Week 5 - Database Technologies Assignment

1)What is meant by triggers? Explain a AFTER INSERT trigger with an example.

1A) Triggers in SQL Server

A trigger is a set of SQL statements that reside in system memory with unique names. It is a specialized category of stored procedure that is called automatically when a database server event occurs. Each trigger is always associated with a table.

A trigger is called a special procedure because it cannot be called directly like a stored procedure. The key distinction between the trigger and procedure is that a trigger is called automatically when a data modification event occurs against a table. A stored procedure, on the other hand, must be invoked directly.

The following are the main characteristics that distinguish triggers from stored procedures:

- ☐ We cannot manually execute/invoked triggers.

- ☐ Triggers have no chance of receiving parameters.

- ☐ A transaction cannot be committed or rolled back inside a trigger.

When we use triggers?

Triggers will be helpful when we need to execute some events automatically on certain desirable scenarios. For example, we have a constantly changing table and need to know the occurrences of changes and when these changes happen. If the primary table made any changes in such scenarios, we could create a trigger to insert the desired data into a separate table.

Example of Trigger in SQL Server

Let us understand how we can work with triggers in the SQL Server. We can do this by first creating a table named 'Employee' using the below statements:

```
CREATE TABLE Employee
(
  Id INT PRIMARY KEY,
  Name VARCHAR(45),
  Salary INT,
  Gender VARCHAR(12),
  DepartmentId INT
)
```

Next, we will insert some record into this table as follows:

```
INSERT INTO Employee VALUES (1,'Steffan', 82000, 'Male', 3),
(2,'Amelie', 52000, 'Female', 2),
(3,'Antonio', 25000, 'male', 1),
```

(4,'Marco', 47000, 'Male', 2),

(5,'Eliana', 46000, 'Female', 3)

We can verify the insert operation by using the SELECT statement. We will get the below output:

```
SELECT * FROM Employee;
```

AFTER INSERT Triggers in SQL Server

The SQL Server AFTER INSERT Triggers will fire after the completion of the Insert operation. The SQL After Insert Triggers not Supported on Views. For this SQL Server After Insert trigger demo, we use the below-shown tables.

As you can see that our Employee table is Empty. Our task is to create SQL AFTER INSERT TRIGGER on this Employee table. And by using this SQL Server After Insert trigger, we want to Insert the records into Employee Table Audit, along with the audit information.

```
USE [SQL Tutorial]
GO
SELECT TOP 1000 [ID]
      ,[Name]
      ,[Education]
      ,[Occupation]
      ,[YearlyIncome]
      ,[Sales]
  FROM [EmployeeTable];

  SELECT COUNT(*) FROM [EmployeeTable]
```

100 % ©tutorialgateway.org

Results | Messages

ID	Name	Education	Occupation	YearlyIncome	Sales

	(No column name)
1	0

and our Employee Table Audit is also empty

```
USE [SQL Tutorial]
GO
SELECT TOP 1000 [ID]
      ,[Name]
      ,[Education]
      ,[Occupation]
      ,[YearlyIncome]
      ,[Sales]
      ,[ServerName]
      ,[ServerInstanceName]
      ,[Insert Time]
  FROM [EmployeeTableAudit];

  SELECT COUNT(*) FROM [EmployeeTableAudit]
```

100 % ©tutorialgateway.org

Results | Messages

ID	Name	Education	Occupation	YearlyIncome	Sales	ServerName	ServerInstanceName	Insert Time

	(No column name)
1	0

After INSERT Triggers in SQL Server Example

In this example, we will create an After Insert Triggers in SQL Server on the Employee table using the CREATE TRIGGER statement.

