DATABASES PROJECT 2023/2024

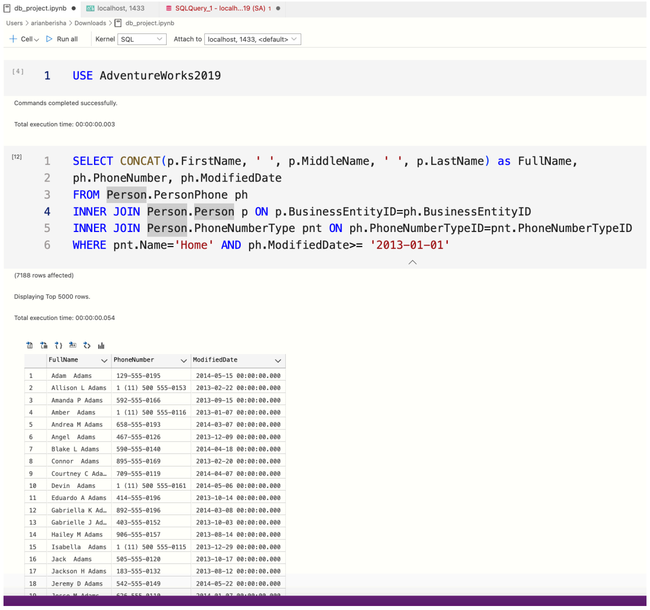
In this Advanced Database Management System project, we are going to show why indexes are important and how they fasten the process. Explores the power of triggers in automating actions based on database events.
We will first start with DML, Triggers & Transactions, and last but not least most important part Indexes.

We will be using an already prepared database AdventureWorks2019.

I. DATA MANIPULATION LANGUAGE

DML statements are responsible for performing operations on the data itself, such as querying, inserting, updating, and deleting records within database tables.

Problem 1: Retrive full names of all persons with their phone numbers, whose phone number types belongs to Home and last update is 2013 and further.


db_project.ipynb ●
localhost, 1433
SQLQuery_1 - localh...19 (SA) 1 ●
Users > arianberisha > Downloads > db_project.ipynb
+ Cell
▷ Run all
Kernel SQL
Attach to localhost, 1433, <default>

[4]

```sql
USE AdventureWorks2019
```

```
Commands completed successfully.

Total execution time: 00:00:00.003
```

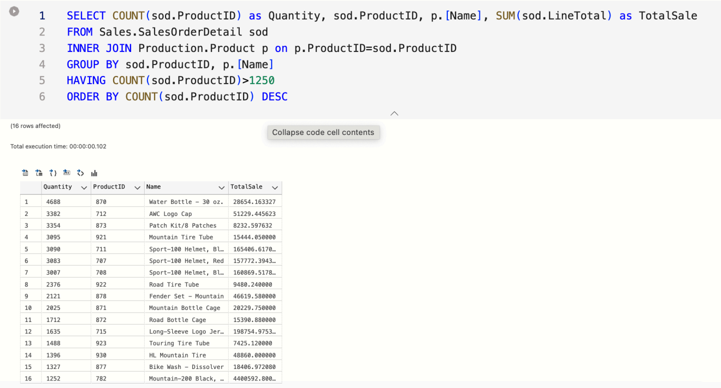
[12]

```sql
SELECT CONCAT(p.FirstName, ' ', p.MiddleName, ' ', p.LastName) as FullName,
ph.PhoneNumber, ph.ModifiedDate
FROM Person.PersonPhone ph
INNER JOIN Person.Person p ON p.BusinessEntityID=ph.BusinessEntityID
INNER JOIN Person.PhoneNumberType pnt ON ph.PhoneNumberTypeID=pnt.PhoneNumberTypeID
WHERE pnt.Name='Home' AND ph.ModifiedDate>= '2013-01-01'
```

```
(7188 rows affected)

Displaying Top 5000 rows.

Total execution time: 00:00:00.054
```

	FullName	PhoneNumber	ModifiedDate
1	Adam Adams	129-555-0195	2014-05-15 00:00:00.000
2	Allison L Adams	1 (11) 500 555-0153	2013-02-22 00:00:00.000
3	Amanda P Adams	592-555-0166	2013-09-15 00:00:00.000
4	Amber Adams	1 (11) 500 555-0116	2013-01-07 00:00:00.000
5	Andrea M Adams	658-555-0193	2014-03-07 00:00:00.000
6	Angel Adams	467-555-0126	2013-12-09 00:00:00.000
7	Blake L Adams	590-555-0140	2014-04-18 00:00:00.000
8	Connor Adams	895-555-0169	2013-02-20 00:00:00.000
9	Courtney C Ada…	709-555-0119	2014-04-07 00:00:00.000
10	Devin Adams	1 (11) 500 555-0161	2014-05-06 00:00:00.000
11	Eduardo A Adams	414-555-0196	2013-10-14 00:00:00.000
12	Gabriella K Ad…	892-555-0196	2014-03-08 00:00:00.000
13	Gabrielle J Ad…	403-555-0152	2013-10-03 00:00:00.000
14	Hailey M Adams	906-555-0157	2013-08-14 00:00:00.000
15	Isabella Adams	1 (11) 500 555-0115	2013-12-29 00:00:00.000
16	Jack Adams	505-555-0120	2013-10-17 00:00:00.000
17	Jackson H Adams	183-555-0132	2013-08-12 00:00:00.000
18	Jeremy D Adams	542-555-0149	2014-05-22 00:00:00.000

Problem 2: Get each product sold more than 1250 pieces and calculate the sum of those. Also order them in descending order of the quantity sold.



