Databases I

Etelka Szendrői (PhD)
Associate professor
szendroi@mik.pte.hu

Stored Procedure

Stored Procedure

- A **stored procedure** is a named collection of procedural and SQL statements. Stored procedures are stored in the database.
- There are two clear advantages to the use of stored procedures:
  - Stored procedures substantially reduce network traffic and increase performance. Because the procedure is stored at the server, there is no transmission of individual SQL statements over the network. The use of stored procedures improves system performance because all transactions are executed locally on the RDBMS, so each SQL statement does not have to travel over the network.
  - Stored procedures help reduce code duplication by means of code isolation and code sharing (creating unique SQL modules that are called by application programs), thereby minimizing the chance of errors and the cost of application development and maintenance.

Stored procedure

Stored procedures can call other stored procedures.

Whenever a RETURN is executed, execution of the stored procedure ends and control returns to the caller.

Syntax:

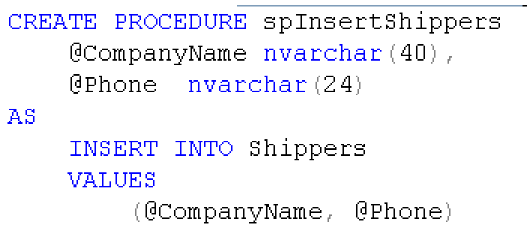
```
CREATE PROCEDURE procName
[parameters]
AS
Begin
    statements
End
```

To Modify: ALTER PROCEDURE procName

To Delete: DROP PROCEDURE procName

Stored procedure

Create a stored procedure to insert a new record to the Shippers table (Northwind database)



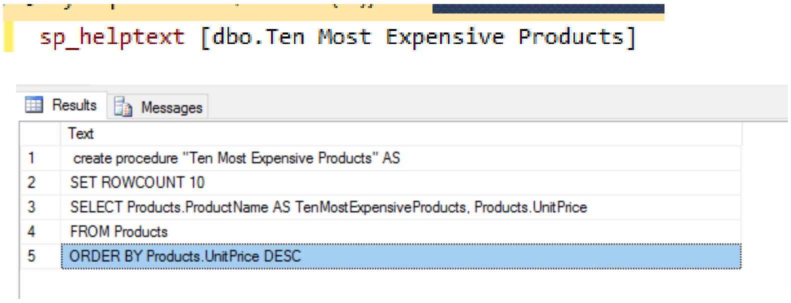
```
CREATE PROCEDURE spInsertShippers
    @CompanyName nvarchar(40),
    @Phone  nvarchar(24)
AS
    INSERT INTO Shippers
    VALUES
        (@CompanyName, @Phone)
```

Execute the stored procedure:

```
EXEC spInsertShippers 'Speedy Shippers Ltd.', '(503) 555-5634'
```

Sp_helptext system stored procedure

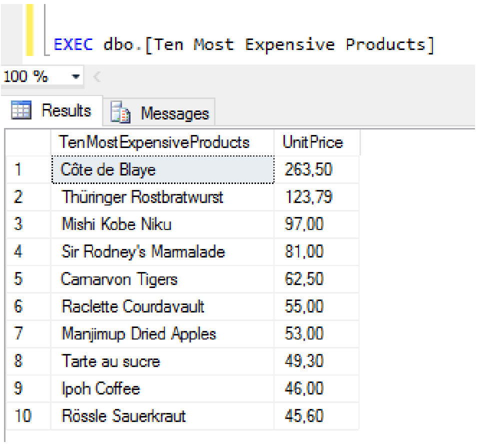
To watch the content of a stored procedure use the sp_helptext system stored procedure



```
sp_helptext [dbo.Ten Most Expensive Products]
```

	Text
1	create procedure "Ten Most Expensive Products" AS
2	SET ROWCOUNT 10
3	SELECT Products.ProductName AS TenMostExpensiveProducts, Products.UnitPrice
4	FROM Products
5	ORDER BY Products.UnitPrice DESC

Execute the Stored procedure



```
EXEC dbo.[Ten Most Expensive Products]
```

	TenMostExpensiveProducts	UnitPrice
1	Côte de Blaye	263,50
2	Thüringer Rostbratwurst	123,79
3	Mishi Kobe Niku	97,00
4	Sir Rodney's Marmalade	81,00
5	Carnarvon Tigers	62,50
6	Raclette Courdavault	55,00
7	Manjimup Dried Apples	53,00
8	Tarte au sucre	49,30
9	Ipoh Coffee	46,00
10	Rössle Sauerkraut	45,60

Stored procedures

Create in current database by using the CREATE PROCEDURE statement

```
CREATE PROCEDURE Production.LongLeadProducts
AS
  SELECT  Name, ProductNumber
  FROM    Production.Product
  WHERE   DaysToManufacture >= 1

GO
```

