Control-Of-Flow Language

Change Data Capture

S4

Control-of-Flow Language

```
BEGIN...END
RETURN
BREAK
THROW
CONTINUE
TRY...CATCH
GOTO label
WAITFOR
IF...ELSE
WHILE
```

RETURN

```
RETURN [ integer_expression ]
```

- Exits unconditionally from a query or procedure

- Returning from a procedure
- Returning status codes
  - stored procs return 0 (success), or
  - a nonzero value (failure)

RETURN

```
CREATE PROCEDURE checkstate @param varchar(11)
AS
  IF @param= 'WA'
   RETURN 1
  ELSE
   RETURN 2;
GO

-----------------------------

DECLARE @return_status int;
EXEC @return_status = checkstate ‘AK';
GO
```

WHILE

```
WHILE Boolean_expression
    { sql_statement | statement_block | BREAK |
      CONTINUE }
```

- Sets a condition for the repeated execution of an SQL statement or statement block

BREAK

- Exits the innermost loop in a WHILE statement or an IF…ELSE statement inside a WHILE loop.

CONTINUE

- Restarts a WHILE loop. Any statements after the CONTINUE keyword are ignored.

GOTO

- Alters the flow of execution to a label

```
Label:
GOTO Label
```

WAITFOR

```
WAITFOR {  DELAY 'time_to_pass' |
           TIME 'time_to_execute' |
           [ ( receive_statement ) |
           ( get_conversation_group_statement ) ]
           [ , TIMEOUT timeout ] }
```

- Blocks the execution of a batch, stored procedure, or transaction

WAITFOR

- Execution continues at 08:35

```
WAITFOR TIME ‘08:35';
```

- Execution continues after 2 hours

```
WAITFOR DELAY '02:00';
```

- if the server is busy → the counter does not start immediately → the counter the delay may be longer than specified.

THROW

```
THROW [
      { error_number | @local_variable},
      { message | @local_variable},
      { state | @local_variable } ] [ ; ]
```

- Raises an exception and transfers execution to a CATCH block of a TRY…CATCH construct
- The exception severity is always set to 16.

```
THROW 51000, ‘Record does not exist', 1;
```

TRY … CATCH

```
BEGIN TRY
     { sql_statement | statement_block }
END TRY
BEGIN CATCH
      [ { sql_statement | statement_block } ]
END CATCH [ ; ]
```

- Implements error handling for Transact-SQL
- catches all execution errors that have a severity >10 that do not close the database connection

TRY … CATCH

- ERROR_NUMBER() returns the error number
- ERROR_SEVERITY() returns the severity
- ERROR_STATE() returns the error state number
- ERROR_PROCEDURE() returns the name of the stored procedure/trigger where the error occurred
- ERROR_LINE() returns the line number that caused the error
- ERROR_MESSAGE() returns the error message

Error Messages

- Error number
  - Integer value between 1 and 49999
  - Custom error messages: 50001…
- Error severity
  - 26 severity levels
  - Error with severity level ≥ 16 are logged automatically
  - Error with severity level between 20 and 25 are fatal and the connection is terminated
- Error message: up to 255 chars

Triggers

= special types of stored procedures that automatically execute when a DML or DDL statement is executed

- cannot be executed directly
- DML statements: INSERT, UPDATE, DELETE
- DDL statements: CREATE_DATABASE, DROP_LOGIN, UPDATE_STATISTICS, DROP_TRIGGER, ALTER_TABLE

Triggers

```
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 [;] > }
```

Triggers

- Moment of execution:
  - FOR
  - AFTER (multiple triggers could be defined)
  - INSTEAD OF
- if multiple triggers are defined for the same action they are executed in random order
- when a trigger is executed 2 special tables named *inserted* and *deleted* are available

Triggers

```
CREATE TRIGGER [dbo].[On_Product_Insert]
   ON  [dbo].[Products]
   FOR INSERT
AS
BEGIN
      SET NOCOUNT ON;

      insert into LogBuys (Name, Date, Quantity)
      select Name, GETDATE(), Quantity
      from inserted

END
```

Triggers

```
CREATE TRIGGER [dbo].[On_Product_Delete]
   ON  [dbo].[Products]
   FOR DELETE
AS
BEGIN
      SET NOCOUNT ON;

      insert into LogSells (Name, Date, Quantity)
      select Name, GETDATE(), Quantity
      from deleted
END
```

