Telerik Academy

Transact SQL (T-SQL)

Creating Stored Procedures, Functions and Triggers

Databases

Telerik Software Academy

http://academy.telerik.com

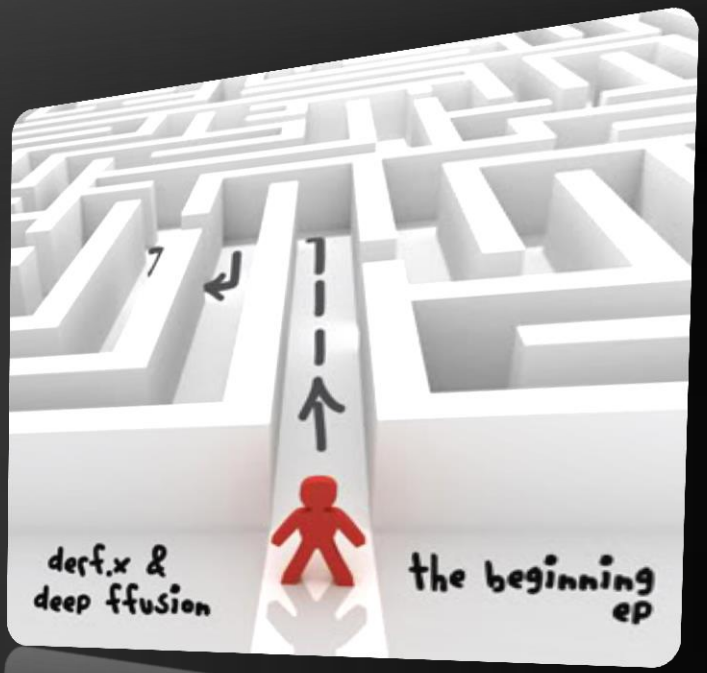
Telerik Academy

Table of Contents

Telerik Academy

Table of Contents (2)

Telerik Academy



Transact-SQL Language

Introduction

What is Transact-SQL

- Transact-SQL (T-SQL) is database manipulation language, an extension to SQL
  - Supported by Microsoft SQL Server and Sybase
  - Used for stored procedures, functions, triggers
- Transact-SQL extends SQL with few additional features:
  - Local variables
  - Control flow constructs (ifs, loops, etc.)
  - Functions for strings, dates, math, etc.

Types of T-SQL Statements

- There are 3 types of statements in the Transact-SQL (T-SQL) language:
  - Data Definition Language (DDL) Statements
  - Data Control Language (DCL) Statements
  - Data Manipulation Language (DML) Statements

Telerik Academy

Data Definition Language (DDL)

- Used to create, change and delete database objects (tables and others)
  - `CREATE <object> <definition>`
  - `ALTER <object> <command>`
  - `DROP <object>`
- The `<object>` can be a table, view, stored procedure, function, etc.
  - Some DDL commands require specific permissions

Telerik Academy

Data Control Language (DCL)

- Used to set / change permissions
  - GRANT – grants permissions
  - DENY – denies permissions
  - REVOKE – cancels the granted or denied permissions

```
USE Northwind
GRANT SELECT ON Products TO Public
GO
```

- As with DDL statements you must have the proper permissions

Data Manipulation Language (DML)

- Used to retrieve and modify table data
  - SELECT – query table data
  - INSERT – insert new records
  - UPDATE – modify existing table data (records)
  - DELETE – delete table data (records)

```
USE Northwind

SELECT CategoryId, ProductName, ProductId, UnitPrice
FROM Products
WHERE UnitPrice BETWEEN 10 and 20
ORDER BY ProductName
```

T-SQL Syntax Elements

- Batch Directives
- Identifiers
- Data Types
- Variables
- System Functions
- Operators
- Expressions
- Control-of-Flow Language Elements

Batch Directives

- `USE <database>`
  - Switch the active database
- `GO`
  - Separates batches (sequences of commands)
- `EXEC(<command>)`
  - Executes a user-defined or system function stored procedure, or an extended stored procedure
  - Can supply parameters to be passed as input
  - Can execute SQL command given as string

