

# Module 6:

## Statements for Modifying Data

### Contents

<b>Module Overview .....</b>	<b>1</b>
<b>Lesson 1: Inserting Data .....</b>	<b>2</b>
INSERT Fundamentals .....	3
INSERT Statement Definitions .....	4
INSERT Statement Examples .....	5
Inserting Values into Identity Columns .....	8
Practice: Working with Inserting Data .....	9
<b>Lesson 2: Deleting Data .....</b>	<b>11</b>
DELETE Fundamentals .....	12
DELETE Statement Definitions .....	13
Using the TRUNCATE Statement .....	14
TRUNCATE versus DELETE .....	15
Practice: Working with Deleting Data .....	16
<b>Lesson 3: Updating Data .....</b>	<b>18</b>
UPDATE Fundamentals .....	19
UPDATE Statement Definitions .....	20
Updating Using another Table .....	21
Practice: Working with Updating Data .....	24
<b>Lesson 4: Fundamentals About Transactions .....</b>	<b>26</b>
Transaction Fundamentals .....	27
Transactions and the Database Engine .....	29
Basic Transaction Statement Definitions .....	31
<b>Summary .....</b>	<b>33</b>



# Module Overview

- 
- Inserting Data
  - Deleting Data
  - Updating Data
  - Fundamentals About Transactions

In past five module of this course, you have learned how to select data from user tables in database. SELECT is powerful clause in TSQL language and it is almost synonymous for SQL language. But this command is only one from the DML group - Data Manipulation Language statements which is contains of four members. Other three are: INSERT, UPDATE and DELETE.

In this module you will learn when and how to use those statements. Also you will understand concept of transaction, very important thing when you are dealing with data modifications.

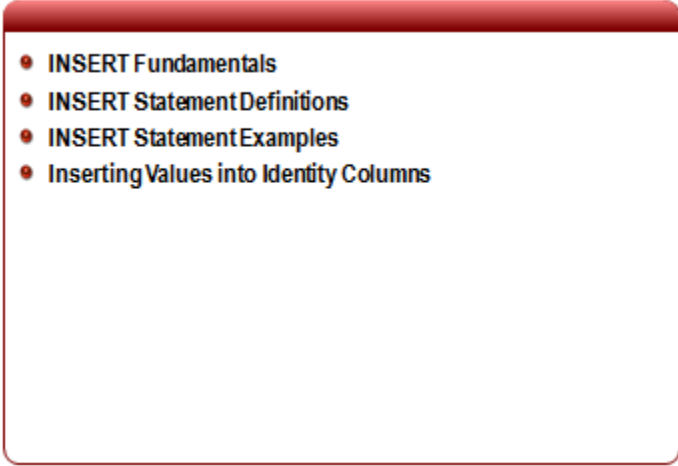
## Objectives

After completing this module, you will be able to:

- Insert data into users tables
- Delete data from users tables
- Update data in users tables
- Understand fundamentals about transactions

*In process of inserting, modifying and deleting data, we will use AdventureWorks2012 sample database. If you want to keep database in original state for latter user, create a full database backup and restore it after completing this module.*

# Lesson 1: Inserting Data

- 
- **INSERT Fundamentals**
  - **INSERT Statement Definitions**
  - **INSERT Statement Examples**
  - **Inserting Values into Identity Columns**

There is no other way for populating database with data besides using INSERT statements. Implementation scenarios can be different:

- User applications
- Import/Export process
- Database replications
- etc.

In any of those, INSERT is in background. Common scenario is implementing data access layer in form of stored procedures (CRUD) – Create, Read, Update and Delete. Those four procedures are used from the application layer for passing user input into database tables.

## Objectives

After completing this lesson, you will be able to:

- Understand INSERT Fundamentals
- Use INSERT Statement Definitions
- Implement INSERT Statement Examples
- Insert Values into Identity Columns

## INSERT Fundamentals

- Primary function of INSERT statement is to insert values of data in to tables

```
INSERT [INTO] table_or view  
[(column_list)]  
data_values
```

- INSERT - is DML statement for initiating operation of adding an data rows
- INTO - is optional keyword that can be used to specified target
- table\_or\_view - is the name of the table or view that is target
- column\_list - defined list of table attributes where we wants to insert data.
- data\_values - contains a list of values that needs to be inserted.

Primary function of INSERT statement is to insert values of data in form of table row-s.  
Destination can be table or view.

Basic syntax is:

```
INSERT [INTO] table_or view  
[(column_list)]  
data_values
```

- INSERT - is DML statement for initiating operation of adding an data rows
- INTO - is optional keyword that can be used to specified target (table or view)
- table\_or\_view - is the name of the table or view that is target for data receiving
- column\_list - defined list of table attributes where we wants to insert data. List needs to be enclosed in parentheses and delimited by commas.
- data\_values - contains a list of values that needs to be inserted. The value list must be enclosed in parentheses

If the column satisfied on of the following definitions, database engine will automatically provide a value for the column without need to put it in data\_values list:

- Has an IDENTITY property. The next incremental identity value is used.
- Has a default. The default value for the column is used.
- Has a timestamp data type. The current timestamp value is used.
- Is nullable. A null value is used.
- Is a computed column. The calculated value is used.

Example of one INSERT statement

```
INSERT INTO Person.CountryRegion  
VALUES ('BA', 'Bosnia and Herzegovina', GETDATE())
```

## INSERT Statement Definitions

- **INSERT statement can be placed on many different places**
- **You can choose one of the following:**
  - Using simple INSERT statement (list of values)
  - Inserting multiple rows of data within an single INSERT statement
  - Adding rows using SELECT statement (subquery)
  - INSERT using EXECUTE
  - INSERT using TOP
  - Adding data with SELECT INTO

INSERT statement can be placed on different places. All of them on the end achieving same task of append new rows to a table.

