_min_sal stored procedure using the named parameters @min_sal and @max_sal.

```
EXEC Find_Employee_max_min_sal @min_sal = 18000, @max_sal = 45000;
```

Mentioned @min_sal parameter

Mentioned @max_sal parameter

Creating text parameters

The following statement adds the @name parameter as a character string parameter to the stored procedure.

```
CREATE PROCEDURE Employee_sal_name
(@min_sal AS INT, @max_sal AS INT, @name AS VARCHAR (10))
AS
BEGIN
   SELECT * FROM
      Employee
   WHERE
      Salary >= @min_sal AND
      Salary <= @max_sal AND
      Ename LIKE @name + '%'
   ORDER BY
      Eno;
```

Text parameter

Here using the Like Operator, we can pass the name to filter the Employees whose name start with the given value

```
END;

EXEC Employee_sal_name @min_sal = 16000, @max_sal = 45000, @name = 'A';
```

This value for @name parameter

SQL Server Triggers

SQL Server triggers are special stored procedures that are executed automatically in response to the database object, database, and server events. SQL Server provides three type of triggers:

- Data manipulation language (DML) triggers which are invoked automatically in response to INSERT, UPDATE, and DELETE events against tables.
- Data definition language (DDL) triggers which fire in response to CREATE, ALTER, and DROP statements. DDL triggers also fire in response to some system stored procedures that perform DDL-like operations.
- Logon triggers which fire in response to LOGON events

The CREATE TRIGGER statement allows you to create a new trigger that is fired automatically whenever an event such as INSERT, DELETE, or UPDATE occurs against a table.

Syntax:

```
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.**

Example:

We will use the Employee table from the Company database for this example.

1. Create a table for logging the changes

The following statement creates a table named Employee_Audits to record information when an INSERT or DELETE event occurs against the Employee table:

```
CREATE TABLE Employee_Audits (
    change_id INT IDENTITY PRIMARY KEY,
    Eno varchar(10),
    Ename varchar(10),
    Address varchar(10),
    Position varchar(10),
    Salary int,
    Dno varchar(10),
    Updated_at DATETIME,
    Operation CHAR(3),
    CHECK (operation = 'INS' or operation='DEL'));
```

Identity used for auto increment the primary key

All the columns of Employee table

DateTime of Updated

Check whether the operation is only from Inserted and Deleted table

2. Creating an after DML trigger

```
CREATE TRIGGER Trg_EMP_audit
ON Employee
AFTER INSERT, DELETE
AS
BEGIN
   SET NOCOUNT ON;
   INSERT INTO Employee_audits (
      Eno, Ename, Address, Position, Salary, Dno, Updated_at, Operation )
  SELECT
      i.Eno, Ename, Address, Position, Salary, Dno, GETDATE(), 'INS'
   FROM
      inserted i
   UNION ALL
   SELECT
      d.Eno, Ename, Address, Position, Salary, Dno, GETDATE(), 'DEL'
   FROM
      deleted d;
END
```

Create the Trigger on the Employee table

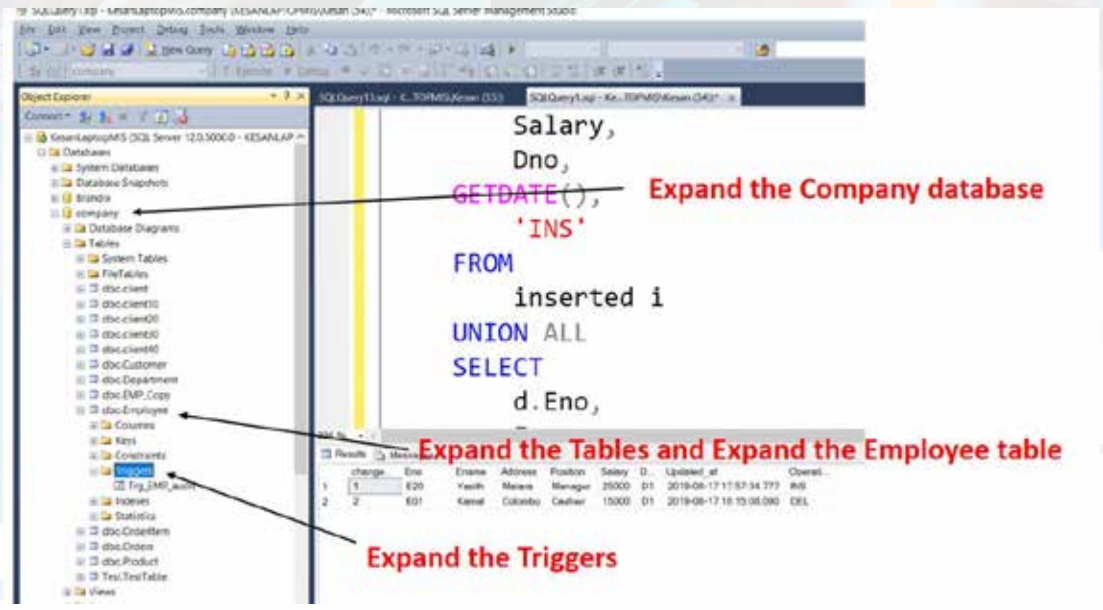
After Insert and Delete operation on Employee table

Insert the data to the Employee_audits table once the new record is insert or delete from Employee table

Once the new data is insert to Employee table INSERTED virtual table update. Therefore, get the new data from that INSERTED virtual table

Once a data is delete from Employee table DELETED virtual table update. Therefore, get the new data from that DELETED virtual table

2. Check your Trigger


Expand the Company database
Expand the Tables and Expand the Employee table
Expand the Triggers