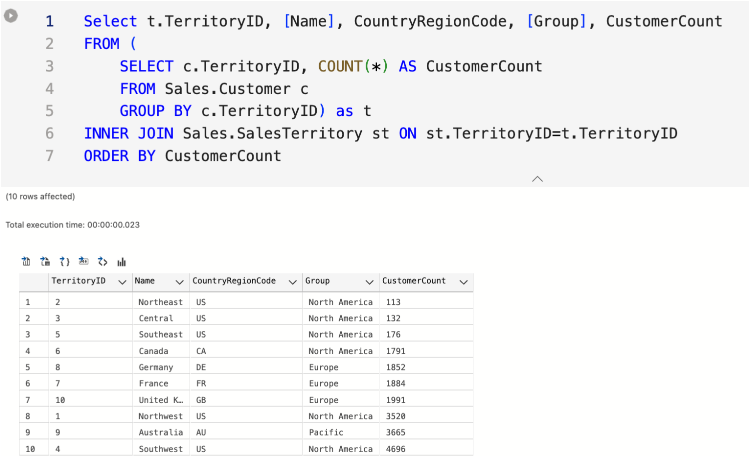
```sql
SELECT COUNT(sod.ProductID) as Quantity, sod.ProductID, p.[Name], SUM(sod.LineTotal) as TotalSale
FROM Sales.SalesOrderDetail sod
INNER JOIN Production.Product p on p.ProductID=sod.ProductID
GROUP BY sod.ProductID, p.[Name]
HAVING COUNT(sod.ProductID)>1250
ORDER BY COUNT(sod.ProductID) DESC
```

(16 rows affected)

Total execution time: 00:00:00.102

	Quantity	ProductID	Name	TotalSale
1	4688	870	Water Bottle – 30 oz.	28654.163327
2	3382	712	AWC Logo Cap	51229.445623
3	3354	873	Patch Kit/8 Patches	8232.597632
4	3095	921	Mountain Tire Tube	15444.050000
5	3090	711	Sport–100 Helmet, Bl…	165406.6170…
6	3083	707	Sport–100 Helmet, Red	157772.3943…
7	3007	708	Sport–100 Helmet, Bl…	160869.5178…
8	2376	922	Road Tire Tube	9480.240000
9	2121	878	Fender Set – Mountain	46619.580000
10	2025	871	Mountain Bottle Cage	20229.750000
11	1712	872	Road Bottle Cage	15390.880000
12	1635	715	Long–Sleeve Logo Jer…	198754.9753…
13	1488	923	Touring Tire Tube	7425.120000
14	1396	930	HL Mountain Tire	48860.000000
15	1327	877	Bike Wash – Dissolver	18406.972080
16	1252	782	Mountain–200 Black, …	4400592.800…

Problem 3: List the territory ID, Name, Country Region code, Group and how many customers each territory has.



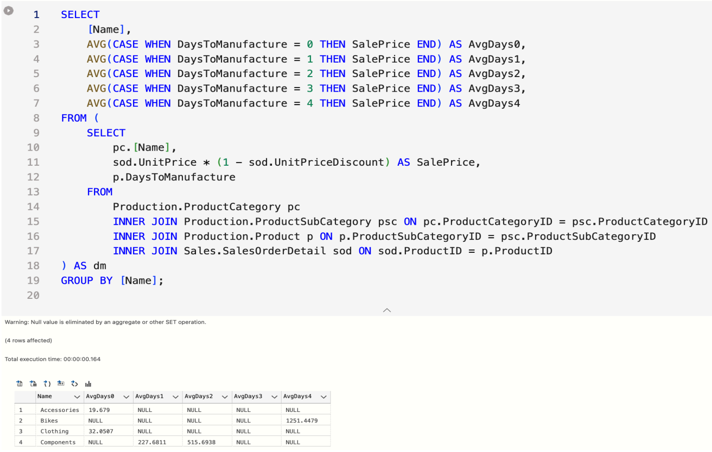
```sql
Select t.TerritoryID, [Name], CountryRegionCode, [Group], CustomerCount
FROM (
    SELECT c.TerritoryID, COUNT(*) AS CustomerCount
    FROM Sales.Customer c
    GROUP BY c.TerritoryID) as t
INNER JOIN Sales.SalesTerritory st ON st.TerritoryID=t.TerritoryID
ORDER BY CustomerCount
```

(10 rows affected)

Total execution time: 00:00:00.023

	TerritoryID	Name	CountryRegionCode	Group	CustomerCount
1	2	Northeast	US	North America	113
2	3	Central	US	North America	132
3	5	Southeast	US	North America	176
4	6	Canada	CA	North America	1791
5	8	Germany	DE	Europe	1852
6	7	France	FR	Europe	1884
7	10	United K…	GB	Europe	1991
8	1	Northwest	US	North America	3520
9	9	Australia	AU	Pacific	3665
10	4	Southwest	US	North America	4696

Problem 4: List each product category by name and the average sale price by days to manufacture. The days to manufacture (0, 1, 2, 3, 4) are to be headers across the top and product categories down the side. Sale price is the unit price * (1 - unit price discount)



```sql
SELECT
    [Name],
    AVG(CASE WHEN DaysToManufacture = 0 THEN SalePrice END) AS AvgDays0,
    AVG(CASE WHEN DaysToManufacture = 1 THEN SalePrice END) AS AvgDays1,
    AVG(CASE WHEN DaysToManufacture = 2 THEN SalePrice END) AS AvgDays2,
    AVG(CASE WHEN DaysToManufacture = 3 THEN SalePrice END) AS AvgDays3,
    AVG(CASE WHEN DaysToManufacture = 4 THEN SalePrice END) AS AvgDays4
FROM (
    SELECT
        pc.[Name],
        sod.UnitPrice * (1 - sod.UnitPriceDiscount) AS SalePrice,
        p.DaysToManufacture
    FROM
        Production.ProductCategory pc
        INNER JOIN Production.ProductSubCategory psc ON pc.ProductCategoryID = psc.ProductCategoryID
        INNER JOIN Production.Product p ON p.ProductSubCategoryID = psc.ProductSubCategoryID
        INNER JOIN Sales.SalesOrderDetail sod ON sod.ProductID = p.ProductID
) AS dm
GROUP BY [Name];

