Functions, Views, System Catalog, Triggers, MERGE - in SQL Server

User-Defined Functions

- the developer can define his / her own functions, which can be used in SQL queries;
- 3 types of user-defined functions in SQL Server:

a. scalar
b. inline table-valued
c. multi-statement table-valued.

a. scalar functions:
- return a value;
- drawback - when a scalar function is operating on multiple rows, SQL Server executes the function once / row in the result set; this can have a significant impact on performance;
- example:

```
CREATE FUNCTION ufNoStudents(@age INT)
RETURNS INT AS
BEGIN
  DECLARE @no INT
  SET @no = 0
  SELECT @no= COUNT(*)
  FROM Students
  WHERE age = @age
  RETURN @no
END
GO

PRINT dbo.ufNoStudents(20)
```

b. inline table-valued functions
- return a table;
- can be called in the FROM clause of a T-SQL query;
- example:

```
CREATE FUNCTION ufStudentsNames(@age INT)
RETURNS TABLE
AS
  RETURN
    SELECT sname
    FROM Students
    WHERE age = @age
GO

SELECT *
FROM ufStudentsNames(20)
```

c. multi-statement table-valued functions
- return a table;
- unlike inline table-valued functions, multi-statement table-valued functions can contain more than one statement;

- example:

```
CREATE FUNCTION ufCoursesFilteredByCredits(@credits INT)
RETURNS @CoursesCredits TABLE (cid INT, cname VARCHAR(70))
AS
BEGIN
  INSERT INTO @CoursesCredits
  SELECT cid, cname
  FROM Courses
  WHERE credits = @credits

  IF @@ROWCOUNT = 0
    INSERT INTO @CoursesCredits
    VALUES (0,'No courses found with specified number of credits.')

  RETURN
END
GO

SELECT *
FROM ufCoursesFilteredByCredits(5)
```

Views

- a view creates a virtual table representing data from one or more tables in an alternative manner;
- contents of the virtual table (columns & rows) - defined by a query;
- can have at most 1024 columns;
- syntax:

```
CREATE VIEW view_name
AS SELECT_statement
```

- example:

```
CREATE OR ALTER VIEW vExaminations
AS
SELECT S.sid, S.sname, S.sgroup, C.cid, C.cname
FROM Students S INNER JOIN Exams E ON S.sid= E.studentid
  INNER JOIN Courses C ON E.courseid = C.cid
GO

SELECT *
FROM vExaminations
```

System Catalog

- stores data about the objects in the database (tables, columns, indexes, stored procedures, user-defined functions, views, etc);
- managed by the server (they are not modified directly by the user);

- examples:
- *sys.objects* – has one row for every object (constraint, stored procedure, table, etc) created in the database;
- *sys.columns* – one row for every column of an object that has columns, e.g., tables, views;
- *sys.sql_modules* – one row for every object that is a module defined in the SQL language in SQL Server (e.g., objects like procedures, functions, etc have an associated SQL module).

Triggers

- special type of stored procedure;
- automatically executed when a DML (INSERT, UPDATE, DELETE) or DDL statement (e.g., creating a database / table, removing / changing a table, eliminating a login, etc) is executed;
- they are not executed directly;
- syntax for INSERT / UPDATE / DELETE trigger on table / view;

```
CREATE TRIGGER <trigger_name >
 ON { table | view}
 [ WITH <dml_trigger_option> [ ,...n ] ]
 { FOR | AFTER | INSTEAD OF }
 { [INSERT] [,] [UPDATE] [,] [DELETE] }
 [ WITH APPEND ]
 [ NOT FOR REPLICATION ]
AS
 { sql_statement [;] [ ,...n ] |
EXTERNAL NAME <method specifier[;] > }
```

- the moment a trigger is executed is specified through one of the options:
  - FOR, AFTER - the DML trigger is fired only when all the operations specified in the triggering statement have executed successfully (multiple such triggers can be defined);
  - INSTEAD OF - the DML trigger is executed instead of the triggering statement;
- if there are multiple triggers defined for the same action, they are executed in a random order;
- when a trigger is executed, 2 special tables can be accessed: *inserted* and *deleted*;
- example:

```
CREATE TRIGGER When_adding_prod
  ON Products
  FOR INSERT
AS
BEGIN
  INSERT INTO BuyLog(PName, OperationDate, Quantity)
  SELECT PName, GETDATE(), Quantity
  FROM inserted
END
GO
```

```
CREATE TRIGGER [dbo].[When_deleting_prod]
  ON [dbo].[Products]
  FOR DELETE
```

```
AS
BEGIN
  SET NOCOUNT ON;
  INSERT INTO SellLog(PName, OperationDate, Quantity)
  SELECT PName, GETDATE(), Quantity
  FROM deleted
END
GO
```