Use EXECUTE to run stored procedure

```
EXECUTE Production.LongLeadProducts
```

Syntax for Altering and Dropping Stored Procedures

- ALTER PROCEDURE

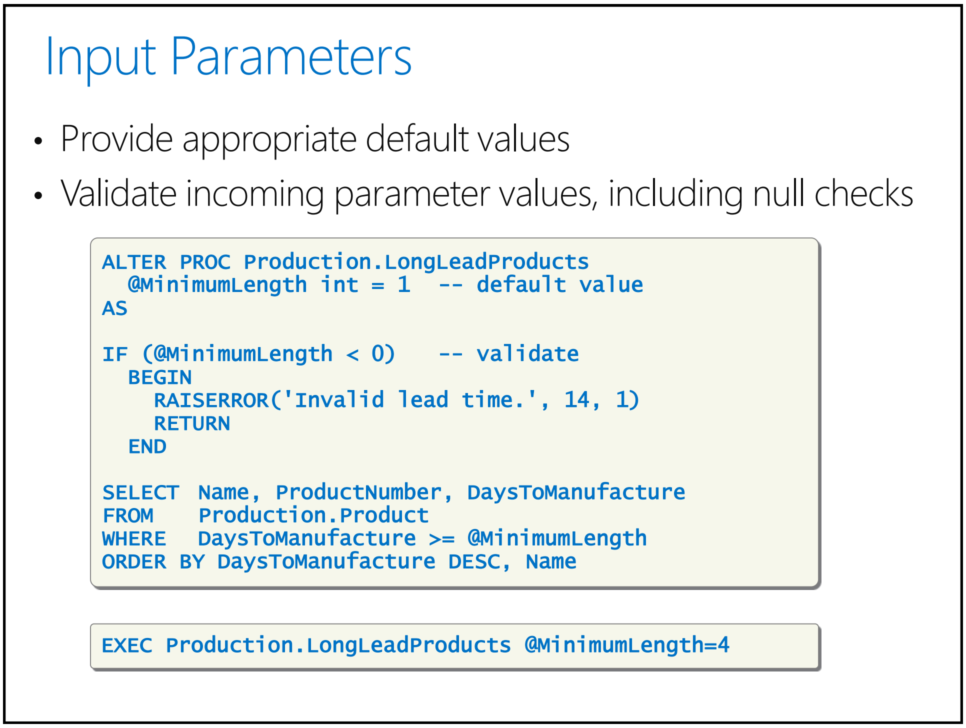
```
ALTER PROC Production.LongLeadProducts
AS
  SELECT  Name, ProductNumber, DaysToManufacture
  FROM    Production.Product
  WHERE   DaysToManufacture >= 1
  ORDER BY DaysToManufacture DESC, Name

GO
```

- DROP PROCEDURE

```
DROP PROC Production.LongLeadProducts
```

Stored Procedure

- *argument* specifies the parameters that are passed to the stored procedure. A stored procedure could have zero or more arguments or parameters.
- *IN/OUT* indicates whether the parameter is for input, output, or both.
- *data-type* is one of the procedural SQL data types used in the RDBMS. The data types normally match those used in the RDBMS table creation statement.
- Variables can be declared. You must specify the variable name, its data type, and (optionally) an initial value.

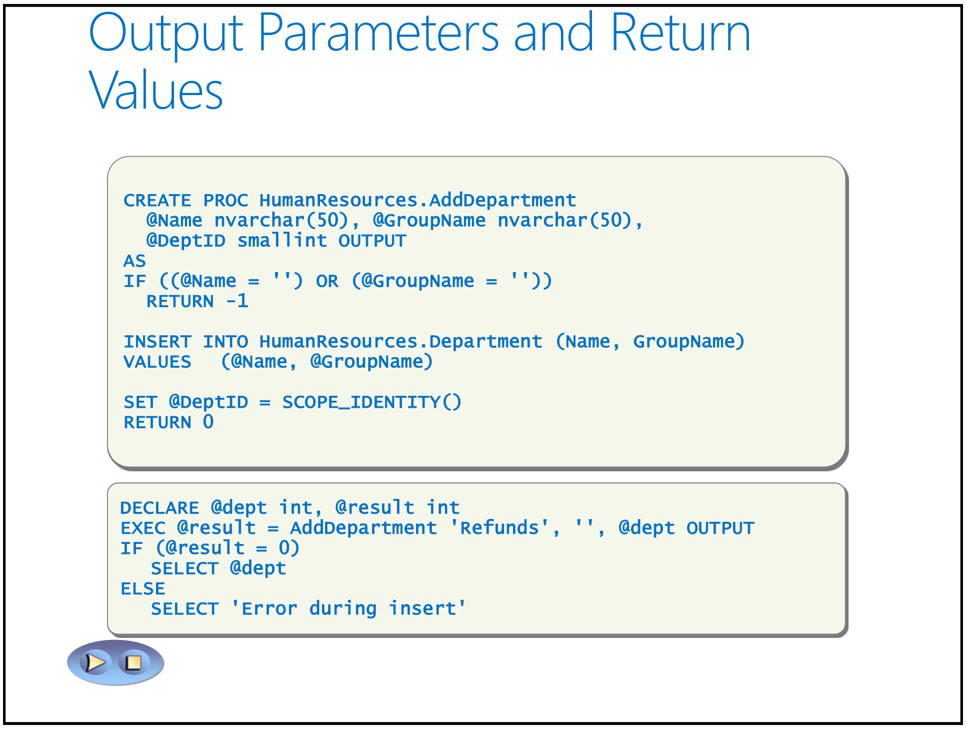


Input Parameters

- Provide appropriate default values
- Validate incoming parameter values, including null checks

```
ALTER PROC Production.LongLeadProducts
  @MinimumLength int = 1  -- default value
AS

IF (@MinimumLength < 0)   -- validate
  BEGIN
    RAISERROR('Invalid lead time.', 14, 1)
    RETURN
  END

SELECT  Name, ProductNumber, DaysToManufacture
FROM    Production.Product
WHERE   DaysToManufacture >= @MinimumLength
ORDER BY DaysToManufacture DESC, Name
```

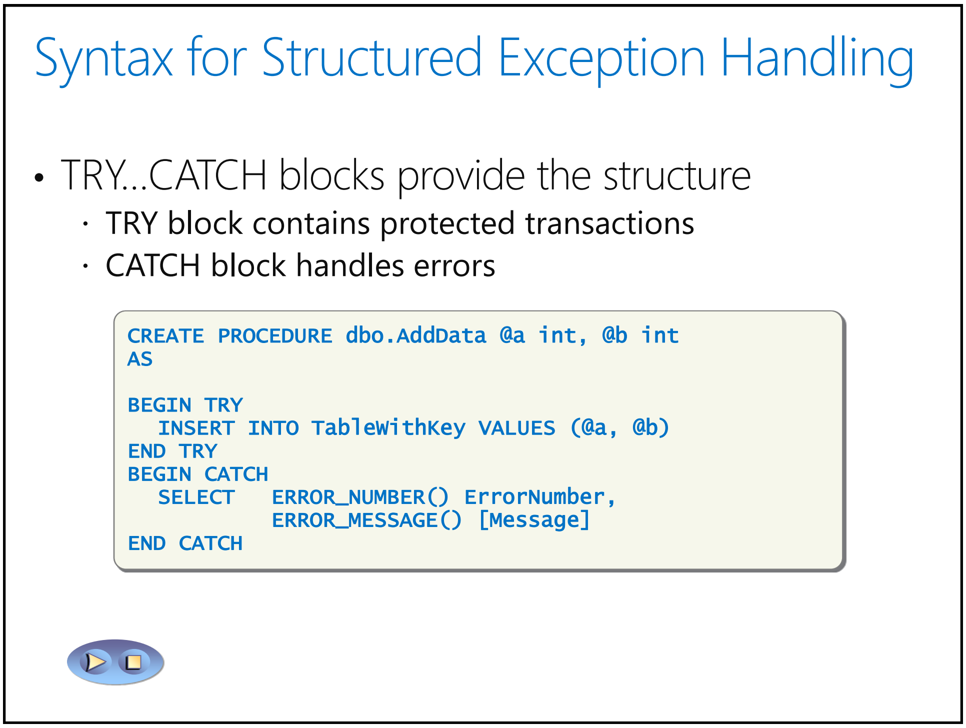
```
EXEC Production.LongLeadProducts @MinimumLength=4
```



