

Module 4:

Querying Multiple Tables

Contents

Module Overview	1
Lesson 1: Querying Multiple Tables Using Joins	2
What are Joins?	3
Types of Joins	4
Querying using INNER JOIN	5
Querying using OUTER JOIN	6
Querying using CROSS JOIN.....	8
Advanced Usage of Joins	9
Practice: Examples of Join Operations	11
Lesson 2: Limiting Result Sets	13
Combining Result Sets by Using the UNION Operator	14
By Using the EXCEPT and INTERSECT Operators	16
By Using the TOP and TABLESAMPLE Operators.....	18
Practice: Limiting Result Sets	20
Summary	22

Module Overview



Databases based on relational model, such as the one created in SQL Server 2012, commonly use more than one table to store data about single entity. For example, data about products are stored in tables: Product, ProductCostHistory, ProductDocument and etc. which are connected with appropriate type of relationships. Working with an entity (for example: customer, product) that has a feature with a number of values, whose history you want to preserve (for example, a frequent change of product price), can cause unnecessary data multiplication. However, by using multiple tables and their relations you are able to significantly reduce data multiplication. Taking into account the fact that the data about the entities are separated in different tables, it is logical to expect that their retrieval will require joining those data into a single result set.

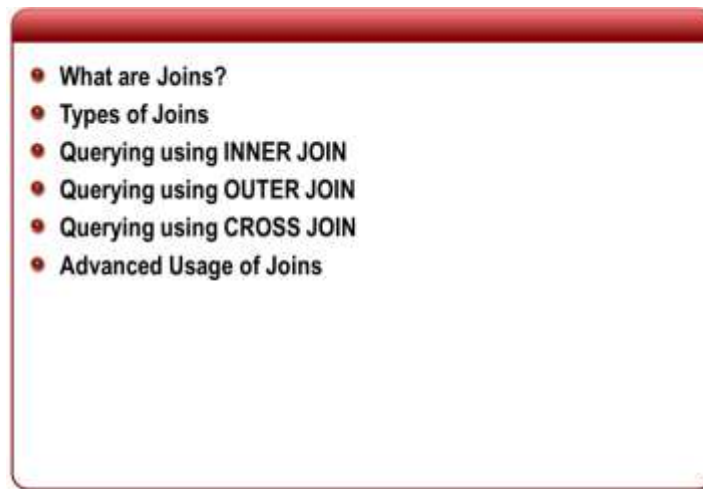
In previous modules we mainly dealt with data stored in one table, while in this module we will talk about how to retrieve data stored in multiple tables.

Objectives

After completing this module, you will be able to:

- Understand different types of joins
- Query multiple tables using appropriate type of join
- Limit queried result set

Lesson 1: Querying Multiple Tables Using Joins



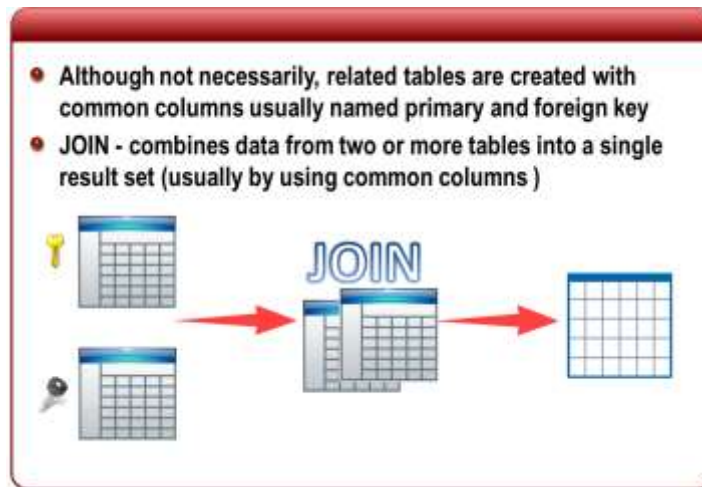
Queries created on a single table can sometimes provide you necessary data, but in practice, most queries are those whose data is acquired from multiple tables. To create a query that will combine data from multiple tables into a single result set requires you to use a powerful relational operator called JOIN. In order to meet the needs of practice, T-SQL provides support to a number of JOIN types such as INNER, OUTER and CROSS discussed in this lesson.

