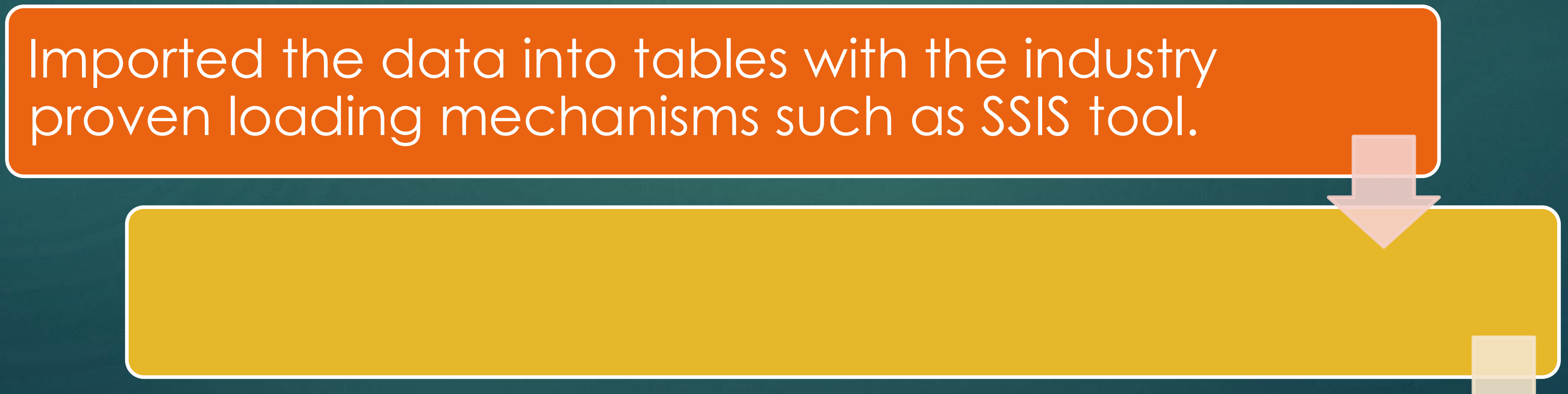
PARTITION TECHNIQUES

ON HEALTH INSURANCE MARKETPLACE

TASKS DONE:



Imported the data into tables with the industry proven loading mechanisms such as SSIS tool.

Implemented the test scenarios successfully with the usage of partition in the queries..

Improvising on the performance using Partitioning :

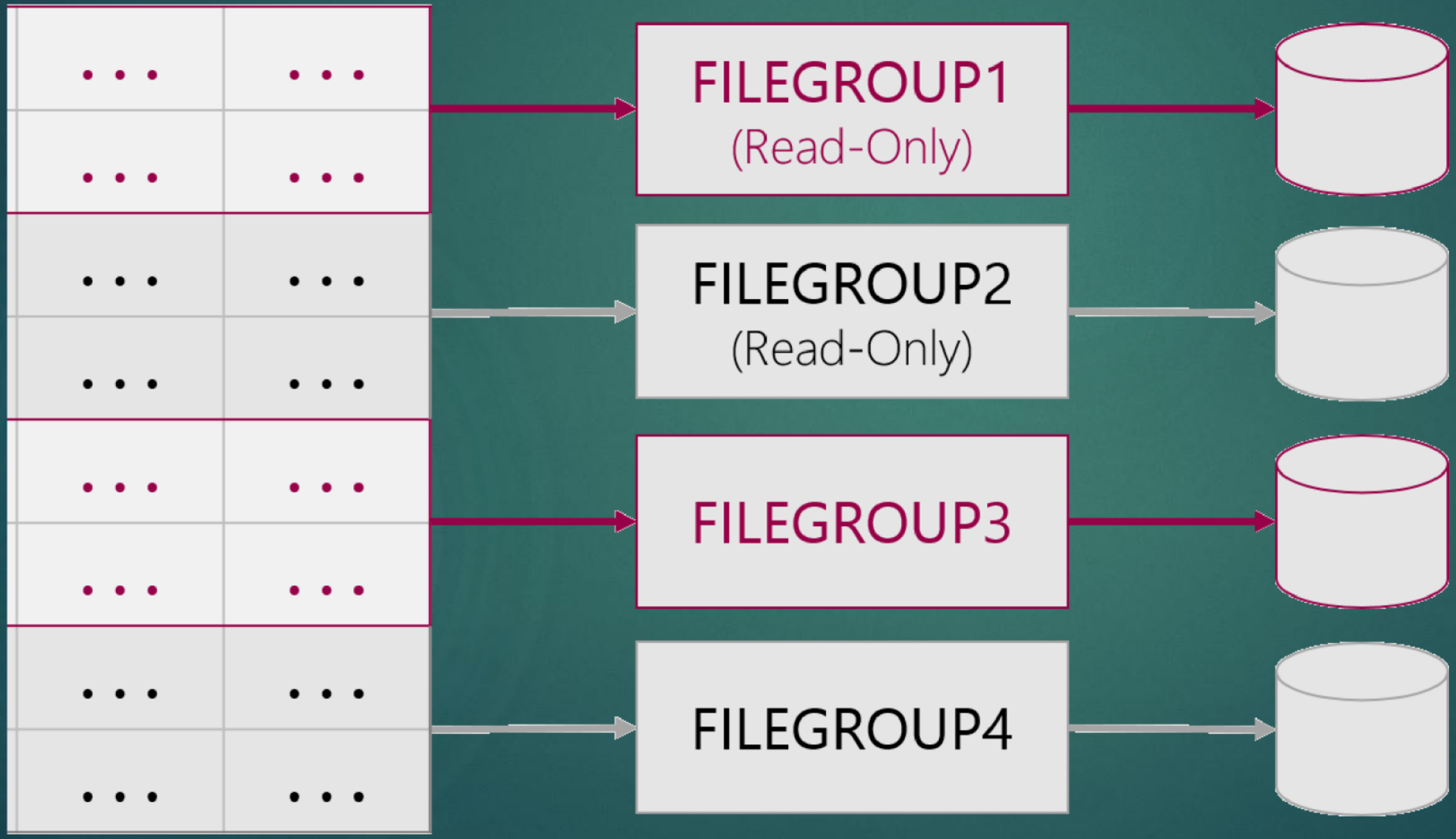
Partitioning is implemented as performance tuning and will be explained in further slides

WHAT IS PARTITIONING

?

WHEN IS PARTITIONING USEFUL ?

A WAY TO BREAK TABLES DOWN INTO SMALLER CHUNKS FO MANAGEABILITY AND PERFORMA



Partitioning is useful in reading data when the partitioning key is part of the SQL statement.

One benefit of a good partitioning strategy is for aging data.

WHY TO USE PARTITIONING ?

The relative speed of the queries is increased.

Faster data load.

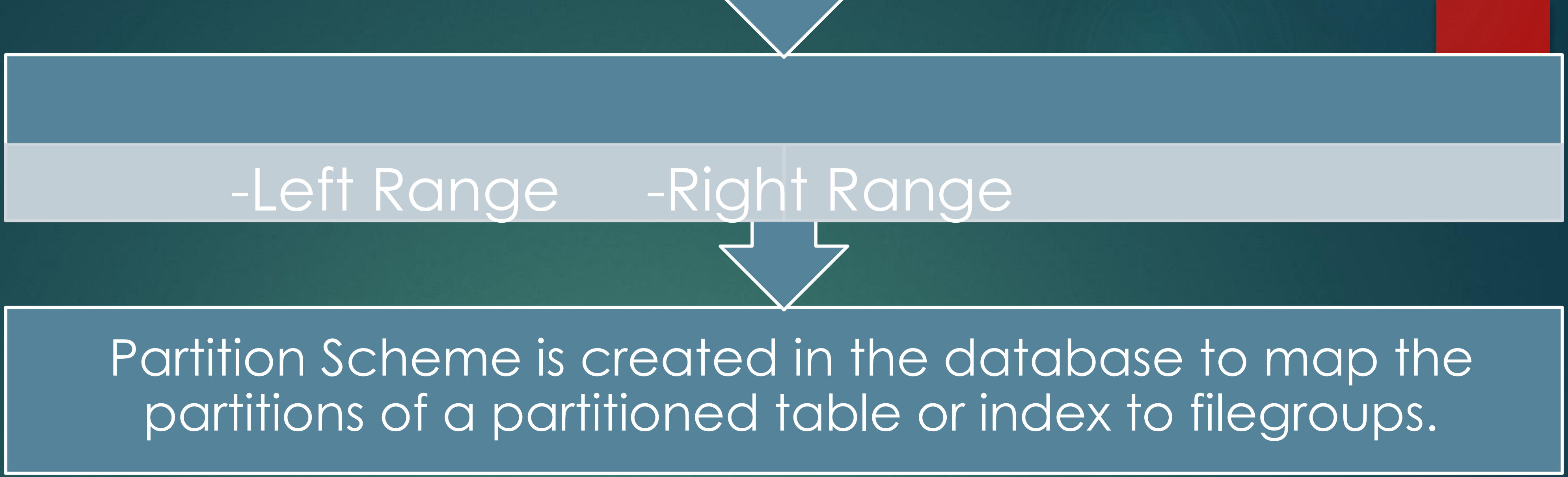
Faster deletion of old data will be limited to certain partitions, if they are no longer needed.