```

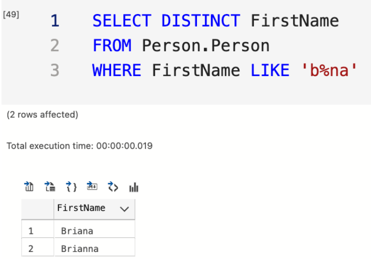
Warning: Null value is eliminated by an aggregate or other SET operation.

(4 rows affected)

Total execution time: 00:00:00.164

	Name	AvgDays0	AvgDays1	AvgDays2	AvgDays3	AvgDays4
1	Accessories	19.679	NULL	NULL	NULL	NULL
2	Bikes	NULL	NULL	NULL	NULL	1251.4479
3	Clothing	32.0507	NULL	NULL	NULL	NULL
4	Components	NULL	227.6811	515.6938	NULL	NULL

Problem 5: Return the records from the Person table where users first names end in 'na' and start with 'b'



[49]

```
SELECT DISTINCT FirstName
FROM Person.Person
WHERE FirstName LIKE 'b%na'
```

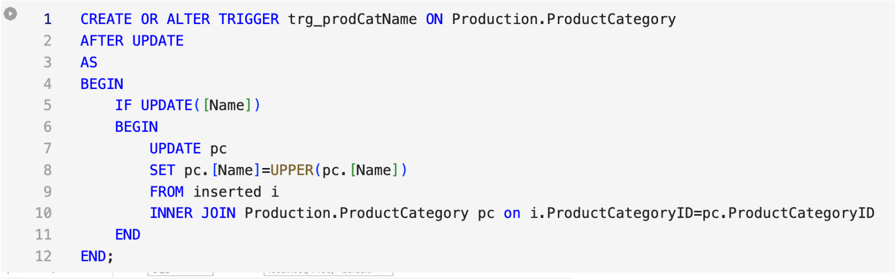
```
(2 rows affected)

Total execution time: 00:00:00.019
```

	FirstName
1	Briana
2	Brianna

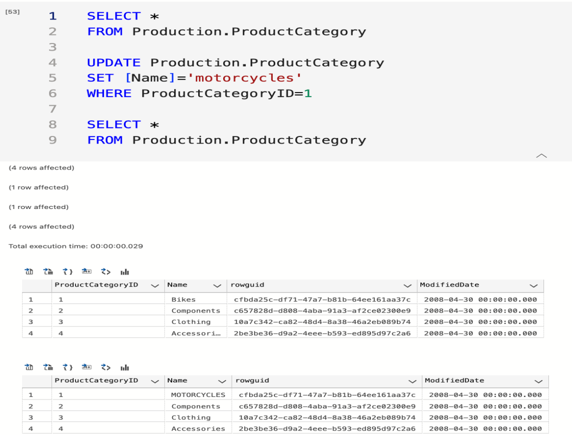
II. TRIGGERS AND TRANSACTIONS

Trigger is a set of instructions or code that is automatically executed ("triggered") in response to certain events on a particular table or view. These events include data manipulation language (DML) statements like INSERT, UPDATE, DELETE, or even a combination of these actions. Triggers are used to enforce business rules, maintain data integrity, and automate certain database-related tasks.
Whereas Transaction is a set of SQL statements that should be executed as one unit.
That means a transaction ensures that either all of the command succeeds or none of them.

Problem 6: When a ProductCategory name is updated, make sure it is saved in uppercases.



```
CREATE OR ALTER TRIGGER trg_prodCatName ON Production.ProductCategory
AFTER UPDATE
AS
BEGIN
    IF UPDATE([Name])
    BEGIN
        UPDATE pc
        SET pc.[Name]=UPPER(pc.[Name])
        FROM inserted i
        INNER JOIN Production.ProductCategory pc on i.ProductCategoryID=pc.ProductCategoryID
    END
END;
```



[53]

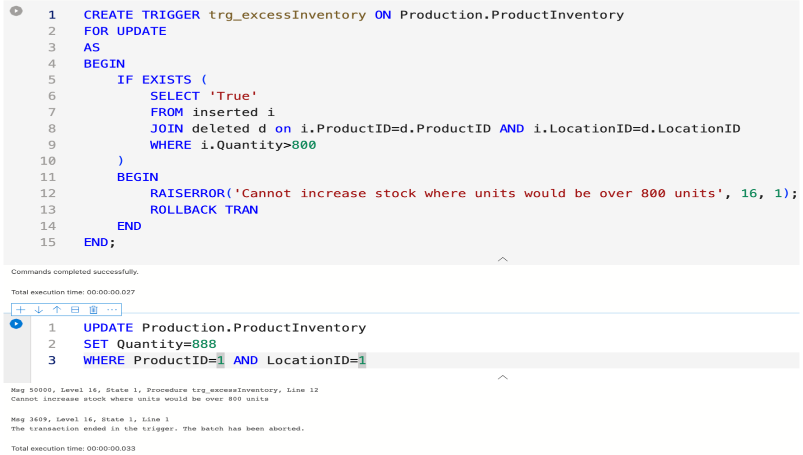
```
SELECT *
FROM Production.ProductCategory

UPDATE Production.ProductCategory
SET [Name]='motorcycles'
WHERE ProductCategoryID=1

SELECT *
FROM Production.ProductCategory
```

```
(4 rows affected)

(1 row affected)

(1 row affected)

(4 rows affected)

Total execution time: 00:00:00.029
```