```
        Salary,
        Dno,
        GETDATE(),
        'INS'
    FROM
        inserted i
    UNION ALL
    SELECT
        d.Eno,
```

	change	Eno	Ename	Address	Position	Salary	D...	Updated_at	Operat...
1	1	E20	Yasith	Malabe	Manager	25000	D1	2019-08-17 17:57:34.777	INS
2	2	E01	Kamal	Colombo	Cashier	15000	D1	2019-08-17 18:15:05.090	DEL

Figure 12.0.3 Check the Trigger in SQL Server

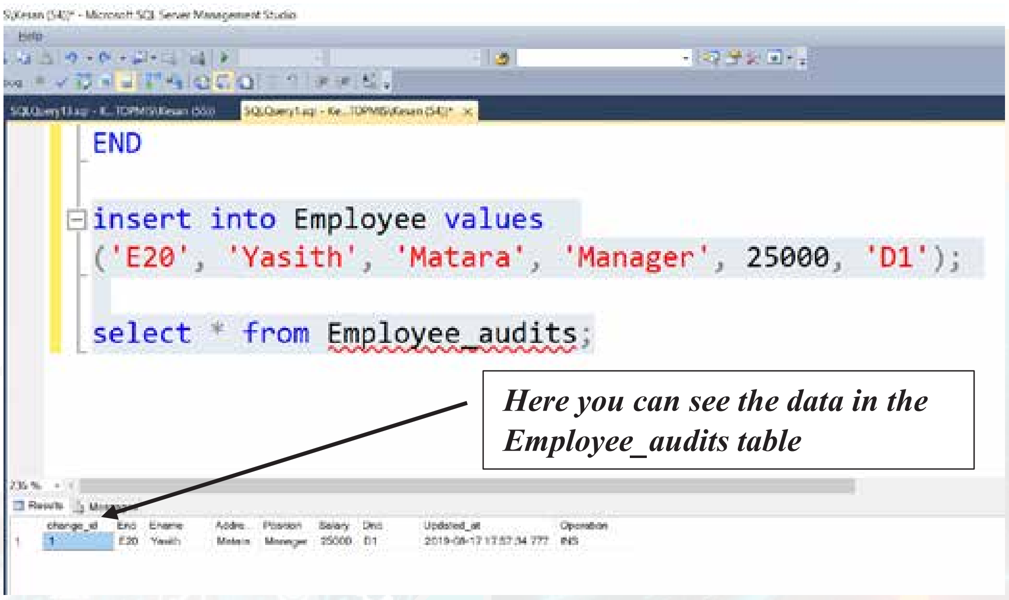
3. Insert a new data to the Employee table and check in the Employee Audits table