Faster archival of rarely used or old data, for migration to cheaper and slower storage media.

PARTITIONING PROCEDURE :

Partition Key is comprised of one or more columns that determine the partition.

Partition Function is a function in the current database that maps the rows of a table or index into partitions based on the values of a specified column.



-Left Range -Right Range

Partition Scheme is created in the database to map the partitions of a partitioned table or index to filegroups.

PARTITIONING TYPES :

Horizontal Partitioning

Horizontal partitioning divides a table into multiple tables. Each table then contains the same number of columns, but fewer rows.

List Partitioning

Range Partitioning

Vertical Partitioning

Vertical partitioning divides a table into multiple tables that contain fewer columns.

SOFTWARE USED:

- SQL Server 2017 Express Edition
- SQL Server Management Studio
- SQL Server Data Tools
- PostgreSQL

DATA SET :

BenefitsCostSharing	*In the BenefitsCostSharing csv file, each record pertains to the coverage of a single benefit by one issuer's insurance plan. This data contains plan-level data on essential health benefits, coverage limits, and cost sharing for each QHP and SADP.*
BusinessRules	*Rules associated with each plan are distinguished based on TIN Number. This helps in determining the price for the user.*
Network	*It contains details of all the network coverage areas for each plan.*
PlanAttributes	*different plan attributes.* *It contains details of each plan with all the covered benefits under each plan. Columns under this are of*

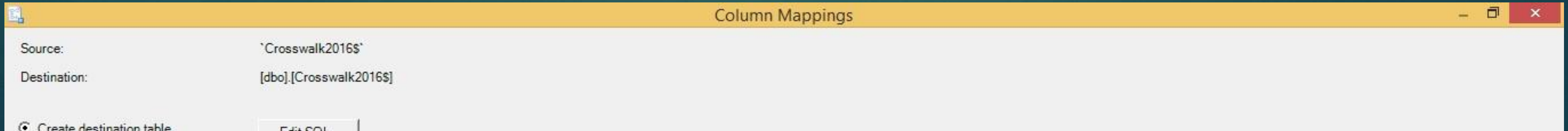
Rate	***This csv describes the variables contained in the Rate-PUF. Each record relates to one issuer's rates based on plan, geographic rating area, and subscriber eligibility requirements. The RatePUF is available for plan year 2014, plan year 2015, and plan year 2016.***
ServiceArea	***This file describes each plan and their corresponding service areas associated with it.***
Cross walk 2015:	The Plan ID Crosswalk PUF (CW-PUF) is one of the seven files that make up the Marketplace PUF. The purpose of the CW-PUF is to map QHPs and SADPs offered through the Marketplaces in 2014 to plans that will be offered through the Marketplaces in 2015. These data either originate from the Plan Crosswalk template
Cross walk 2016	The purpose of the CW-PUF is to map QHPs and SADPs offered through the Marketplaces in 2015 to plans that will be offered through the Marketplaces in 2016. These data either originate from the Plan Crosswalk template (i.e., template field), an Excel-based form used by issuers to describe their plans in the QHP application process.

Table Name Partition

BenefitCostSharing	IssuerID	BenefitCostName,StateCode,PlanId,CoinsOutOfnet
BusinessRules	IssuerID	StateCode,IssuerId,ProductId
PlanAttributes	IssuerID	IssuerId2,Statecode, PlanId
Network	IssuerID	
Rate	IssuerID	PlanId,Age,IndividualRate
Servicearea	IssuerID	StateCode,MarketCoverage
CrossWalk2015	IssuerID	PlanId,Dentalcode,Fipscode,state
Crosswalk2016	IssuerID	PlanId,Dentalcode,Fipscode,state

DATA SET CHOSEN :

CREATION OF FILE GROUPS :



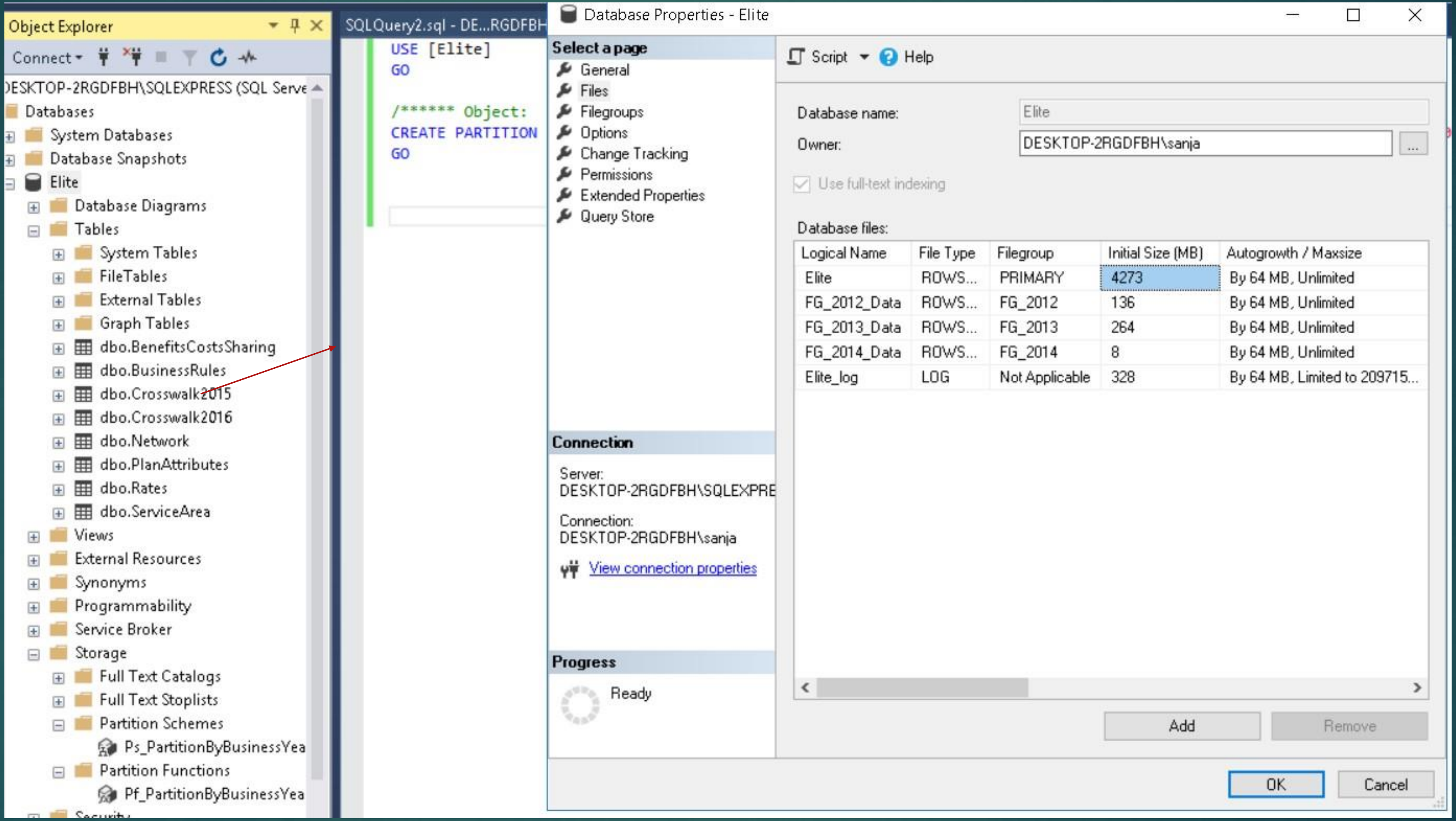
Column Mappings

Source: `Crosswalk2016$`

Destination: [dbo].[Crosswalk2016$]

● Create destination table

ASSIGNING FILES TO EACH FILEGROUP:



```
USE [Elite]
GO

/****** Object:
CREATE PARTITION
GO
```

Database name: Elite

Owner: DESKTOP-2RGDFBH\sanja

☑ Use full-text indexing

Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth / Maxsize
Elite	ROWS...	PRIMARY	4273	By 64 MB, Unlimited
FG_2012_Data	ROWS...	FG_2012	136	By 64 MB, Unlimited
FG_2013_Data	ROWS...	FG_2013	264	By 64 MB, Unlimited
FG_2014_Data	ROWS...	FG_2014	8	By 64 MB, Unlimited
Elite_log	LOG	Not Applicable	328	By 64 MB, Limited to 209715...

