DATABASE MANAGEMENT SYSTEMS (IT 2040)

LECTURE 05 – DATABASE PROGRAMMING

LECTURE CONTENT

- Views
- SQL extensions (specifically T-SQL)
- Functions & Procedures
- Triggers

LEARNING OUTCOMES

- At the end of this lecture, students should be able to
  - Identify situations where views, functions, stored procedures and triggers are applicable.
  - Write syntactically correct sql statements to create functions, procedures and triggers on RDBMS to cater user requirements.

VIEWS

- A **view** is a virtual table which is derived from other tables which are called **base tables** or **defining tables.**
- A view does not necessarily exist in a physical form.
- In SQL, CREATE VIEW statement is used to define views.
- Syntax :

```
CREATE VIEW <view_name> AS
SELECT <column_name(s)>
FROM <table_name>
WHERE <condition>
```

VIEWS (CONTD.)

- emp (eid, ename, age, salary) ,dept (did, dname, budget,mangerId), works (eid, did, pct_time)

- Create a view named dept_info that contains name of the department, budget and manager's name

```
CREATE VIEW dept_info(dname,budget,manager)
AS
SELECT d.dname, d.budget, e.ename  FROM emp e, dept d
WHERE e.eid=d.managerId
```

EXERCISE 1

- Create a view named emp_info which contains eid, name, salary and total percentage of time.

  emp (<u>eid</u>, ename, age, salary)

  dept (<u>did</u>, dname, budget,mangerId)

  works (eid, did, pct_time)

QUERYING AND DELETING A VIEW

- Querying a view can be done similar to a table.
  - Ex : SELECT * FROM dept_info
  - Ex : SELECT dname FROM dep_info WHERE budget> 500,000

- Dropping a view
  - Ex : DROP VIEW dept_info

UPDATING VIEWS (CONTD.)

- Updating a view can be ambiguous…
  - Views containing aggregate functions are not updateable

  - Ex :

```
UPDATE emp_info
SET  tot_pct= 90
WHERE eid = 1000
```

**This is not possible

UPDATING VIEWS (CONTD.)

- Views containing a join can be ambiguous

A	B
b	1

B	C
1	d
1	e

A	B	C
b	1	d
b	1	e

```
UPDATE V1
SET A = a
WHERE C = e
```

- Thus, in many DBMSs, views are updateable only if they are defined on a single base table.

WHY VIEWS ?

- Advantages
  - **Security :** Each user can be given permission to access the database only through a small set of views that contain the specific data the user is authorized to see, thus restricting the user's access to stored data
  - **Query Simplicity :** A view can draw data from several different tables and present it as a single table, turning multi-table queries into single-table queries against the view.
- Disadvantages
  - **Performance :** Views create the appearance of a table, but the DBMS must still translate queries against the view into queries against the underlying source tables. If the view is defined by a complex, multi-table query then simple queries on the views may take considerable time
  - **Update restrictions**

PROGRAMMING IN T-SQL

- Similar to a programming language, certain extensions have been made in SQL to program simple server-side logic.
- Some of the statements include:
  - Variables
  - Selection conditions
    - IF (…)… ELSE …
  - Looping
    - WHILE (…)

T-SQL: VARIABLES

- A Transact-SQL local variable is an object that can hold a single data value of a specific type.
- Variables in scripts are typically used:
  - As a counter either to count the number of times a loop is performed or to control how many times the loop is performed
  - To hold a data value to be tested by a control-of-flow statement
  - To save a data value to be returned by a stored procedure return code.

T-SQL: VARIABLES (CONTD.)

- The DECLARE statement initializes a Transact-SQL variable.
  - Syntax: DECLARE @<variable name> <data type>
  - Ex: DECLARE @DName VARCHAR(20)
  - The created variable will be holding a null value

- To assign a value to a variable, use the SET statement.
  - Syntax : SET @<variable name> =<value>
  - SET @DName = ‘SESD’

T-SQL:VARIABLES (CONTD.)