Which you will use depends on specific task you want to solve and layer of application from where you want to add data. You can choose one of the following:

- Using simple INSERT statement (list of values)
- Inserting multiple rows of data within an single INSERT statement
- Adding rows using SELECT statement (subquery)
- INSERT using EXECUTE
- INSERT using TOP
- Adding data with SELECT INTO

*All this examples will be practically demonstrated in the next topic.*

## INSERT Statement Examples

- **INSERT using TOP**
  - You can define how much exactly data you want to insert
  - Use ORDER BY so define what a criterion for TOP values is

```
INSERT TOP (10) INTO AddExample
SELECT FirstName, LastName
FROM Person.Person
ORDER BY LastName DESC
```
- **Adding data with SELECT INTO**

```
SELECT Title + ', ' + FirstName + ' ' +
+LastName AS Customer
INTO #tempPersons
FROM Person.Person
WHERE Title IS NOT NULL
```

### Adding data using simple INSERT statement:

This is common way to add data in to tables. It uses list of columns and corresponding list of values. It's important that list of values follow list of columns (number, data types and defaults).

Let's analyze following example:

```
INSERT INTO Production.ProductReview
(ProductID, ReviewerName, ReviewDate, EmailAddress,
Rating, Comments, ModifiedDate)
VALUES (937, 'Jasmin', GETDATE(), 'jasmin@mvp-press.com', 4,
'This is some cool product', GETDATE())
```

This statement should add one record (comment) about actual product in Production.ProductReview table with following table and column definition.

Production.ProductReview
Columns
ProductReviewID (PK, int, not null)
ProductID (FK, int, not null)
ReviewerName (Name(nvarchar(50)), not null)
ReviewDate (datetime, not null)
EmailAddress (nvarchar(50), not null)
Rating (int, not null)
Comments (nvarchar(3850), null)
ModifiedDate (datetime, not null)

In this case table has eight columns and if you notice, column list in INSERT statement only seven. Column ProductReviewID is PK and has Is Identity property, so database engine takes care about adding this data. Also we use twice function GETDATE() to add a current date.

## Inserting multiple rows of data

You can also put more than one record at the time. It is good feature when you need to insert more records within a single statement.

Let's analyze following example:

```
INSERT INTO Production.ProductReview
    (ProductID,ReviewerName,ReviewDate,
     EmailAddress,Rating,Comments,
     ModifiedDate)
VALUES (937,'Mattias',GETDATE(), 'mattias@mvp-press.com',4,
       'Yeah it is cool',GETDATE()),
       (937,'Denis',GETDATE(), 'denis@mvp-press.com',4,
       'It is best!',GETDATE())
```

You need to follow same rules like in single row insert version of statement. Limitation is that within single statement you can add 1000 rows. If you need more than that you can create multiple INSERT statements.

## Adding rows using SELECT statement

You can easily use SELECT statement collecting specific rows from database tables, do any operation with data on the fly then INSERT it in another table. In a nutshell this is subquery.

Let's analyze following example:

First we will create simple table with three columns

```
CREATE TABLE AddExample (
    AddExampleID int IDENTITY (1,1) Primary key,
    FirstName nvarchar (50),
    LastName nvarchar (50)
)
```

Then we will SELECT all last and first names from table Person.Person and populate new table with that data.

Let's analyze following example:

```
INSERT INTO AddExample
    SELECT FirstName, LastName
    FROM Person.Person
```



## INSERT using EXECUTE

You can put INSERT statement in any valid TSQL form and save it like a stored procedure. After that step use EXECUTE to run and pass input parameters.

Let's analyze following example:

```
CREATE PROC ProductReviewInsert
    @ProductID int,
    @ReviewerName Name,
    @ReviewDate datetime,
    @EmailAddress nvarchar(50),
    @Rating int,
    @Comments nvarchar(3850),
    @ModifiedDate datetime
AS
INSERT INTO ProductReview, ProductID, ReviewerName,
    ReviewDate, EmailAddress, Rating,
    Comments, ModifiedDate
SELECT @ProductID, @ReviewerName, @ReviewDate, @EmailAddress,
    @Rating, @Comments, @ModifiedDate

--Execution of stored procedure
EXECUTE Production.ProductReviewInsert
```

## INSERT using TOP

You can define how much exactly data you want to insert. Modified example of INSERT with subquery illustrates that situation. Notice that when you want that TOP makes sense, you need to use ORDER BY so define what a criterion for TOP values is.

```
INSERT TOP (10) INTO AddExample
SELECT FirstName, LastName
FROM Person.Person
ORDER BY LastName DESC
```

## Adding data with SELECT INTO

This way of adding data is subquery of SELECT, but in nested part you can find INSERT statement. Following example get a data with SELECT part then insert it into temp table.

Let's analyze following example:

```
SELECT Title + ', ' + FirstName + ' ' + LastName AS Customer
INTO #tempPersons
FROM Person.Person
WHERE Title IS NOT NULL
```

## Inserting Values into Identity Columns

- In some situation you have need to insert specific value into IDENTITY column
- Use SET IDENTITY\_INSERT ON

```
SET IDENTITY_INSERT T1 ON;
GO
--Add one more row
INSERT INTO T1 (col1,col2)
VALUES (99, 'Overrirde identity value')
```

In some situation you have need to insert specific value into IDENTITY column. If you want to do that you need to override automatic insert with SET IDENTITY\_INSERT ON.