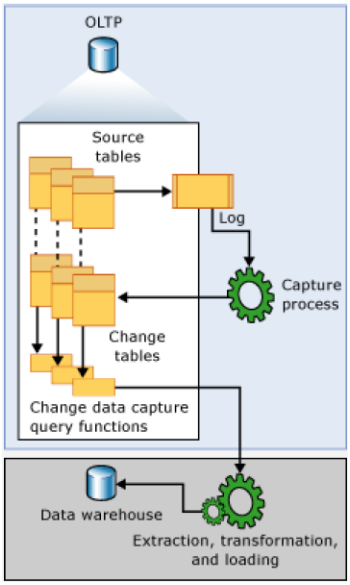
```
CREATE TRIGGER [dbo].[When_changing_prod]
  ON [dbo].[Products]
  FOR UPDATE
AS
BEGIN
  SET NOCOUNT ON;
  INSERT INTO SellLog(PName, OperationDate, Quantity)
  SELECT d.PName, GETDATE(), d.Quantity - i.Quantity
  FROM deleted d INNER JOIN inserted i ON d.PID = i.PID
  WHERE i.Quantity < d.Quantity

  INSERT INTO BuyLog(PName, OperationDate, Quantity)
  SELECT i.PName, GETDATE(), i.Quantity - d.Quantity
  FROM deleted d INNER JOIN inserted i ON d.PID = i.PID
  WHERE i.Quantity > d.Quantity
END
GO
```

SET NOCOUNT ON/OFF
- the number of rows affected by a T-SQL statement or stored procedure:
    - is not returned (ON);
    - is returned (OFF);
- @@ROWCOUNT is always modified.

Change Data Capture
- data about DML changes in the table / db;
- introduced in SQL Server 2008;
- sys.sp_cdc_enable_db – CDC for the database;
- sys.sp_cdc_enable_table – CDC for the monitored tables;
- data is archived and monitored without an additional programming effort (e.g., through triggers);
- user-created tables are monitored; corresponding data is stored in tables that can be queried using SQL, i.e., mirror tables - they contain the monitored tables' columns + metadata describing changes.

Databases
Seminar 4

Functions, Views, System Catalog, Triggers, MERGE - in SQL Server



The MERGE Statement

- a source table is compared with a target table; INSERT, UPDATE, DELETE statements can be executed based on the result of the comparison, i.e., INSERT / UPDATE / DELETE operations can be executed on the target table based on the result of a join with the source table.

```
MERGE TargetTable AS Target
USING SourceTable AS Source
ON (Search terms)
WHEN MATCHED THEN
  UPDATE SET
    or
  DELETE
WHEN NOT MATCHED [BY TARGET] THEN
  INSERT
WHEN NOT MATCHED BY SOURCE THEN
  UPDATE SET
    or
  DELETE
```

- example:
- the Books table:

	BookID	Title	Author	ISBN	Pages
1	1	In Search of Lost Time	Marcel Proust	NULL	NULL
2	2	In Search of Lost Time	NULL	NULL	350
3	3	In Search of Lost Time	NULL	9789731246420	NULL

Functions, Views, System Catalog, Triggers, MERGE - in SQL Server

```
MERGE Books
USING
      (SELECT MAX(BookID) BookID, Title, MAX(Author) Author,
        MAX(ISBN) ISBN, MAX(Pages) Pages
      FROM Books
      GROUP BY Title
      ) MergeData ON Books.BookID = MergeData.BookID
WHEN MATCHED THEN
      UPDATE SET Books.Title = MergeData.Title,
      Books.Author = MergeData.Author,
      Books.ISBN = MergeData.ISBN,
      Books.Pages = MergeData.Pages
WHEN NOT MATCHED BY SOURCE THEN DELETE;
```

	BookID	Title	Author	ISBN	Pages
1	3	In Search of Lost Time	Marcel Proust	9789731246420	350