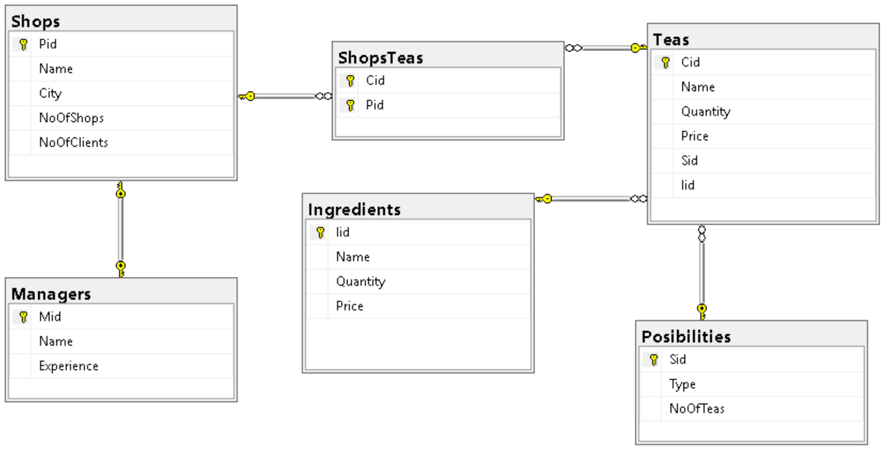
Laboratory 5

CRUD Operations

Create =Create new table or INSERT
Read=SELECT
Update=UPDATE
Delete=DELETE

You have to create stored procedures for CRUD operations for 3 tables (3 stored procedures for CRUD on each table and a main procedure, or 4 operations*3 tables =12 stored procedures, or …).
You will have to consider your own database and work on it. Here, we consider the database



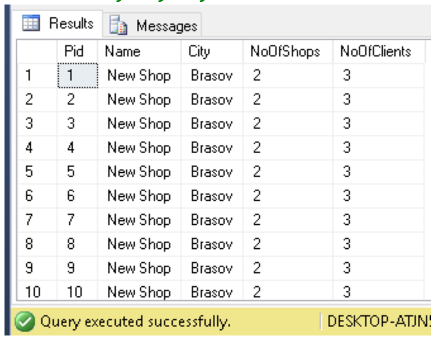
We can choose tables Shops, ShopsTeas, Teas, Ingredients and Posibilities.
The way that you implement your **stored procedures** is up to you!!!

1. An example of crud operation on a table can be (similar for Shops, Ingredients, Posibilities – these tables have only primary keys, no foreign keys)

```
USE [Example_Lab1]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[CRUD_Shops]
       @table_name Varchar(50),
       @name varchar(50),
       @city varchar(50),
       @nos int,
       @noc int,
       @noOfRows int
AS
BEGIN
       SET NOCOUNT ON;
       -- verify the parameters - at least one from the list -
with the help of a scalar function or a stored procedure with
output parameter

       -- CREATE=INSERT
       declare @n int =1
       -- we add as many rows as the parameter indicate us
       -- not all the fields must be given as parameters
       while @n<=@noOfRows begin
        insert into Shops(Name, City, NoOfShops, NoOfClients)
              Values(@name, @city, @nos, @noc)
        set @n=@n+1
       end
```

```
-- execute
EXEC CRUD_Shops 'Shops', 'New Shop',
'Brasov', 2, 3, 10
```



```
    Pid  Name      City    NoOfShops  NoOfClients
1   1    New Shop  Brasov  2          3
2   2    New Shop  Brasov  2          3
3   3    New Shop  Brasov  2          3
4   4    New Shop  Brasov  2          3
5   5    New Shop  Brasov  2          3
6   6    New Shop  Brasov  2          3
7   7    New Shop  Brasov  2          3
8   8    New Shop  Brasov  2          3
9   9    New Shop  Brasov  2          3
10  10   New Shop  Brasov  2          3
Query executed successfully.          DESKTOP-ATJN
```