Output Parameters and Return Values

```
CREATE PROC HumanResources.AddDepartment
  @Name nvarchar(50), @GroupName nvarchar(50),
  @DeptID smallint OUTPUT
AS
IF ((@Name = '') OR (@GroupName = ''))
  RETURN -1

INSERT INTO HumanResources.Department (Name, GroupName)
VALUES   (@Name, @GroupName)

SET @DeptID = SCOPE_IDENTITY()
RETURN 0
```

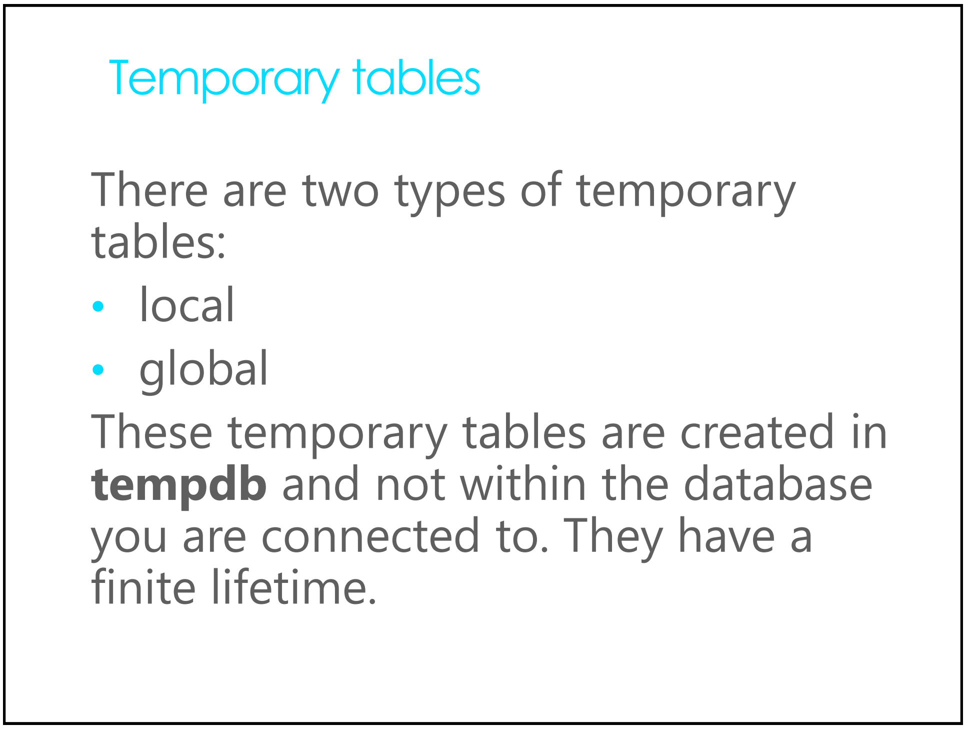
```
DECLARE @dept int, @result int
EXEC @result = AddDepartment 'Refunds', '', @dept OUTPUT
IF (@result = 0)
   SELECT @dept
ELSE
   SELECT 'Error during insert'
```



Syntax for Structured Exception Handling

- TRY...CATCH blocks provide the structure
  - TRY block contains protected transactions
  - CATCH block handles errors

```
CREATE PROCEDURE dbo.AddData @a int, @b int
AS

BEGIN TRY
   INSERT INTO TableWithKey VALUES (@a, @b)
END TRY
BEGIN CATCH
   SELECT   ERROR_NUMBER() ErrorNumber,
            ERROR_MESSAGE() [Message]
END CATCH
```



Temporary tables

There are two types of temporary tables:

- local
- global

These temporary tables are created in **tempdb** and not within the database you are connected to. They have a finite lifetime.

Local temporary tables

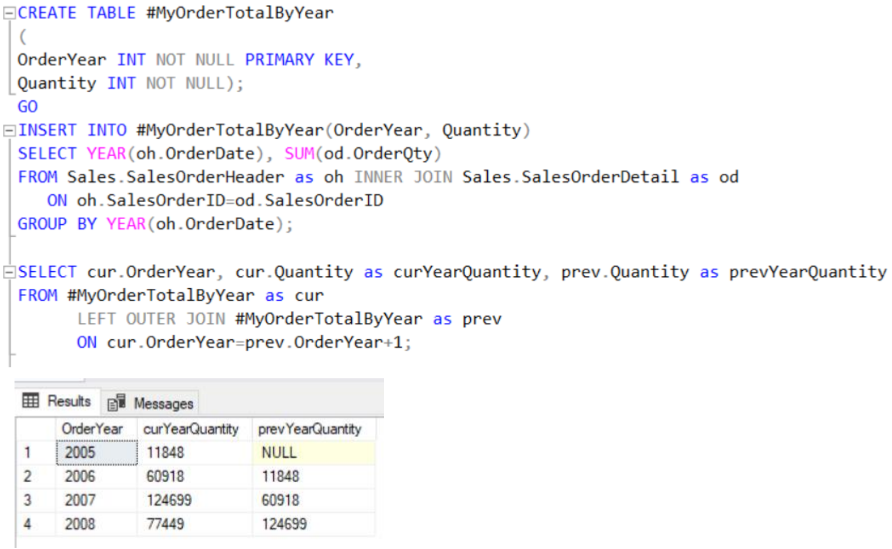
A local temporary table is defined by prefixing the table name by a single hash mark: #.