	ProductCategoryID	Name	rowguid	ModifiedDate
1	1	Bikes	cfbda25c-df71-47a7-b81b-64ee161aa37c	2008-04-30 00:00:00.000
2	2	Components	c657828d-d808-4aba-91a3-af2ce02300e9	2008-04-30 00:00:00.000
3	3	Clothing	10a7c342-ca82-48d4-8a38-46a2eb089b74	2008-04-30 00:00:00.000
4	4	Accessori…	2be3be36-d9a2-4eee-b593-ed895d97c2a6	2008-04-30 00:00:00.000

	ProductCategoryID	Name	rowguid	ModifiedDate
1	1	MOTORCYCLES	cfbda25c-df71-47a7-b81b-64ee161aa37c	2008-04-30 00:00:00.000
2	2	Components	c657828d-d808-4aba-91a3-af2ce02300e9	2008-04-30 00:00:00.000
3	3	Clothing	10a7c342-ca82-48d4-8a38-46a2eb089b74	2008-04-30 00:00:00.000
4	4	Accessories	2be3be36-d9a2-4eee-b593-ed895d97c2a6	2008-04-30 00:00:00.000

Problem 7: Write a trigger for the ProductInventory table to make sure inventory cannot exceed 800 units when there is an update



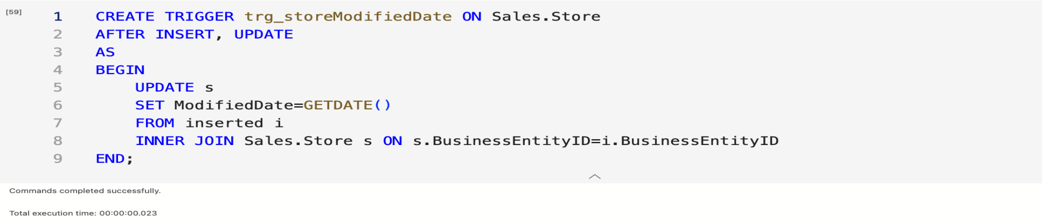
```
CREATE TRIGGER trg_excessInventory ON Production.ProductInventory
FOR UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 'True'
        FROM inserted i
        JOIN deleted d on i.ProductID=d.ProductID AND i.LocationID=d.LocationID
        WHERE i.Quantity>800
    )
    BEGIN
        RAISERROR('Cannot increase stock where units would be over 800 units', 16, 1);
        ROLLBACK TRAN
    END
END;
```

```
Commands completed successfully.

Total execution time: 00:00:00.027
```

```
UPDATE Production.ProductInventory
SET Quantity=888
WHERE ProductID=1 AND LocationID=1
```

```
Msg 50000, Level 16, State 1, Procedure trg_excessInventory, Line 12
Cannot increase stock where units would be over 800 units

Msg 3609, Level 16, State 1, Line 1
The transaction ended in the trigger. The batch has been aborted.

Total execution time: 00:00:00.033
```

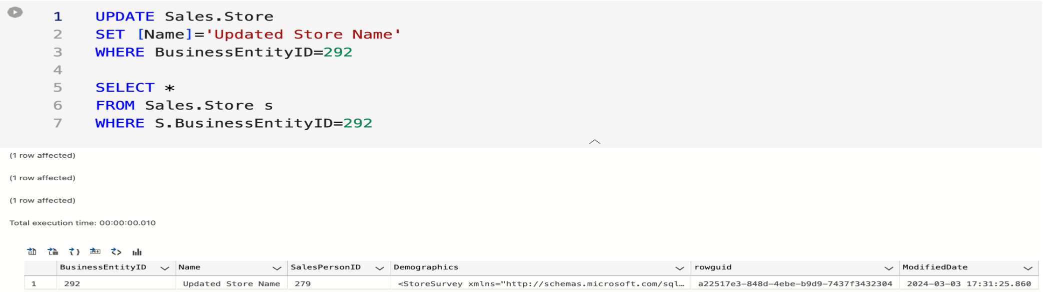
Problem 8: Update the ModifiedDate column of Sales.Store table whenever a record is inserted or updated.



[59]

```
CREATE TRIGGER trg_storeModifiedDate ON Sales.Store
AFTER INSERT, UPDATE
AS
BEGIN
    UPDATE s
    SET ModifiedDate=GETDATE()
    FROM inserted i
    INNER JOIN Sales.Store s ON s.BusinessEntityID=i.BusinessEntityID
END;
```

```
Commands completed successfully.

Total execution time: 00:00:00.023
```

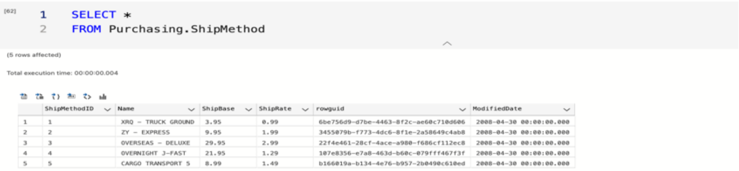


```
UPDATE Sales.Store
SET [Name]='Updated Store Name'
WHERE BusinessEntityID=292

SELECT *
FROM Sales.Store s
WHERE S.BusinessEntityID=292
```

```
(1 row affected)

(1 row affected)

(1 row affected)

Total execution time: 00:00:00.010
```

	BusinessEntityID	Name	SalesPersonID	Demographics	rowguid	ModifiedDate
1	292	Updated Store Name	279	<StoreSurvey xmlns="http://schemas.microsoft.com/sql…	a22517e3-848d-4ebe-b9d9-7437f3432304	2024-03-03 17:31:25.860

Problem 9: Using transactions insert a new record to Purchasing.ShipMethod, update another existing record and then commit the changes if there is no error occured, otherwice rollback to maintain data consistency.