Telerik Academy

Batch Directives – Examples

```
EXEC sp_who – this will show all active users
```

```
USE TelerikAcademy

GO

DECLARE @table VARCHAR(50) = 'Projects'
SELECT 'The table is: ' + @table
DECLARE @query VARCHAR(50) = 'SELECT * FROM ' + @table;
EXEC(@query)

GO

-- The following will cause an error because
-- @table is defined in different batch
SELECT 'The table is: ' + @table
```

Identifiers

- Identifiers in SQL Server (e.g. table names)
  - Alphabetical character + sequence of letters, numerals and symbols, e.g. `FirstName`
  - Identifiers starting with symbols are special
- Delimited identifiers
  - Used when names use reserved words or contain embedded spaces and other characters
  - Enclose in brackets ([ ]) or quotation marks (" ")
  - E.g. `[First Name]`, `[INT]`, `"First + Last"`

Good Naming Practices

- Keep names short but meaningful
- Use clear and simple naming conventions
- Use a prefix that distinguishes types of object
  - Views – `V_AllUsers`, `V_CustomersInBulgaria`
  - Stored procedures – `usp_FindUsersByTown(…)`
- Keep object names and user names unique
  - Example of naming collision:
    - `Sales` as table name
    - `sales` as database role

Variables

- Variables are defined by DECLARE @ statement
  - Always prefixed by @, e.g. @EmpID
- Assigned by SET or SELECT @ statement
- Variables have local scope (until GO is executed)

```
DECLARE @EmpID varchar(11),
  @LastName char(20)
SET @LastName = 'King'
SELECT @EmpID = EmployeeId
 FROM Employees
 WHERE LastName = @LastName
SELECT @EmpID AS EmployeeID
GO
```

Telerik Academy

Data Types in SQL Server

- Numbers, e.g. `int`
- Dates, e.g. `datatime`
- Characters, e.g. `varchar`
- Binary, e.g. `image`
- Unique Identifiers (GUID)
- Unspecified type – `sql_variant`
- Table – set of data records
- Cursor – iterator over record sets
- User-defined types

System Functions

- Aggregate functions – multiple values → value

```
SELECT AVG(Salary) AS AvgSalary
FROM Employees
```

- Scalar functions – single value → single value

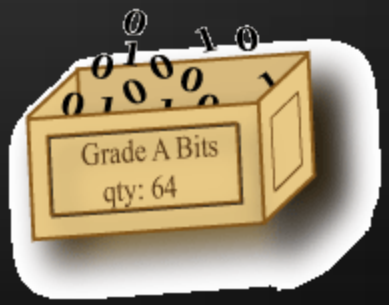
```
SELECT DB_NAME() AS [Active Database]
```

- Rowset functions – return a record set

```
SELECT *
FROM OPENDATASOURCE('SQLNCLI','Data Source =
  London\Payroll;Integrated Security = SSPI').
  AdventureWorks.HumanResources.Employee
```

Operators in SQL Server

- Types of operators
  - Arithmetic, e.g. +, -, *, /
  - Comparison, e.g. =, <>
  - String concatenation (+)
  - Logical, e.g. `AND`, `OR`, `EXISTS`



Expressions

- Expressions are combination of symbols and operators
  - Evaluated to single scalar value
  - Result data type is dependent on the elements within the expression

```
SELECT
  DATEDIFF(Year, HireDate, GETDATE()) * Salary / 1000
  AS [Annual Bonus]
FROM Employees
```

Control-of-Flow Language Elements

- Statement Level
  - `BEGIN ... END` block
  - `IF ... ELSE` block
  - `WHILE` constructs
- Row Level
  - `CASE` statements

IF … ELSE

- The IF … ELSE conditional statement is like in C#

```
IF ((SELECT COUNT(*) FROM Employees) >= 100)
  BEGIN
    PRINT 'Employees are at least 100'
  END
```

```
IF ((SELECT COUNT(*) FROM Employees) >= 100)
  BEGIN
    PRINT 'Employees are at least 100'
  END
ELSE
  BEGIN
    PRINT 'Employees are less than 100'
  END
```

WHILE Loops

- While loops are like in C#

```
DECLARE @n int = 10
PRINT 'Calculating factoriel of ' +
  CAST(@n as varchar) + ' ...'

DECLARE @factorial numeric(38) = 1
WHILE (@n > 1)
  BEGIN
    SET @factorial = @factorial * @n
    SET @n = @n - 1
  END

PRINT @factorial
```