Remember, After Insert trigger will fire after the completion of Insert operation on the Employee table. Once it completes inserting into the Employee table, it will start inserting into the Employee audit table. If it fails to insert into the Employee table, then it won't insert into the Audit table

-- Example for After INSERT Triggers in SQL Server

CREATE TRIGGER AfterINSERTTrigger on [EmployeeTable]

FOR INSERT

AS DECLARE @EmpID INT,

```
        @EmpName VARCHAR(50),
        @EmpEducation VARCHAR(50),
        @EmpOccupation VARCHAR(50),
        @EmpYearlyIncome DECIMAL (10, 2),
        @EmpSales DECIMAL (10, 2);

SELECT @EmpID = ins.ID FROM INSERTED ins;
SELECT @EmpName = ins.Name FROM
INSERTED ins;
SELECT @EmpEducation = ins.Education FROM
INSERTED ins;
SELECT @EmpOccupation = ins.Occupation
FROM INSERTED ins;
SELECT @EmpYearlyIncome = ins.YearlyIncome
FROM INSERTED ins;
SELECT @EmpSales = ins.Sales FROM INSERTED
ins;

INSERT INTO [EmployeeTableAudit](
      [ID]
     ,[Name]
     ,[Education]
```

```
	,[Occupation]
	,[YearlyIncome]
	,[Sales]
	,[ServerName]
	,[ServerInstanceName]
	,[Insert Time])
VALUES (@EmpID,
	@EmpName,
	@EmpEducation,
	@EmpOccupation,
	@EmpYearlyIncome,
	@EmpSales,
	CAST( SERVERPROPERTY('MachineName') AS VARCHAR(50)),
	CAST( SERVERPROPERTY('ServerName') AS VARCHAR(50)),
	GETDATE()
	);
PRINT 'We Successfully Fired the AFTER INSERT Triggers in SQL Server.'
GO
```

2)Write the syntax for a basic form of SQL.

2A)What is Syntax?

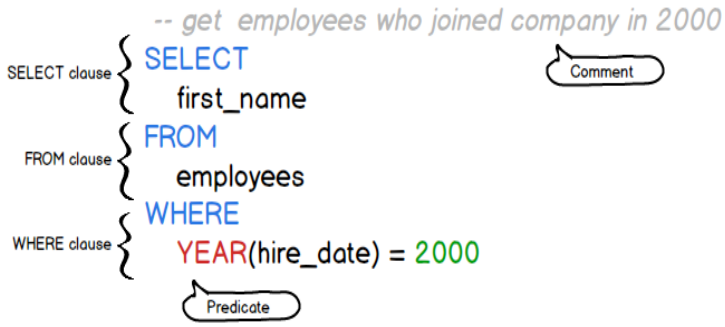
The term syntax refers to strict structural patterns used when creating a query. As soon as you enter the search criteria using the correct syntax, the query should execute, and the requested records retrieved from the target database. Remember that database query languages may be database-dependent and may not execute across different database platforms. It is always a good idea to check for compatibility before beginning your programming work.

```
SELECT
    first_name
FROM
    employees
WHERE
    YEAR(hire_date) = 2000;
```

As you see, it reads like a normal sentence.

Get the first names of employees who were hired in 2000.

The SELECT first_name, FROM employees, and WHERE are clauses in the SQL statement. Some clauses are mandatory e.g., the SELECT and FROM clauses whereas others are optional such as the WHERE clause.



SQL Syntax

Because SQL was designed specifically for the non-technical people in mind, it is very simple and easy to understand. To write an SQL statement, you just need to tell what you want instead of how you want it like other imperative languages such as PHP, Java, and C++.

SQL is a user-friendly language because it is mainly for the users who perform ad-hoc queries and generate reports.

Nowadays, SQL is used by the highly technical people like data analysts, data scientists, developers, and database administrators.

A Simple Example :

```
CREATE DATABASE mydb;

USE mydb;

CREATE TABLE mytable ( id INT PRIMARY KEY, name VARCHAR(20) );

INSERT INTO mytable VALUES ( 1, 'Will' );

INSERT INTO mytable VALUES ( 2, 'Marry' );

INSERT INTO mytable VALUES ( 3, 'Dean' );

SELECT id, name FROM mytable WHERE id = 1;

UPDATE mytable SET name = 'Willy' WHERE id = 1;

SELECT id, name FROM mytable;

DELETE FROM mytable WHERE id = 1;
```

SELECT id, name FROM mytable;

DROP DATABASE mydb;

SELECT count(1) from mytable; gives the number of records in the table

3)Write and Sample Nested Query SQL Statement.