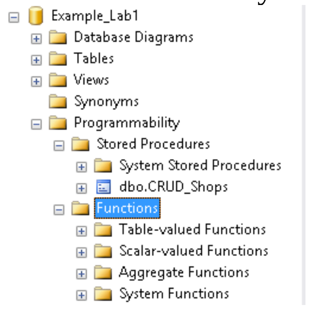
```
        -- READ=SELECT
        select * from Shops

        -- UPDATE
        update Shops set City='Cluj-Napoca' where NoOfClients>10
and Name LIKE 'A%'

        -- DELETE
        delete from Shops where NoOfClients=0

        print 'CRUD operations for table ' + @table_name
END
```

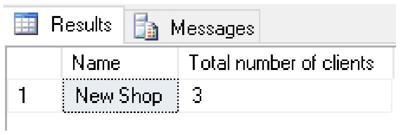
For the instruction SELECT, one can also use a **function** that returns a table. There are also scalar functions that can be used for verify the parameters.



- *inline Table-Valued Functions – RETURNS a TABLE*

```
USE Example_Lab1
GO
IF OBJECT_ID (N'ShopsF', N'IF') IS NOT NULL
    DROP FUNCTION ShopsF;
GO
CREATE FUNCTION ShopsF (@Pid int)
RETURNS TABLE
AS
RETURN
(
    SELECT Name, SUM(NoOfClients) AS 'Total number of clients'
    FROM Shops
    WHERE Pid=@Pid
    GROUP BY Name
);
GO
```

```
-- EXECUTE
SELECT * FROM ShopsF (2);
```



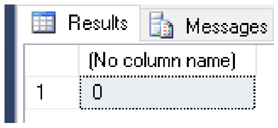
	Name	Total number of clients
1	New Shop	3

Return the Name and the Number of Clients for a Shop given with ID

- *Scalar-Valued Functions*

```
Use Example_Lab1
IF OBJECT_ID (N'dbo.Sfunction', N'FN') IS NOT NULL
    DROP FUNCTION dbo.Sfunction;
GO
-- return the Total number of clients for a given Shops Pid
CREATE FUNCTION dbo.Sfunction(@Pid int)
RETURNS int
AS
BEGIN
    DECLARE @r int;
    SELECT @r = SUM(NoOfClients)
    FROM Shops
    WHERE Pid = @Pid AND Name LIKE 'S%';
    IF (@r IS NULL) SET @r = 0;
    RETURN @r;
END;
GO
```

```
-- EXECUTION
SELECT dbo.Sfunction(2)
```



	(No column name)
1	0

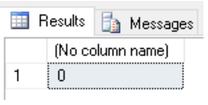
One **must create at least one function for each table** in which will verify some conditions related to a field of the table (that will appear as a parameter in the crud stored procedure)

For example: verify that the name of the Shop start with a letter, verify that the quantity for Ingredient to be positive, verify the e-mail address to respect a format, …

TestPrice – check for table Ingredients the field Price - when will be executed

```
Create function dbo.TestPrice(@p int)
RETURNS INT
AS
  BEGIN
       IF @p BETWEEN 10 AND 20 SET @p=1
       ELSE SET @p=0
       RETURN @p
END
```

```
-- execute
SELECT dbo.TestPrice(2)
```



```
   (No column name)
1  0
```

--or verification like

```
if dbo.TestPrice(@p)=1
  insert into...
else print 'no insertion..'
```

Function that add a constraint with check

```
CREATE TABLE CheckTbl (col1 int, col2 int);
GO
CREATE FUNCTION CheckFnctn()
RETURNS int
AS
BEGIN
  DECLARE @retval int
  SELECT @retval = COUNT(*) FROM CheckTbl
  RETURN @retval
END;
GO
ALTER TABLE CheckTbl
ADD CONSTRAINT chkRowCount CHECK (dbo.CheckFnctn() >= 1 );
GO
```