Server:
DESKTOP-2RGDFBH\SQLEXPRE

Connection:
DESKTOP-2RGDFBH\sanja

Ready

Partition	Operation Performed	Partitioned Table (Sec)	Non-Partitioned Table (Sec)
Vertical	Select	0.02	0.114
	Update	0.11	2.80
	Delete	0.38	3.30

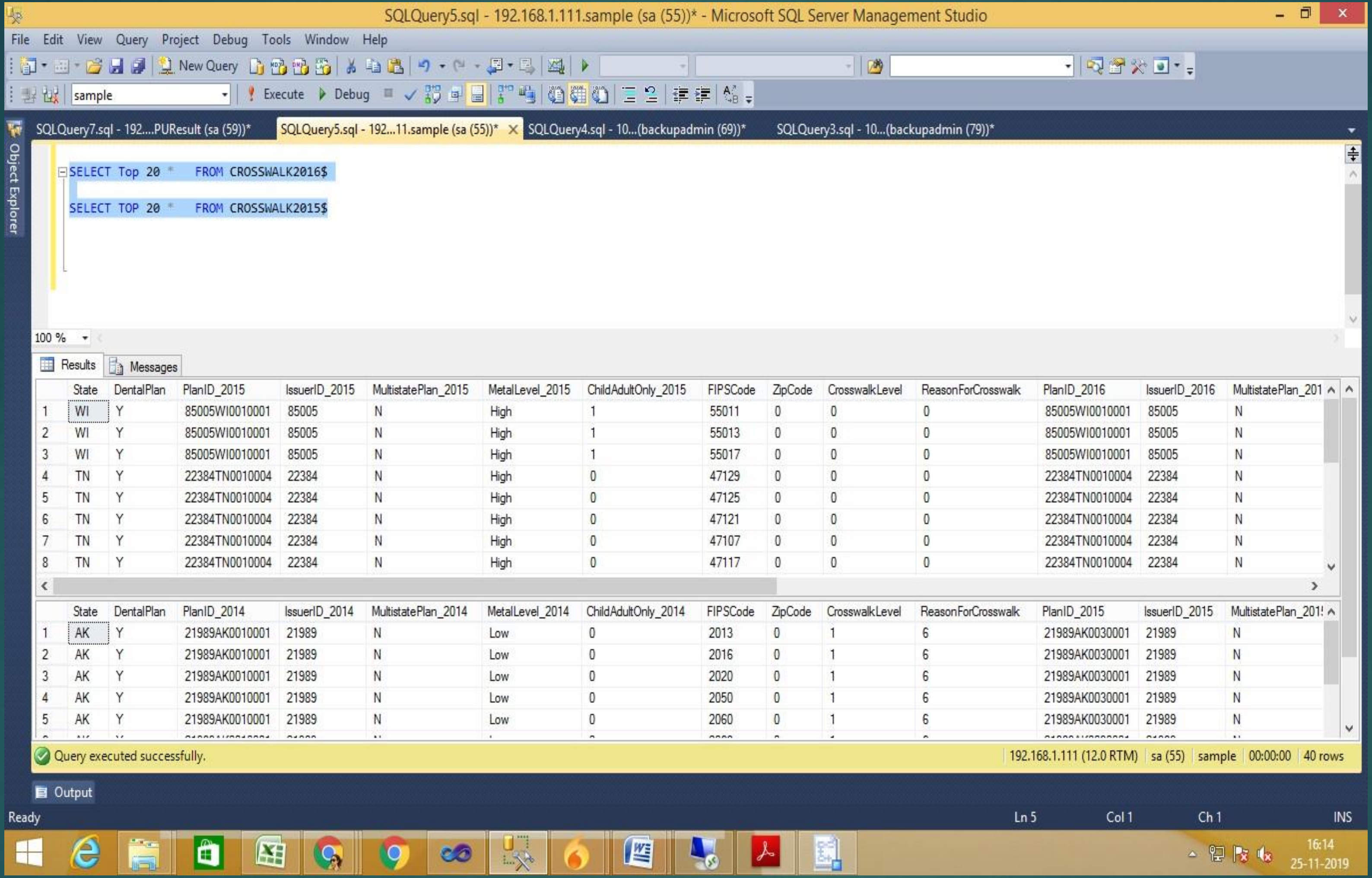
Vertical Partitioning:

	Inner Join	0.10	10.0

- This partitioning scheme is traditionally used to reduce the width of a target table by splitting a table vertically.
- so that only certain columns are included in a particular dataset, with each partition including all rows.
- Below are the Query execution time of different operations performed on the Nonpartitioned tables.

PARTITION TABLE DETAILS

Vertical partitioning



SQLQuery5.sql - 192.168.1.111.sample (sa (55))* - Microsoft SQL Server Management Studio
File
Edit
View
Query
Project
Debug
Tools
Window
Help
New Query
sample
Execute
Debug
Object Explorer
SQLQuery7.sql - 192....PUResult (sa (59))*
SQLQuery5.sql - 192...11.sample (sa (55))*
SQLQuery4.sql - 10...(backupadmin (69))*
SQLQuery3.sql - 10...(backupadmin (79))*
100 %
Results
Messages
Output

```sql
SELECT Top 20 *   FROM CROSSWALK2016$

SELECT TOP 20 *   FROM CROSSWALK2015$
```

	State	DentalPlan	PlanID_2015	IssuerID_2015	MultistatePlan_2015	MetalLevel_2015	ChildAdultOnly_2015	FIPSCode	ZipCode	CrosswalkLevel	ReasonForCrosswalk	PlanID_2016	IssuerID_2016	MultistatePlan_201
1	WI	Y	85005WI0010001	85005	N	High	1	55011	0	0	0	85005WI0010001	85005	N
2	WI	Y	85005WI0010001	85005	N	High	1	55013	0	0	0	85005WI0010001	85005	N
3	WI	Y	85005WI0010001	85005	N	High	1	55017	0	0	0	85005WI0010001	85005	N
4	TN	Y	22384TN0010004	22384	N	High	0	47129	0	0	0	22384TN0010004	22384	N
5	TN	Y	22384TN0010004	22384	N	High	0	47125	0	0	0	22384TN0010004	22384	N
6	TN	Y	22384TN0010004	22384	N	High	0	47121	0	0	0	22384TN0010004	22384	N
7	TN	Y	22384TN0010004	22384	N	High	0	47107	0	0	0	22384TN0010004	22384	N
8	TN	Y	22384TN0010004	22384	N	High	0	47117	0	0	0	22384TN0010004	22384	N

	State	DentalPlan	PlanID_2014	IssuerID_2014	MultistatePlan_2014	MetalLevel_2014	ChildAdultOnly_2014	FIPSCode	ZipCode	CrosswalkLevel	ReasonForCrosswalk	PlanID_2015	IssuerID_2015	MultistatePlan_201!
1	AK	Y	21989AK0010001	21989	N	Low	0	2013	0	1	6	21989AK0030001	21989	N
2	AK	Y	21989AK0010001	21989	N	Low	0	2016	0	1	6	21989AK0030001	21989	N
3	AK	Y	21989AK0010001	21989	N	Low	0	2020	0	1	6	21989AK0030001	21989	N
4	AK	Y	21989AK0010001	21989	N	Low	0	2050	0	1	6	21989AK0030001	21989	N
5	AK	Y	21989AK0010001	21989	N	Low	0	2060	0	1	6	21989AK0030001	21989	N