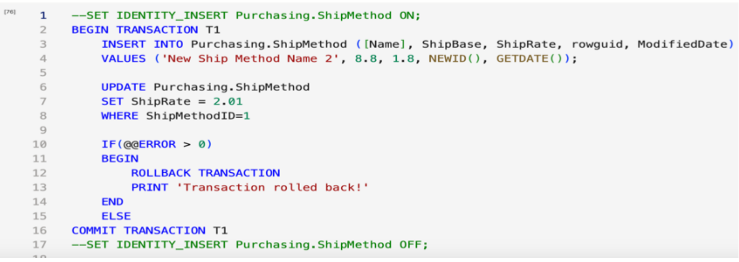


[62]

```
SELECT *
FROM Purchasing.ShipMethod
```

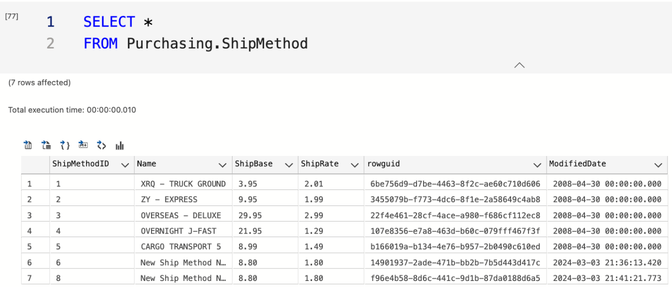
```
(5 rows affected)

Total execution time: 00:00:00.004
```

	ShipMethodID	Name	ShipBase	ShipRate	rowguid	ModifiedDate
1	1	XRQ - TRUCK GROUND	3.95	0.99	6be756d9-d7be-4463-8f2c-ae60c710d606	2008-04-30 00:00:00.000
2	2	ZY - EXPRESS	9.95	1.99	3455079b-f773-4dc6-8f1e-2a58649c4ab8	2008-04-30 00:00:00.000
3	3	OVERSEAS - DELUXE	29.95	2.99	22f4e461-28cf-4ace-a980-f686cf112ec8	2008-04-30 00:00:00.000
4	4	OVERNIGHT J-FAST	21.95	1.29	107e8356-e7a8-463d-b60c-079fff467f3f	2008-04-30 00:00:00.000
5	5	CARGO TRANSPORT 5	8.99	1.49	b166019a-b134-4e76-b957-2b0490c610ed	2008-04-30 00:00:00.000



[76]

```
--SET IDENTITY_INSERT Purchasing.ShipMethod ON;
BEGIN TRANSACTION T1
    INSERT INTO Purchasing.ShipMethod ([Name], ShipBase, ShipRate, rowguid, ModifiedDate)
    VALUES ('New Ship Method Name 2', 8.8, 1.8, NEWID(), GETDATE());

    UPDATE Purchasing.ShipMethod
    SET ShipRate = 2.01
    WHERE ShipMethodID=1

    IF(@@ERROR > 0)
    BEGIN
        ROLLBACK TRANSACTION
        PRINT 'Transaction rolled back!'
    END
    ELSE
COMMIT TRANSACTION T1
--SET IDENTITY_INSERT Purchasing.ShipMethod OFF;
```



[77]

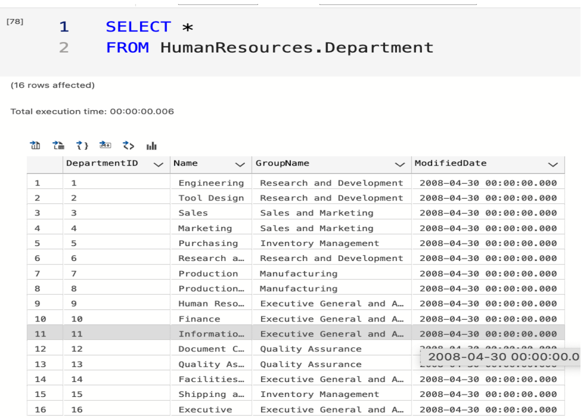
```sql
SELECT *
FROM Purchasing.ShipMethod
```

(7 rows affected)

Total execution time: 00:00:00.010

	ShipMethodID	Name	ShipBase	ShipRate	rowguid	ModifiedDate
1	1	XRQ – TRUCK GROUND	3.95	2.01	6be756d9-d7be-4463-8f2c-ae60c710d606	2008-04-30 00:00:00.000
2	2	ZY – EXPRESS	9.95	1.99	3455079b-f773-4dc6-8f1e-2a58649c4ab8	2008-04-30 00:00:00.000
3	3	OVERSEAS – DELUXE	29.95	2.99	22f4e461-28cf-4ace-a980-f686cf112ec8	2008-04-30 00:00:00.000
4	4	OVERNIGHT J-FAST	21.95	1.29	107e8356-e7a8-463d-b60c-079fff467f3f	2008-04-30 00:00:00.000
5	5	CARGO TRANSPORT 5	8.99	1.49	b166019a-b134-4e76-b957-2b0490c610ed	2008-04-30 00:00:00.000
6	6	New Ship Method N…	8.80	1.80	14901937-2ade-471b-bb2b-7b5d443d417c	2024-03-03 21:36:13.420
7	8	New Ship Method N…	8.80	1.80	f96e4b58-8d6c-441c-9d1b-87da0188d6a5	2024-03-03 21:41:21.773

Problem 10: Use nested transactions for HumanResources.Department on your purpose.