```
insert into Employee values
('E20', 'Yasith', 'Matara', 'Manager', 25000, 'D1');
```

Insert the data to the Employee table

```
select * from Employee_audits;
```

Check the data in the Employee_audits



```
END

insert into Employee values
('E20', 'Yasith', 'Matara', 'Manager', 25000, 'D1');

select * from Employee_audits;
```

Here you can see the data in the Employee_audits table

```
   change_id  Eno  Ename   Addre...  Position  Salary  Dno  Updated_at               Operation
1  1          E20  Yasith  Matara    Manager   25000   D1   2019-08-17 17:57:34.777  INS
```

Figure 12.0.4 insert data into the Employee Table

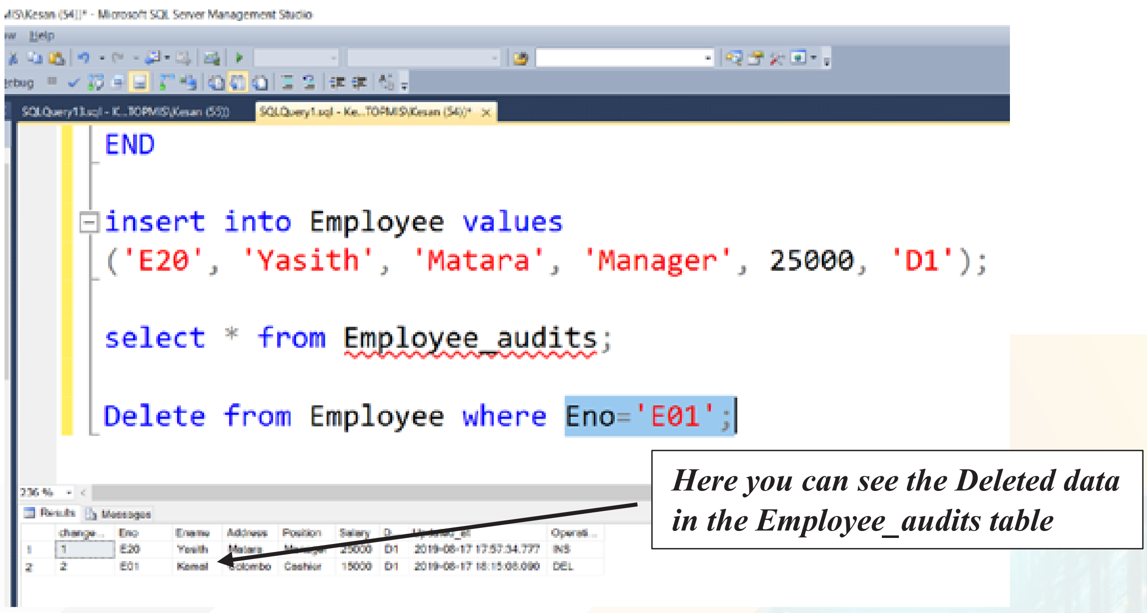
4. Delete a data from the Employee table and check in the Employee Audits table

```
Delete from Employee where Eno='E01';
```

Delete the data to the Employee table

```
select * from Employee_audits;
```

Check the data in the Employee_audits table



```
END

insert into Employee values
('E20', 'Yasith', 'Matara', 'Manager', 25000, 'D1');

select * from Employee_audits;

Delete from Employee where Eno='E01';
```

	change_...	Eno	Ename	Address	Position	Salary	D...	Updated_at	Operati...
1	1	E20	Yasith	Matara	Manager	25000	D1	2019-08-17 17:57:34.777	INS
2	2	E01	Kamal	Colombo	Cashier	15000	D1	2019-08-17 18:15:08.090	DEL

Here you can see the Deleted data in the Employee_audits table

Figure 12.0.5 Delete data from the Employee Table

Additional Readings to improve your Knowledge

1. **Fundamentals of Database Systems by Elmasri, Navathe.**
2. **SQL Server Tutorial Guide : https://www.sqlservertutorial.net/**
3. **Database Design Using Entity-Relationship Diagrams (Foundations of Database Design) 1st Edition by Sikha Bagui and Richard Earp.**