The validation functions can check a variable to be in a specified interval, a varchar to have a specified length, a specified varchar to have just letters, a date to be smaller/greater than the current date (e.g. expiration_date>GETDATE()), the e-mail address to include the character '@', and so on…

Stored Procedure with INPUT/OUTPUT parameters

```
CREATE PROCEDURE test_InsertShops
@flag bit OUTPUT,-- return 0 for fail,1 for success
@Name varchar(50),
@City varchar(100),
@NoOfShops int,
@NoOfClients int
AS
BEGIN
 Insert into Shops(Name, City, NoOfShops, NoOfClients) Values(@Name, @City, @NoOfShops, @NoOfClients)
 IF @@TRANCOUNT > 0 SET @flag=1;
 ELSE SET @flag=0;
END
```

```
--Execute above created procedure to insert rows into table
Declare @flag bit
EXEC test_InsertShops @flag OUTPUT,'Shop 1','Bucuresti', 14, 12
if @flag=1 print 'Successfully inserted'
else print 'There is some error'
```

```
(1 row(s) affected)
There is some error
```

The parameters of the stored procedures must be parameters that refers to the columns from the table considered and also others.

Insert - can be performed for one row or multiple rows.

Select – a simple select or a complex one, that can be also saved in a table-valued function, or a view, or a stored procedure and returned from there.

Update – particular one on the fields of the table. NOT on the primary key. If the update is on the foreign key, first must be checked the existence of the new foreign key in the table in which is primary key. Update is performed for less records, maybe a group, but not very often. – preferable with WHERE condition. The update SHOULD NOT BE PERFORMED for the PRIMARY KEY. If in the intermediate table of the relation many to many are only the foreign keys sett it also to be primary key, nothing should be update (just a message will be ok).
Delete – particular one, for only some records or for all. The order of inserting is first the table in which is primary key, so that we can extract that primary key and set it to be foreign key, and only after the table with the foreign key. The order when we delete is the opposite one: first from the table that has the foreign key and then from the table that has the primary key. So, maybe when a delete from a table needs to be performed, other delete's will be need it to be performed (e.g. Delete from Shops may involve first the delete from ShopsTeas and only then the delete from Shops). – preferable with WHERE condition.

If consider proper, can be performed INSERT + SELECT + UPDATE (on the same record from insert – on a field) + SELECT (for double feedback) + DELETE (on the same record from insert and update) + SELECT (for double feedback).

Constraint(s) per table/column related to the validation of the data, can refer to primary key, foreign key, or other constraints.

PRIMARY KEY + FOREIGN KEY – can be threated as follows:
- If are given as parameters – MUST be VALIDATED first
- If are not given as parameters – MUST be EXTRACTED from the corresponding tables.

PRIMARY KEY:
- If it is IDENTITY – nothing to be done – inserted automatically
- If it is NOT IDENTITY –
  - If it is given as PARAMETER – MUST be VALIDATED – TO NOT EXIST IN THAT TABLE (for example, a count can be performed for that primary key given as parameter and the count to be 0)
  - If it is NOT given as a parameter – the MAXIMUM value should be extracted and increased with 1, to get to a new value for the primary key (it should work also for the varchar type)

FOREIGN KEY:
- If it is given as PARAMETER – MUST be VALIDATED – TO EXIST IN THE TABLE IN WHICH IS PRIMARY KEY (for example, a count can be performed for that foreign key given as parameter in the table in which is primary key, and we check to be found there)
- If it is NOT given as a parameter – should be extracted from the table in which is primary key (the first value – top 1, minimum value, maximum value, average value, ..) and put it on the foreign key position – a variable can be used for this operation.

