# Triggers

- Triggers define code to be executed automatically when a certain event happens. Triggers takes no parameters.
- DDL Trigger
  - CREATE, ALTER, DROP
- DML trigger
  - INSERT, UPDATE, DELETE
- Trigger can be set on Server level, Database level and Table level
- logon triggers are skipped
  - Logon triggers fire in response to a LOGON event

# Triggers

- You need to be familiar with the different types of DDL and DML triggers, but don’t have to dig too deep
- Being familiar with the core concepts and knowing how to create basic DDL and DML triggers should be sufficient
- Additional topics like logon triggers and different optional settings will not be covered.

# DML Triggers

- FOR and AFTER triggers are equivalent

```
CREATE [ OR ALTER ] TRIGGER [ schema_name . ]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 [ ; ] > }

--<dml_trigger_option> ::=
--    [ ENCRYPTION ]
--    [ EXECUTE AS Clause ]

--<method_specifier> ::=
--    assembly_name.class_name.method_name
```



Antra

# DML Triggers

```
CREATE TRIGGER NewPODetail3
ON Purchasing.PurchaseOrderDetail
FOR INSERT AS
IF @@ROWCOUNT = 1
BEGIN
   UPDATE Purchasing.PurchaseOrderHeader
   SET SubTotal = SubTotal + LineTotal
   FROM inserted
   WHERE PurchaseOrderHeader.PurchaseOrderID = inserted.PurchaseOrderID

END
ELSE
BEGIN
      UPDATE Purchasing.PurchaseOrderHeader
   SET SubTotal = SubTotal +
      (SELECT SUM(LineTotal)
      FROM inserted
      WHERE PurchaseOrderHeader.PurchaseOrderID
       = inserted.PurchaseOrderID)
   WHERE PurchaseOrderHeader.PurchaseOrderID IN
      (SELECT PurchaseOrderID FROM inserted)
END;
```

- @@rowcount returns the number of rows affected by the last statement

# DDL Triggers

- Check documentation for different database/server level events

```
CREATE TABLE TableSchemaChanges (ChangeEvent xml, DateModified datetime)

CREATE TRIGGER TR_ALTERTABLE ON DATABASE -- scope is a database
FOR ALTER_TABLE --DDL event                rather than a table
AS
BEGIN

INSERT INTO TableSchemaChanges
SELECT EVENTDATA(),GETDATE()

END
```



Antra

# Control Flow



Antra

# TRY CATCH

- Error handling is done by using try catch.

```
SET @saleid = NEWID()
  BEGIN TRY
    BEGIN TRANSACTION
    INSERT INTO Sales.Sales
         SELECT
           @saleid,
           @productid,
           @employeeid,
           @quantity
    COMMIT TRANSACTION
  END TRY
  BEGIN CATCH
    INSERT INTO dbo.DB_Errors
    VALUES
  (SUSER_SNAME(),
   ERROR_NUMBER(),
   ERROR_STATE(),
   ERROR_SEVERITY(),
   ERROR_LINE(),
   ERROR_PROCEDURE(),
   ERROR_MESSAGE(),
   GETDATE());

-- Transaction uncommittable
    IF (XACT_STATE()) = -1
      ROLLBACK TRANSACTION

-- Transaction committable
    IF (XACT_STATE()) = 1
      COMMIT TRANSACTION
  END CATCH
```

# THROW

- Used to manually raise an error
- often used when database operations are working fine, but violating business rules
- meaning of error_number and state are defined by user, error_number needs to be between 50000 and 2147483647, state needs to be between 0 and 255

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

```
THROW 51000, 'The record does not exist.', 1;
```



Antra

# IF ELSE

```
IF Boolean_expression
     { sql_statement | statement_block }
[ ELSE
     { sql_statement | statement_block } ]
```

```
DECLARE @Number INT;
SET @Number = 50;
IF @Number > 100
   PRINT 'The number is large.';
ELSE
   BEGIN
      IF @Number < 10
      PRINT 'The number is small.';
   ELSE
      PRINT 'The number is medium.';
   END ;
GO
```

```
IF DATENAME(weekday, GETDATE()) IN (N'Saturday', N'Sunday')
       SELECT 'Weekend';
ELSE
       SELECT 'Weekday';
```



Antra

# WHILE

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

```
USE AdventureWorks2012;
GO
WHILE (SELECT AVG(ListPrice) FROM Production.Product) < $300
BEGIN
   UPDATE Production.Product
      SET ListPrice = ListPrice * 2
   SELECT MAX(ListPrice) FROM Production.Product
   IF (SELECT MAX(ListPrice) FROM Production.Product) > $500
      BREAK
   ELSE
      CONTINUE
END
PRINT 'Too much for the market to bear';
```

# BREAK CONTINUE

- BREAK
  - exits the current WHILE loop.
  - If the current WHILE loop is nested inside another, BREAK exits only the current loop, and control is given to the next statement in the outer loop.
- Continue
  - Skip the current iteration of the loop. Any statements after the CONTINUE keyword are ignored.