CASE Statement

- CASE examines a sequence of expressions and returns different value depending on the evaluation results

```
SELECT Salary, [Salary Level] =
  CASE
    WHEN Salary BETWEEN 0 and 9999 THEN 'Low'
    WHEN Salary BETWEEN 10000 and 30000 THEN 'Average'
    WHEN Salary > 30000 THEN 'High'
    ELSE 'Unknown'
  END
FROM Employees
```

Control-of-Flow – Example

```
DECLARE @n tinyint
SET @n = 5
IF (@n BETWEEN 4 and 6)
 BEGIN
  WHILE (@n > 0)
   BEGIN
    SELECT @n AS 'Number',
      CASE
        WHEN (@n % 2) = 1
          THEN 'EVEN'
        ELSE 'ODD'
      END AS 'Type'
    SET @n = @n - 1
   END
 END
ELSE
 PRINT 'NO ANALYSIS'
GO
```

Telerik Academy

Stored Procedures

Telerik Academy

What are Stored Procedures?

- Stored procedures are named sequences of T-SQL statements
  - Encapsulate repetitive program logic
  - Can accept input parameters
  - Can return output results
- Benefits of stored procedures
  - Share application logic
  - Improved performance
  - Reduced network traffic

Telerik Academy

Creating Stored Procedures

- CREATE PROCEDURE … AS …
- Example:

```
USE TelerikAcademy
GO

CREATE PROC dbo.usp_SelectSeniorEmployees
AS
  SELECT *
  FROM Employees
  WHERE DATEDIFF(Year, HireDate, GETDATE()) > 5
GO
```

Telerik Academy

Executing Stored Procedures

- Executing a stored procedure by EXEC

```
EXEC usp_SelectSeniorEmployees
```

- Executing a stored procedure within an INSERT statement

```
INSERT INTO Customers
EXEC usp_SelectSpecialCustomers
```

Telerik Academy

Altering Stored Procedures

- Use the ALTER PROCEDURE statement

```
USE TelerikAcademy
GO

ALTER PROC dbo.usp_SelectSeniorEmployees
AS
  SELECT FirstName, LastName, HireDate,
    DATEDIFF(Year, HireDate, GETDATE()) as Years
  FROM Employees
  WHERE DATEDIFF(Year, HireDate, GETDATE()) > 5
  ORDER BY HireDate
GO
```

Telerik Academy

Dropping Stored Procedures

- DROP PROCEDURE

```
DROP PROC usp_SelectSeniorEmployees
```

  - Procedure information is removed from the sysobjects and syscomments system tables
- You could check if any objects depend on the stored procedure by executing the system stored procedure sp_depends

```
EXEC sp_depends 'usp_SelectSeniorEmployees'
```

Telerik Academy

Stored Procedures

Using Parameters

Defining Parameterized Procedures

- To define a parameterized procedure use the syntax:

```
CREATE PROCEDURE usp_ProcedureName
[(@parameter1Name parameterType,
  @parameter2Name parameterType,…)] AS
```

- Choose carefully the parameter types, and provide appropriate default values

```
CREATE PROC usp_SelectEmployeesBySeniority(
  @minYearsAtWork int = 5) AS
…
```

Parameterized Stored Procedures – Example

```
CREATE PROC usp_SelectEmployeesBySeniority(
  @minYearsAtWork int = 5)
AS
  SELECT FirstName, LastName, HireDate,
    DATEDIFF(Year, HireDate, GETDATE()) as Years
  FROM Employees
  WHERE DATEDIFF(Year, HireDate, GETDATE()) >
    @minYearsAtWork
  ORDER BY HireDate
GO

EXEC usp_SelectEmployeesBySeniority 10

EXEC usp_SelectEmployeesBySeniority
```

Passing Parameter Values

- Passing values by parameter name

```
EXEC usp_AddCustomer
    @customerID = 'ALFKI',
    @contactName = 'Maria Anders',
    @companyName = 'Alfreds Futterkiste',
    @contactTitle = 'Sales Representative',
    @address = 'Obere Str. 57',
    @city = 'Berlin',
    @postalCode = '12209',
    @country = 'Germany',
    @phone = '030-0074321'
```