Validation:
IF validation is performed with success THEN INSERT …
ELSE PRINT 'Cannot be performed due to the validation'

Stored procedure with a function used to validate the parameters:

```
USE Example_Lab1
GO

ALTER PROCEDURE [dbo].[CRUD_Shops]
@table_name Varchar(50),
@name varchar(50),
@city varchar(50),
@nos int,
@noc int,
@noOfRows int
AS
BEGIN
SET NOCOUNT ON;
       -- verify the parameters - at least one from the list
       IF (dbo.TestPrice(@nos)=1 and dbo.TestPrice(@noc)=1)
       BEGIN
              -- CREATE=INSERT
              insert into Shops(Name, City, NoOfShops, NoOfClients) Values(@name, @city, @nos, @noc)
              -- READ=SELECT
              select * from Shops
              -- UPDATE
              update Shops set City='Cluj-Napoca'
              where NoOfClients>10 and Name LIKE 'A%'
              -- DELETE
              delete from Shops
              where NoOfClients=0
              print 'CRUD operations for table ' + @table_name
       END
       ELSE
       BEGIN
              PRINT 'Error'
              RETURN
       END
END
```

```
Stored procedure list_list_of_Parameters
Begin
        Validation ok – INSERT + SELECT + UPDATE + DELETE
        Validation NOT ok – message 'Not ok – cannot be performed'
End
```

OR

```
Stored procedure list_list_of_Parameters
Begin
        Validation ok – INSERT
        Validation NOT ok – message_Insert 'Not ok – cannot be performed'
        SELECT
        UPDATE
        DELETE
End
```

ORGANIZE THE STRUCTURE OF THE CRUD OPERATIONS:

- 3 stored procedures with INSERT + SELECT + UPDATE + DELETE – for every table separated (one stored procedure per table).
- Separate stored procedures for INSERT per each table, SELECT per each table, UPDATE per each table, DELETE per each table, AND a “union” procedure to execute CRUD per each table (15 stored procedures).
- INSERT stored procedure for all tables (with if for each table), SELECT stored procedure for all tables (with if for each table), UPDATE stored procedure for all tables (with if for each table), DELETE stored

procedure for all tables (with if for each table), AND a "union" procedure to execute CRUD per each table

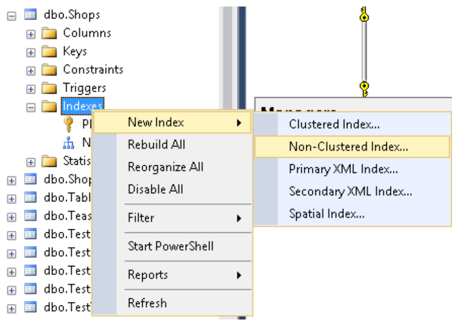
- A single stored procedure with a lot of if's and then correspondingly to each table take from there.
- Or, any other idea.

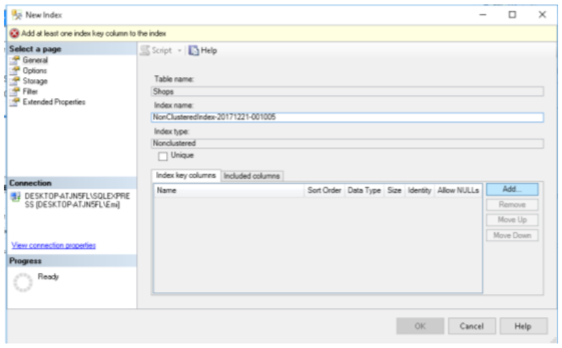
Also, please, prepare EXECUTION scenarios – with success and without success for each table.

Views - Must be created on the tables used for the CRUD operations and be relevant and also to include ALL the tables used in the stored procedures (for example, view_1 uses 2 tables and view_2 uses 3 tables, so, 5 tables).