Objectives

After completing this lesson, you will be able to:

- Identify the required JOIN type
- Create query using INNER JOIN
- Create query using OUTER JOIN
- Create query using CROSS JOIN
- Create queries with different types of JOINS

What are Joins?



JOIN operator provides you the functionality to combine data from two or more tables into a single result set. Although not necessarily, related tables are created with common columns usually named primary and foreign key. These keys are used to join related tables to another one and their absence sometimes significantly complicates joining process or leads to creation of result sets that have large amount of unnecessary data.

As you can predict, the column names of joining tables whose data you want to include in the result set are specified in the `SELECT` clause. After that it is recommended that you specify JOIN operator as a part of `FROM` clause, but it can also be specified in the `WHERE` clause. By following this recommendation you will certainly be able to improve query readability and use `WHERE` clause primarily to specify the additional filtering conditions. In addition to JOIN, it is important to mention the keyword `ON` which is used to specify columns for joining tables.

Based on the aforementioned, you certainly realize that the table joining is an operation that affects the performance and therefore you should take into account the number of tables that you specify in the JOIN operator.

Types of Joins



Before we start with describing different types of joins, let us describe basic JOIN syntax.

```
SELECT LeftTable.Column1, RightTable.Column1
FROM LeftTable JoinType RightTable
ON JoinCondition
```

Database engine performs table joins in a way that it takes the record from the LeftTable and usually based on the common field checks for one or more matches in the RightTable. Depending on the type of JOIN and matching results, database engine incorporates appropriate records in the query result set.

There are several types of JOIN operator:

- **INNER JOIN** – returns only matching rows
- **OUTER JOIN** – returns matching the unmatched rows
- **CROSS JOIN** – returns all rows from the LeftTable combined with all rows from the RightTable.

Querying using INNER JOIN

- Cross-section of two sets
- Resulting set will contain only those elements that are common to both tables

```
SELECT P.ProductNumber, R.Comments
FROM Production.Product P
INNER JOIN Production.ProductReview R
ON P.ProductID = R.ProductID
```

ProductNumber	Comments
SO-B909-M	I can't believe I'm singing the....
PD-M562	A little on the heavy side, but overall ...
PD-M562	Maybe it's just because I'm new to
BK-R64Y-40	The Road-550-W from Adventure Works Cycles ...

(4 row(s) affected)

An INNER JOIN can be viewed as a cross-section of two sets. Therefore, when you create an INNER JOIN with two tables, depending on the conditions of connectivity, the resulting set will contain only those elements that are common to both tables. In other words, an INNER JOIN matches rows from two tables based on the common columns values in each table. The following query will create a list of all products for which a review is created. Since the products and reviews data are stored in two separate tables, that is a signal for JOIN operator. As we have emphasized that we want only those products for which the review has been created, or only those data that are common in both tables, this means that you need an INNER JOIN.

```
SELECT P.ProductNumber, R.Comments
FROM Production.Product P
INNER JOIN Production.ProductReview R
ON P.ProductID = R.ProductID
```

Result is:

ProductNumber	Comments
SO-B909-M	I can't believe I'm singing the....
PD-M562	A little on the heavy side, but overall ...
PD-M562	Maybe it's just because I'm new to
BK-R64Y-40	The Road-550-W from Adventure Works Cycles ...

(4 row(s) affected)

As you can check, table Product contains 504 rows, and table ProductReview only 4. However, since we have used INNER JOIN, the result set will be consisted of those rows that match both tables. Table joining is made based on value of the ProductID column which we specified by using ON operator. It is also important to note the way in which we have used aliases, and based on them we have referenced columns in different tables. As an exercise, try to make the previous example without using aliases. Since INNER JOIN is the default type of join, instead you can use keyword JOIN only.