Let's analyze following example

```
--Create table
CREATE TABLE dbo.T1(
    col1 int IDENTITY,
    col2 nvarchar(50));

GO

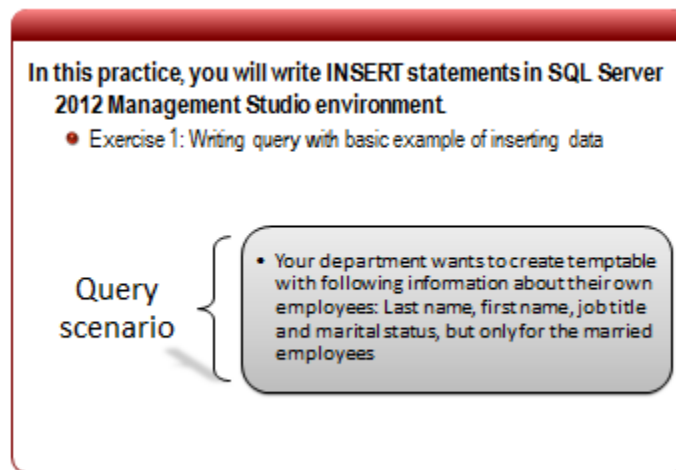
--Put two rows with regular IDENTITY insert
INSERT INTO T1 VALUES ('Row No.1');
INSERT INTO T1 (col2) VALUES ('Row No.2');
GO

--Set option for IDENTITY override
SET IDENTITY_INSERT T1 ON;
GO
--Add one more row
INSERT INTO T1 (col1,col2)
VALUES (99, 'Overrirde identity value');
GO

SELECT * FROM T1

--Delete table
DROP Table T1
```

## Practice: Working with Inserting Data



In this practice you will write queries in SQL Server 2012 Management Studio environment. Practice is based on Lesson 1.

To successfully complete exercise you need following resources:

- SQL Server Management Studio 2012
- AdventureWorks2012 sample database
  - <http://msftdbprodsamples.codeplex.com/>

### Exercise: Writing query with basic examples of inserting data

1. Click Start→Microsoft SQL Server 2012→SQL Server Management Studio
2. On the Connect to Server dialog windows, under Server name type the name of your local instance.
  - a. If it is default then just type (local)
3. Use Windows Authentication
4. Open File menu →New →Database Engine Query

#### *Query Scenario*

Your department wants to create temp table with following information about their own employees: Last name, first name, job title and marital status, but only for the married employees.

5. Write the following TSQL query and run it:

```
SELECT P.FirstName, P.LastName, E.JobTitle, E.MaritalStatus
INTO #tempEmployee
FROM Person.Person AS P
INNER JOIN HumanResources.Employee AS E
ON P.BusinessEntityID = E.BusinessEntityID
WHERE E.MaritalStatus = 'M'
```

6. Check is there temple table with data:

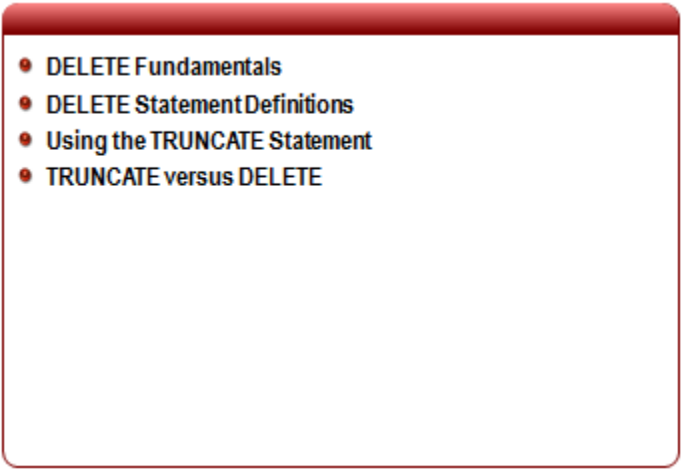
```
SELECT * FROM #tempEmployee
```

Result should be inserted 146 rows.

7. Now add yourself into temp table (write the following TSQL query and replace with your data)

```
INSERT INTO #tempEmployee
(FirstName, LastName, JobTitle, MaritalStatus)
VALUES ('Your First Name', 'Your Last Name',
'Your Job Title', 'Your Marital Status')
```

## Lesson 2: Deleting Data

- 
- DELETE Fundamentals
  - DELETE Statement Definitions
  - Using the TRUNCATE Statement
  - TRUNCATE versus DELETE

So far we know how to read and add new data into database tables. Now we will learn how to delete one or more rows in table or view. Statement that we use in this process is DELETE and belongs to the DML group of statements.

Also you will learn what is truncating table and difference between delete operations.

### Objectives

After completing this lesson, you will be able to:

- Understand DELETE fundamentals
- Use DELETE statement
- Use TRUNCATE statement
- Understand difference between TRUNCATE and DELETE statements

## DELETE Fundamentals

- TSQL statement, DELETE, removes one or more rows;
- When you working with DELETE, you should be aware following facts
  - You can delete one or more rows at the time
  - DELETE without WHERE clause, all data from table will be deleted
  - This rule will not apply if table have FK constraint,
  - but if cascade delete option is ON then you can have delete chain reaction
  - Only WHERE clause is guarantee that specific row(s) will be deleted.
  - When deleting large data sets, delete can create locks

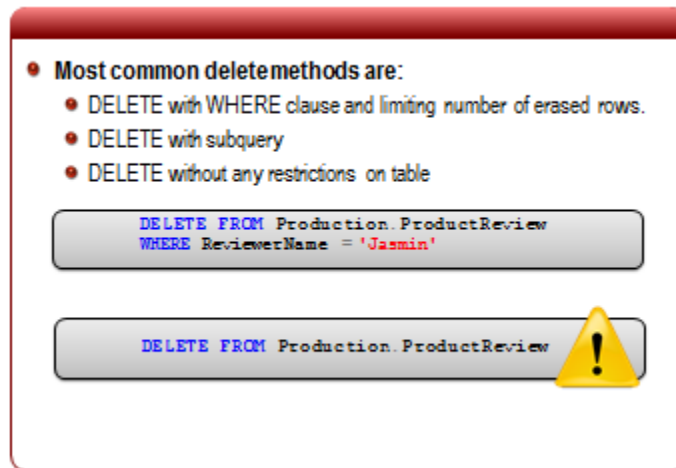
TSQL statement, DELETE, removes one or more rows from a table or view in SQL Server 2012 (or any other version of SQL Server). It is standard SQL command for deleting data. When you working with DELETE, you should be aware following facts and satisfied certain rules.