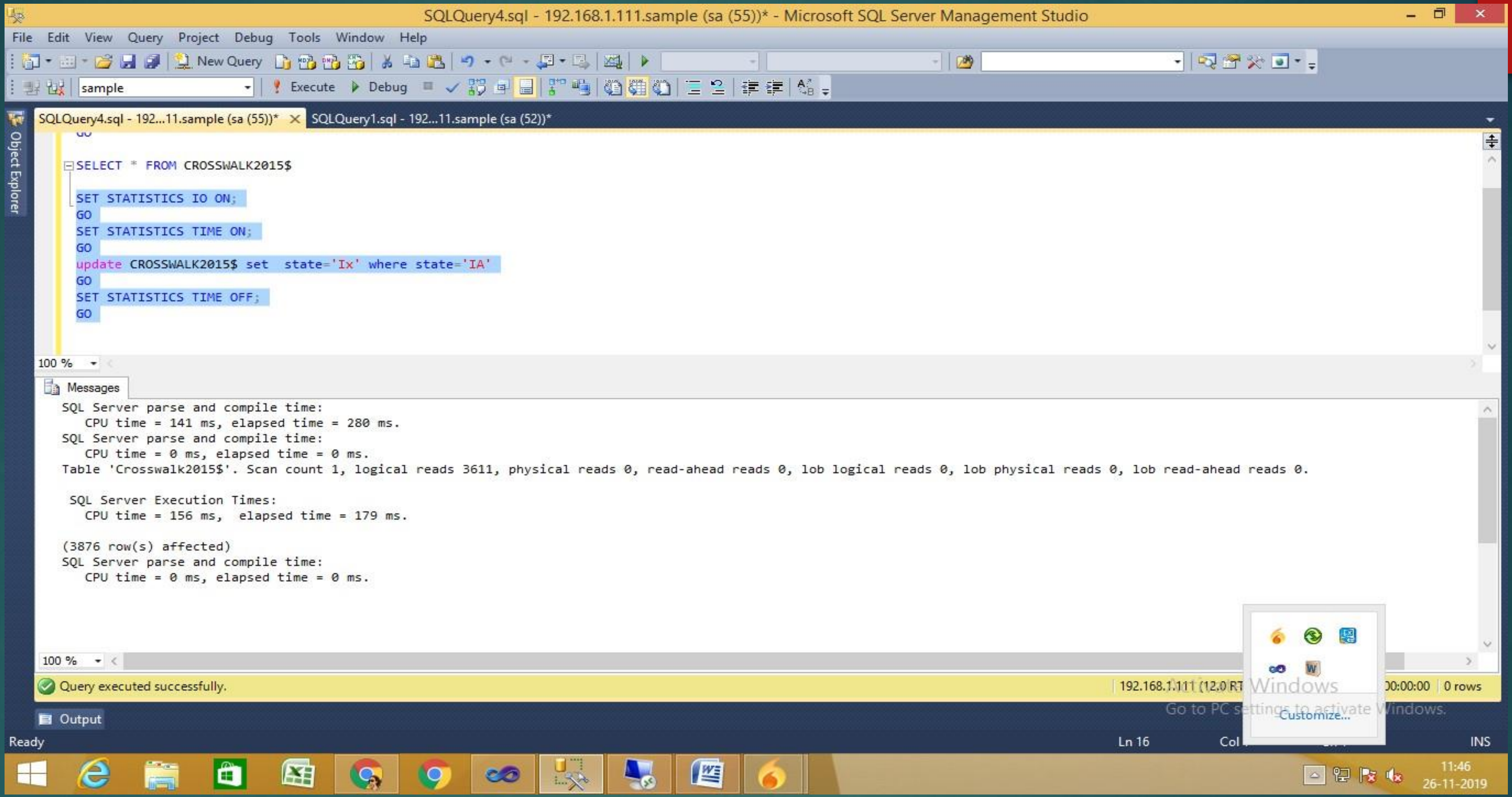
Query executed successfully.

192.168.1.111 (12.0 RTM) | sa (55) | sample | 00:00:00 | 40 rows

Ready   Ln 5   Col 1   Ch 1   INS

16:14
25-11-2019

Update operation: Non partition:

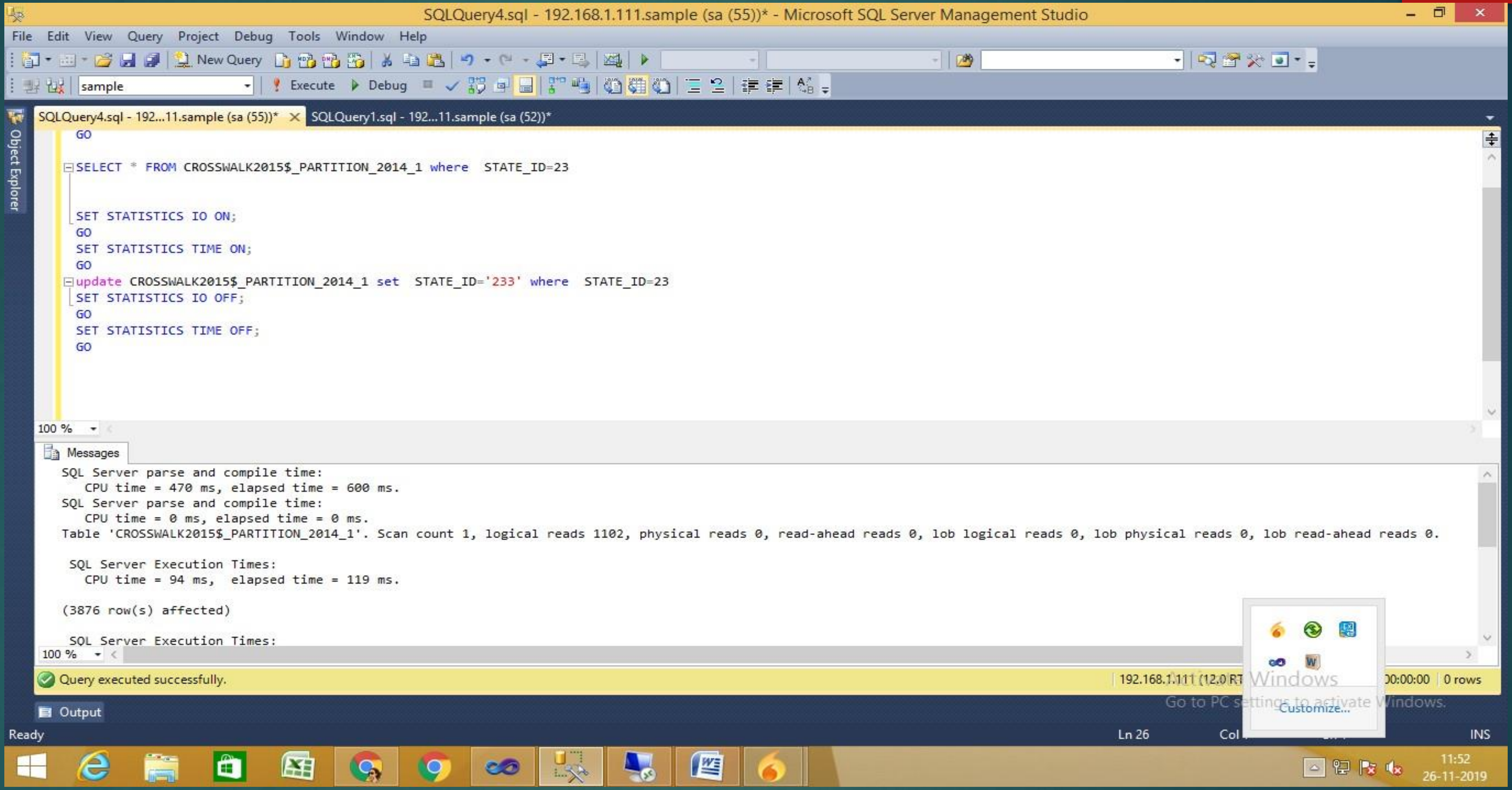

SQLQuery4.sql - 192.168.1.111.sample (sa (55))* - Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Help
New Query
sample
Execute
Debug
SQLQuery4.sql - 192...11.sample (sa (55))*
SQLQuery1.sql - 192...11.sample (sa (52))*
Object Explorer
100 %
Messages
Output
Ready
Ln 16
Col
INS
192.168.1.111 (12.0 RT
00:00:00
0 rows
Customize...
11:46
26-11-2019

```sql
SELECT * FROM CROSSWALK2015$

SET STATISTICS IO ON;
GO
SET STATISTICS TIME ON;
GO
update CROSSWALK2015$ set  state='Ix' where state='IA'
GO
SET STATISTICS TIME OFF;
GO
```

```
SQL Server parse and compile time:
   CPU time = 141 ms, elapsed time = 280 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'Crosswalk2015$'. Scan count 1, logical reads 3611, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 156 ms,  elapsed time = 179 ms.

(3876 row(s) affected)
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
```

Query executed successfully.

