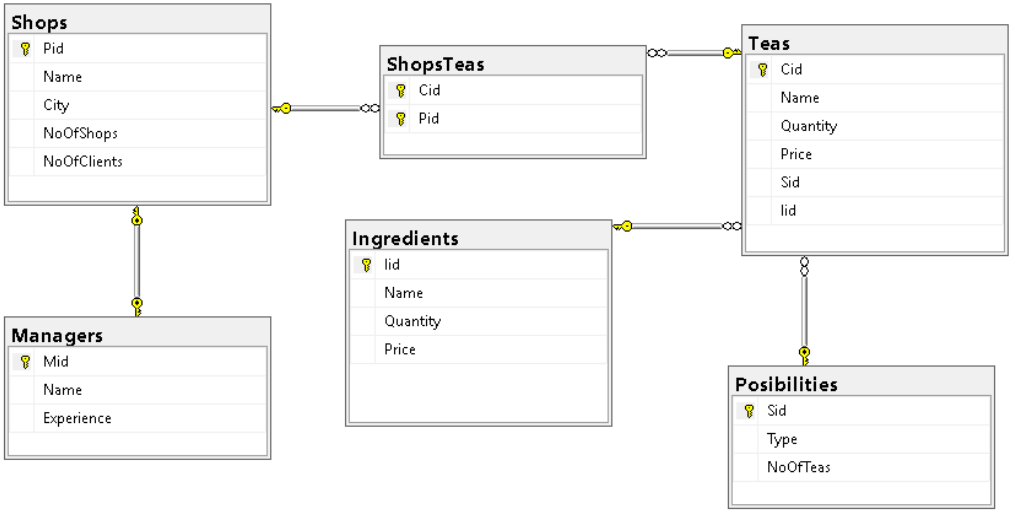
Laboratory 5

CRUD Operations

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

In this laboratory, one has to create stored procedures for CRUD operations for 5 tables (you will have 5 stored procedures and a main procedure, or 4 operations*5 tables =20 stored procedures, or …).

We consider the database



Here, for example, one 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),
```

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

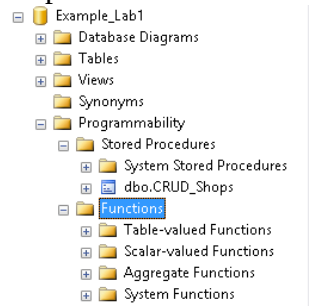
```
        @nos int,
        @noc int,
        @noOfRows int
AS
BEGIN
        SET NOCOUNT ON;
        -- verify the parameters - at least one from the list -
        -- with the help of a scalar function or a stored procedure with output parameter

        -- 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

        -- 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
```

Results | Messages

	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

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*

```sql
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
```

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

Results | Messages

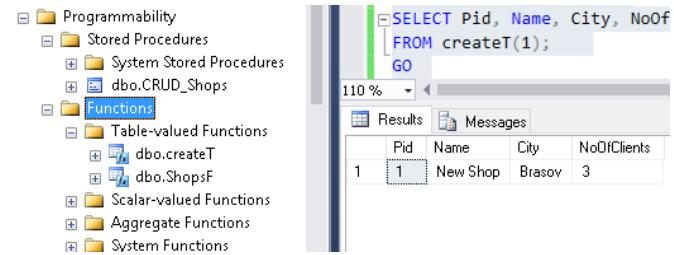
	Name	Total number of clients
1	New Shop	3

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

- *multi-statement Table-Valued Functions – RETURNS a TABLE*

```sql
USE Example_Lab1
IF OBJECT_ID (N'dbo.createT', N'TF') IS NOT NULL
    DROP FUNCTION dbo.createT;
GO
CREATE FUNCTION dbo.createT (@Pid int)
RETURNS @rett TABLE
(
    Pid int primary key,
    Name varchar(50) NOT NULL,
    City varchar(50) NOT NULL,
       NoOfClients int
)
--Returns a result set that lists the shops with the Pid given
AS
BEGIN
WITH IntermediateT(Pid, Name, City, NoOfClients)
-- IntermediateT - the Intermediate table for Shops -  name and columns
    AS (
        -- Get the initial list of Shops
        SELECT Pid, Name, City, NoOfClients
        FROM Shops
        WHERE Pid=@Pid
       )
-- copy the required columns to the result of the function
   INSERT @rett
```

```sql
-- Example execute
SELECT Pid, Name, City, NoOfClients
FROM createT(1);
GO
```