- You can delete one or more rows at the time from table or view
- If you use DELETE without WHERE clause, all data from table will be deleted
- This rule will not apply if table have FK constraint, but if cascade delete option is ON then you can have delete chain reaction
- Only WHERE clause is guarantee that specific row(s) will be deleted. Rows that satisfies WHERE condition
- When deleting large data sets, delete can create locks and affect overall system performance

--Basic DELETE syntax

```
DELETE Table_Or_View
WHERE Delete_Condition
```

## DELETE Statement Definitions



You can delete data on many ways, depends on your need and business process requirements. Most common methods are:

- DELETE with WHERE clause and limiting number of erased rows.
- DELETE with subquery
- DELETE without any restrictions on table

*Warning: Following examples will delete actual data from Production.ProductReview table. You should create backup to keep data (if you did not do it already)*

### DELETE with WHERE clause

Erase record where reviewer name is Jasmin.

```
DELETE FROM Production.ProductReview
WHERE ReviewerName = 'Jasmin'
```

### DELETE with subquery

Erase records where reviewers name are Jasmin and Denis, using subquery

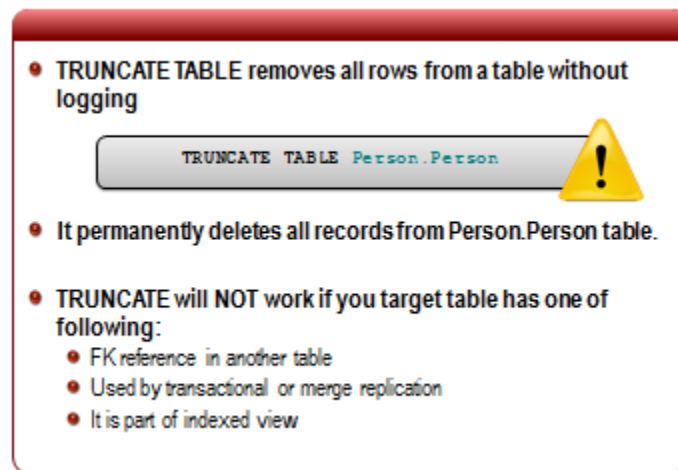
```
DELETE FROM Production.ProductReview
WHERE ReviewerName IN (SELECT ReviewerName
                        FROM Production.ProductReview
                        WHERE ReviewerName = 'Jasmin'
                        OR ReviewerName = 'Denis')
```

### DELETE without any restrictions

Delete all record from table.

```
DELETE FROM Production.ProductReview
```

## Using the TRUNCATE Statement



TRUNCATE TABLE removes all rows from a table without logging, for every single row erase, operation and it is very fast.

Let's analyze following example:

```
TRUNCATE TABLE Person.Person
```

It permanently deletes all records from Person.Person table.

**You should pay attention when execute this command.** If you don't have backups, data from target table is permanently lost because there are no records about truncate process in transaction log.

TRUNCATE will NOT work if you target table has one of following:

- FK reference in another table
- Used by transactional or merge replication
- It is part of indexed view



## TRUNCATE versus DELETE

- DELETE has performance issue with erasing large data sets.
- TRUNCATE is faster operation and consume less system and transaction log resources.

Feature	DELETE	TRUNCATE
Log file	Removes rows one at the time and records operation in log file for each row.	Removes data by deallocating data pages, and only record page deallocation in log file
Locks	Each row in table is locked for deleting operation.	Each page is locked for deleting operation
Speed	Slower	Faster

DELETE statement sometimes is just not enough to complete the task. Main reason is performance issue with erasing large data sets.

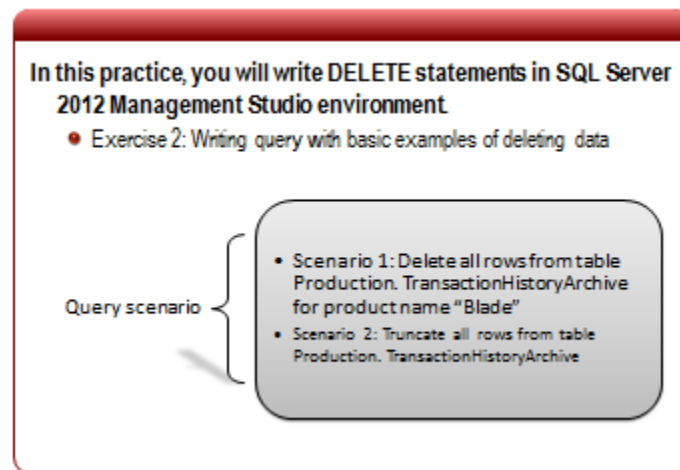
Problem with DELETE is that, when row is erased from table operation is logged in transaction log file. Process is repeated for every row in the table.

On the contrary TRUNCATE is faster operation and consume less system and transaction log resources. After operation is completed, zero pages are left in the table, while after DELETE statement table can contains empty pages

Feature	DELETE	TRUNCATE
<b>Log file</b>	Removes rows one at the time and records operation in log file for each row.	Removes data by deallocating data pages, and only record page deallocation in log file
<b>Locks</b>	Each row in table is locked for deleting operation.	Each page is locked for deleting operation
<b>Speed</b>	Slower	Faster

Depends on your needs you should be very careful when using DELETE or TRUNCATE, but TRUNCATE requires extra attention because recovery operation is very hard (if you don't have backup it's impossible)

## Practice: Working with Deleting Data



In this practice you will write queries in SQL Server 2012 Management Studio environment. Practice is based on Lesson 2.