Non-Clustered Indexes

- *with Object Explorer:* Database -> Tables -> expand the table used to create a non-clustered index -> Right-click the Indexes -> New Index -> select Non-Clustered Index…





Add at least one index key column to the index

Table name: Shops

Index name: NonClusteredIndex-20171221-001005

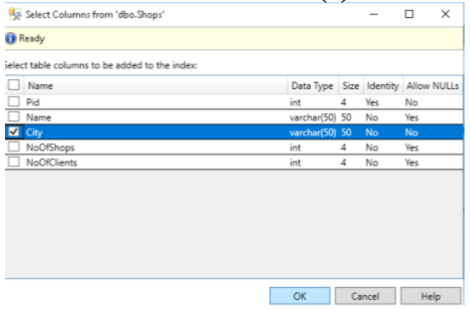
Index type: Nonclustered

☐ Unique

DESKTOP-ATJN5FL\SQLEXPRESS [DESKTOP-ATJN5FL\Emi]

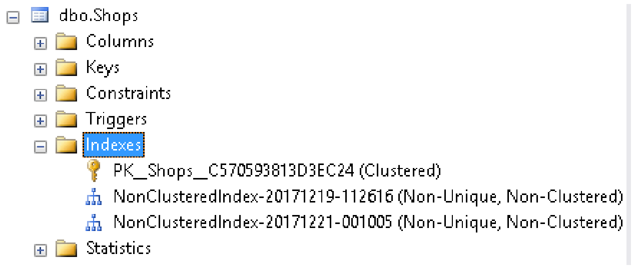
-> Index name box (=enter the name of the new index)

-> Add… -> check the column(s) that will be included in the nonclustered index ->Ok -> Ok.



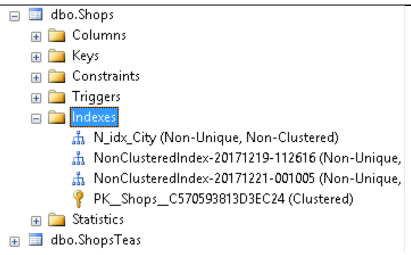
Select table columns to be added to the index:

	Name	Data Type	Size	Identity	Allow NULLs
☐	Pid	int	4	Yes	No
☐	Name	varchar(50)	50	No	Yes
✓	City	varchar(50)	50	No	No
☐	NoOfShops	int	4	No	Yes
☐	NoOfClients	int	4	No	Yes


dbo.Shops
Columns
Keys
Constraints
Triggers
Indexes
PK__Shops__C570593813D3EC24 (Clustered)
NonClusteredIndex-20171219-112616 (Non-Unique, Non-Clustered)
NonClusteredIndex-20171221-001005 (Non-Unique, Non-Clustered)
Statistics

- *with Transact-SQL:* New Query -> write the code -> Execute

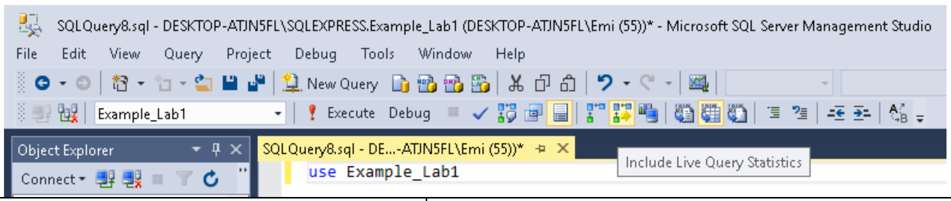
```
USE Example_Lab1
GO
-- Find an existing index named Nix_Name and delete it if found.
IF EXISTS (SELECT name FROM sys.indexes WHERE name = N'N_idx_City')
    DROP INDEX N_idx_City ON Shops;
GO
-- Create a nonclustered index called N_idx_City on the Shops table
using the City column.
CREATE NONCLUSTERED INDEX N_idx_City ON Shops (City);
GO
```


dbo.Shops
Columns
Keys
Constraints
Triggers
Indexes
N_idx_City (Non-Unique, Non-Clustered)
NonClusteredIndex-20171219-112616 (Non-Unique,
NonClusteredIndex-20171221-001005 (Non-Unique,
PK__Shops__C570593813D3EC24 (Clustered)
Statistics
dbo.ShopsTeas