Querying using OUTER JOIN

- Returns even those data that does not have a match in the joining table
- There are three variations of OUTER JOIN operator:
 - **LEFT OUTER JOIN** – returns all the rows from the LeftTable and only the matching rows from the RightTable
 - **RIGHT OUTER JOIN** – returns all the rows from the RightTable and only the matching rows from the LeftTable
 - **FULL OUTER JOIN** – returns all rows from both the LeftTable and the RightTable

An OUTER JOIN will return all rows that exist in LeftTable, even though corresponding rows do not exist in the RightTable. Therefore, OUTER JOIN enables you to create a result set that includes even those data that do not have a match in the joining table, and in the place of a matching value they will be assigned a NULL. There are three variations of OUTER JOIN operator: LEFT, RIGHT and FULL OUTER.

The result set of a **LEFT OUTER JOIN** includes all the rows from the LeftTable and only the matching rows from the RightTable.

```
SELECT P.ProductNumber, R.Comments
FROM Production.Product P
LEFT OUTER JOIN Production.ProductReview R
ON P.ProductID = R.ProductID
```

Result is:

ProductNumber	Comments
AR-5381	NULL
BA-8327	NULL
...	
PD-M340	NULL
PD-M562	A little on the heavy side, ...
PD-M562	Maybe it's just because I'm new to ...
...	
BK-R19B-48	NULL
BK-R19B-52	NULL

(505 row(s) affected)

A **RIGHT OUTER JOIN** is the reverse of a LEFT OUTER JOIN and therefore it includes all the rows from the RightTable and only the matching rows from the LeftTable which is in this case the same as the result of the INNER JOIN.


```

SELECT P.ProductNumber, R.Comments
FROM Production.Product P
RIGHT OUTER JOIN Production.ProductReview R
ON P.ProductID = R.ProductID

```

Result is:

ProductNumber	Comments
SO-B909-M	I can't believe I'm singing the....
PD-M562	A little on the heavy side, but overall ...
PD-M562	Maybe it's just because I'm new to
BK-R64Y-40	The Road-550-W from Adventure Works Cycles ...

(4 row(s) affected)

A **FULL OUTER JOIN** returns all rows from both the LeftTable and the RightTable which is in this case the same as the result of the LEFT INNER JOIN.

```

SELECT P.ProductNumber, R.Comments
FROM Production.Product P
FULL OUTER JOIN Production.ProductReview R
ON P.ProductID = R.ProductID

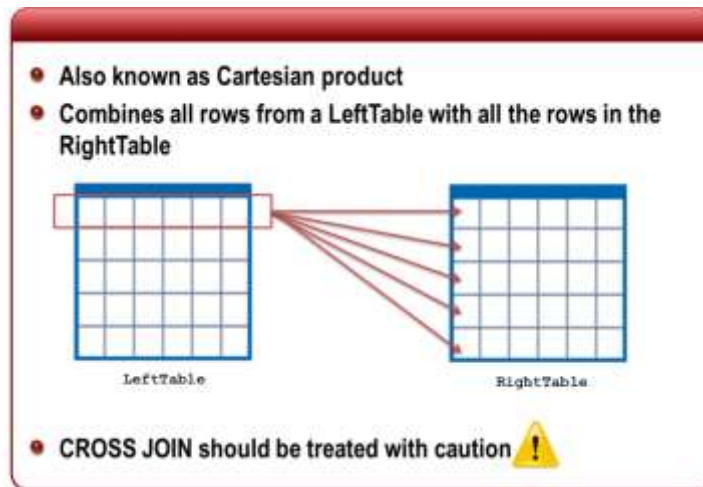
```

Result is:

ProductNumber	Comments
AR-5381	NULL
BA-8327	NULL
...	
PD-M340	NULL
PD-M562	A little on the heavy side, ...
PD-M562	Maybe it's just because I'm new to ...
...	
BK-R19B-48	NULL
BK-R19B-52	NULL

(505 row(s) affected)