```
    SELECT Pid, Name, City, NoOfClients
    FROM IntermediateT
    RETURN
END;
GO
```

- *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)
```

Results | Messages

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

Email Validation: This function below basically checks for Nulls, invalid or duplicate characters, and looks for '@' and '.' in the formatting.

```
-- scalar-valued function
 Create FUNCTION udf_ValidateEmail (@email varChar(255))
RETURNS bit
AS
begin
return
(
select
        Case
                When    @Email is null then 0                   --NULL Email is invalid
                When    charindex(' ', @email)      <> 0 or     --Check for invalid character
                        charindex('/', @email)      <> 0 or --Check for invalid character
                        charindex(':', @email)      <> 0 or --Check for invalid character
```

```
-- Execution
select
dbo.udf_ValidateEmail('myemailaddress@somedomain.com')
```

Results | Messages

	(No column name)
1	1

```
                                        charindex(';', @email)        <> 0 then 0 --Check for invalid character
                    When len(@Email)-1 <= charindex('.', @Email) then 0--check for '%._' at end of string
                    When     @Email like '%@%@%'or
                                        @Email Not Like '%@%.%'  then 0--Check for duplicate @ or invalid format
                    Else 1
          END
)
end
```

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)
```

Results | Messages

	(No column name)
1	0

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
```

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

2. Another example of crud operation on a table with primary key and foreign keys

```
USE [Example_Lab1]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[CRUD_Teas]
       @table_name Varchar(50),
       @name varchar(50),
       @quantity int,
       @price int,
       @noOfRows int
AS
BEGIN
       SET NOCOUNT ON;
       -- MUST 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
       set @n=@noOfRows
       -- we add as many rows as the parameter indicate us
       -- not all the fields must be given as parameters
       declare @fks int
       set @fks = (select MAX(Sid) from Posibilities)
       declare @fki int
       set @fki = (select MIN(Iid) from Ingredients)
       -- before, please check here if you have records in tables Posibilities and Ingredients
       while @n>=1 begin
              insert into Teas(Name, Quantity, Price, Sid, Iid)
              Values(@name, @quantity, @price, @fks, @fki)
              set @n=@n-1
       end
```

```
-- execute
EXEC CRUD_Teas 'Teas',
'Mint', 3, 8, 100
```

Results | Messages

	Cid	Name	Quantity	Price	Sid	Iid
122	1...	Mint	3	8	3	1
123	1...	Mint	3	8	3	1
124	1...	Mint	3	8	3	1
125	1...	Mint	3	8	3	1
126	1...	Mint	3	8	3	1
127	1...	Mint	3	8	3	1
128	1...	Mint	3	8	3	1
129	1...	Mint	3	8	3	1
130	1...	Mint	3	8	3	1
131	1...	Mint	3	8	3	1
132	1...	Mint	3	8	3	1

Query executed successfully. DESKTO

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

        -- UPDATE
        update Teas set Price=Price+10 where Cid=2

        -- DELETE
        delete from ShopsTeas
        delete from Teas where Name='Cherry'

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

3. Another example of crud operation on a table with 2 primary keys that are also foreign keys – only a solution with cross join, or you can try with while…

```
-- ShopsTeas
-- to insert all the possible records in ShopsTeas
-- you can go with while and take all from Shops and Teas or with cross join or...

delete from ShopsTeas

if OBJECT_ID('Interm', 'U') IS NOT NULL
        drop table Interm

select Cid, Pid into Interm from Shops CROSS JOIN Teas

insert into ShopsTeas
select Cid, Pid from Interm

select * from ShopsTeas
```

Messages

(0 row(s) affected)

(4200 row(s) affected)

(4200 row(s) affected)

110 %

Query executed successfully.

Results | Messages

	Cid	Pid
1	1	1
2	1	2
3	1	3
4	1	4
5	1	5
6	1	6
7	1	7
8	1	8
9	1	9
10	1	10
11	1	11

Query executed successfully. | DESKTOP-ATJN5FL\SQLEXPRESS ... | DESKTOP-ATJN5FL\Emi (55) | Example_Lab1 | 00:00:00 | 4200 rows

