

# **NET3000**

# **Database Concepts and SQL**

Instructor: Phil Kaufman

## **Lecture 8**

## **Stored Procedures and Triggers**

November 16, 2015

# Agenda

- What is a Stored Procedure
  - ◆ Definition of a stored procedure
  - ◆ Syntax to create a stored procedure
  - ◆ How to drop/delete
  - ◆ How to run a stored procedure
- Useful system stored procedures for stored procedures
- What can you do with a stored procedure
- How can we return values from a stored procedure

# Agenda (con't)

- What is a Database Trigger
- What triggers a Trigger
- Types of Triggers
- Benefits of Triggers
- Pseudo-tables in Triggers
- How to create a Trigger
- How to drop a trigger

# What is a Stored Procedure

## ■ A stored Procedure

- ♦ Is a sequence of SQL commands stored in the database catalogue so that it can be invoked later by a program
- ♦ Has parameters that can be passed to the the procedure
- ♦ May contain declared variables, comments
- ♦ May contain flow control statements (eg. Logic like IF-THEN-ELSE, Looping)
- ♦ May use cursors
  - ♦ Cursor is a control that iterates through a set of rows returned by a query so that data can be processed
- ♦ Is a DDL statement

# What is a Stored Procedure (con't)

- The syntax to create a stored procedure is:

```
CREATE PROCEDURE procedure_name
    [ { @parameter data_type }
    ] [ , ...n ]
AS
    { [ BEGIN ] sql_statement [ ; ] [ ...n ] [ END ] }
```

# What is a Stored Procedure (con't)

- An Example Stored Procedure is:

```
CREATE PROCEDURE usp_GetAddress @City nvarchar(30)
AS
    BEGIN
        SELECT *
        FROM AdventureWorks.Person.Address
        WHERE City = @City;
    END
GO
```

# What is a Stored Procedure (con't)

- The syntax to drop a stored procedure is:

```
DROP PROCEDURE procedure_name
```

- For Example:

```
DROP PROCEDURE usp_GetAddress  
GO
```

# What is a Stored Procedure (con't)

- The syntax to execute a stored procedure is:

```
EXEC[UTE] sp_name [parameters]
```

- For Example:

```
EXECUTE usp_GetAddress @City = 'New York'  
GO
```



# Useful system stored procedures

- Useful system Stored Procedures for Stored procedures:
  - ♦ **sp\_help** 'sp\_name'
    - ♦ returns general information about the stored procedure, such as storage, parameters, etc
  - ♦ **sp\_helptext** 'sp\_name'
    - ♦ returns the definition of the stored procedure
  - ♦ **sp\_depends** 'sp\_name'
    - ♦ returns dependencies for the stored procedure

# What can you do with a stored procedure

- Procedures are useful for a host of actions within the database
  - ♦ Return a result set (for display, for inserting)
  - ♦ Complete a number of DML operations (such as UPDATES, INSERTS, DELETES)
  - ♦ Return a calculated value to a calling SQL statement (such as a SELECT statement)
  - ♦ Many many more uses!

# Returning data from a stored procedure

- Return a **result set**
- Limited to returning a single result set
- Example

```
CREATE PROCEDURE dbo.usp_GetPeopleByLastName (@LastName NVARCHAR(50))
AS
    SELECT ContactID, FirstName, LastName
    FROM Person.Contact
    WHERE LastName = @LastName
    ORDER BY ContactID
GO
```

# Returning data from a stored procedure (con't)

## ■ Example (con't)

```
EXEC dbo.usp_GetPeopleByLastName @LastName = 'Alexander'  
GO
```

ContactID	FirstName	LastName
22	J. Phillip	Alexander
23	Michelle	Alexander
430	Mary	Alexander

. . .