Update operation: Partitioning:



SQLQuery4.sql - 192.168.1.111.sample (sa (55))* - Microsoft SQL Server Management Studio
File Edit View Query Project Debug Tools Window Help
New Query
sample
Execute
Debug
SQLQuery4.sql - 192...11.sample (sa (55))*
SQLQuery1.sql - 192...11.sample (sa (52))*
Object Explorer
100 %
Messages
192.168.1.111 (12.0 RT
00:00:00
0 rows
Output
Ready
Ln 26
Col
INS
Windows
Go to PC settings to activate Windows.
Customize...
11:52
26-11-2019

```sql
GO

SELECT * FROM CROSSWALK2015$_PARTITION_2014_1 where  STATE_ID=23


SET STATISTICS IO ON;
GO
SET STATISTICS TIME ON;
GO
update CROSSWALK2015$_PARTITION_2014_1 set  STATE_ID='233' where  STATE_ID=23
SET STATISTICS IO OFF;
GO
SET STATISTICS TIME OFF;
GO
```

```
SQL Server parse and compile time:
   CPU time = 470 ms, elapsed time = 600 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'CROSSWALK2015$_PARTITION_2014_1'. Scan count 1, logical reads 1102, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 94 ms,  elapsed time = 119 ms.

(3876 row(s) affected)

 SQL Server Execution Times:
```

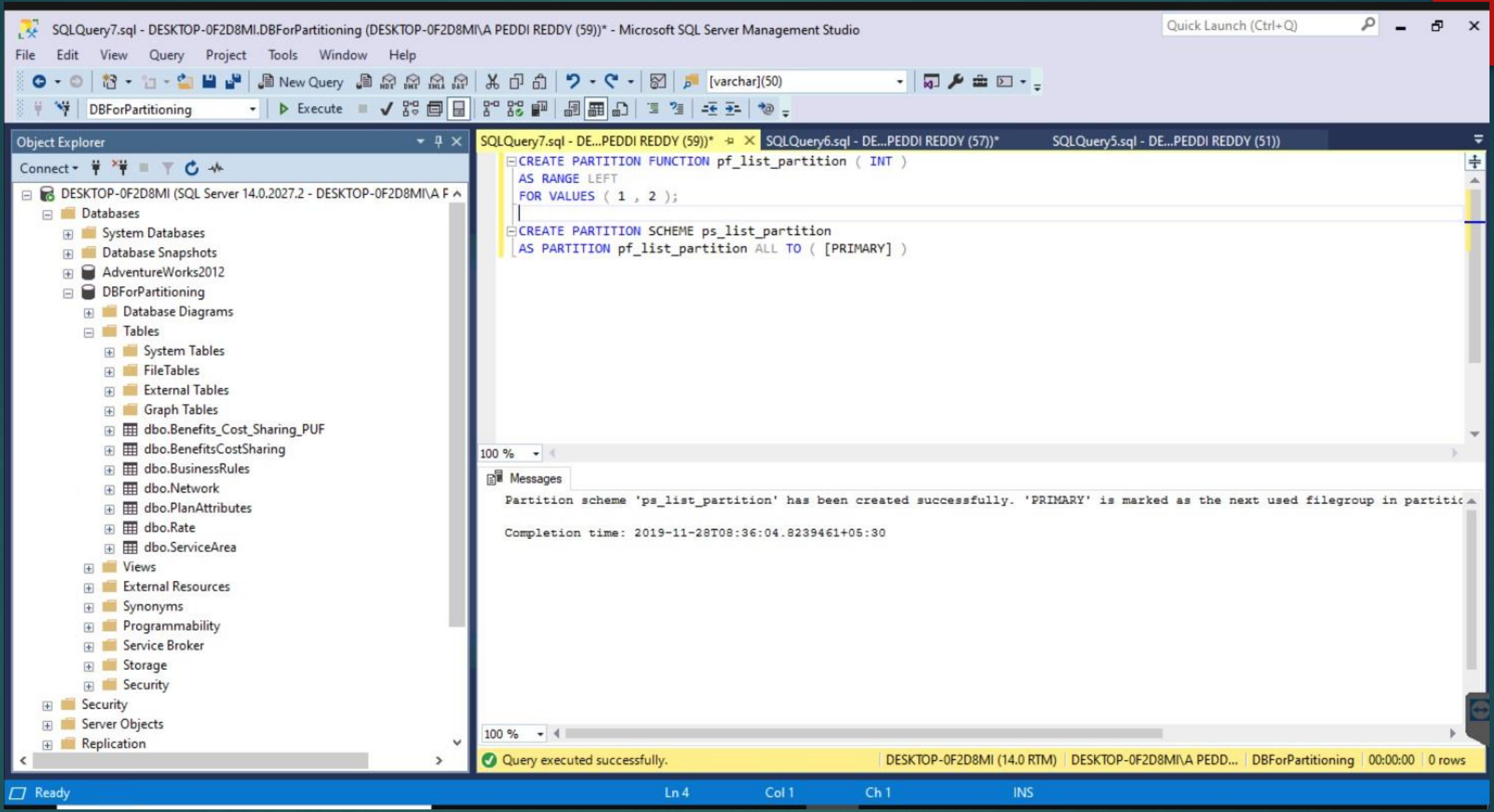
Query executed successfully.

List partitioning:

- List partitioning is used when there are a predefined set of discrete values.
- Using which we can group and organize unordered sets of data and when we plan to access medium data aggregations/ dumps on a frequent basis using the partition key column.
- If a table is partitioned by list, the partitioning key can only consist of a single column of the table.

Partition perfomed	Operation	Partitioned table (sec)	Non partitioned table (sec)
List	Select	0.15	1.23
	Update	0.97	5.501
	Delete	0.20	6.036
	Inner Join	0.7	5.721

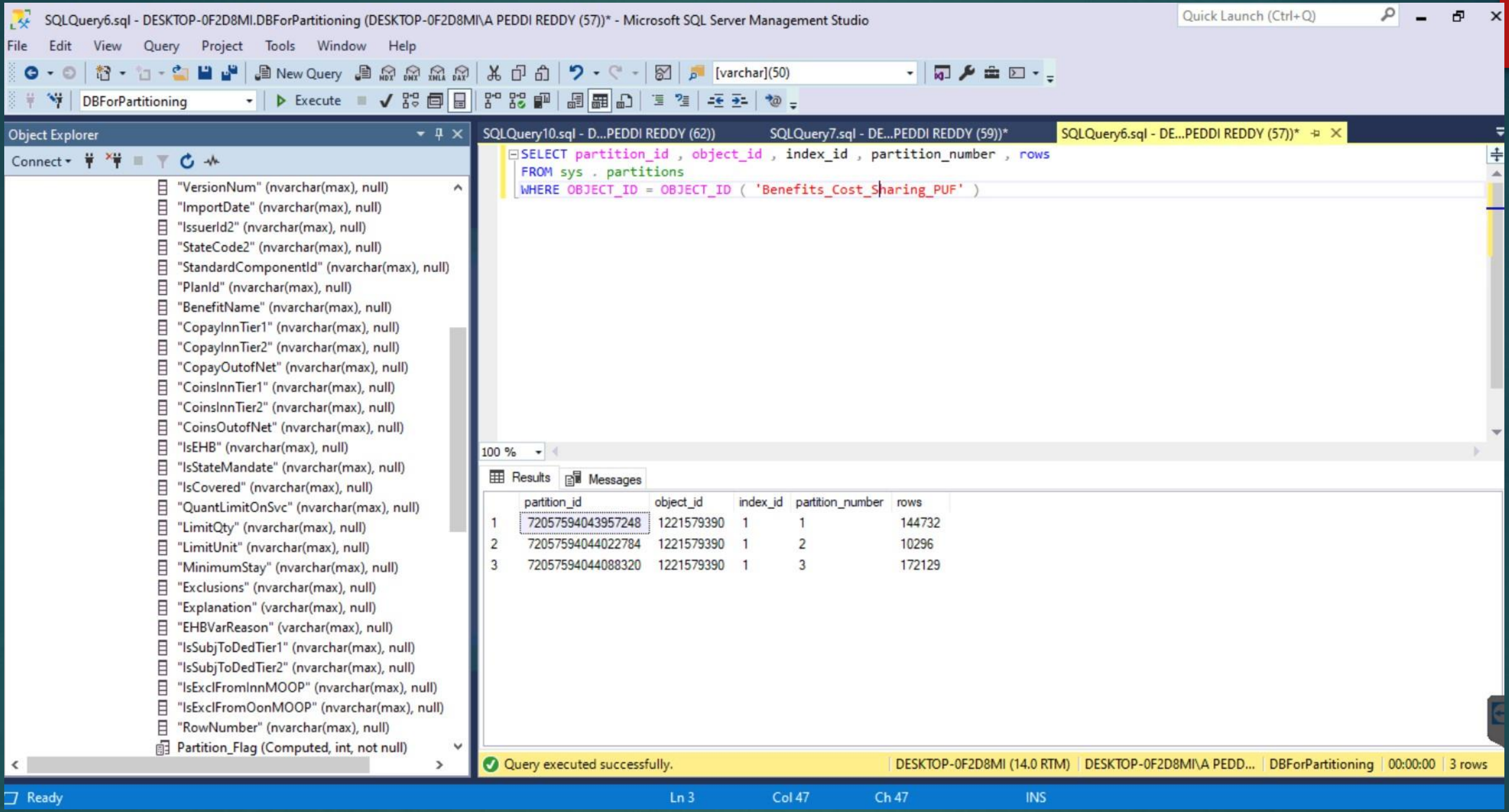
Creation of list Partition:



```sql
CREATE PARTITION FUNCTION pf_list_partition ( INT )
AS RANGE LEFT
FOR VALUES ( 1 , 2 );

CREATE PARTITION SCHEME ps_list_partition
AS PARTITION pf_list_partition ALL TO ( [PRIMARY] )
```

```
Partition scheme 'ps_list_partition' has been created successfully. 'PRIMARY' is marked as the next used filegroup in partitic

Completion time: 2019-11-28T08:36:04.8239461+05:30
```