Check Clustered / Non-Clustered Indexes (Instead of Dynamic Management Views and Functions)

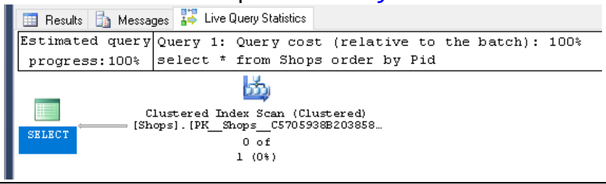
Check the indexes – check **Include Live Query Statistics -** when run a query.

After an update / order by / …, the order of the records is changed. For example, the indexes become 'un-ordered' (1, 2, 3, … -> 3, 1, 2, …). To choose the 'best' index, verify with the menu Include Live Query Statistics.


SQLQuery8.sql - DESKTOP-ATJN5FL\SQLEXPRESS.Example_Lab1 (DESKTOP-ATJN5FL\Emi (55))* - Microsoft SQL Server Management Studio
File
Edit
View
Query
Project
Debug
Tools
Window
Help
New Query
Example_Lab1
Execute
Debug
Object Explorer
Connect
SQLQuery8.sql - DE...-ATJN5FL\Emi (55))*
Include Live Query Statistics

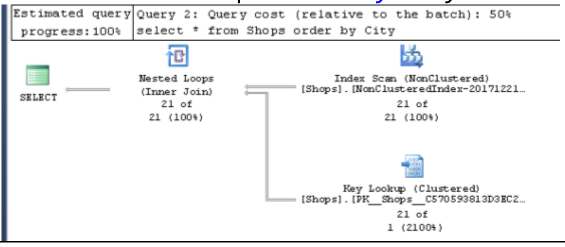
```
use Example_Lab1
```

```
-- check Include Live Query Statistics
-- on Pid (the primary key) there is a
clustered index
select * from Shops order by Pid
```


Results
Messages
Live Query Statistics
SELECT
Clustered Index Scan (Clustered)
[Shops].[PK__Shops__C5705938B203858…
0 of
1 (0%)

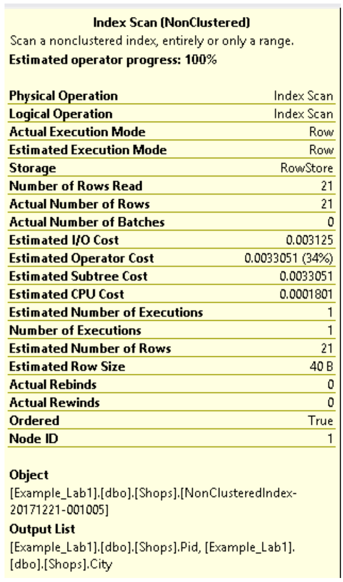
```
Estimated query progress:100%
Query 1: Query cost (relative to the batch): 100%
select * from Shops order by Pid
```

```
-- on City I have a nonclustered index
select * from Shops order by City
```


SELECT
Nested Loops
(Inner Join)
21 of
21 (100%)
Index Scan (NonClustered)
[Shops].[NonClusteredIndex-20171221…
21 of
21 (100%)
Key Lookup (Clustered)
[Shops].[PK__Shops__C570593813D3EC2…
21 of
1 (2100%)

```
Estimated query progress:100%
Query 2: Query cost (relative to the batch): 50%
select * from Shops order by City
```