Querying using CROSS JOIN



Comparing it with other types of JOIN operators, CROSS JOIN is certainly one that is least used in daily work with databases. Rare usage of this JOIN type is probably caused by its functionality. The CROSS JOIN, also known as Cartesian product, combines all rows from the LeftTable with all rows in the RightTable. In other words, CROSS JOIN takes one row from the LeftTable and combines it with all rows in the RightTable. Expressed numerically, if the LeftTable has 504 rows and the RightRow has 4 rows, then as a result of the CROSS JOIN you can expect 2016 rows. CROSS JOIN functionality eliminates the need for defining the conditions of joining previously done with ON operator.

```
SELECT P.ProductNumber, R.Comments
FROM Production.Product P
CROSS JOIN Production.ProductReview R
```

Result is:

ProductNumber	Comments
AR-5381	I can't believe I'm singing ...
BA-8327	I can't believe I'm singing ...
...	
VE-C304-S	The Road-550-W from Adventure ...
WB-H098	The Road-550-W from Adventure ...

(2016 row(s) affected)

Based on the way it performs joining, you can certainly conclude that the CROSS JOIN should be treated with caution, especially when we have a lot of data in both tables. However, the characteristics of CROSS JOIN in some cases prove to be very useful as is the case with when you just need a larger amount of data to test all or any portion of the system (generation of test data).

Advanced Usage of Joins

- You will often require data from more than two tables
 - Multiple JOINS
- Self-join refers to any kind of join used to join a table to itself

```
SELECT P.Name AS Product, L.Name AS Location, I.Quantity
FROM Production.Product P
      INNER JOIN Production.ProductInventory I
ON P.ProductID = I.ProductID
      INNER JOIN Production.Location L
ON L.LocationID = I.LocationID
```

Product	Location	Quantity
Adjustable Race	Tool Crib	408
Adjustable Race	Miscellaneous Storage	324
....		
Road-750 Black, 52	Final Assembly	116

(1069 row(s) affected)

Generating detailed information about an entity in a single result set will often require data from multiple tables. This means you will have to apply a corresponding type of JOIN on more than two tables. As with previous examples JOIN specification is made in FROM clause. However, since we are combining more than two tables, JOIN conditions must be connected with AND or with OR operator.

The next query will collect information on the quantity of every product on each production location. The required data are located in 3 different tables: Product (contains data about products), ProductInventory (contains data about quantity of every product on each location) and Location (contains data about location of production). It is important to note which common columns or keys are used to join tables.

```
SELECT P.Name AS Product, L.Name AS Location, I.Quantity
FROM Production.Product P
      INNER JOIN Production.ProductInventory I
ON P.ProductID = I.ProductID
      INNER JOIN Production.Location L
ON L.LocationID = I.LocationID
```

Result is:

Product	Location	Quantity
Adjustable Race	Tool Crib	408
Adjustable Race	Miscellaneous Storage	324
....		
Road-750 Black, 52	Final Assembly	116

(1069 row(s) affected)

If you want to narrow the search to a specific location and quantity, it is necessary to define the search conditions in the WHERE clause. For example:

```
WHERE L.Name = 'Tool Crib' AND I.Quantity < 200
```

As a special form of JOIN we can consider *self-join* which refers to any kind of join used to join a table to itself. Self-join is used in cases where the inner and outer queries refer to the same table. The following query example will return only those products whose price is higher than average prices of the same sub-category.

```
SELECT P1.Name, P1.ListPrice
FROM Production.Product AS P1
     INNER JOIN Production.Product AS P2
     ON P1.ProductSubcategoryID=P2.ProductSubcategoryID
GROUP BY P1.Name, P1.ListPrice
HAVING P1.ListPrice > AVG (P2.ListPrice)
```

Result is:

Name	ListPrice
-----	-----
Mountain-100 Silver, 38	3399,99
Mountain-100 Silver, 42	3399,99
....	
HL Road Tire	32,60
Touring Tire	28,99
(116 row(s) affected)	

Practice: Examples of Join Operations

In this practice, you will:

- Create a query that retrieves data as described in the following scenario

Query scenario

Company manager requires a list of all products, which next to the product name should contain a name of the product vendor, and name of the unit measure. Example of the result set:

Product	Vendor	Measure
Adjustable Rack	Shelvard, Inc.	Case

In this practice you will write queries in SQL Server 2012 Management Studio environment. Practice is based on Lesson 1 from Module 4.