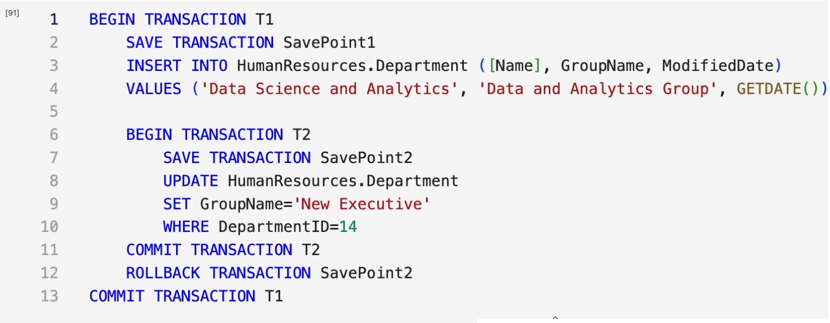
[78]

```sql
SELECT *
FROM HumanResources.Department
```

(16 rows affected)

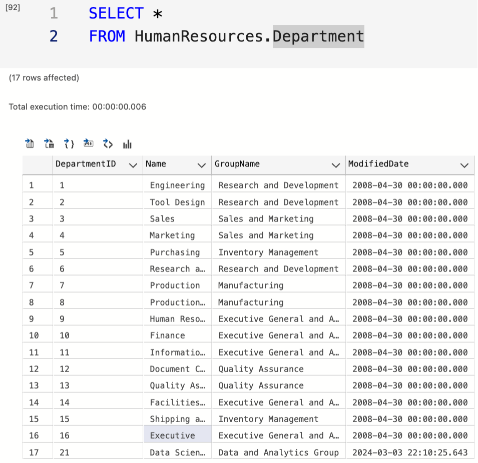
Total execution time: 00:00:00.006

	DepartmentID	Name	GroupName	ModifiedDate
1	1	Engineering	Research and Development	2008-04-30 00:00:00.000
2	2	Tool Design	Research and Development	2008-04-30 00:00:00.000
3	3	Sales	Sales and Marketing	2008-04-30 00:00:00.000
4	4	Marketing	Sales and Marketing	2008-04-30 00:00:00.000
5	5	Purchasing	Inventory Management	2008-04-30 00:00:00.000
6	6	Research a…	Research and Development	2008-04-30 00:00:00.000
7	7	Production	Manufacturing	2008-04-30 00:00:00.000
8	8	Production…	Manufacturing	2008-04-30 00:00:00.000
9	9	Human Reso…	Executive General and A…	2008-04-30 00:00:00.000
10	10	Finance	Executive General and A…	2008-04-30 00:00:00.000
11	11	Informatio…	Executive General and A…	2008-04-30 00:00:00.000
12	12	Document C…	Quality Assurance	2008-04-30 00:00:00.000
13	13	Quality As…	Quality Assurance	2008-04-30 00:00:00.0
14	14	Facilities…	Executive General and A…	2008-04-30 00:00:00.000
15	15	Shipping a…	Inventory Management	2008-04-30 00:00:00.000
16	16	Executive	Executive General and A…	2008-04-30 00:00:00.000



[91]

```sql
BEGIN TRANSACTION T1
    SAVE TRANSACTION SavePoint1
    INSERT INTO HumanResources.Department ([Name], GroupName, ModifiedDate)
    VALUES ('Data Science and Analytics', 'Data and Analytics Group', GETDATE())

    BEGIN TRANSACTION T2
        SAVE TRANSACTION SavePoint2
        UPDATE HumanResources.Department
        SET GroupName='New Executive'
        WHERE DepartmentID=14
    COMMIT TRANSACTION T2
    ROLLBACK TRANSACTION SavePoint2
COMMIT TRANSACTION T1
```



[92]

```sql
SELECT *
FROM HumanResources.Department
```

(17 rows affected)

Total execution time: 00:00:00.006

	DepartmentID	Name	GroupName	ModifiedDate
1	1	Engineering	Research and Development	2008-04-30 00:00:00.000
2	2	Tool Design	Research and Development	2008-04-30 00:00:00.000
3	3	Sales	Sales and Marketing	2008-04-30 00:00:00.000
4	4	Marketing	Sales and Marketing	2008-04-30 00:00:00.000
5	5	Purchasing	Inventory Management	2008-04-30 00:00:00.000
6	6	Research a…	Research and Development	2008-04-30 00:00:00.000
7	7	Production	Manufacturing	2008-04-30 00:00:00.000
8	8	Production…	Manufacturing	2008-04-30 00:00:00.000
9	9	Human Reso…	Executive General and A…	2008-04-30 00:00:00.000
10	10	Finance	Executive General and A…	2008-04-30 00:00:00.000
11	11	Informatio…	Executive General and A…	2008-04-30 00:00:00.000
12	12	Document C…	Quality Assurance	2008-04-30 00:00:00.000
13	13	Quality As…	Quality Assurance	2008-04-30 00:00:00.000
14	14	Facilities…	Executive General and A…	2008-04-30 00:00:00.000
15	15	Shipping a…	Inventory Management	2008-04-30 00:00:00.000
16	16	Executive	Executive General and A…	2008-04-30 00:00:00.000
17	21	Data Scien…	Data and Analytics Group	2024-03-03 22:10:25.643