- Passing values by position

```
EXEC usp_AddCustomer 'ALFKI2', 'Alfreds Futterkiste',
'Maria Anders', 'Sales Representative', 'Obere Str. 57',
'Berlin', NULL, '12209', 'Germany', '030-0074321'
```

Returning Values Using OUTPUT Parameters

```
CREATE PROCEDURE dbo.usp_AddNumbers
    @firstNumber smallint,
    @secondNumber smallint,
    @result int OUTPUT
AS
    SET @result = @firstNumber + @secondNumber
GO

DECLARE @answer smallint
EXECUTE usp_AddNumbers 5, 6, @answer OUTPUT
SELECT 'The result is: ', @answer

-- The result is: 11
```

Creating stored procedure

Executing stored procedure

Execution results

Returning Values Using The Return Statement

```
CREATE PROC usp_NewEmployee(
  @firstName nvarchar(50), @lastName nvarchar(50),
  @jobTitle nvarchar(50), @deptId int, @salary money)
AS
  INSERT INTO Employees(FirstName, LastName,
    JobTitle, DepartmentID, HireDate, Salary)
  VALUES (@firstName, @lastName, @jobTitle, @deptId,
    GETDATE(), @salary)
  RETURN SCOPE_IDENTITY()
GO

DECLARE @newEmployeeId int
EXEC @newEmployeeId = usp_NewEmployee
  @firstName='Steve', @lastName='Jobs', @jobTitle='Trainee',
  @deptId=1, @salary=7500

SELECT EmployeeID, FirstName, LastName
FROM Employees
WHERE EmployeeId = @newEmployeeId
```

Telerik Academy

Triggers

What Are Triggers?

- Triggers are very much like stored procedures
  - Called in case of specific event
- We do not call triggers explicitly
  - Triggers are attached to a table
  - Triggers are fired when a certain SQL statement is executed against the contents of the table
  - E.g. when a new row is inserted in given table

Types of Triggers

- There are two types of triggers
  - After triggers
  - Instead-of triggers
- After triggers
  - Fired when the SQL operation has completed and just before committing to the database
- Instead-of triggers
  - Replace the actual database operations

After Triggers

- Also known as "for-triggers" or just "triggers"
- Defined by the keyword FOR

```
CREATE TRIGGER tr_TownsUpdate ON Towns FOR UPDATE
AS
  IF (EXISTS(SELECT * FROM inserted WHERE Name IS NULL) OR
     EXISTS(SELECT * FROM inserted WHERE LEN(Name) = 0))
    BEGIN
      RAISERROR('Town name cannot be empty.', 16, 1)
      ROLLBACK TRAN
      RETURN
    END
GO

UPDATE Towns SET Name='' WHERE TownId=1
```

This will cause and error

Instead Of Triggers

- Defined by using INSTEAD OF

```
CREATE TABLE Accounts(
  Username varchar(10) NOT NULL PRIMARY KEY,
  [Password] varchar(20) NOT NULL,
  Active CHAR NOT NULL DEFAULT 'Y')
GO

CREATE TRIGGER tr_AccountsDelete ON Accounts
  INSTEAD OF DELETE
AS
  UPDATE a SET Active = 'N'
  FROM Accounts a JOIN DELETED d
    ON d.Username = a.Username
  WHERE a.Active = 'Y'
GO
```

Telerik Academy

User-Defined Functions

Telerik Academy

Types of User-Defined Functions

- Scalar functions (like SQRT(…))
  - Similar to the built-in functions
- Table-valued functions
  - Similar to a view with parameters
  - Return a table as a result of single SELECT statement
- Aggregate functions (like SUM(…))
  - Perform calculation over set of inputs values
  - Defined through external .NET functions

Telerik Academy

Creating and Modifying Functions

- To create / modify / delete function use:
  - CREATE FUNCTION <function_name> RETURNS <datatype> AS …
  - ALTER FUNCTION / DROP FUNCTION

```
CREATE FUNCTION ufn_CalcBonus(@salary money)
  RETURNS money
AS
BEGIN
  IF (@salary < 10000)
    RETURN 1000
  ELSE IF (@salary BETWEEN 10000 and 30000)
    RETURN @salary / 20
  RETURN 3500
END
```