List Partition Filegroups:


SQLQuery6.sql - DESKTOP-0F2D8MI.DBForPartitioning (DESKTOP-0F2D8MI\A PEDDI REDDY (57))* - Microsoft SQL Server Management Studio
Quick Launch (Ctrl+Q)
File
Edit
View
Query
Project
Tools
Window
Help
New Query
[varchar](50)
DBForPartitioning
Execute
Object Explorer
Connect
SQLQuery10.sql - D...PEDDI REDDY (62))
SQLQuery7.sql - DE...PEDDI REDDY (59))*
SQLQuery6.sql - DE...PEDDI REDDY (57))*
100 %
Results
Messages

"VersionNum" (nvarchar(max), null)
"ImportDate" (nvarchar(max), null)
"IssuerId2" (nvarchar(max), null)
"StateCode2" (nvarchar(max), null)
"StandardComponentId" (nvarchar(max), null)
"PlanId" (nvarchar(max), null)
"BenefitName" (nvarchar(max), null)
"CopayInnTier1" (nvarchar(max), null)
"CopayInnTier2" (nvarchar(max), null)
"CopayOutofNet" (nvarchar(max), null)
"CoinsInnTier1" (nvarchar(max), null)
"CoinsInnTier2" (nvarchar(max), null)
"CoinsOutofNet" (nvarchar(max), null)
"IsEHB" (nvarchar(max), null)
"IsStateMandate" (nvarchar(max), null)
"IsCovered" (nvarchar(max), null)
"QuantLimitOnSvc" (nvarchar(max), null)
"LimitQty" (nvarchar(max), null)
"LimitUnit" (nvarchar(max), null)
"MinimumStay" (nvarchar(max), null)
"Exclusions" (nvarchar(max), null)
"Explanation" (varchar(max), null)
"EHBVarReason" (varchar(max), null)
"IsSubjToDedTier1" (nvarchar(max), null)
"IsSubjToDedTier2" (nvarchar(max), null)
"IsExclFromInnMOOP" (nvarchar(max), null)
"IsExclFromOonMOOP" (nvarchar(max), null)
"RowNumber" (nvarchar(max), null)
Partition_Flag (Computed, int, not null)

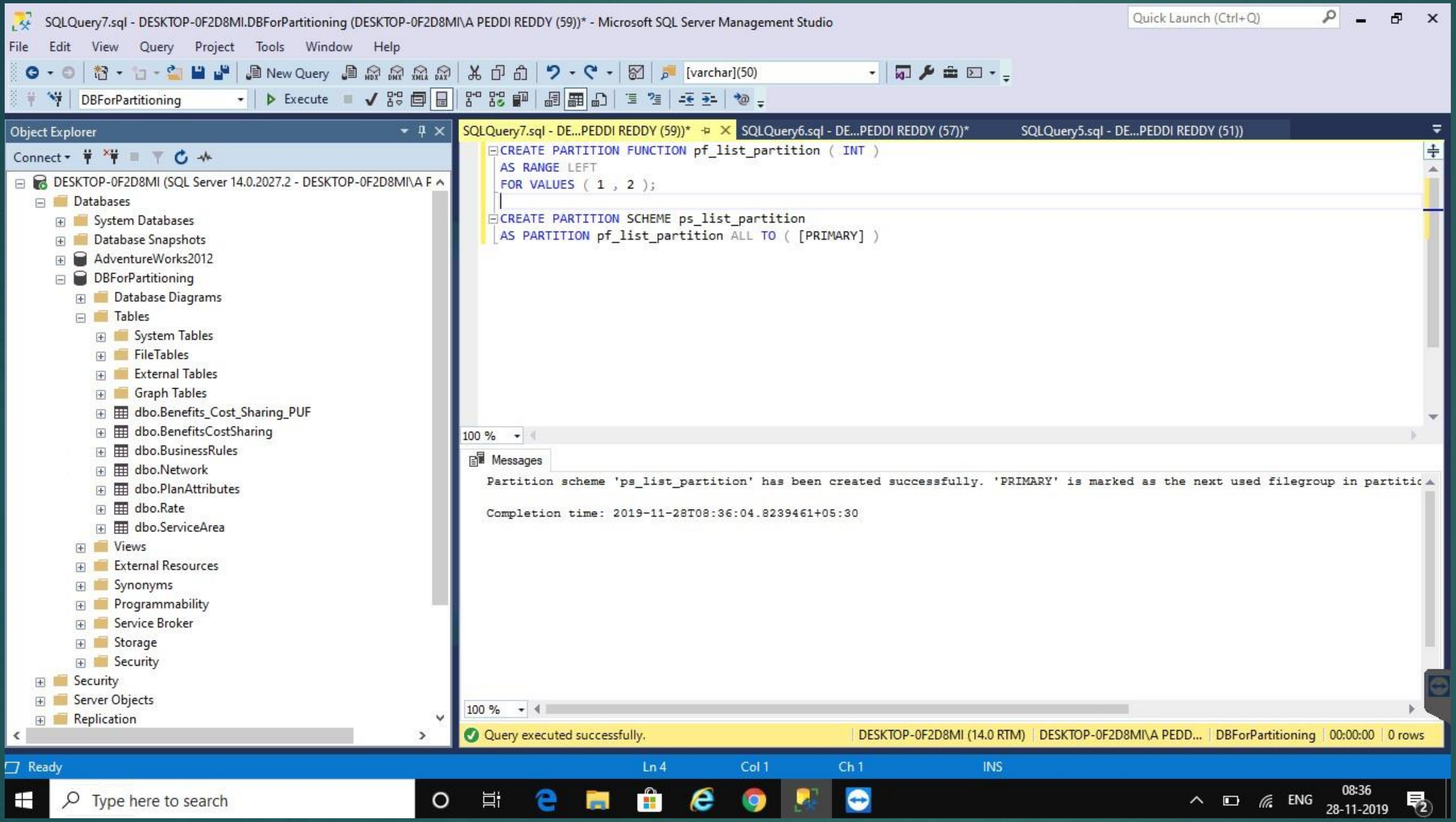
```sql
SELECT partition_id , object_id , index_id , partition_number , rows
FROM sys . partitions
WHERE OBJECT_ID = OBJECT_ID ( 'Benefits_Cost_Sharing_PUF' )
```

	partition_id	object_id	index_id	partition_number	rows
1	72057594043957248	1221579390	1	1	144732
2	72057594044022784	1221579390	1	2	10296
3	72057594044088320	1221579390	1	3	172129