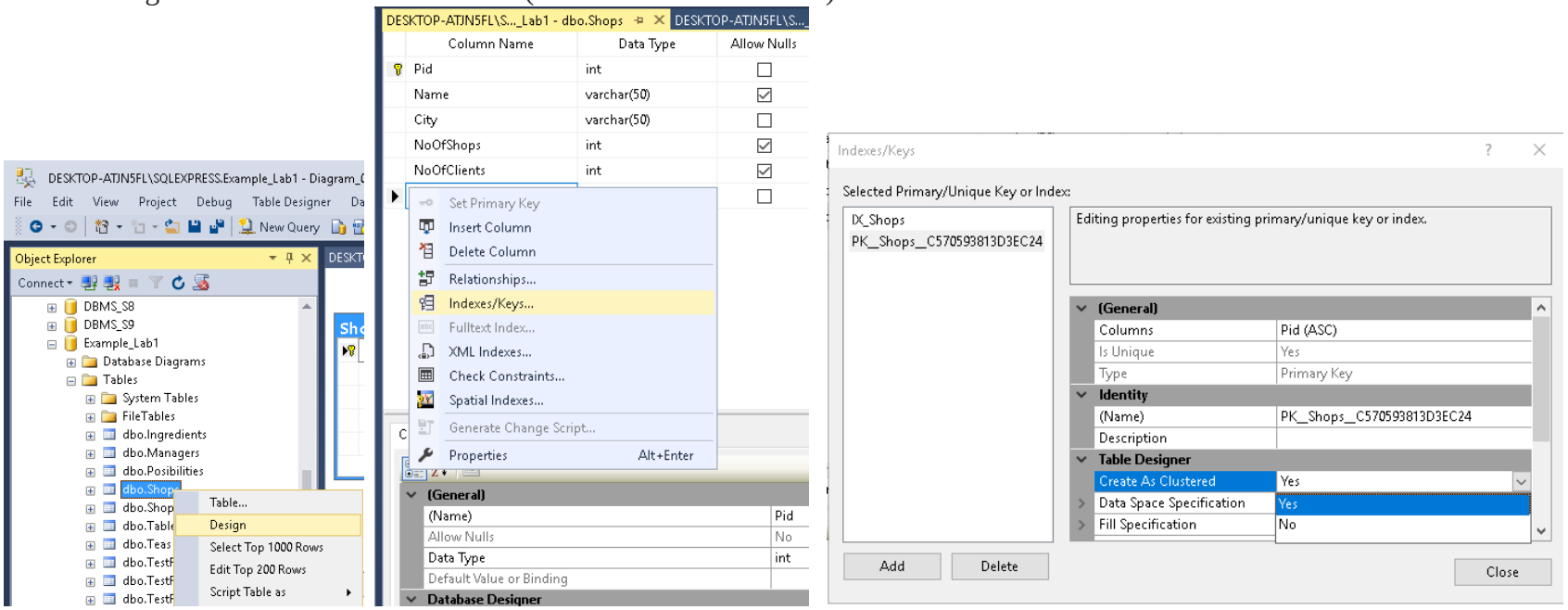
Views

Must be created on the tables used for the CRUD operations and be relevant.

Non-Clustered Indexes

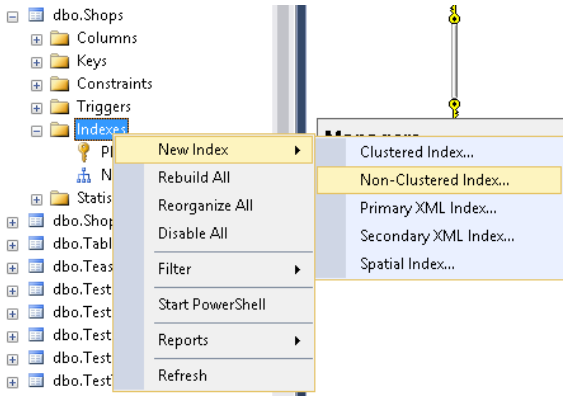
To create a nonclustered index by using the Table Designer