- The declared variables could be used in scripts
- Ex :
  - SELECT budget
    FROM Dept
    WHERE dname = @DName
  - DECLARE @empId INT
    SELECT @empId = MAX(eid)
    FROM emp

T-SQL: IF STATEMENT

- Imposes conditions on the execution of a Transact-SQL statement.
- Ex:

```
IF (SELECT count(eid) FROM emp) > 1000
BEGIN
    PRINT ‘Inside the IF statement’
    PRINT ‘There are lesser than 1000 employees ’
END
ELSE
    PRINT ‘There are more than 1000 emloyees ! '
```

T-SQL: WHILE STATEMENT

- Sets a condition for the repeated execution of an SQL statement or statement block.
- The statements are executed repeatedly as long as the specified condition is true.
- The execution of statements in the WHILE loop can be controlled from inside the loop with the BREAK and CONTINUE keywords.

T-SQL: WHILE STATEMENT (CONTD.)

- BREAK
  - Causes an exit from the innermost WHILE loop. Any statements appearing after the END keyword, marking the end of the loop, are executed.

- CONTINUE
  - Causes the WHILE loop to restart, ignoring any statements after the CONTINUE keyword.

T-SQL: WHILE STATEMENT(CONTD.)

- Ex :

```
WHILE @count<=100
BEGIN
    INSERT INTO Employees VALUES(@count,CONCAT('Employee',@count))
    SET @count=@count+1
END
```

STORED FUNCTIONS/PROCEDURES

- Business logic is maintained in database tier for data intensive operations
  - E.g. Calculating all interest earned in bank accounts

- In SQL Server 2005, Stored Procedure/ Functions can be written in
  - T-SQL (we will study only this)
  - Any .NET Language

STORED FUNCTIONS/PROCEDURES (CONTD.)

- Syntax of a function

```
CREATE FUNCTION <function name> (parameters)
RETURNS <return type>
<function body>
```

- Parameter mode of parameters for functions is IN which parameters allow the calling code to pass values into the procedure

- Syntax of a procedure

```
CREATE PROCEDURE <procedure name> (parameters)
 <proedure body>
```

- Each parameter to a procedure should have a data type and a parameter mode (IN or OUT).
  - IN: This is the default mode. IN parameters allow the calling code to pass values into the procedure
  - OUT: OUT parameters allow the procedure to pass values back to the calling code

FUNCTIONS

- Ex : Create a function that returns the number of employees in a given department.

Function name

Input parameters

```
create function getEmpCount (@did char(12)) returns int
as
begin
        declare @ecount int
        select @ecount=count(*)
        from works w
        where w.did=@did
        return @ecount
end
```

Return data type

FUNCTIONS (CONTD.)

- Calling the function created previously

```
declare @result int
exec @result=get_empCount 'Admin'
print @result
```

EXERCISE 2

- Create a function to return the total percentage of time a person works given the employee id.

STORED PROCEDURES

- Ex : Create a procedure to give a salary increment to all the employee by a given percentage from their existing salary

```
CREATE PROCEDURE increaseSalary (@pct float)
as
begin
    Update emp
    Set salary=salary+salary * (pct/100)
end;
```

STORED PROCEDURES (CONTD.)

- Calling the procedure

```
exec increaseSalary 10
```

STORED PROCEDURES (CONTD.)

- Ex 2: create a procedure that outputs statistics of salary (min, max) for a given department.

```
create procedure get_stats(@did varchar(12),@maxm real output,@minm real output)
as
begin
    select @maxm=max(e.salary),@minm=min(e.salary)    from dept d, works w, emp e
    where d.did=w.did and w.eid=e.eid and d.did=@did
end
```

STORED PROCEDURES (CONTD.)

- Calling the procedure

```
declare @max int,@min int
exec get_stats 'Admin', @max output,@min output
print @max
print @min
```

EXERCISE 3

- Create a procedure that outputs the name of the manager and his salary in a given department.