Query executed successfully. DESKTOP-0F2D8MI (14.0 RTM) DESKTOP-0F2D8MI\A PEDD... DBForPartitioning 00:00:00 3 rows

Ready Ln 3 Col 47 Ch 47 INS

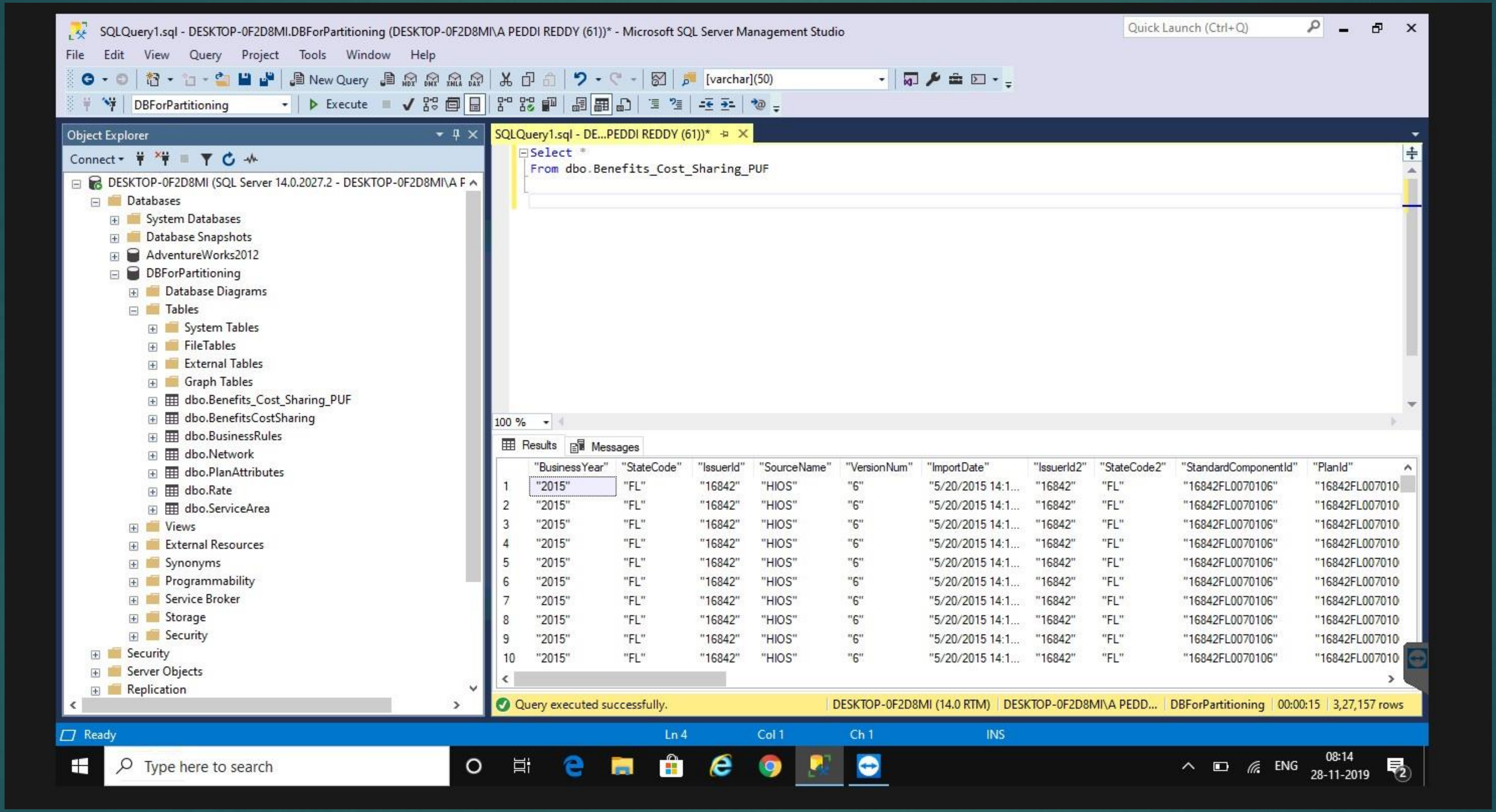
PARTITION CREATION:



```sql
CREATE PARTITION FUNCTION pf_list_partition ( INT )
AS RANGE LEFT
FOR VALUES ( 1 , 2 );

CREATE PARTITION SCHEME ps_list_partition
AS PARTITION pf_list_partition ALL TO ( [PRIMARY] )
```

```
Partition scheme 'ps_list_partition' has been created successfully. 'PRIMARY' is marked as the next used filegroup in partitic

Completion time: 2019-11-28T08:36:04.8239461+05:30
```

List partition:

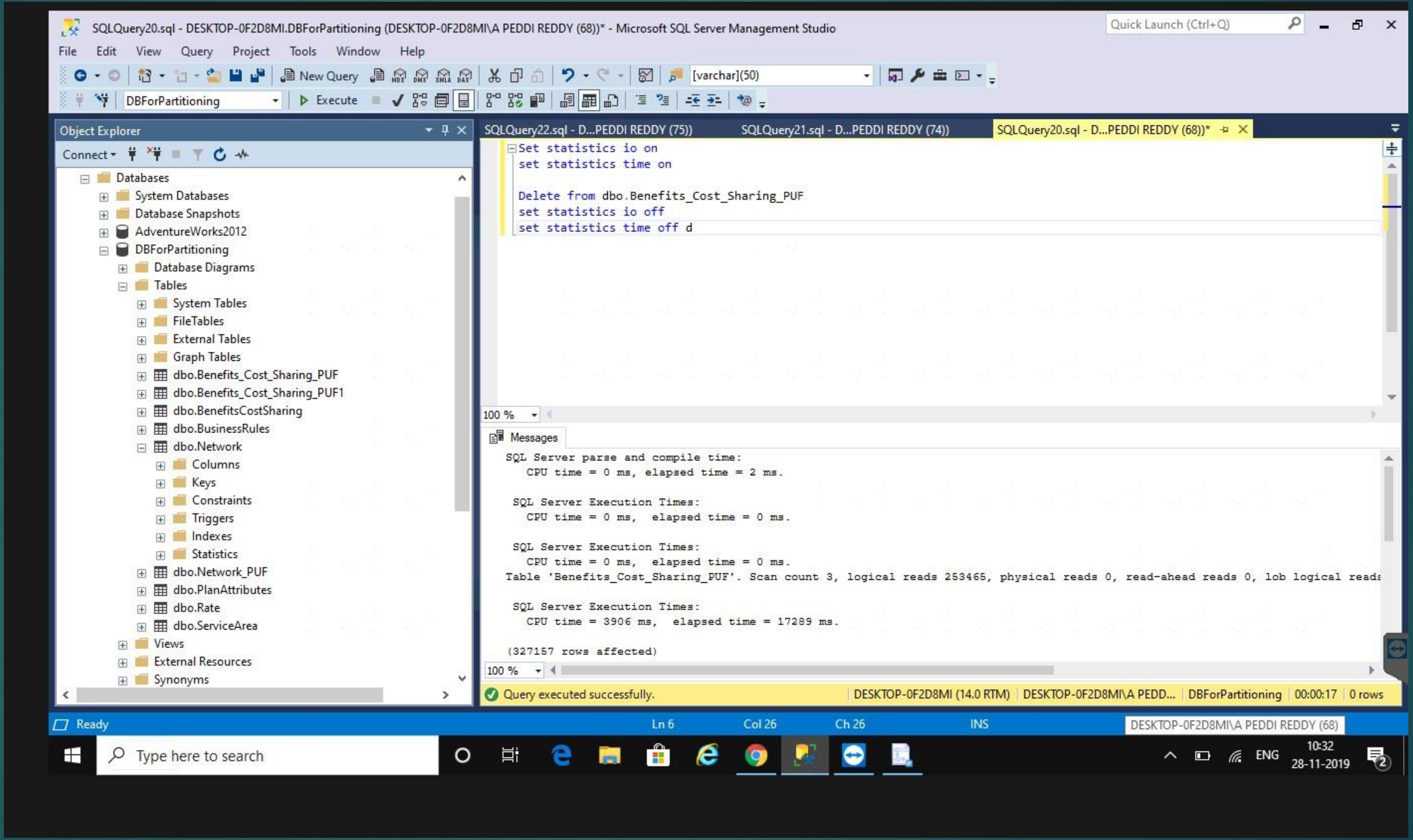


```
Select *
From dbo.Benefits_Cost_Sharing_PUF
```