To successfully complete the exercise you need the following resources:

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

Exercise 1: Examples of Join Operations

- Click Start→Microsoft SQL Server 2012→SQL Server Management Studio
- 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)
- Use Windows Authentication
- Open File menu →New →Database Engine Query

Query Scenario

Company manager requires a list of all products, which next to the product name should contain a name of the product vendor, and name of the unit measure.

- Type following TSQL code in query windows:

```
SELECT P.Name AS Product, V.Name AS Vendor, M.Name as Measure
FROM Production.Product P
INNER JOIN Purchasing.ProductVendor PV
ON P.ProductID = PV.ProductID
INNER JOIN Purchasing.Vendor V
ON PV.BusinessEntityID= V.BusinessEntityID
```

```
INNER JOIN Production.UnitMeasure M
ON M.UnitMeasureCode = PV.UnitMeasureCode
```

6. Click on Execute or press F5 to run

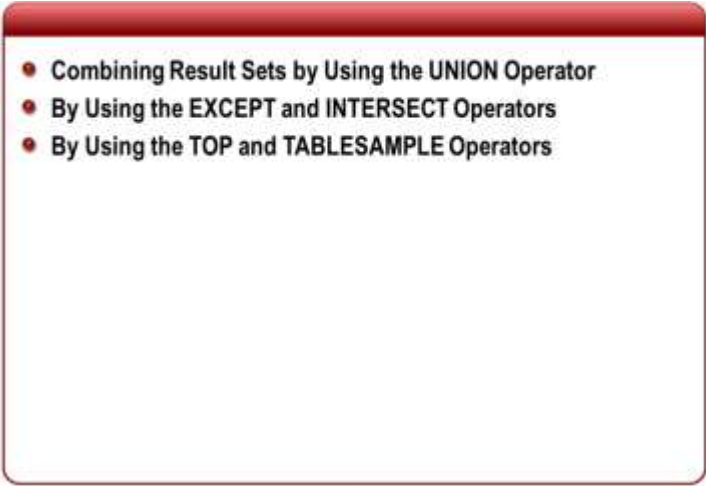
Result is:

Product	Vendor	Measure
Adjustable Race	Litware, Inc.	Case
All-Purpose Bike Stand	Green Lake Bike Company	Each
AWC Logo Cap	Integrated Sport Products	Each
....		
Women's Tights, M	Fitness Association	Each
Women's Tights, S	Fitness Association	Each

(460 row(s) affected)

7. Analyze the previous query with special emphasis on joining columns

Lesson 2: Limiting Result Sets

- 
- Combining Result Sets by Using the UNION Operator
 - By Using the EXCEPT and INTERSECT Operators
 - By Using the TOP and TABLESAMPLE Operators

There are not so rare cases in which you need to combine results returned from two or more SELECT statements into a single result set. One of the easiest ways to combine result sets is by using the relational operators UNION, EXCEPT and INTERSECT. All these operators take two result sets as input and perform a very specific combining operation in order to create different output.

Objectives

After completing this lesson, you will be able to combine result sets by:

- Using the UNION Operator
- Using the EXCEPT and INTERSECT Operators
- Using the TOP and TABLESAMPLE Operators

Combining Result Sets by Using the UNION Operator

- **UNION** combine the results of two or more **SELECT** statements without returning duplicate rows
- **SELECT** statements used with **UNION** operator must comply with certain rules. They must have:
 - the same number of columns
 - the same data type
 - and the same column order

```
SELECT ProductModelID AS [Product ID],
       ModifiedDate AS [Date of modification]
FROM Production.ProductModelProductDescriptionCulture
UNION
SELECT ProductModelID AS [Product ID],
       ModifiedDate AS [Date of modification]
FROM Production.ProductModel
```

UNION operator enables you to combine the results of two or more SELECT statements without returning duplicate rows. This means that UNION operator functions identically to the UNION operator with the DISTINCT keyword. However, each SELECT statement used with UNION operator must comply with certain rules: have the same number, data type and order of columns.