3A)Nested Queries in SQL

In nested queries, a query is written inside a query. The result of inner query is used in execution of outer query. We will use STUDENT, COURSE, STUDENT_COURSE tables for understanding nested queries.

STUDENT

S_ID	S_NAME	S_ADDRESS	S_PHONE	S_AGE
1	RAM	DELHI	945xxxx451	18
2	RAMESH	GURGAON	965xxxx543	18
3	SUJIT	ROHTAK	915xxxx131	20
4	SURESH	DELHI	915xxxx971	21

COURSE

C_ID	C_NAME
C1	DSA
C2	Programming
C3	DBMS

STUDENT_COURSE

S_ID	C_ID
S1	C1
S1	C3
S2	C1
S3	C2
S4	C2
S4	C3

There are mainly two types of nested queries:

- **Independent Nested Queries:** In independent nested queries, query execution starts from innermost query to outermost queries. The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like IN, NOT IN, ANY, ALL etc are used in writing independent nested queries.

IN: If we want to find out S_ID who are enrolled in C_NAME ‘DSA’ or ‘DBMS’, we can write it with the help of independent nested query and IN operator. From COURSE table, we can find out C_ID for C_NAME ‘DSA’ or DBMS’ and we can use these C_IDs for finding S_IDs from STUDENT_COURSE TABLE.

STEP 1: Finding C_ID for C_NAME =’DSA’ or ‘DBMS’

Select C_ID from COURSE where C_NAME = ‘DSA’ or C_NAME = ‘DBMS’

STEP 2: Using C_ID of step 1 for finding S_ID

Select S_ID from STUDENT_COURSE where C_ID IN

(SELECT C_ID from COURSE where C_NAME = ‘DSA’ or C_NAME=’DBMS’);

The inner query will return a set with members C1 and C3 and outer query will return those S_IDs for which C_ID is equal to any member of set (C1 and C3 in this case). So, it will return S1, S2 and S4.

Note: If we want to find out names of STUDENTs who have either enrolled in ‘DSA’ or ‘DBMS’, it can be done as:

Select S_NAME from STUDENT where S_ID IN

(Select S_ID from STUDENT_COURSE where
C_ID IN

(SELECT C_ID from COURSE where
C_NAME='DSA' or C_NAME='DBMS'));

NOT IN: If we want to find out S_IDs of STUDENTs who have neither enrolled in 'DSA' nor in 'DBMS', it can be done as:

Select S_ID from STUDENT where S_ID NOT IN

(Select S_ID from STUDENT_COURSE where
C_ID IN

(SELECT C_ID from COURSE where

C_NAME='DSA' or C_NAME='DBMS'));

The innermost query will return a set with members C1 and C3. Second inner query will return those S_IDs for which C_ID is equal to any member of set (C1 and C3 in this case) which are S1, S2 and S4. The outermost query will return those S_IDs where S_ID is not a member of set (S1, S2 and S4). So it will return S3.

- **Co-related Nested Queries:** In co-related nested queries, the output of inner query depends on the row which is being currently executed in outer query. e.g.; If we want to find out S_NAME of STUDENTs who are enrolled in C_ID 'C1', it can be done with the help of co-related nested query as:

Select S_NAME from STUDENT S where EXISTS

(select * from STUDENT_COURSE SC where S.S_ID=SC.S_ID and SC.C_ID='C1');

For each row of STUDENT S, it will find the rows from STUDENT_COURSE where S.S_ID = SC.S_ID and SC.C_ID='C1'. If for a S_ID from STUDENT S, atleast a row exists in STUDENT_COURSE SC with C_ID='C1', then

inner query will return true and corresponding S_ID will be returned as output.

4)Create student, department t and college table and execute the below 1. Retrieve student name and department names from student and department 2. Retrieve the students those who have average maths marks greater than 50 3. Retrieve departments having students less than 50 4. Retrieve college having departments more than 5 Note: Generate 20 matching records

4A)