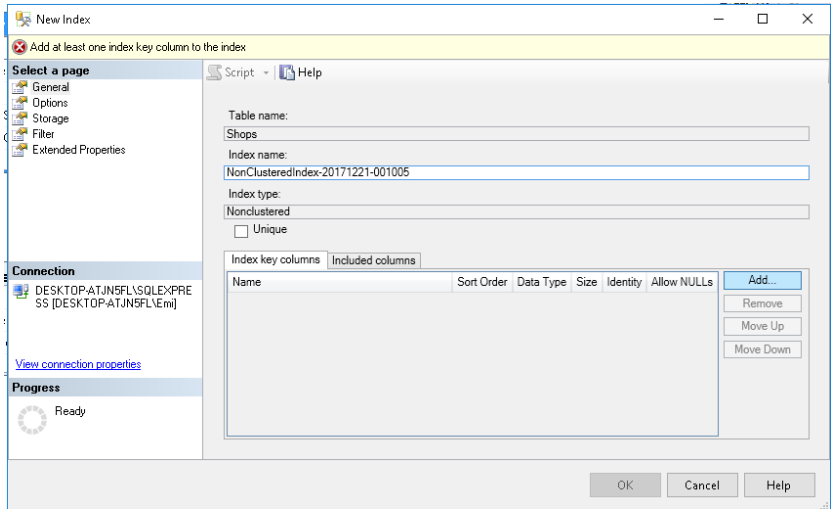
- Choose the database -> Tables -> right click on the table that will be used to create a non-clustered index -> Design -> Indexes/Keys -> Add
- Select the new index in the Selected Primary/Unique Key or Index text box.
- In the grid -> Create as Clustered: No (for nonclustered indexes) -> Close -> Save table



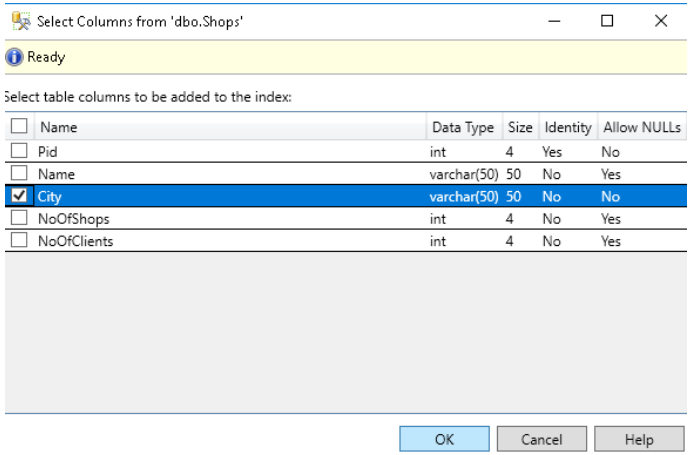
To create a nonclustered index by using Object Explorer

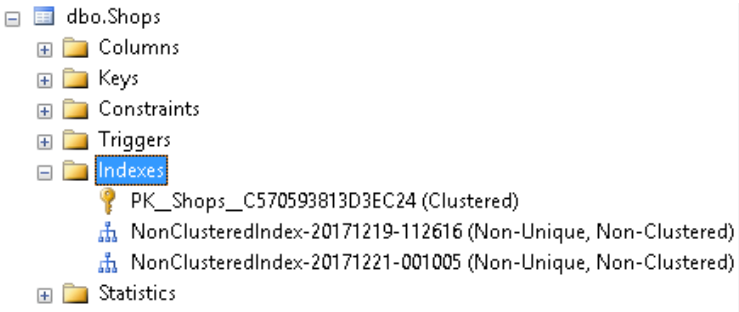
- Choose the database -> Tables (folder) -> expand the table that will be used to create a non-clustered index
- Right-click the Indexes folder -> New Index -> select Non-Clustered Index…





- In the New Index dialog box -> General page -> Index name box (=enter the name of the new index)
- Under Index key columns -> click Add…
- In the Select Columns from table_name dialog box -> select the check box(es) of the table column(s) to be added to the nonclustered index ->Ok -> Ok.

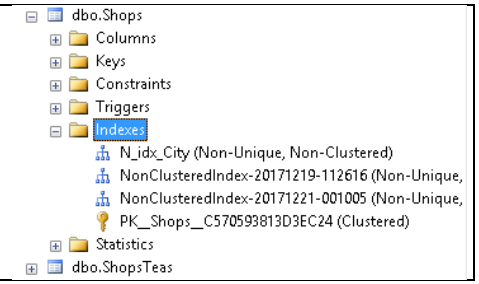




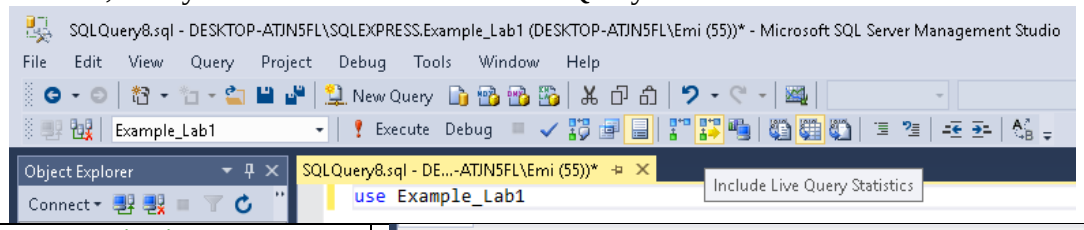
To create a nonclustered index on a table by using Transact-SQL