To successfully complete exercise you need following resources:

- SQL Server Management Studio 2012
- AdventureWorks2012 sample database
  - <http://msftdbprodsamples.codeplex.com/>

### Exercise: Writing query with basic examples of deleting data

1. Click Start → Microsoft SQL Server 2012 → SQL Server Management Studio
2. On the Connect to Server dialog windows, under Server name type the name of your local instance.
  - If it is default then just type (local)
3. Use Windows Authentication
4. Open File menu → New → Database Engine Query

#### *Query Scenario*

Scenario 1: Delete all rows from table Production.TransactionHistoryArchive for product name "Blade"

Scenario 2: Truncate all rows from table Production.TransactionHistoryArchive

**Warning:** Following examples will delete actual data from Production.TransactionHistoryArchive table. You should create backup to keep data (if you did not do it already)

Scenario 1:

5. Write the following TSQL query and run it:

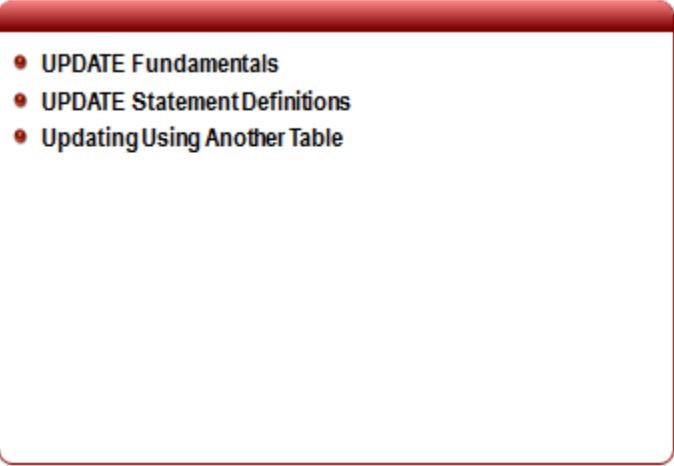
```
DELETE FROM Production.TransactionHistoryArchive
WHERE ProductID IN (
    SELECT THS.ProductID
    FROM Production.TransactionHistoryArchive AS THS
    INNER JOIN Production.Product AS P
    ON THS.ProductID = P.ProductID
    WHERE P.Name = 'Blade')
```

Scenario 2:

6. Write the following TSQL query and run it:

```
TRUNCATE TABLE Production.TransactionHistoryArchive
```

## Lesson 3: Updating Data

- 
- UPDATE Fundamentals
  - UPDATE Statement Definitions
  - Updating Using Another Table

Last of four DML statement (SELECT, INSERT, DELETE) is UPDATE. This command is used for change in existing data in the tables. In this lesson you will learn how to update row in the table and how to update rows based on another tables (subquery).

### Objectives

After completing this lesson, you will be able to:

- Understand UPDATE fundamentals
- Use UPDATE statement
- Update using another table

## UPDATE Fundamentals

- **UPDATE** statement can change record data values in one or many rows in the table or view

```
UPDATE Table_Or_View
SET column(s) = newValue(s)
FROM External_Table_Source
WHERE Update_Condition
```

- **SET** – this clause contains list of columns separated with commas that is going to be updated.
- **FROM** – you can use external source of data (another table or view)
  - this is subquery in UPDATE statement.
- **WHERE** – place to define what you want to update

With UPDATE statement you can change record data values in one or many rows in the table or view. How will UPDATE work with your records, depends on the usage of statement clauses.

```
UPDATE Table_Or_View
SET column(s) = newValue(s)
FROM External_Table_Source
WHERE Update_Condition
```

**SET** – this clause contains single or list of columns separated with commas that is going to be updated. Each column contains a new data value in form of expression. Rules for expression are like any other TSQL expression that you had learned so far.

**FROM** – you use external source of data (another table or view) fill in values expression in SET clause. Basically, this is subquery in UPDATE statement.

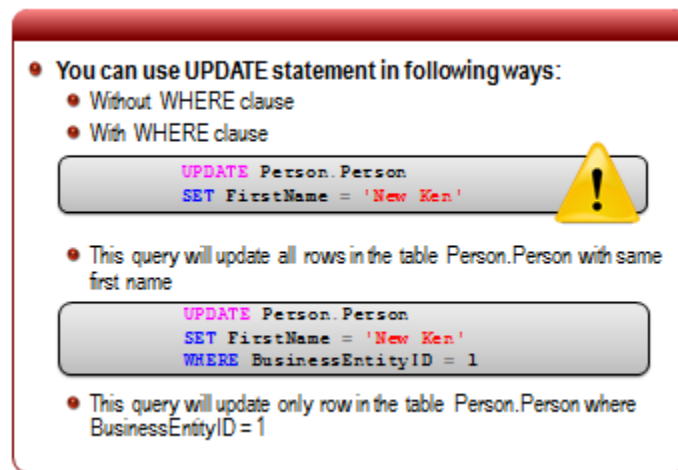
**WHERE** – places to define update (search) condition that defines row from the source table where update operation will occur. Without WHERE part update will occur on all table data (columns in SET part).

Let's analyze following example:

```
UPDATE Person.Person
SET FirstName = 'New Ken', LastName = 'New Sánchez'
WHERE BusinessEntityID = 1
```

This query updates first name and last name of person where BusinessEntityID is equal to 1.

## UPDATE Statement Definitions



- You can use UPDATE statement in following ways:
  - Without WHERE clause
  - With WHERE clause

```
UPDATE Person.Person
SET FirstName = 'New Ken'
```

This query will update all rows in the table Person.Person with same first name

```
UPDATE Person.Person
SET FirstName = 'New Ken'
WHERE BusinessEntityID = 1
```

This query will update only row in the table Person.Person where BusinessEntityID = 1

You can use UPDATE statement in following ways:

- Without WHERE clause
- With WHERE clause

**Warning:** Following examples will update actual data from Person.Person table. You should create backup to keep data (if you did not do it already)

### Without WHERE clause

Let's analyze following example:

```
UPDATE Person.Person
SET FirstName = 'New Ken'
```

This query will update all rows in the table Person.Person with same first name.