(123 row(s) affected)

# Returning data from a stored procedure

- Return using **OUTPUT** variable(s)
- Can return multiple values
- Example

```
CREATE PROCEDURE dbo.usp_GetCountByLastName (@LastName NVARCHAR(50),  
                                              @LastNameCount INT OUTPUT)  
AS  
    SELECT @LastNameCount = COUNT(*)  
    FROM Person.Contact  
    WHERE LastName = @LastName  
GO
```

# Returning data from a stored procedure (con't)

## ■ Example (con't)

```
DECLARE @TheCount INT
```

```
EXEC dbo.usp_GetCountByLastName @LastName = 'Alexander',  
                                @LastNameCount = @TheCount OUTPUT
```

```
SELECT TheCount = @TheCount  
GO
```

```
TheCount  
-----  
123
```

```
(1 row(s) affected)
```

# Returning data from a stored procedure

- Return using **RETURN**
- Is most limiting as returns a single value
- Example

```
CREATE PROCEDURE dbo.usp_TestReturn (@InValue int)
AS
    Return @Invalue
GO
```

# Returning data from a stored procedure (con't)

## ■ Example (con't)

```
DECLARE @ReturnValue INT
```

```
EXECUTE @ReturnValue = usp_TestReturn 3
```

```
SELECT ReturnValue = @ReturnValue
```

```
ReturnValue
```

```
-----
```

```
3
```

```
(1 row(s) affected)
```



# What is a Database Trigger

- A Database Trigger is a stored procedure code block that is automatically executed in response to certain database events (as defined by the database engine)
- Events are based on tables or views data
- Triggers are most commonly used for maintaining the data integrity of the information in the database
- Triggers can be used in creating audit records and reflecting changes to crucial business tables, and validating changes against a set of business rules
- Triggers are assigned to a single tables and views

# What triggers a Trigger!

- A Database Trigger is triggered by a **DML** statement (DDL and login triggers are beyond the scope of this course)
- DML statements that may trigger a Trigger are
  - ◆ INSERT
  - ◆ UPDATE
  - ◆ DELETE
- Fired once per DML statement (not per affected row)
- Cannot be directly called like a stored procedure

# Types of Triggers

- Types of Triggers (defines when they fire) are:
  - ◆ AFTER
  - ◆ INSTEAD OF
- AFTER
  - ◆ is executed after the action of the INSERT, UPDATE, or DELETE statement is performed
- INSTEAD OF
  - ◆ overrides the standard action of the triggering statement

# Benefits of Triggers

- Triggers can cascade changes through related tables
- Triggers can guard against malicious or incorrect INSERT, UPDATE, and DELETE operations.
- Triggers can evaluate the state of a table before and after a data modification and take action based on that difference (see pseudo-tables)
- Triggers can supply superior error messaging in your application and database

# Pseudo-tables in Triggers

- Triggers have two pseudo-tables available within the trigger at execution time
  - ◆ Deleted
  - ◆ Inserted
- Deleted
  - Contains rows before the DML affected rows
- Inserted
  - Contains rows after the DML affected rows

# Pseudo-tables (con't)

Operation	<i>deleted</i> table	<i>inserted</i> table
INSERT	(not used)	Contains rows being inserted
DELETE	Contains rows being deleted	(not used)
UPDATE	Contains rows as they were before the UPDATE statement	Contains rows as they are after the UPDATE statement

# How to create a Trigger

- The syntax to create a trigger is:

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

- { sql\_statement [ ; ] [ ,...n ] }

# How to create a Trigger (con't)

- An Example Trigger is:

```
CREATE TRIGGER trg_EmployeeUpdate
ON dbo.Employees AFTER UPDATE
AS
    INSERT INTO dbo.LogTable(ID, OldValue, NewValue)
    SELECT i.ID, d.Name, i.Name
    FROM inserted i
    INNER JOIN deleted d ON i.ID = d.ID
GO
```



# How to delete a Trigger

- The syntax to drop a trigger is:

```
DROP TRIGGER trigger_name
```

- For Example:

```
DROP TRIGGER trg_EmployeeUpdate  
GO
```

# Conclusion

We learned about:

- **Stored Procedures**

- ♦ **How to define them**
- ♦ **Example uses of them**
- ♦ **How to return values from them**

- **Database Triggers**

- ♦ **What triggers a Trigger**
- ♦ **Types of Triggers**
- ♦ **Benefits**