Telerik Academy

Scalar User-Defined Functions

- Can be invoked at any place where a scalar expression of the same data type is allowed
- RETURNS clause
  - Specifies the returned data type
  - Return type is any data type except `text`, `ntext`, `image`, `cursor` or `timestamp`
- Function body is defined within a `BEGIN...END` block
  - Should end with `RETURN <some value>`

Inline Table-Valued Functions

- Inline table-valued functions
  - Return a table as result (just like a view)
  - Could take some parameters
- The content of the function is a single `SELECT` statement
  - The function body does not use `BEGIN` and `END`
  - `RETURNS` specifies `TABLE` as data type
- The returned table structure is defined by the result set

Inline Table-Valued Functions Example

- Defining the function

```
USE Northwind
GO
CREATE FUNCTION fn_CustomerNamesInRegion
  ( @regionParameter nvarchar(30) )
RETURNS TABLE
AS
RETURN (
  SELECT CustomerID, CompanyName
  FROM Northwind.dbo.Customers
  WHERE Region = @regionParameter
)
```

- Calling the function with a parameter

```
SELECT * FROM fn_CustomerNamesInRegion(N'WA')
```

Multi-Statement Table-Valued Functions

- BEGIN and END enclose multiple statements
- RETURNS clause – specifies table data type
- RETURNS clause – names and defines the table

Multi-Statement Table-Valued Function – Example

```
CREATE FUNCTION fn_ListEmployees(@format nvarchar(5))
RETURNS @tbl_Employees TABLE
  (EmployeeID int PRIMARY KEY NOT NULL,
  [Employee Name] Nvarchar(61) NOT NULL)
AS
BEGIN
  IF @format = 'short'
    INSERT @tbl_Employees
    SELECT EmployeeID, LastName FROM Employees
  ELSE IF @format = 'long'
    INSERT @tbl_Employees SELECT EmployeeID,
    (FirstName + ' ' + LastName) FROM Employees
  RETURN
END
```

Telerik Academy

InvoiceNo	InvoiceDate	PaymentDate	CustomerNo	SalesPerson	ProductNo	Quantity	UnitPrice	Amount	Descripti
20,000.00	1/1/2008	3/5/2008 2:10:05 PM	10,220.00	8.00	8.00	4.00	299.00	1,196.00	Excel for
20,001.00	1/1/2008	2/9/2008 8:17:45 PM	10,491.00	4.00	4.00	4.00	279.00	1,116.00	Excel ver
20,002.00	1/1/2008	2/22/2008 12:58:04 PM	10,704.00	3.00	1.00	3.00	299.90	899.70	ActiveDa
20,003.00	1/1/2008	2/9/2008 6:20:35 PM	10,430.00	5.00	54.00	4.00	199.00	796.00	Word ver
20,004.00	1/1/2008	2/28/2008 12:15:48 PM	10,841.00	17.00	11.00	2.00	129.00	458.00	Expressio
20,005.00	1/1/2008	2/24/2008 6:44:27 AM	10,777.00	1.00	5.00	4.00	229.00	916.00	CCR DS
20,006.00	1/1/2008	2/5/2008 5:12:19 PM	10,653.00	19.00	58.00	2.00	129.00	458.00	Windows
20,007.00	1/1/2008	2/27/2008 3:36:27 PM	10,413.00	12.00	61.00	3.00	3,429.00	10,287...	Applicatio
20,008.00	1/1/2008	1/23/2008 9:39:27 AM	10,654.00	12.00	4.00	3.00	279.00	837.00	Excel ver
20,009.00	1/1/2008	1/26/2008 5:07:01 PM	10,300.00	1.00	10.00	2.00	279.00	558.00	Expressio
20,010.00	1/1/2008	2/21/2008 8:23:11 PM	10,439.00	19.00	38.00	4.00	268.00	1,072.00	Office Vis
20,010.00	1/2/2008	2/6/2008 11:54:01 PM	10,439.00	99.00	38.00	4.00	268.00	1,072.00	Office Vis
20,012.00	1/2/2008	1/30/2008 7:49:39 PM	10,919.00	11.00	31.00	4.00	179.00	716.00	Office Or
20,013.00	1/2/2008	2/19/2008 1:43:16 PM	10,600.00	8.00	27.00	2.00	279.00	558.00	Office SH
20,014.00	1/2/2008	3/1/2008 3:16:14 AM	10,511.00	19.00	22.00	4.00	689.00	2,756.00	Office Er
20,015.00	1/2/2008	2/17/2008 3:26:46 PM	10,353.00	17.00	45.00	2.00	47.89	295.78	TechNet
20,016.00	1/2/2008	2/1/2008 7:29:23 AM	10,792.00	21.00	4.00	8.00	279.00	2,232.00	Excel ver
20,017.00	1/2/2008	2/22/2008 2:36:32 AM	10,516.00	7.00	38.00	3.00	268.00	804.00	Office Vis
20,018.00	1/2/2008	2/7/2008 6:46:09 PM	10,411.00	11.00	65.00	3.00	779.00	2,337.00	BizTalk®
20,019.00	1/2/2008	1/26/2008 10:52:18 PM	10,004.00	4.00	16.00	3.00	179.00	537.00	MapPoin