Executions of subsequent queries would return result set containing only the rows from individual tables, or more precisely the first query would return 762 records, and the second query 128 records. By using the UNION operator, we have created a unique result set of 201 records.

```
SELECT ProductModelID AS [Product ID],
       ModifiedDate AS [Date of modification]
FROM Production.ProductModelProductDescriptionCulture
UNION
SELECT ProductModelID AS [Product ID],
       ModifiedDate AS [Date of modification]
FROM Production.ProductModel
```

Result is:

Product ID	Date of modification
1	2007-06-01 00:00:00.000
2	2005-06-01 00:00:00.000
2	2007-06-01 00:00:00.000
...	
127	2007-06-01 00:00:00.000
128	2007-06-01 00:00:00.000

(201 row(s) affected)

If you wish to create a UNION without eliminating duplicated rows, then you can use the UNION ALL operator, which does not use DISTINCT feature. In this case, the previous query would return 890 records.

By Using the EXCEPT and INTERSECT Operators

- **EXCEPT** - produces a result set by removing all rows from the first result set which are present in the second result set.
- **INTERSECT** - produces a result set by identifying common rows in both sets

```

SELECT ProductModelID AS [Product ID],
       ModifiedDate AS [Date of modification]
FROM Production.ProductModelProductDescriptionCulture
INTERSECT
SELECT ProductModelID AS [Product ID],
       ModifiedDate AS [Date of modification]
FROM Production.ProductModel
  
```

Product ID	Date of modification
1	2007-05-01 00:00:00.000
22	2007-05-01 00:00:00.000
....	
127	2007-06-01 00:00:00.000

(54 row(s) affected)

Operator EXCEPT produces a result set by removing all rows from the first result set which are present in the second result set. Once again, it is important to mention that the same rules that apply to the UNION operator, also apply to the EXCEPT operator. Based on the described functionality of the EXCEPT operator, you can see why the following query returns only 73 rows.

```

SELECT ProductModelID AS [Product ID],
       ModifiedDate AS [Date of modification]
FROM Production.ProductModelProductDescriptionCulture
EXCEPT
SELECT ProductModelID AS [Product ID],
       ModifiedDate AS [Date of modification]
FROM Production.ProductModel
  
```

Result is:

Product ID	Date of modification
2	2007-06-01 00:00:00.000
3	2007-06-01 00:00:00.000
4	2007-06-01 00:00:00.000
....	
125	2007-06-01 00:00:00.000
126	2007-06-01 00:00:00.000

(73 row(s) affected)

Operator INTERSECT produces a result set by identifying common rows in both sets returned by SELECT statement.

```

SELECT ProductModelID AS [Product ID],
       ModifiedDate AS [Date of modification]
FROM Production.ProductModelProductDescriptionCulture
INTERSECT
  
```

```
SELECT ProductModelID AS [Product ID],
       ModifiedDate AS [Date of modification]
FROM Production.ProductModel
```

Result is:

Product ID	Date of modification
1	2007-06-01 00:00:00.000
22	2007-06-01 00:00:00.000
24	2007-06-01 00:00:00.000
....	
121	2007-06-01 00:00:00.000
122	2007-06-01 00:00:00.000
127	2007-06-01 00:00:00.000

(54 row(s) affected)

If you use relational operators UNION, EXCEPT and INTERSECT together in the same expression, you must be aware of their precedence. Therefore, the operator INTERSECT has the highest precedence, but EXCEPT and UNION are evaluated from left to right, based on their position in the expression.