# Additional Topics



Antra

# Cascading Referential Integrity

- NO ACTION
  - The Database Engine raises an error and the delete or update action on the row in the parent table is rolled back.
- CASCADE
  - Corresponding rows are updated or deleted in the referencing table when that row is updated or deleted in the parent table.
  - CASCADE cannot be specified if a timestamp column is part of either the foreign key or the referenced key.
  - ON DELETE CASCADE cannot be specified for a table that has an INSTEAD OF DELETE trigger.
  - ON UPDATE CASCADE cannot be specified for tables that have INSTEAD OF UPDATE triggers.

# Cascading Referential Integrity

- SET NULL
  - All the values that make up the foreign key are set to NULL when the corresponding row in the parent table is updated or deleted.
  - the foreign key columns must be nullable.
  - Cannot be specified for tables that have INSTEAD OF UPDATE triggers.
- SET DEFAULT
  - All the values that make up the foreign key are set to their default values if the corresponding row in the parent table is updated or deleted.
  - all foreign key columns must have default definitions. If a column is nullable, and there is no explicit default value set, NULL becomes the implicit default value of the column.
  - Cannot be specified for tables that have INSTEAD OF UPDATE triggers.

# Cascading Referential Integrity

```sql
CREATE TABLE products
( product_id INT PRIMARY KEY,
  product_name VARCHAR(50) NOT NULL,
  category VARCHAR(25)
);

CREATE TABLE inventory
( inventory_id INT PRIMARY KEY,
  product_id INT NOT NULL,
  quantity INT,
  min_level INT,
  max_level INT,
  CONSTRAINT fk_inv_product_id
    FOREIGN KEY (product_id)
    REFERENCES products (product_id)
    ON DELETE CASCADE
);
```

```sql
ALTER TABLE child_table
ADD CONSTRAINT fk_name
    FOREIGN KEY (child_col1, child_col2, ... child_col_n)
    REFERENCES parent_table (parent_col1, parent_col2, ... parent_col_n)
    ON DELETE CASCADE;
```



Antra

# Index

- Indexes are used to improve **read** performance. It will increase write time by a little, best for read-heavy tables.
- **Clustered** Indexes are physical indexes to sort and store the data rows. Only 1 clustered index per table. Created automatically for PK
- **Nonclustered** indexes have a key value pairs structure which contains a pointer to the clustered index address or hard drive address. Created automatically for UNIQUE
- Can both be unique or non-unique
- Index can be created on combinations of columns.



Antra

# Index

- There are a total of 12 types of indexes in SQL Server
- For additional information, check https://docs.microsoft.com/en-us/sql/relational-databases/indexes/indexes?view=sql-server-ver15

# Finding Missing Index

- Locate slow but frequently used queries using Extended Events or Logs, then use EXECUTION PLAN to view any potential index placement.
- Use TUNING ADVISOR to automatically locate missing indexes.



Antra

# QUERY HINTS

- Query Hints are optional commands to tell SQL Server how to execute a particular query.
- When Selecting, we can use table hints to apply certain types of locks to a table, including NOLOCK, TABLOCK, ROWLOCK, SNAPSHOT, etc
- Hash | Order GROUP BY
- Merge | Hash | Concat UNION
- Loop | Merge | Hash JOIN

**Caution**

Because the SQL Server Query Optimizer typically selects the best execution plan for a query, we recommend only using hints as a last resort for experienced developers and database administrators.



Antra

# CURSOR

- Kind of a for-each loop for each row of a result set
- HUGE performance hit, but could be useful in certain scenarios.
- ODBC/JDBC might force you to use cursor

```sql
DECLARE @LastName VARCHAR(50), @FirstName VARCHAR(50);

DECLARE contact_cursor CURSOR FOR
SELECT LastName, FirstName FROM Person.Person
WHERE LastName LIKE 'B%'
ORDER BY LastName, FirstName;

OPEN contact_cursor;

-- Perform the first fetch and store the values in variables.
-- Note: The variables are in the same order as the columns
-- in the SELECT statement.

FETCH NEXT FROM contact_cursor
INTO @LastName, @FirstName;

-- Check @@FETCH_STATUS to see if there are any more rows to fetch.
WHILE @@FETCH_STATUS = 0
BEGIN

   -- Concatenate and display the current values in the variables.
   PRINT 'Contact Name: ' + @FirstName + ' ' +  @LastName

   -- This is executed as long as the previous fetch succeeds.
   FETCH NEXT FROM contact_cursor
   INTO @LastName, @FirstName;
END

CLOSE contact_cursor;
DEALLOCATE contact_cursor;
```



Antra

# Schema

- Collection of database objects
- predefined schemas
  - dbo, guest(guest is rare to see, even the official doc doesn't talk about it)
  - sys, and INFORMATION_SCHEMA(no modifications allowed)
- Useful for grouping tables, views by business logic.
- Useful for permission assignment

```
CREATE SCHEMA schema_name
    [AUTHORIZATION owner_name]
```

```
CREATE TABLE customer_services.jobs(
    job_id INT PRIMARY KEY IDENTITY,
    customer_id INT NOT NULL,
    description VARCHAR(200),
    created_at DATETIME2 NOT NULL
);
```



Antra

# SQL Server Agent

- Microsoft Windows service that executes scheduled administrative tasks, which are called jobs in SQL Server
- Schedule future jobs, execute maintenance plans
- https://docs.microsoft.com/en-us/sql/ssms/agent/sql-server-agent?view=sql-server-ver15