TRIGGERS

- Triggers are useful in enforcing business rules and data integrity.
- They are more powerful than general constraints.
- For example,
  - The employees salary is always less than his/her manager’s salary

T-SQL: TRIGGERS

- A trigger is a special type of stored procedure that automatically takes effect when the data in a specified table is modified.
- A trigger is invoked in response to a
  - DDL statement (CREATE, ALTER etc.) or
  - DML statement (INSERT, UPDATE, or DELETE statement).‘

T-SQL: TRIGGER SYNTAX

- We will learn DML triggers…
- Syntax :

```
CREATE TRIGGER trigger_name
ON { table | view }
{
   { { FOR | AFTER | INSTEAD OF }
 { [ INSERT ] [ , ] [ UPDATE ] [ , ]
   [DELETE ] }
     AS
           sql_statement [ ...n ]
      }
  }
```

T-SQL: TRIGGER SYNTAX (CONTD.)

- FOR|AFTER
  - AFTER specifies that the DML trigger is fired only when all operations specified in the triggering SQL statement have executed successfully.
  - AFTER is the default when FOR is the only keyword specified.
  - AFTER triggers cannot be defined on views.

- INSTEAD OF
  - Specifies that the trigger is executed *instead of* the triggering SQL statement, thus overriding the actions of the triggering statements.
  - Specifies At most, one INSTEAD OF trigger per INSERT, UPDATE, or DELETE statement can be defined on a table or view.

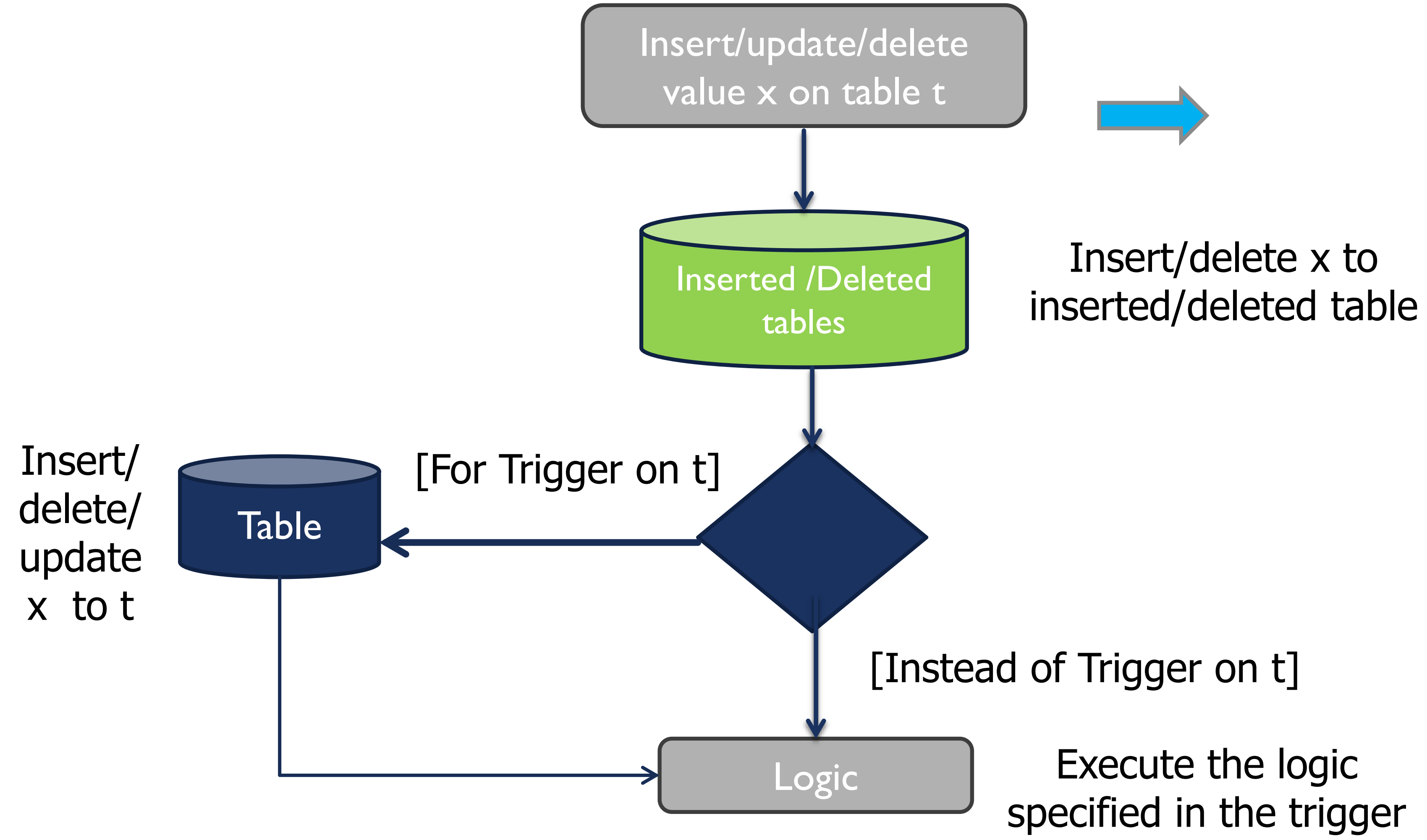
T-SQL: TRIGGER SYNTAX (CONTD.)