### With WHERE clause

Let's analyze following example:

```
UPDATE Person.Person
SET FirstName = 'New Ken'
WHERE BusinessEntityID = 1
```

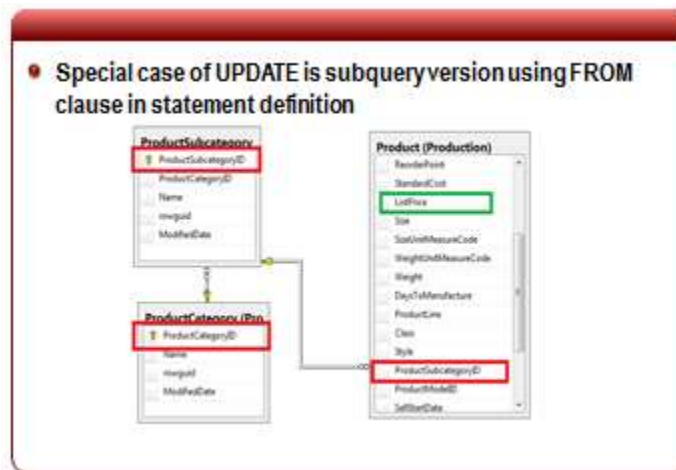
This query will update only row in the table Person.Person where BusinessEntityID = 1

Let's analyze following example:

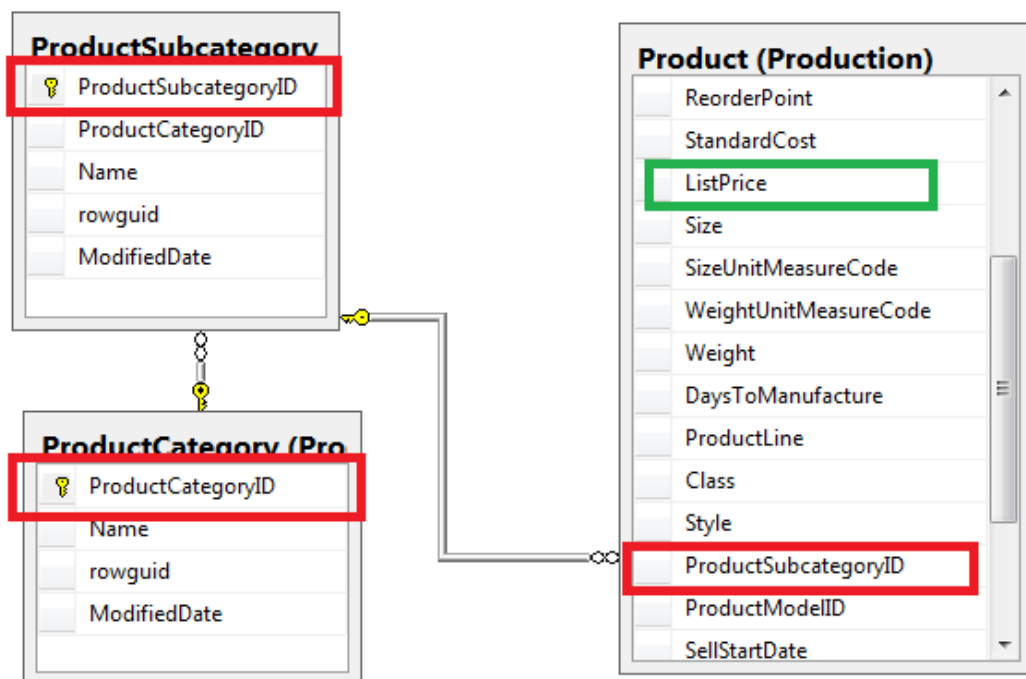
```
UPDATE Production.Product
SET ListPrice = ListPrice * 1.1
WHERE ProductID = 2
```

This query will increase list price for 10%, for ProductID 1

## Updating Using another Table



Special case of UPDATE is subquery version using FROM clause in statement definition. It updates rows in one table with data from another table. This is very handful in situation when update process depends on data from highly relational normalized database tables. Next examples illustrate this situation.



We want to increase ListPrice for 10 % for all products in 'Bikes' category. Problem is that Name of category is two tables away from Products. So we need to use classic PK-FK join operation.



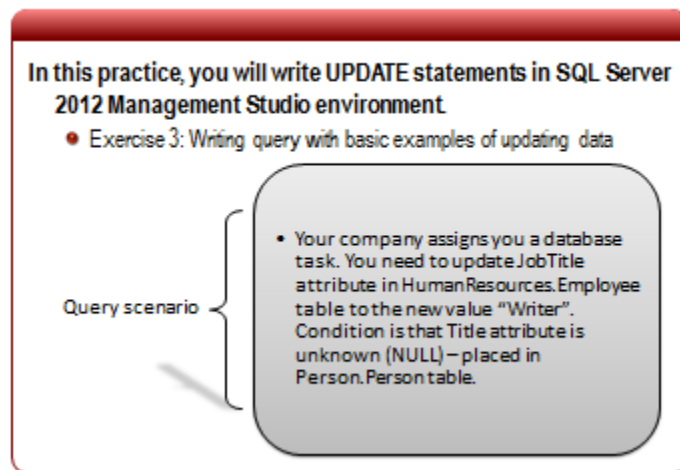
***Warning: Following example will update actual data from Production.Product table. You should create backup to keep data (if you did not do it already)***

Let's analyze following example:

```
UPDATE Production.Product
SET ListPrice = ListPrice * 1.1
FROM Production.ProductCategory AS PC
INNER JOIN Production.ProductSubcategory AS PS
ON PC.ProductCategoryID = PS.ProductCategoryID
INNER JOIN Production.Product AS P
ON PS.ProductSubcategoryID = P.ProductSubcategoryID
WHERE PC.Name = 'Bikes'
```

This query will increase list price for 10% for every product from Bikes category.

## Practice: Working with Updating Data



In this practice you will write queries in SQL Server 2012 Management Studio environment. Practice is based on Lesson 3.