CREATE TABLE #TempTableName (…..);

The scope of a local temporary table is the connection that created it only.

A local temporary table survives until the connection it was created within is dropped (when the Query Editor window is closed).

Local temporary table

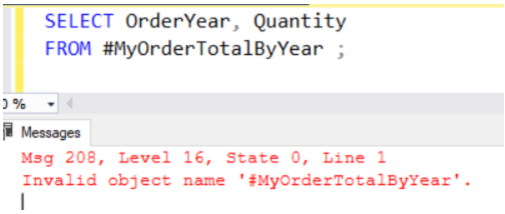


```
CREATE TABLE #MyOrderTotalByYear
(
OrderYear INT NOT NULL PRIMARY KEY,
Quantity INT NOT NULL);
GO
INSERT INTO #MyOrderTotalByYear(OrderYear, Quantity)
SELECT YEAR(oh.OrderDate), SUM(od.OrderQty)
FROM Sales.SalesOrderHeader as oh INNER JOIN Sales.SalesOrderDetail as od
   ON oh.SalesOrderID=od.SalesOrderID
GROUP BY YEAR(oh.OrderDate);

SELECT cur.OrderYear, cur.Quantity as curYearQuantity, prev.Quantity as prevYearQuantity
FROM #MyOrderTotalByYear as cur
      LEFT OUTER JOIN #MyOrderTotalByYear as prev
      ON cur.OrderYear=prev.OrderYear+1;
```

	OrderYear	curYearQuantity	prevYearQuantity
1	2005	11848	NULL
2	2006	60918	11848
3	2007	124699	60918
4	2008	77449	124699

Local temporary table

In a new query pan has been typed the same SELECT statement => error!(another session!!!)



```
SELECT OrderYear, Quantity
FROM #MyOrderTotalByYear ;
```

```
Msg 208, Level 16, State 0, Line 1
Invalid object name '#MyOrderTotalByYear'.
```

Global temporary table

A global temporary table is defined by prefixing the table name by a double hash mark: ##.

The scope of a global temporary table differs significantly.

When a connection creates the table, it is then available to be used by any user and any connection, just like a permanent table.

A global temporary table will be “deleted” only when all connections to it have been closed.

TRIGGERS

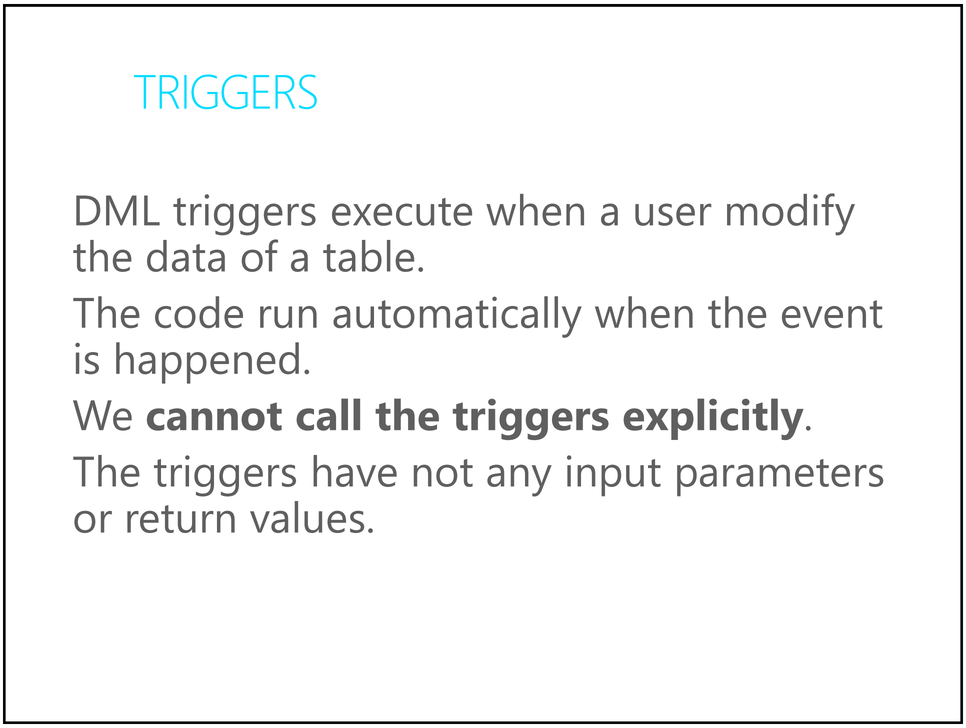
TRIGGERS

The trigger is a special stored procedure, which answers an event in the database

Types:

- DDL (data definition)
- DML (data manipulation)

DDL triggers fire (execute) if a user modify the structure of the database objects (CREATE, ALTER, DROP)



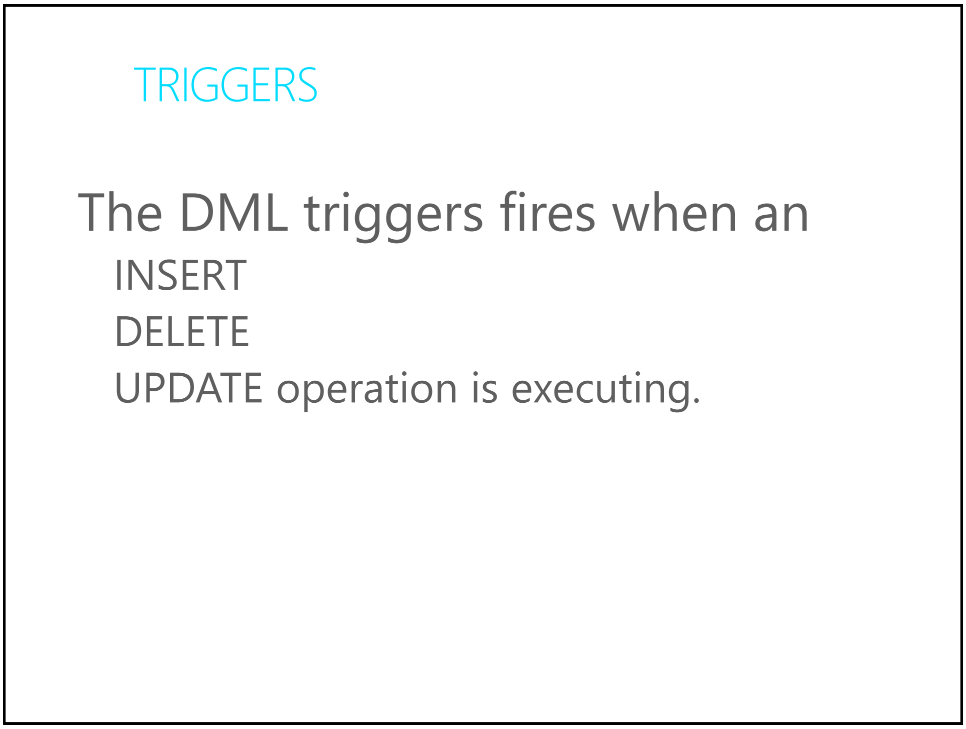
TRIGGERS

DML triggers execute when a user modify the data of a table.