Working with Cursors

Processing Each Record in a Record Set

Working with Cursors

```
DECLARE empCursor CURSOR READ_ONLY FOR
  SELECT FirstName, LastName FROM Employees

OPEN empCursor
DECLARE @firstName char(50), @lastName char(50)
FETCH NEXT FROM empCursor INTO @firstName, @lastName

WHILE @@FETCH_STATUS = 0
  BEGIN
    PRINT @firstName + ' ' + @lastName
    FETCH NEXT FROM empCursor
    INTO @firstName, @lastName
  END

CLOSE empCursor
DEALLOCATE empCursor
```

Telerik Academy

Transact SQL

Questions?

http://academy.telerik.com

Exercises

1. Create a database with two tables: `Persons(Id(PK), FirstName, LastName, SSN)` and `Accounts(Id(PK), PersonId(FK), Balance)`. Insert few records for testing. Write a stored procedure that selects the full names of all persons.
2. Create a stored procedure that accepts a number as a parameter and returns all persons who have more money in their accounts than the supplied number.
3. Create a function that accepts as parameters – sum, yearly interest rate and number of months. It should calculate and return the new sum. Write a `SELECT` to test whether the function works as expected.

Exercises (2)

4. Create a stored procedure that uses the function from the previous example to give an interest to a person's account for one month. It should take the `AccountId` and the interest rate as parameters.
5. Add two more stored procedures `WithdrawMoney(AccountId, money)` and `DepositMoney (AccountId, money)` that operate in transactions.
6. Create another table – `Logs(LogID, AccountID, OldSum, NewSum)`. Add a trigger to the `Accounts` table that enters a new entry into the `Logs` table every time the sum on an account changes.

Exercises (3)

7. Define a function in the database `TelerikAcademy` that returns all Employee's names (first or middle or last name) and all town's names that are comprised of given set of letters. Example '`oistmiahf`' will return '`Sofia`', '`Smith`', … but not '`Rob`' and '`Guy`'.
8. Using database cursor write a T-SQL script that scans all employees and their addresses and prints all pairs of employees that live in the same town.
9. * Write a T-SQL script that shows for each town a list of all employees that live in it. Sample output:

```
Sofia -> Svetlin Nakov, Martin Kulov, George Denchev
Ottawa -> Jose Saraiva
…
```

Exercises (4)

10. Define a .NET aggregate function `StrConcat` that takes as input a sequence of strings and return a single string that consists of the input strings separated by `','`. For example the following SQL statement should return a single string:

```
SELECT StrConcat(FirstName + ' ' + LastName)
FROM Employees
```

Free Trainings @ Telerik Academy

- "Web Design with HTML 5, CSS 3 and JavaScript" course @ Telerik Academy
  - html5course.telerik.com

- Telerik Software Academy
  - academy.telerik.com

- Telerik Academy @ Facebook
  - facebook.com/TelerikAcademy

- Telerik Software Academy Forums
  - forums.academy.telerik.com