Triggers

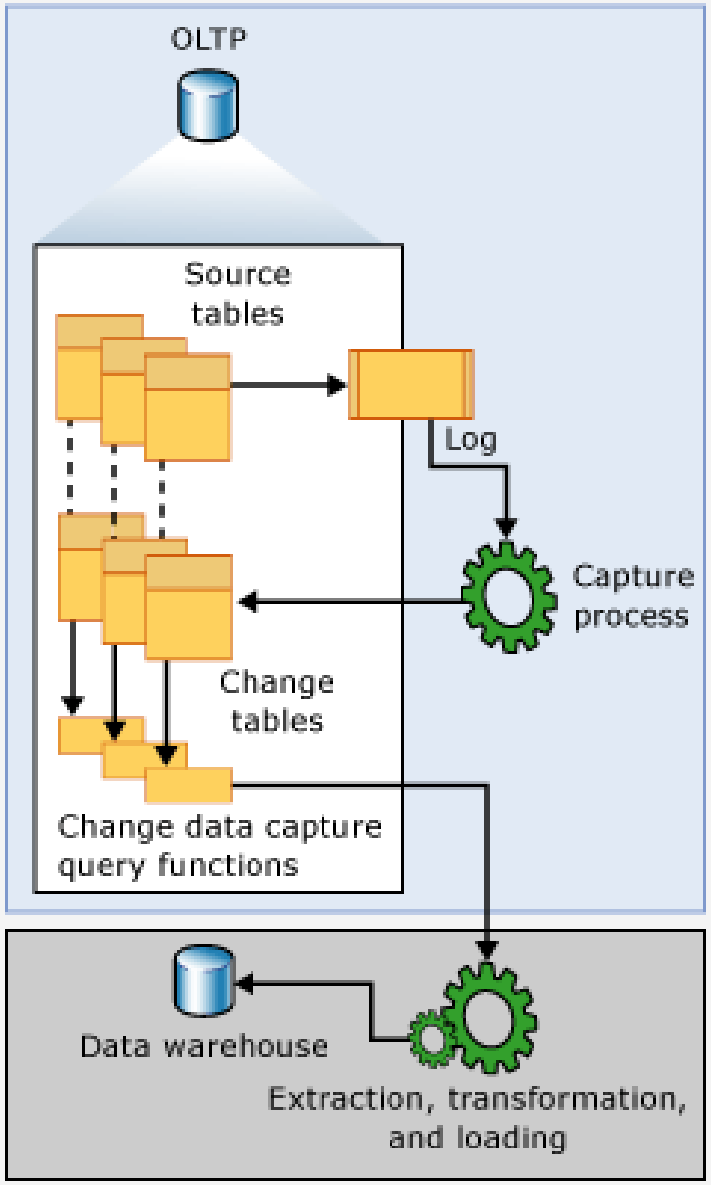
```
ALTER TRIGGER [dbo].[On_Product_Update]
   ON  [dbo].[Products]
   FOR UPDATE
AS
BEGIN
 SET NOCOUNT ON;
 insert into LogSells (Name, Date, Quantity)
 select deleted.Name,GETDATE(),deleted.Quantity-
inserted.Quantity
   from deleted d inner join inserted i on d.ID=i.ID
       where i.Quantity < d.Quantity

 insert into LogBuys (Name, Date, Quantity)
 select i.Name, GETDATE(), i.Quantity-d.Quantity
   from deleted d inner join inserted i on d.ID = i.ID
       where i.Quantity > d.Quantity
END
```

SET NOCOUNT ON/OFF

- ON - the count is not returned.
- OFF - the count is returned

- @@ROWCOUNT is always updated.

Change Data Capture



- provides information about DML changes on a table and a database.
- Introduced in SQL Server 2008

OUTPUT clause

- provides access to *inserted, updated* or *deleted* records
- can implement certain functionalities performed only through triggers

```
UPDATE Categories
SET CategoryName = 'Dried Produce'
OUTPUT inserted.CategoryID,
       deleted.CategoryName,
       inserted.CategoryName, get_date(),
       SUSER_SNAME()
INTO CategoryChanges
WHERE CategoryID = 7
```

MERGE statement and CDC

- MERGE – gives the ability to compare rows in a source and a destination table.
- INSERT, UPDATE or DELETE commands could be performed based on the result of this comparison

MERGE – General syntax

```
Merge Table definition as Target
Using ( Table Source ) as Source
(
Column Keys
)
ON (
Search Terms
)
WHEN MATCHED THEN
      UPDATE SET
         or
      DELETE
WHEN NOT MATCHED BY TARGET/SOURCE THEN
INSERT
```

MERGE sample

Books table

	BookId	Title	Author	ISBN	Pages
1	1	Microsoft SQL Server 2005 For Dummies	Andrew Watt	NULL	NULL
2	2	Microsoft SQL Server 2005 For Dummies	NULL	NULL	432
3	3	Microsoft SQL Server 2005 For Dummies	NULL	978-0-7645-7755-0	NULL

MERGE sample

```
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;
```