The code run automatically when the event is happened.

We **cannot call the triggers explicitly**.

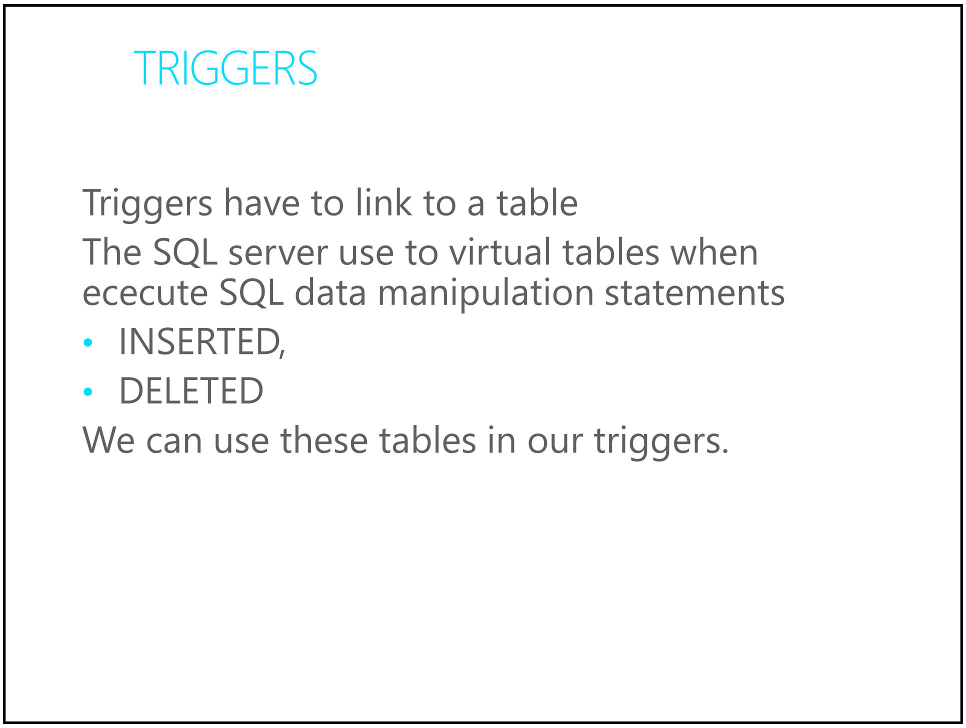
The triggers have not any input parameters or return values.



TRIGGERS

The DML triggers fires when an

INSERT
DELETE
UPDATE operation is executing.



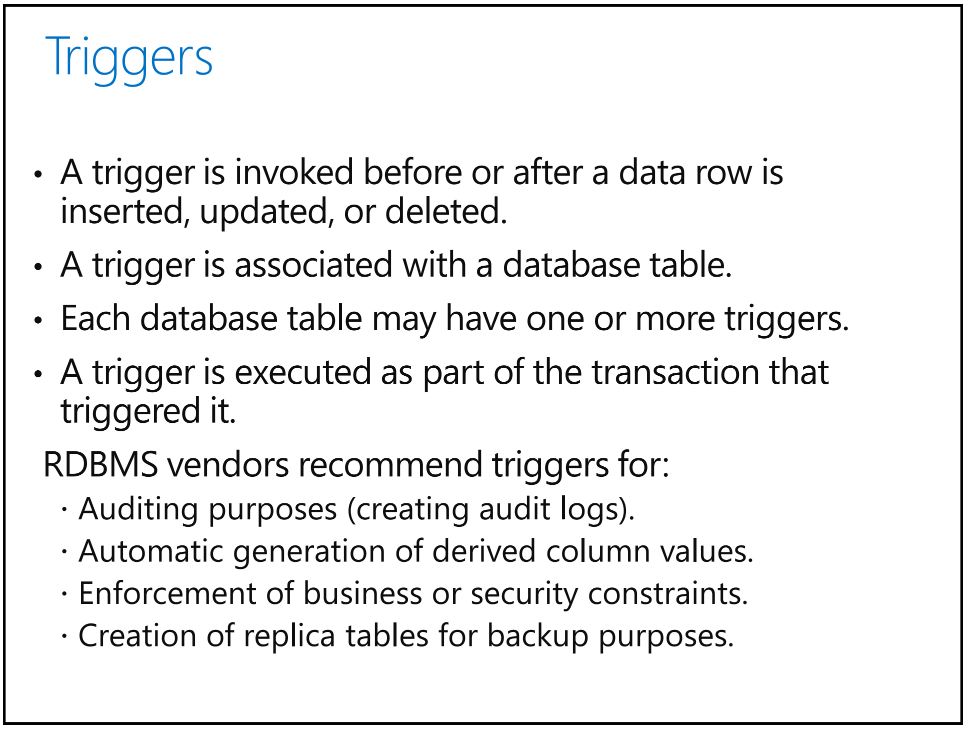
TRIGGERS

Triggers have to link to a table

The SQL server use to virtual tables when ececute SQL data manipulation statements

- INSERTED,
- DELETED

We can use these tables in our triggers.



Triggers

- A trigger is invoked before or after a data row is inserted, updated, or deleted.
- A trigger is associated with a database table.
- Each database table may have one or more triggers.
- A trigger is executed as part of the transaction that triggered it.