To successfully complete exercise you need following resources:

- SQL Server Management Studio 2012
- AdventureWorks2012 sample database
  - <http://msftdbprodsamples.codeplex.com/>

### Exercise: Writing query with basic examples of updating data

1. Click Start→Microsoft SQL Server 2012→SQL Server Management Studio
2. On the Connect to Server dialog windows, under Server name type the name of your local instance.
  - If it is default then just type (local)
3. Use Windows Authentication
4. Open File menu →New →Database Engine Query

#### *Query Scenario*

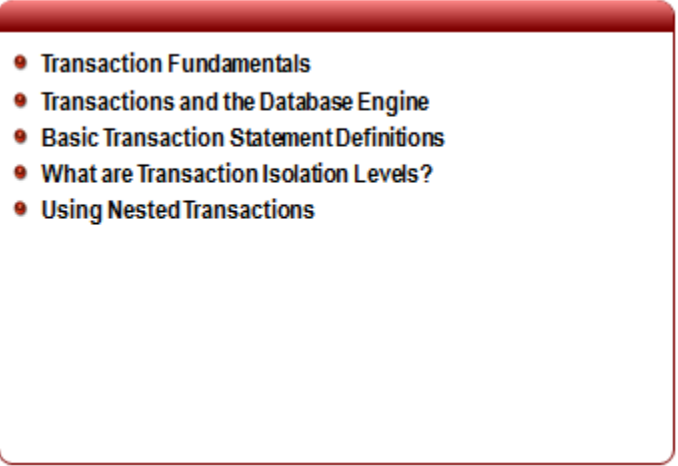
Your company assigns you a database task. You need to update JobTitle attribute in HumanResources.Employee table to the new value “Writer”. Condition is that Title attribute is unknown (NULL) – placed in Person.Person table.

**Warning:** Following example will update actual data from HumanResources.Employee table. You should create backup to keep data (if you did not do it already)

5. Write the following TSQL query and run it:

```
UPDATE HumanResources.Employee
SET JobTitle = 'Writer'
FROM Person.Person AS P
INNER JOIN HumanResources.Employee AS E
ON P.BusinessEntityID = E.BusinessEntityID
WHERE P.Title IS NULL
```

## Lesson 4: Fundamentals About Transactions

- 
- Transaction Fundamentals
  - Transactions and the Database Engine
  - Basic Transaction Statement Definitions
  - What are Transaction Isolation Levels?
  - Using Nested Transactions

Database is built on concept of transactions and it very important in many aspects of developments, especially when you create data access layer in TSQL code. In a nutshell, transaction is set of operation (TSQL statements) performed as a single unit. This means that all of them need to be executed or none. We can say all or nothing.

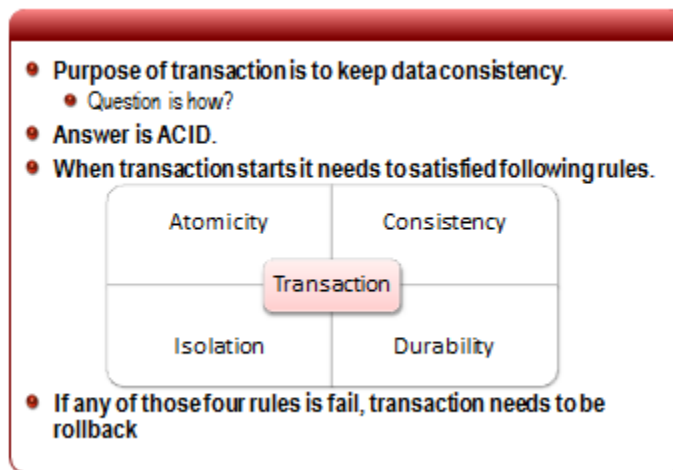
Purpose of transaction is to keep data consistency during executing TSQL statements. Why? Because during transaction something can goes wrong: hardware or software failure. Imagine that you have partial updated data in to the system. In this lesson you will learn what the transactions are and learn ACID concepts also how to implement it in your TSQL code.

### Objectives

After completing this lesson, you will be able to:

- Understand transaction fundamentals
- Understand transactions and the database engine
- Recognize different types of transaction
- Use transaction isolation levels
- Use nested transactions

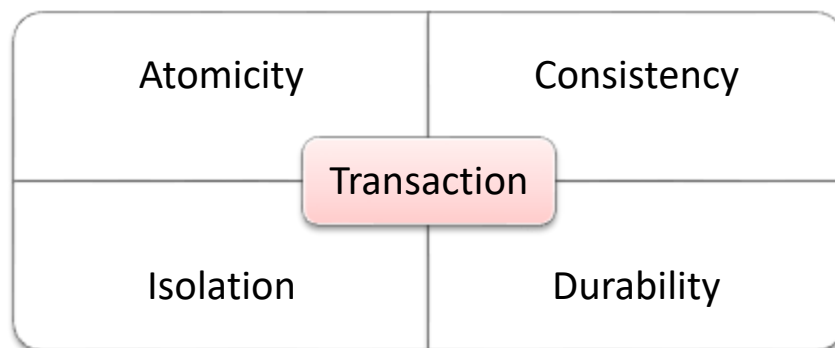
## Transaction Fundamentals



So, transaction is set (sequence) of data operations (Read, Write, Update and Delete) executed as a single unit. Unit can be implementation of some business process. For an example: online order, operation on ATM machine, online testing etc. Common for all of this that those operations are not juts single TSQL statement it's a sequence of theme executed in some logical order.

Purpose of transaction is to keep data consistency. Question is how? Answer is ACID. When transaction starts it needs to satisfied following rules.

- Atomicity
- Consistency
- Isolation
- Durability



If any of those four rules is fail, transaction does not make sense and it need to be rollback,

Atomicity – all operation from transaction needs to reflect into database

**Consistency** – Data integrity needs to be kept during transaction and leave all data in a consistent state.

**Isolation** - Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.

**Durability** – All modifications need to be permanent and persistent even in a case of hardware and system errors.

**Example:** Transaction to transfer \$100 from account A to account B:

```
1. read(A)
2. A := A - 100
3. write(A)
4. read(B)
5. B := B + 100
6. write(B)
7.
```

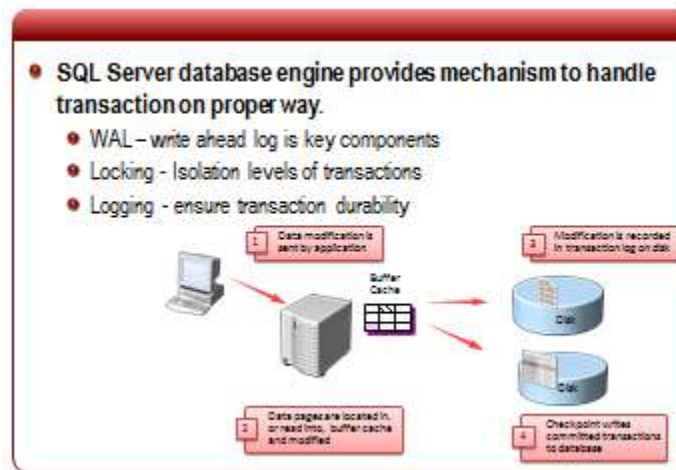
**Atomicity requirement** — if the transaction fails after step 3 and before step 6, the system should ensure that its updates are not reflected in the database, else an inconsistency will result.