- [DELETE] [,] [INSERT] [,] [UPDATE]
  - Are keywords that specify which data modification statements, when attempted against this table or view, activate the trigger. At least one option must be specified.

TRIGGERS : INSERTED/DELETED TABLES

- When a trigger is executed SQL Server creates two virtual tables called INSERTED and DELETED.
- The **deleted** table stores copies of the affected rows during DELETE and UPDATE statements
- The **inserted** table stores copies of the affected rows during INSERT and UPDATE statements
    - For example when inserting a record to a table, SQL Server creates a virtual table call INSERTED and loads data into the inserted table then executes the trigger statements and writes the related data pages.
- The format of the inserted and deleted tables is the same as the format of the table on which the trigger is defined
- Each column in the inserted and deleted tables maps directly to a column in the base table

TRIGGERS : HOW DO THEY WORK?


Insert/update/delete value x on table t
Inserted /Deleted tables
Insert/delete x to inserted/deleted table
[For Trigger on t]
Table
Insert/ delete/ update x to t
[Instead of Trigger on t]
Logic
Execute the logic specified in the trigger

T-SQL: TRIGGERS(CONTD.)

- Example 1:
  - Consider tables below
    - Account (accountNo, custId, branch, balance)
    - AccountAudit (accountNo, balance, date)
  - Create a trigger to track all inserts/updates done to the balance field of an Account table at a bank in the AccountAudit table

T-SQL: TRIGGERS (CONTD.)

```
Create trigger account_audit_trigg
On Account
For Insert, update
As
Begin
    Declare @ano int
    Declare @balance float

    Select @ano=accountNo,@balance=balance from inserted
    Insert into accountAudit(@ano,@balance,getdate())
end
```

T-SQL: TRIGGERS (CONTD.)

- Example 2:
  - Consider following tables :
    - Emp(eid ,ename, age, salary)
    - Works (eid, did, pct-time)
    - Dept(did, budget, managerid)
  - Create a trigger to ensure that an employee doesn't work in more than 2 departments

EXERCISE 4

- Consider the following table
    - Transaction(tid, accountNo, type,amount,date)
        - Type may contain ‘credit’ or ‘debit’
- Assuming that the bank’s maximum withdrawal limit per day is 40000, write a trigger to ensure that no customer withdraws more than the given limit.

EXERCISE 5

- Consider the tables given below
  - Employee(nic, name, salary, dno)
  - Dept (dno, dname, mgrNic)
- Create a trigger to ensure that no employee has a salary greater than his/her manager.

SUMMARY

- Views
- Transaction Basics
- T-SQL extensions
- Stored Procedures
- Triggers