RDBMS vendors recommend triggers for:

- Auditing purposes (creating audit logs).
- Automatic generation of derived column values.
- Enforcement of business or security constraints.
- Creation of replica tables for backup purposes.

TRIGGERS

Syntax:

```
CREATE TRIGGER triggername
ON [shema.]tablename
{FOR|AFTER
[DELETE],[INSERT],[UPDATE] |INSTEAD OF}
AS
Sql statements
```

The triggering timing: FOR or AFTER. This timing indicates when the trigger's SQL code executes— in this case, before or after the triggering statement is completed.

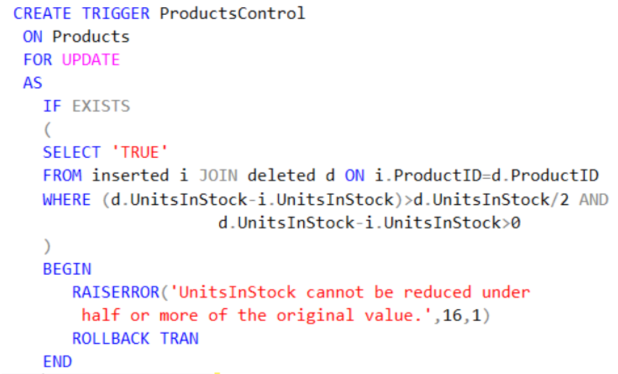
Some Notes with TRIGGERS

- Oracle and MS SQL Server allow a trigger to include multiple triggering conditions; that is, any combination of INSERT, UPDATE, and/or DELETE.
- MySQL allows only one triggering condition per trigger. Therefore, if a certain set of actions should be taken in the case of multiple events, for example, during an UPDATE or an INSERT, then two separate triggers are required in MySQL. To reduce having duplicate code in both triggers, it is a common practice to create a stored procedure that performs the common actions, then have both triggers call the same stored procedure.

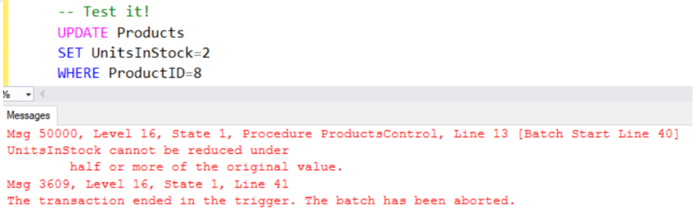
Trigger Example

Create a trigger which rejects the modification of the UnitsInStock value of a Product (Northwind DB), if its new value will be lower than the half of the original value was.

Triggers



```
CREATE TRIGGER ProductsControl
 ON Products
 FOR UPDATE
 AS
   IF EXISTS
   (
   SELECT 'TRUE'
   FROM inserted i JOIN deleted d ON i.ProductID=d.ProductID
   WHERE (d.UnitsInStock-i.UnitsInStock)>d.UnitsInStock/2 AND
                    d.UnitsInStock-i.UnitsInStock>0
   )
   BEGIN
      RAISERROR('UnitsInStock cannot be reduced under
       half or more of the original value.',16,1)
      ROLLBACK TRAN
   END
```



```
-- Test it!
UPDATE Products
SET UnitsInStock=2
WHERE ProductID=8
```

```
Msg 50000, Level 16, State 1, Procedure ProductsControl, Line 13 [Batch Start Line 40]
UnitsInStock cannot be reduced under
       half or more of the original value.
Msg 3609, Level 16, State 1, Line 41
The transaction ended in the trigger. The batch has been aborted.
```

Special Type of data

Geometry and Geography data types

- Geometry data type
  - The GEOMETRY data type might be used for a warehouse application to store the location of each product in the warehouse
  - The GEOMETRY data type follows a “flat Earth” model, with basically X, Y, and Z coordinates.
- Geography data type
  - The GEOGRAPHY data type can be used to store data that can be used in mapping software. You may wonder why two types that both store locations exist. The GEOGRAPHY data type represents the “round Earth” model, storing longitude and latitude. These data types implement international standards for spatial data.
- These data types supports the OpenGIS Simple Features for SQL standard, which is a specification published by an international regulatory body known as the Open Geospatial Consortium (OGC):
  - Well-Known Text (WKT),
  - Well-Known Binary (WKB),
  - Geography Markup Language (GML).

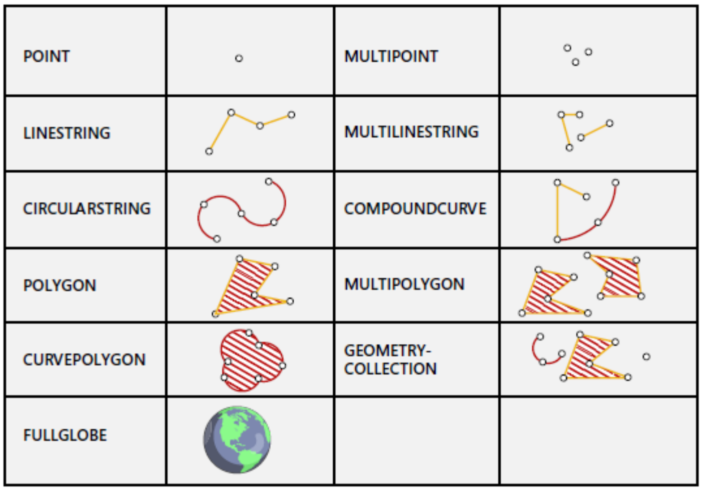
- Shapes are projected onto spatial models using vector objects—which are collections of points, lines, and polygons (closed shapes). Both the geometry and geography data types support the same Well-Known Text (WKT) markup language, which is a convention that expresses the vector objects that you define using a syntax governed by the OGC.

Examples of WKT Strings

WKT String	Description
POINT(6 10)	A single point at xy-coordinates 6, 10
POINT(-111.06687 45.01188)	A single point on the earth (longitude/latitude coordinates)
LINESTRING(3 4,10 50,20 25)	A two-part line, drawn between three points specified as xy-coordinates
POLYGON((-75.17031 39.95601, -75.16786 39.95778, -75.17921 39.96874, -75.18441 39.96512, -75.17031 39.95601))	An enclosed shape on the earth drawn between the points specified as longitude/latitude coordinates

As you can see, the same WKT syntax is used for expressing spatial elements using either the planar or the geodetic model. Also notice that geodetic coordinates are always expressed in WKT with the longitude value first, followed by the latitude value.