By moving the mouse through the indexes, one can check the properties…



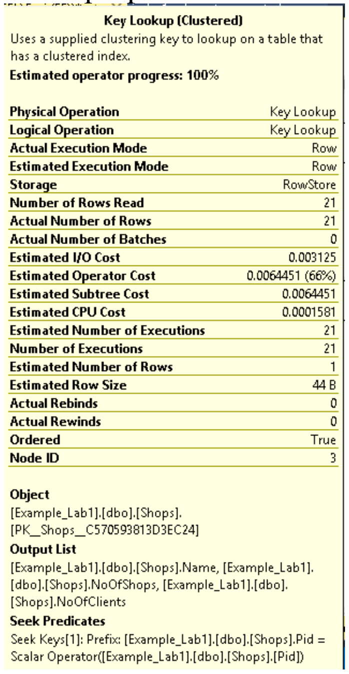
Index Scan (NonClustered)

Scan a nonclustered index, entirely or only a range.

Estimated operator progress: 100%

Physical Operation	Index Scan
Logical Operation	Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	21
Actual Number of Rows	21
Actual Number of Batches	0
Estimated I/O Cost	0.003125
Estimated Operator Cost	0.0033051 (34%)
Estimated Subtree Cost	0.0033051
Estimated CPU Cost	0.0001801
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	21
Estimated Row Size	40 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	1

Object
[Example_Lab1].[dbo].[Shops].[NonClusteredIndex-20171221-001005]
Output List
[Example_Lab1].[dbo].[Shops].Pid, [Example_Lab1].[dbo].[Shops].City



Key Lookup (Clustered)

Uses a supplied clustering key to lookup on a table that has a clustered index.

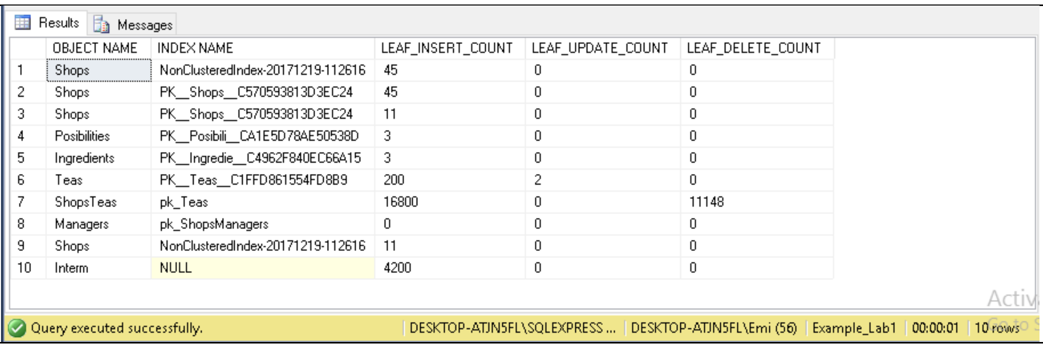
Estimated operator progress: 100%

Physical Operation	Key Lookup
Logical Operation	Key Lookup
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Number of Rows Read	21
Actual Number of Rows	21
Actual Number of Batches	0
Estimated I/O Cost	0.003125
Estimated Operator Cost	0.0064451 (66%)
Estimated Subtree Cost	0.0064451
Estimated CPU Cost	0.0001581
Estimated Number of Executions	21
Number of Executions	21
Estimated Number of Rows	1
Estimated Row Size	44 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	True
Node ID	3

Object
[Example_Lab1].[dbo].[Shops].[PK__Shops__C570593813D3EC24]
Output List
[Example_Lab1].[dbo].[Shops].Name, [Example_Lab1].[dbo].[Shops].NoOfShops, [Example_Lab1].[dbo].[Shops].NoOfClients
Seek Predicates
Seek Keys[1]: Prefix: [Example_Lab1].[dbo].[Shops].Pid = Scalar Operator([Example_Lab1].[dbo].[Shops].[Pid])