III. INDEXES

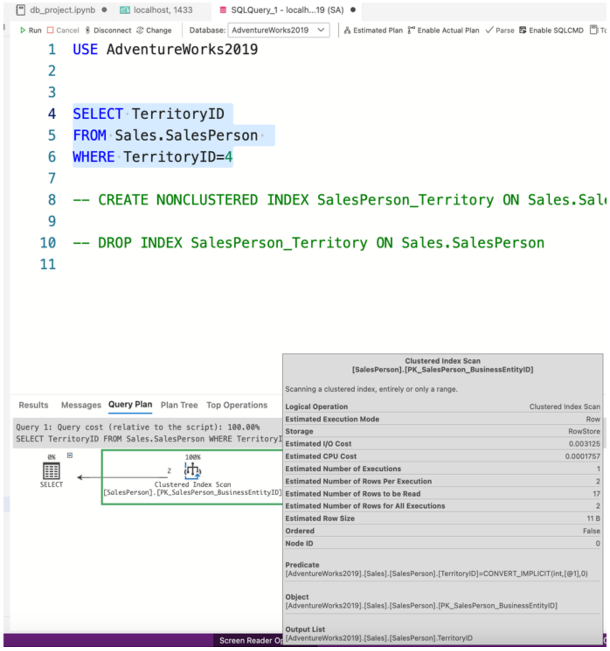
The goal of the index is to make the search operation faster.
Indexes make the search operation faster by creating something called a B-Tree (Balanced Tree) structure internally.
SQL Server Indexes are divided into two types.
They are as follows:

1. Clustered index
2. Non-Clustered index

There is only one clustered index per table and 999 non-clustered ones.
The Clustered Index by default is created when we create the primary key constraint for a table.
That means the primary key column creates a clustered index by default.
In Non-Clustered Index data is stored in one place and the index is stored in another place.

Here we are going to use Non-Clustered Indexes because the database found has used for all tables Clustered ones by default.
We can see in each problem when an index is used, the Estimated Number of Rows to be Read changes. That is the improvement.

Problem 11:



```
USE AdventureWorks2019


SELECT TerritoryID
FROM Sales.SalesPerson
WHERE TerritoryID=4

-- CREATE NONCLUSTERED INDEX SalesPerson_Territory ON Sales.Sal

-- DROP INDEX SalesPerson_Territory ON Sales.SalesPerson

```

```
Query 1: Query cost (relative to the script): 100.00%
SELECT TerritoryID FROM Sales.SalesPerson WHERE TerritoryI
```

Clustered Index Scan
[SalesPerson].[PK_SalesPerson_BusinessEntityID]

Clustered Index Scan
[SalesPerson].[PK_SalesPerson_BusinessEntityID]

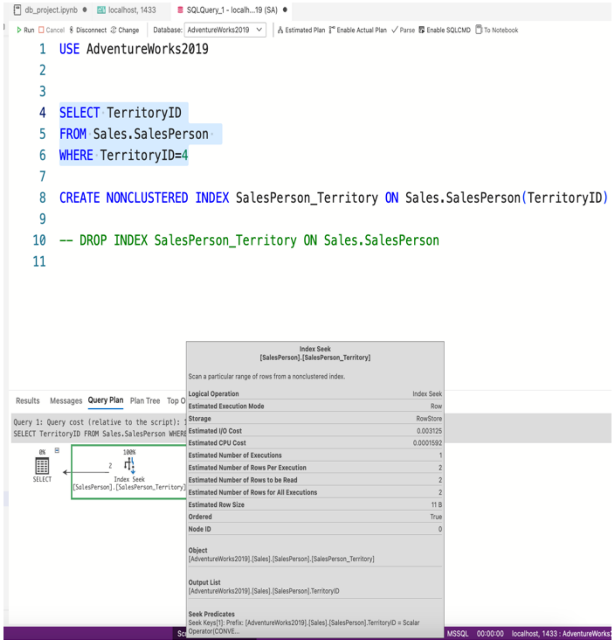
Scanning a clustered index, entirely or only a range.

Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0001757
Estimated Number of Executions	1
Estimated Number of Rows Per Execution	2
Estimated Number of Rows to be Read	17
Estimated Number of Rows for All Executions	2
Estimated Row Size	11 B
Ordered	False
Node ID	0

Predicate
[AdventureWorks2019].[Sales].[SalesPerson].[TerritoryID]=CONVERT_IMPLICIT(int,[@1],0)

Object
[AdventureWorks2019].[Sales].[SalesPerson].[PK_SalesPerson_BusinessEntityID]

Output List
[AdventureWorks2019].[Sales].[SalesPerson].TerritoryID



```
USE AdventureWorks2019


SELECT TerritoryID
FROM Sales.SalesPerson
WHERE TerritoryID=4

CREATE NONCLUSTERED INDEX SalesPerson_Territory ON Sales.SalesPerson(TerritoryID)

-- DROP INDEX SalesPerson_Territory ON Sales.SalesPerson

```

```
Query 1: Query cost (relative to the script): 1
SELECT TerritoryID FROM Sales.SalesPerson WHERE
```

Index Seek
[SalesPerson].[SalesPerson_Territory]

Index Seek
[SalesPerson].[SalesPerson_Territory]

Scan a particular range of rows from a nonclustered index.