Geometry data type



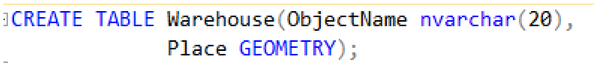
POINT		MULTIPOINT	
LINESTRING		MULTILINESTRING	
CIRCULARSTRING		COMPOUNDCURVE	
POLYGON		MULTIPOLYGON	
CURVEPOLYGON		GEOMETRY-COLLECTION	
FULLGLOBE			

Geometry and Geography data types

- These complex types are based on the CLR (Common Language Runtime). New data types similar to these built-in complex types can be created with a .NET language. We should know how to use the built-in CLR types.
- These types are in .NET as a class library (DLL) and offer us more than 90 methods. The name of these methods begin with the ST characters symbolize that support the OGC standard.
- Some of the methods are static and some of them are object instance methods.

Geometry data type

- Planar model is a flat surface where shapes are plotted using two-dimensional x- and y-coordinates
- These coordinates are based on an arbitrary measurement system, so you can define any measurement unit you want (for example, centimeters, meters, kilometers, inches, feet, miles, pixels, and so on).
- Our first example demonstrates the geometry data type in a very simple scenario. You will define and store shapes representing different objects.
- The first thing you need to do is create tables to hold the shapes that define the warehouse and objects to place in it.
- You can give the area, and distance between them.



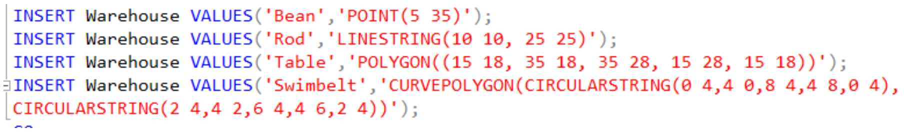
```
CREATE TABLE Warehouse(ObjectName nvarchar(20),
                Place GEOMETRY);
```

Geometry datatype

- Let be walls of the warehouse:

```
INSERT Warehouse VALUES ('Walls', 'LINESTRING(0 0, 40 0, 40 40, 0 40, 0 0)')
```

- Put some object into it.

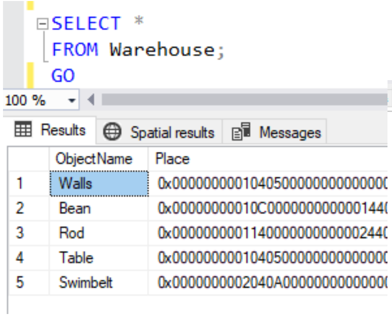


```
INSERT Warehouse VALUES('Bean','POINT(5 35)');
INSERT Warehouse VALUES('Rod','LINESTRING(10 10, 25 25)');
INSERT Warehouse VALUES('Table','POLYGON((15 18, 35 18, 35 28, 15 28, 15 18))');
INSERT Warehouse VALUES('Swimbelt','CURVEPOLYGON(CIRCULARSTRING(0 4,4 0,8 4,4 8,0 4),
CIRCULARSTRING(2 4,4 2,6 4,4 6,2 4))');
```

35

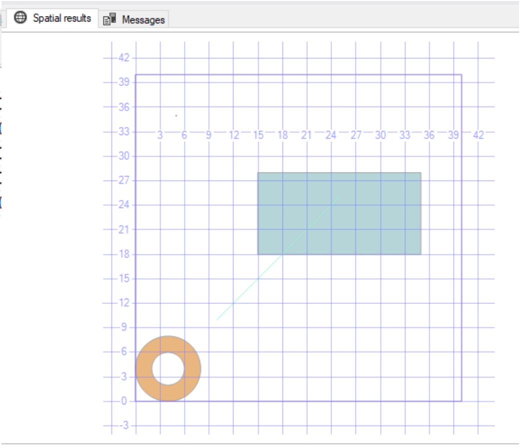
Geometry datatype

- List the content of the warehouse:



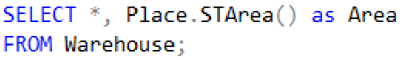
```
SELECT *
FROM Warehouse;
GO
```

	ObjectName	Place
1	Walls	0x0000000001040500000000000000
2	Bean	0x00000000010C0000000000001440
3	Rod	0x0000000001140000000000002440
4	Table	0x0000000001040500000000000000
5	Swimbelt	0x0000000002040A00000000000000

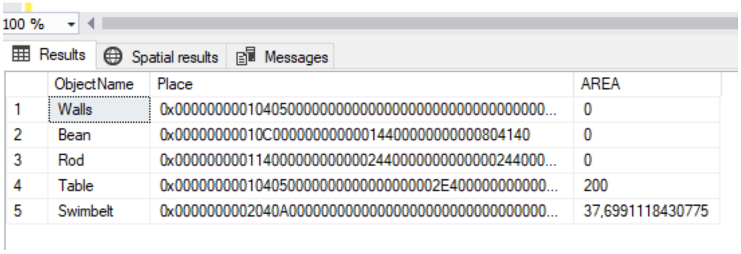


Geometry datatype

- Calculate the area occupied by objects:



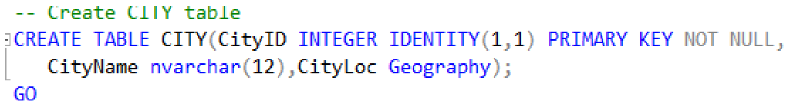
```
SELECT *, Place.STArea() as Area
FROM Warehouse;
```



	ObjectName	Place	AREA
1	Walls	0x000000000104050000000000000000000000000000000...	0
2	Bean	0x00000000010C000000000000001440000000000000804140	0
3	Rod	0x00000000011400000000000000244000000000000000244000...	0
4	Table	0x00000000010405000000000000000000002E400000000000...	200
5	Swimbelt	0x0000000002040A000000000000000000000000000000000...	37,6991118430775