Check all indexes and some other properties (leaf number after Insert, update, delete)

```
use Example_Lab1
GO
SELECT OBJECT_NAME(A.[OBJECT_ID]) AS [OBJECT NAME],
       I.[NAME] AS [INDEX NAME],
       A.LEAF_INSERT_COUNT,
       A.LEAF_UPDATE_COUNT,
       A.LEAF_DELETE_COUNT
FROM   SYS.DM_DB_INDEX_OPERATIONAL_STATS (NULL,NULL,NULL,NULL ) A INNER JOIN SYS.INDEXES AS I
         ON I.[OBJECT_ID] = A.[OBJECT_ID] AND I.INDEX_ID = A.INDEX_ID
WHERE  OBJECTPROPERTY(A.[OBJECT_ID],'IsUserTable') = 1
```



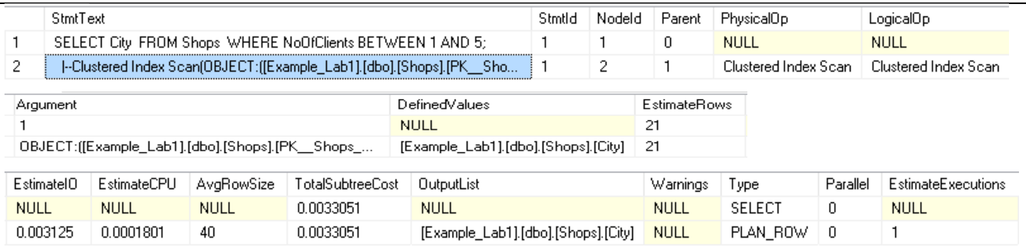
Results | Messages

	OBJECT NAME	INDEX NAME	LEAF_INSERT_COUNT	LEAF_UPDATE_COUNT	LEAF_DELETE_COUNT
1	Shops	NonClusteredIndex-20171219-112616	45	0	0
2	Shops	PK__Shops__C570593813D3EC24	45	0	0
3	Shops	PK__Shops__C570593813D3EC24	11	0	0
4	Posibilities	PK__Posibili__CA1E5D78AE50538D	3	0	0
5	Ingredients	PK__Ingredie__C4962F840EC66A15	3	0	0
6	Teas	PK__Teas__C1FFD861554FD8B9	200	2	0
7	ShopsTeas	pk_Teas	16800	0	11148
8	Managers	pk_ShopsManagers	0	0	0
9	Shops	NonClusteredIndex-20171219-112616	11	0	0
10	Interm	NULL	4200	0	0

```
Query executed successfully. | DESKTOP-ATJN5FL\SQLEXPRESS ... | DESKTOP-ATJN5FL\Emi (56) | Example_Lab1 | 00:00:01 | 10 rows
```

Show index plan

```
use Example_Lab1
go
SET NOCOUNT ON;
GO
SET SHOWPLAN_ALL ON;
GO
SELECT City
FROM Shops
WHERE NoOfClients BETWEEN 1
AND 5;
GO
SET SHOWPLAN_ALL OFF;
```



	StmtText	StmtId	NodeId	Parent	PhysicalOp	LogicalOp
1	SELECT City FROM Shops WHERE NoOfClients BETWEEN 1 AND 5;	1	1	0	NULL	NULL
2	|--Clustered Index Scan(OBJECT:([Example_Lab1].[dbo].[Shops].[PK__Sho...	1	2	1	Clustered Index Scan	Clustered Index Scan

Argument	DefinedValues	EstimateRows
1	NULL	21
OBJECT:([Example_Lab1].[dbo].[Shops].[PK__Shops_...	[Example_Lab1].[dbo].[Shops].[City]	21

EstimateIO	EstimateCPU	AvgRowSize	TotalSubtreeCost	OutputList	Warnings	Type	Parallel	EstimateExecutions
NULL	NULL	NULL	0.0033051	NULL	NULL	SELECT	0	NULL
0.003125	0.0001801	40	0.0033051	[Example_Lab1].[dbo].[Shops].[City]	NULL	PLAN_ROW	0	1