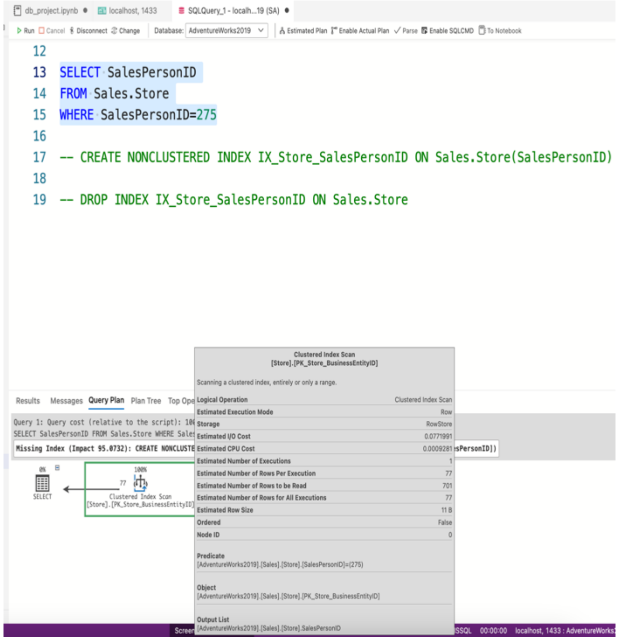
Logical Operation	Index Seek
Estimated Execution Mode	Row
Storage	RowStore
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0001592
Estimated Number of Executions	1
Estimated Number of Rows Per Execution	2
Estimated Number of Rows to be Read	2
Estimated Number of Rows for All Executions	2
Estimated Row Size	11 B
Ordered	True
Node ID	0

Object
[AdventureWorks2019].[Sales].[SalesPerson].[SalesPerson_Territory]

Output List
[AdventureWorks2019].[Sales].[SalesPerson].TerritoryID

Seek Predicates
Seek Keys[1]: Prefix: [AdventureWorks2019].[Sales].[SalesPerson].TerritoryID = Scalar Operator(CONVE...

Problem 12:



```sql
SELECT SalesPersonID
FROM Sales.Store
WHERE SalesPersonID=275

-- CREATE NONCLUSTERED INDEX IX_Store_SalesPersonID ON Sales.Store(SalesPersonID)

-- DROP INDEX IX_Store_SalesPersonID ON Sales.Store
```

```
Query 1: Query cost (relative to the script): 10(
SELECT SalesPersonID FROM Sales.Store WHERE Sales
Missing Index (Impact 95.0732): CREATE NONCLUSTE                                 sPersonID])
```

Clustered Index Scan
[Store].[PK_Store_BusinessEntityID]

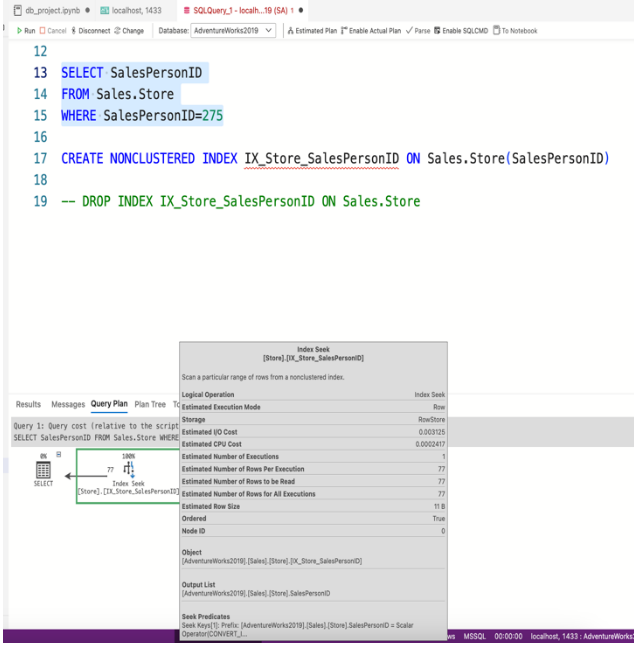
Scanning a clustered index, entirely or only a range.

Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated I/O Cost	0.0771991
Estimated CPU Cost	0.0009281
Estimated Number of Executions	1
Estimated Number of Rows Per Execution	77
Estimated Number of Rows to be Read	701
Estimated Number of Rows for All Executions	77
Estimated Row Size	11 B
Ordered	False
Node ID	0

Predicate
[AdventureWorks2019].[Sales].[Store].[SalesPersonID]=(275)

Object
[AdventureWorks2019].[Sales].[Store].[PK_Store_BusinessEntityID]

Output List
[AdventureWorks2019].[Sales].[Store].SalesPersonID



```sql
SELECT SalesPersonID
FROM Sales.Store
WHERE SalesPersonID=275

CREATE NONCLUSTERED INDEX IX_Store_SalesPersonID ON Sales.Store(SalesPersonID)

-- DROP INDEX IX_Store_SalesPersonID ON Sales.Store
```

```
Query 1: Query cost (relative to the script
SELECT SalesPersonID FROM Sales.Store WHERE
```

Index Seek
[Store].[IX_Store_SalesPersonID]

Scan a particular range of rows from a nonclustered index.

Logical Operation	Index Seek
Estimated Execution Mode	Row
Storage	RowStore
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0002417
Estimated Number of Executions	1
Estimated Number of Rows Per Execution	77
Estimated Number of Rows to be Read	77
Estimated Number of Rows for All Executions	77
Estimated Row Size	11 B
Ordered	True
Node ID	0

Object
[AdventureWorks2019].[Sales].[Store].[IX_Store_SalesPersonID]

Output List
[AdventureWorks2019].[Sales].[Store].SalesPersonID

Seek Predicates
Seek Keys[1]: Prefix: [AdventureWorks2019].[Sales].[Store].SalesPersonID = Scalar Operator(CONVERT_I...