By Using the TOP and TABLESAMPLE Operators

- **TOP** – allows you to limit the number of rows
 - Used in combination with the ORDER BY
 - ORDER BY operator causes TOP to return the specified number of rows but from the sorted result set
- **TABLESAMPLE** – returns randomly selected rows:
 - Sample size in percentage (PERCENT) or row (ROWS) value
 - Number of rows returned?

```
SELECT Name as Product
FROM Production.Product TABLESAMPLE (10 PERCENT)
```

- Used in conjunction with TOP keyword

```
SELECT TOP 50 Name as Product
FROM Production.Product TABLESAMPLE (50 PERCENT)
```

The TOP keyword allows you to limit the number of rows in the data set that results from a query. It is usually used in combination with the ORDER BY clause to obtain a certain number of rows with the highest or lowest values. The following query will return a list of products that should be urgently ordered from suppliers because of their quantity.

```
SELECT TOP 10 P.Name AS Product, L.Name AS Location, I.Quantity
FROM Production.Product P
    INNER JOIN Production.ProductInventory I
ON P.ProductID = I.ProductID
    INNER JOIN Production.Location L
ON L.LocationID = I.LocationID
ORDER BY I.Quantity ASC
```

Result is:

Product	Location	Quantity
Half-Finger Gloves, M	Finished Goods Storage	0
Women's Tights, M	Finished Goods Storage	0
Hitch Rack - 4-Bike	Finished Goods Storage	0
....		
Paint - Red	Paint Storage	24

(10 row(s) affected)

As you already know, the TOP keyword will return a specified number of rows contained in the result set. However, note that the TOP keyword used in conjunction with ORDER BY operator returns the specified number of rows but from the sorted result set.

You will probably find yourself wondering how you can create a query that returns randomly selected rows. The simplest solution is to use TABLESAMPLE operator specified in the FROM clause. Sample size (number of randomly selected rows) can be expressed either by using percentage (keyword PERCENT) or row (keyword ROWS) values. However, sample size expressed in rows will be converted in percentage value based on available statistics.

```
SELECT Name as Product
FROM Production.Product TABLESAMPLE (10 PERCENT)
```

Result is:

```
Product
-----
Touring-1000 Yellow, 54
Touring-1000 Yellow, 60
...
Women's Tights, L
Women's Tights, M
Women's Tights, S

(24 row(s) affected)
```

You've probably already concluded that previous result set does not represent 10% of the total number of rows in the Products table (total number of rows is 504). Also, if you execute the query a few times, you'll notice that almost every time you get a different number of rows.

This behavior is caused by the fact that actual data in your table is stored in pages and TABLESAMPLE operator picks the whole content of randomly chosen data page. However, each data page can contain a different number of rows. Since it is hard to predict the number of rows returned by TABLESAMPLE operator, it is usually used in conjunction with TOP keyword as shown in the following example.

```
SELECT TOP 50 Name as Product
FROM Production.Product TABLESAMPLE (50 PERCENT)
```

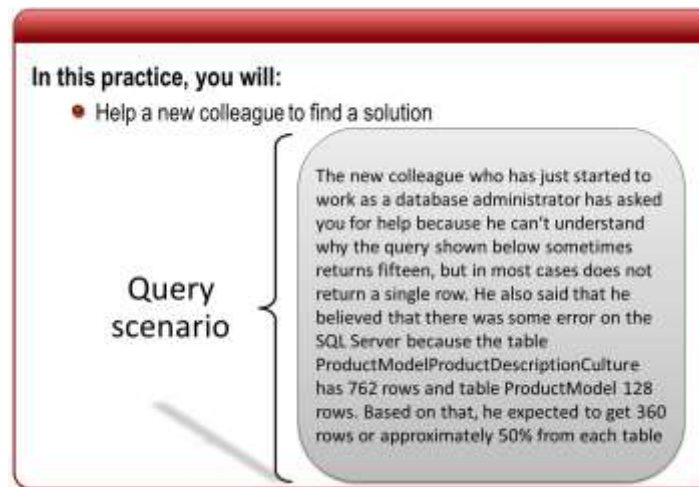
Result is:

```
Product
-----
LL Mountain Frame - Black, 52
LL Mountain Frame - Silver, 40
...
LL Touring Seat Assembly
LL Touring Seat/Saddle

(50 row(s) affected)
```

In addition, it is important to remember that operator TABLESAMPLE cannot be applied to derived tables.