**Consistency requirement** – the sum of A and B is unchanged by the execution of the transaction.

**Isolation requirement** — if between steps 3 and 6, another transaction is allowed to access the partially updated database, it will see an inconsistent database (the sum  $A + B$  will be less than it should be). Can be ensured trivially by running transactions serially, that is one after the other. However, executing multiple transactions concurrently has significant benefits, as we will see latter.

**Durability requirement** — once the user has been notified that the transaction has completed (i.e., the transfer of the \$100 has taken place), the updates to the database by the transaction must persist despite failures.

## Transactions and the Database Engine

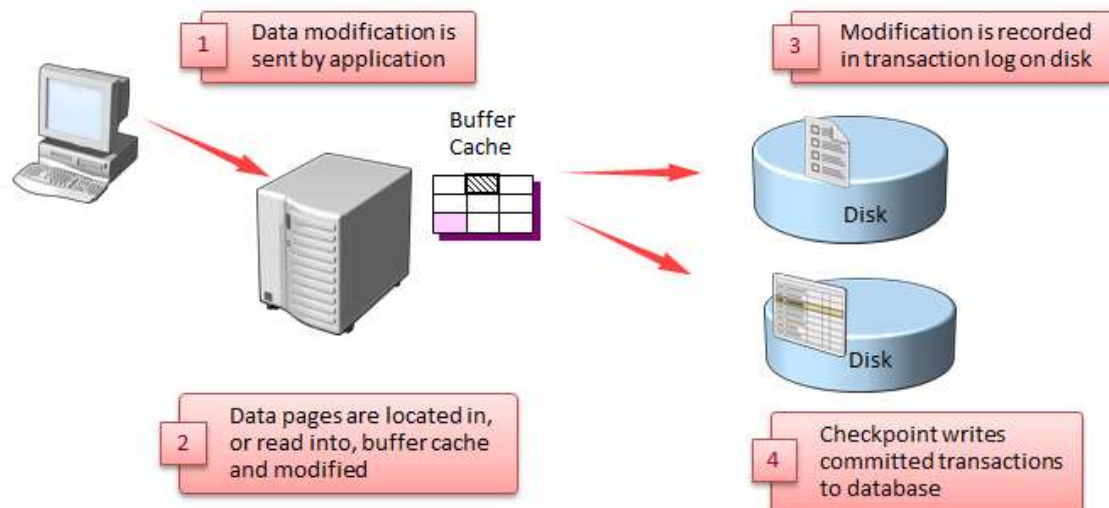


SQL Server database engine provides mechanism to handle transaction on proper way. In that process very important role plays transaction log file. Engine has three major components to ensure ACID in transactions.

**WAL** – write ahead log is key components. Like name sad, any data change operation is first recorded in transaction log then then written to the disk (actual data). This maintains data consistency.

**Locking** - Isolation levels of transactions are there to provide good enough isolation of each transaction based on business process requirements. Isolation means that other transaction can or cannot see partial modified data.

**Logging** - ensure transaction durability. Even if the server hardware, operating system, or the instance of the database engine itself fails, the instance uses the transaction logs upon restart to automatically roll back any uncompleted transactions to the point of the system failure.







## Basic Transaction Statement Definitions

- **BEGIN TRANSACTION** – is a point in TSQL code from where all parts are treated as one for keeping data consistent
- **COMMIT TRANSACTION** - all data modifications performed since the start of the transaction are permanent
- **ROLLBACK TRANSACTION** - Rolls back an explicit or implicit transaction to the beginning of the transaction
- *You cannot rollback transaction after COMMIT*

SQL programmers are responsible for starting and ending transactions that enforce the logical consistency of the data. The programmer must define the sequence of data modifications that leave the data in a consistent state relative to the organization's business rules. The programmer includes these modification statements in a single transaction so that the SQL Server Database Engine can enforce the physical integrity of the transaction. How? SQL Server provides following TSQL statements:

**BEGIN TRANSACTION** – is a point in TSQL code from where all parts are treated as one for keeping data consistent. If error occurs, all data modifications placed in **BEGIN TRANSACTION** can be rollback

**COMMIT TRANSACTION** - all data modifications performed since the start of the transaction a permanent part of the database, frees the resources held by the transaction

**ROLLBACK TRANSACTION** - Rolls back an explicit or implicit transaction to the beginning of the transaction, or to a save point inside the transaction. You can use **ROLLBACK TRANSACTION** to erase all data modifications made from the start of the transaction or to a save point. It also frees resources held by the transaction

*You cannot rollback transaction after COMMIT*

```
BEGIN TRANSACTION;  
GO  
DELETE FROM HumanResources.JobCandidate  
WHERE JobCandidateID = 1;  
GO  
  
BEGIN TRANSACTION;  
GO  
DELETE FROM HumanResources.JobCandidate
```

```
        WHERE JobCandidateID = 1;  
GO  
ROLLBACK TRANSACTION  
  
BEGIN TRANSACTION;  
GO  
DELETE FROM HumanResources.JobCandidate  
        WHERE JobCandidateID = 1;  
GO  
COMMIT TRANSACTION;
```

# Summary

---

Writing statements that actually change data is responsible task for database programmer. In this module you had learned when and how to use INSERT, UPDATE and DELETE statements.

You had learned to combine those statements in to subqueries to achieve more complex task when dealing with data modification.

After completing this module, you learned:

- How to insert data into users tables
- How to delete data from users tables
- How to update data in users tables
- Create transactions in your code