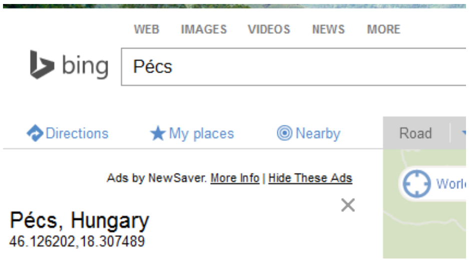
Geography datatype

- Create City table with location



```
  -- Create CITY table
CREATE TABLE CITY(CityID INTEGER IDENTITY(1,1) PRIMARY KEY NOT NULL,
     CityName nvarchar(12),CityLoc Geography);
GO
```

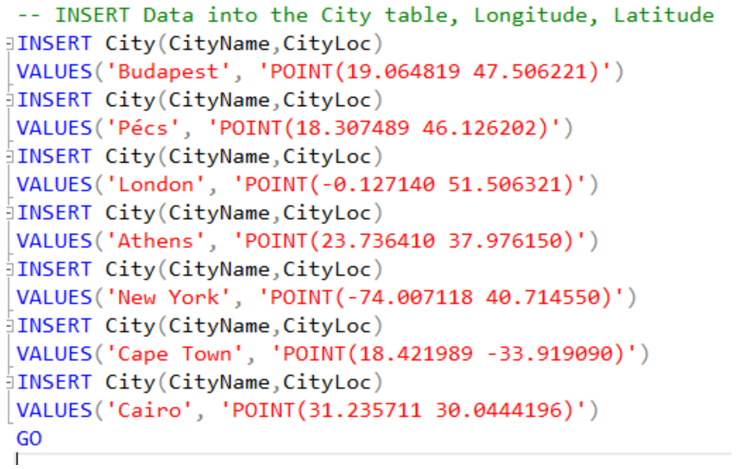
Add data to the City table



Pécs, Hungary
46.126202,18.307489

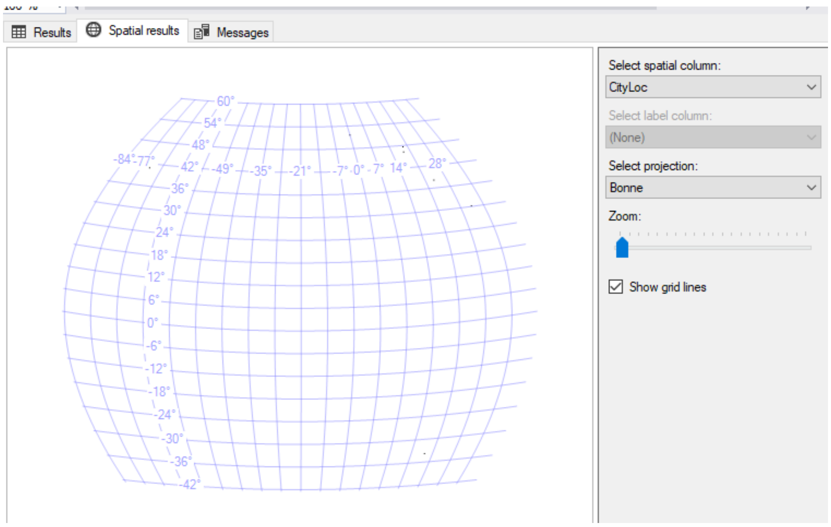
Geography datatype

- Insert data into the City table:



```
-- INSERT Data into the City table, Longitude, Latitude
INSERT City(CityName,CityLoc)
VALUES('Budapest', 'POINT(19.064819 47.506221)')
INSERT City(CityName,CityLoc)
VALUES('Pécs', 'POINT(18.307489 46.126202)')
INSERT City(CityName,CityLoc)
VALUES('London', 'POINT(-0.127140 51.506321)')
INSERT City(CityName,CityLoc)
VALUES('Athens', 'POINT(23.736410 37.976150)')
INSERT City(CityName,CityLoc)
VALUES('New York', 'POINT(-74.007118 40.714550)')
INSERT City(CityName,CityLoc)
VALUES('Cape Town', 'POINT(18.421989 -33.919090)')
INSERT City(CityName,CityLoc)
VALUES('Cairo', 'POINT(31.235711 30.0444196)')
GO
```

The result



Select spatial column: CityLoc

Select label column: (None)

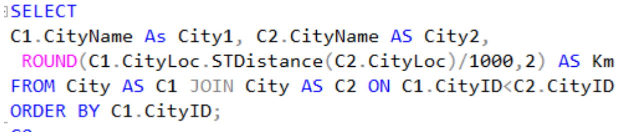
Select projection: Bonne

Zoom:

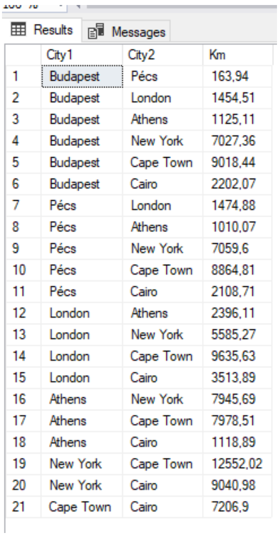
☑ Show grid lines

The distance between two city is calculated with STDISTANCE() function

The distances between cities



```
SELECT
C1.CityName As City1, C2.CityName AS City2,
 ROUND(C1.CityLoc.STDistance(C2.CityLoc)/1000,2) AS Km
FROM City AS C1 JOIN City AS C2 ON C1.CityID<C2.CityID
ORDER BY C1.CityID;
```



	City1	City2	Km
1	Budapest	Pécs	163,94
2	Budapest	London	1454,51
3	Budapest	Athens	1125,11
4	Budapest	New York	7027,36
5	Budapest	Cape Town	9018,44
6	Budapest	Cairo	2202,07
7	Pécs	London	1474,88
8	Pécs	Athens	1010,07
9	Pécs	New York	7059,6
10	Pécs	Cape Town	8864,81
11	Pécs	Cairo	2108,71
12	London	Athens	2396,11
13	London	New York	5585,27
14	London	Cape Town	9635,63
15	London	Cairo	3513,89
16	Athens	New York	7945,69
17	Athens	Cape Town	7978,51
18	Athens	Cairo	1118,89
19	New York	Cape Town	12552,02
20	New York	Cairo	9040,98
21	Cape Town	Cairo	7206,9

Thank you for your attention!