Practice: Limiting Result Sets



In this practice you will write queries in SQL Server 2012 Management Studio environment. The practice is based on Lesson 2 from Module 4.

To successfully complete the exercise you need the following resources:

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

Exercise 2: Limiting Result Sets

8. Click Start→Microsoft SQL Server 2012→SQL Server Management Studio
9. 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)
10. Use Windows Authentication
11. Open File menu →New →Database Engine Query

Query Scenario

The new colleague who has just started to work as a database administrator has asked you for help because he can't understand why the query shown below sometimes returns fifteen, but in most cases does not return a single row. He also said that he believed that there was some error on the SQL server because the table ProductModelProductDescriptionCulture has 762 rows and table ProductModel 128 rows. Based on that, he expected to get 360 rows or approximately 50% from each table.

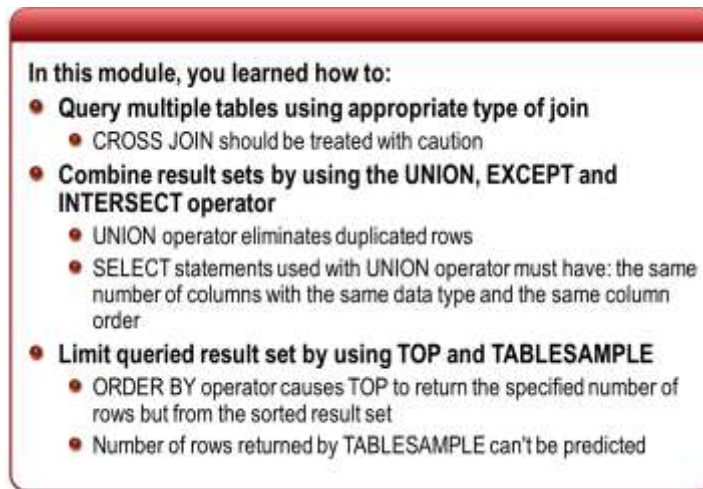
12. Type following TSQL code in query windows:

```
SELECT TOP (300) ProductModelID AS [Product ID] ,  
      ModifiedDate AS [Date of modification]
```

```
FROM Production.ProductModelProductDescriptionCulture
TABLESAMPLE (50 PERCENT)
INTERSECT
SELECT TOP (60) ProductModelID AS [Product ID],
    ModifiedDate AS [Date of modification]
FROM Production.ProductModel
TABLESAMPLE (50 PERCENT)
```

What kind of answer will you give to the new colleague?

Summary



In this module, you learned how to:

- **Query multiple tables using appropriate type of join**
 - CROSS JOIN should be treated with caution
- **Combine result sets by using the UNION, EXCEPT and INTERSECT operator**
 - UNION operator eliminates duplicated rows
 - SELECT statements used with UNION operator must have: the same number of columns with the same data type and the same column order
- **Limit queried result set by using TOP and TABLESAMPLE**
 - ORDER BY operator causes TOP to return the specified number of rows but from the sorted result set
 - Number of rows returned by TABLESAMPLE can't be predicted

At this moment, you are definitely able to create queries that can be used to support the basic forms of reporting. Database typically contains dozens of tables, and is therefore extremely important that you master creating queries over multiple tables. This implies the knowledge of the functioning of INNER, OUTER and CROSS joins.

Relational operator UNION, EXCEPT or INTERSECT are always on hand when you need to combine result sets from two or more SELECT statements. Also, note that the query testing on large production databases can significantly affect performance, and therefore it is advisable to limit the number of results by using the TOP keyword.

In addition to the aforementioned, key points you should remember from this module are:

- CROSS JOIN should be treated with caution
- UNION operator eliminates duplicated rows
- SELECT statements used with UNION operator must have: the same number of columns with the same data type and the same column order
- ORDER BY operator causes TOP to return the specified number of rows but from the sorted result set
- Number of rows returned by TABLESAMPLE can't be predicted