- Choose the database -> New Query -> write the code -> Execute

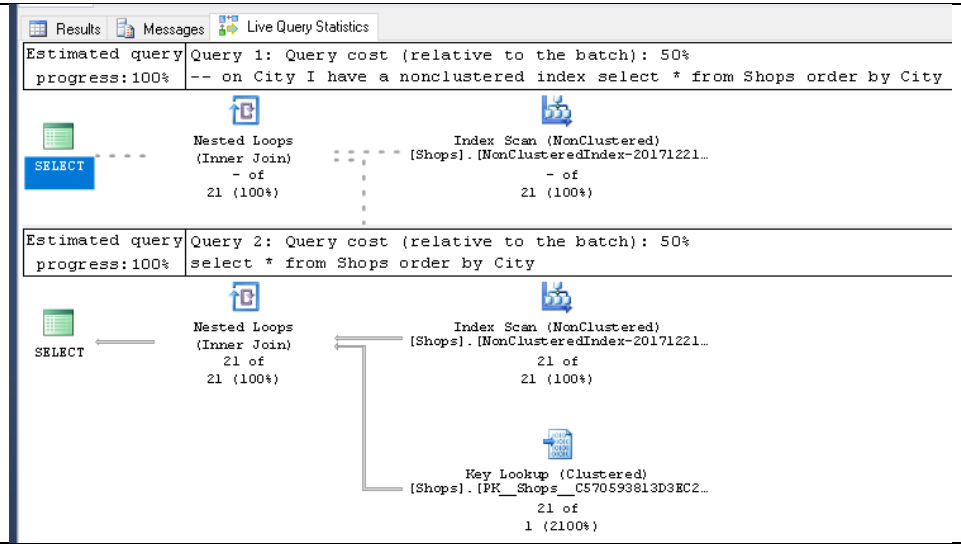
```
USE Example_Lab1
GO
-- Find an existing index named Nix_Name and delete it if found.
IF EXISTS (SELECT name FROM sys.indexes WHERE name = N'N_idx_City')
    DROP INDEX N_idx_City ON Shops;
GO
-- Create a nonclustered index called N_idx_City on the Shops table using the City column.
CREATE NONCLUSTERED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



Check Clustered / Non-Clustered Indexes (Instead of Dynamic Management Views and Functions)
Check the indexes – check **Include Live Query Statistics -** when a query is run.
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.




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

-- on City I have a nonclustered index
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.

Physical Operation	Index Scan
Logical Operation	Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0.0033051 (34%)
Estimated I/O Cost	0.003125
Estimated Subtree Cost	0.0033051
Estimated CPU Cost	0.0001801
Estimated Number of Executions	1
Estimated Number of Rows	21
Estimated Row Size	40 B
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

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

Check the user_scans, user_seeks, user_updates, …

```
use Example_Lab1
GO
select i2.name, i1.user_scans, i1.user_seeks, i1.user_updates,i1.last_user_scan,i1.last_user_seek, i1.last_user_update
from sys.dm_db_index_usage_stats i1 inner join sys.indexes i2 on i1.index_id = i2.index_id
where i1.object_id = OBJECT_ID('Shops') and i1.object_id = i2.object_id
```

Results | Messages

	name	user_scans	user_seeks	user_updates	last_user_scan	last_user_seek	last_user_update
1	NonClusteredIndex-20171221-001005	2	0	0	2017-12-21 00:23:24.550	NULL	NULL
2	NonClusteredIndex-20171219-112616	6	0	12	2017-12-20 23:55:37.790	NULL	2017-12-20 18:43:53.827
3	PK__Shops__C570593813D3EC24	10	4	13	2017-12-21 00:23:24.547	2017-12-20 18:15:46.333	2017-12-20 18:43:53.827

Query executed successfully. | DESKTOP-ATJN5FL\SQLEXPRESS ... | DESKTOP-ATJN5FL\Emi (56) | Example_Lab1 | 00:00:00 | 3 rows