	"BusinessYear"	"StateCode"	"IssuerId"	"SourceName"	"VersionNum"	"ImportDate"	"IssuerId2"	"StateCode2"	"StandardComponentId"	"PlanId"
1	"2015"	"FL"	"16842"	"HIOS"	"6"	"5/20/2015 14:1...	"16842"	"FL"	"16842FL0070106"	"16842FL007010
2	"2015"	"FL"	"16842"	"HIOS"	"6"	"5/20/2015 14:1...	"16842"	"FL"	"16842FL0070106"	"16842FL007010
3	"2015"	"FL"	"16842"	"HIOS"	"6"	"5/20/2015 14:1...	"16842"	"FL"	"16842FL0070106"	"16842FL007010
4	"2015"	"FL"	"16842"	"HIOS"	"6"	"5/20/2015 14:1...	"16842"	"FL"	"16842FL0070106"	"16842FL007010
5	"2015"	"FL"	"16842"	"HIOS"	"6"	"5/20/2015 14:1...	"16842"	"FL"	"16842FL0070106"	"16842FL007010
6	"2015"	"FL"	"16842"	"HIOS"	"6"	"5/20/2015 14:1...	"16842"	"FL"	"16842FL0070106"	"16842FL007010
7	"2015"	"FL"	"16842"	"HIOS"	"6"	"5/20/2015 14:1...	"16842"	"FL"	"16842FL0070106"	"16842FL007010
8	"2015"	"FL"	"16842"	"HIOS"	"6"	"5/20/2015 14:1...	"16842"	"FL"	"16842FL0070106"	"16842FL007010
9	"2015"	"FL"	"16842"	"HIOS"	"6"	"5/20/2015 14:1...	"16842"	"FL"	"16842FL0070106"	"16842FL007010
10	"2015"	"FL"	"16842"	"HIOS"	"6"	"5/20/2015 14:1...	"16842"	"FL"	"16842FL0070106"	"16842FL007010

Query executed successfully. | DESKTOP-0F2D8MI (14.0 RTM) | DESKTOP-0F2D8MI\A PEDD... | DBForPartitioning | 00:00:15 | 3,27,157 rows

Delete operation: partitioning:

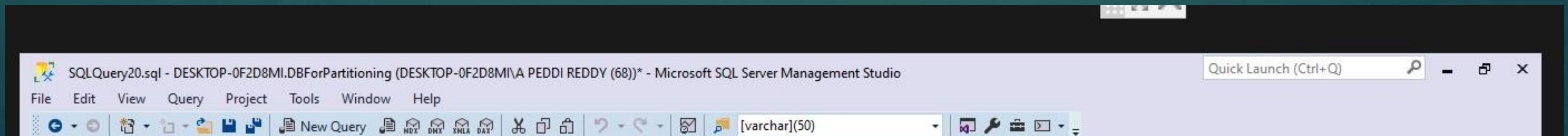


```
Set statistics io on
set statistics time on

Delete from dbo.Benefits_Cost_Sharing_PUF
set statistics io off
set statistics time off d
```

```
 SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 2 ms.

  SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.

  SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
 Table 'Benefits_Cost_Sharing_PUF'. Scan count 3, logical reads 253465, physical reads 0, read-ahead reads 0, lob logical reads

  SQL Server Execution Times:
   CPU time = 3906 ms,  elapsed time = 17289 ms.

 (327157 rows affected)
```

Query executed successfully. | DESKTOP-0F2D8MI (14.0 RTM) | DESKTOP-0F2D8MI\A PEDD... | DBForPartitioning | 00:00:17 | 0 rows

Inner join: Partitioning:

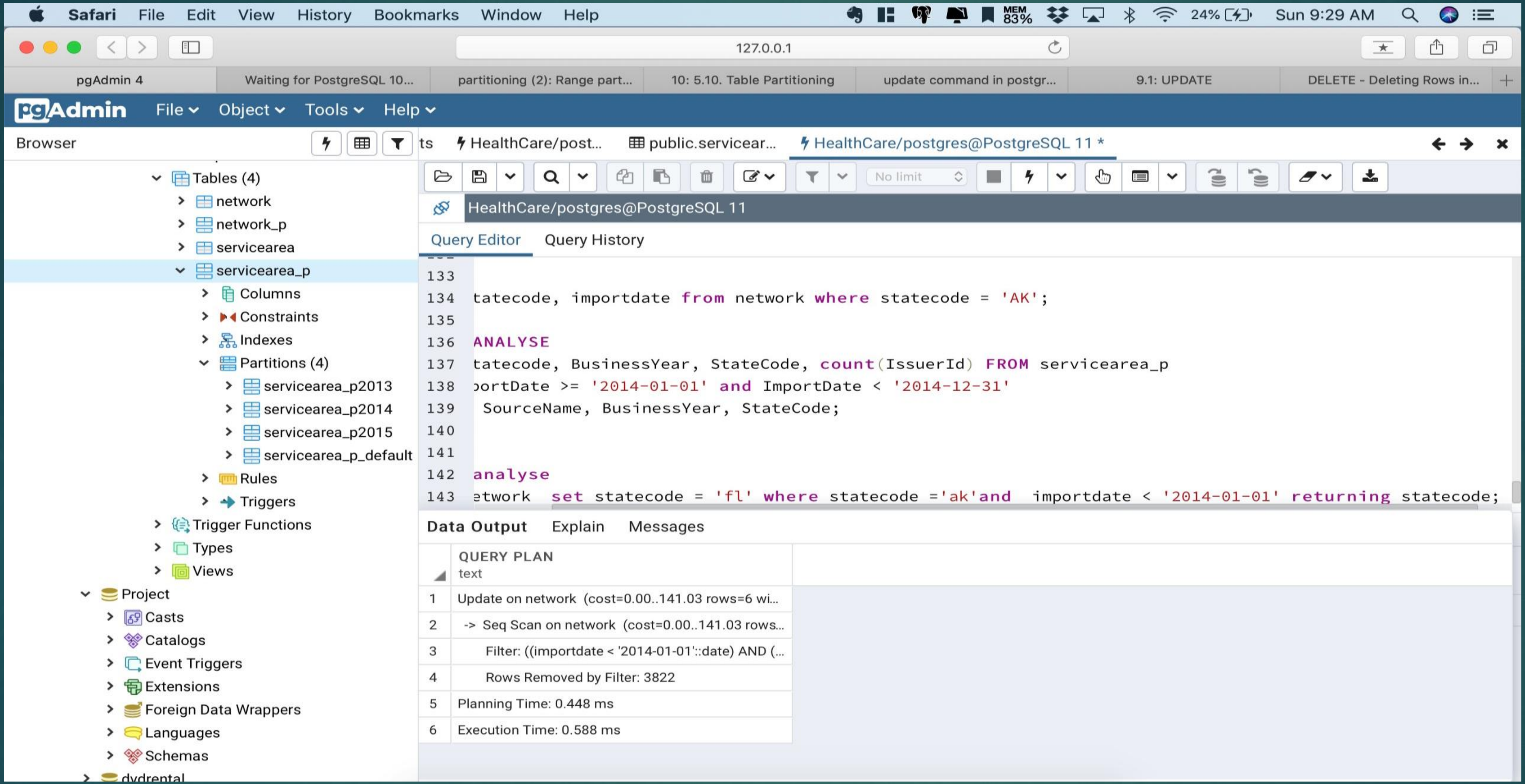

SQLQuery20.sql - DESKTOP-0F2D8MI.DBForPartitioning (DESKTOP-0F2D8MI\A PEDDI REDDY (68))* - Microsoft SQL Server Management Studio
Quick Launch (Ctrl+Q)
File
Edit
View
Query
Project
Tools
Window
Help
New Query
[varchar](50)

Range Partitioning:

- Range partitioning is used when we plan to access medium data aggregations/dumps on a frequent basis based on dates.
- “Service Area” attribute is chosen as the partition key for Range Partitioning.
- Below are the Query execution time of different operations performed on the Non-partitioned Vs Range Partitioned tables.

Operation Performed	Range Partition	Non partition
Select	20.620ms	64.029ms
Update	85.95ms	85.95ms

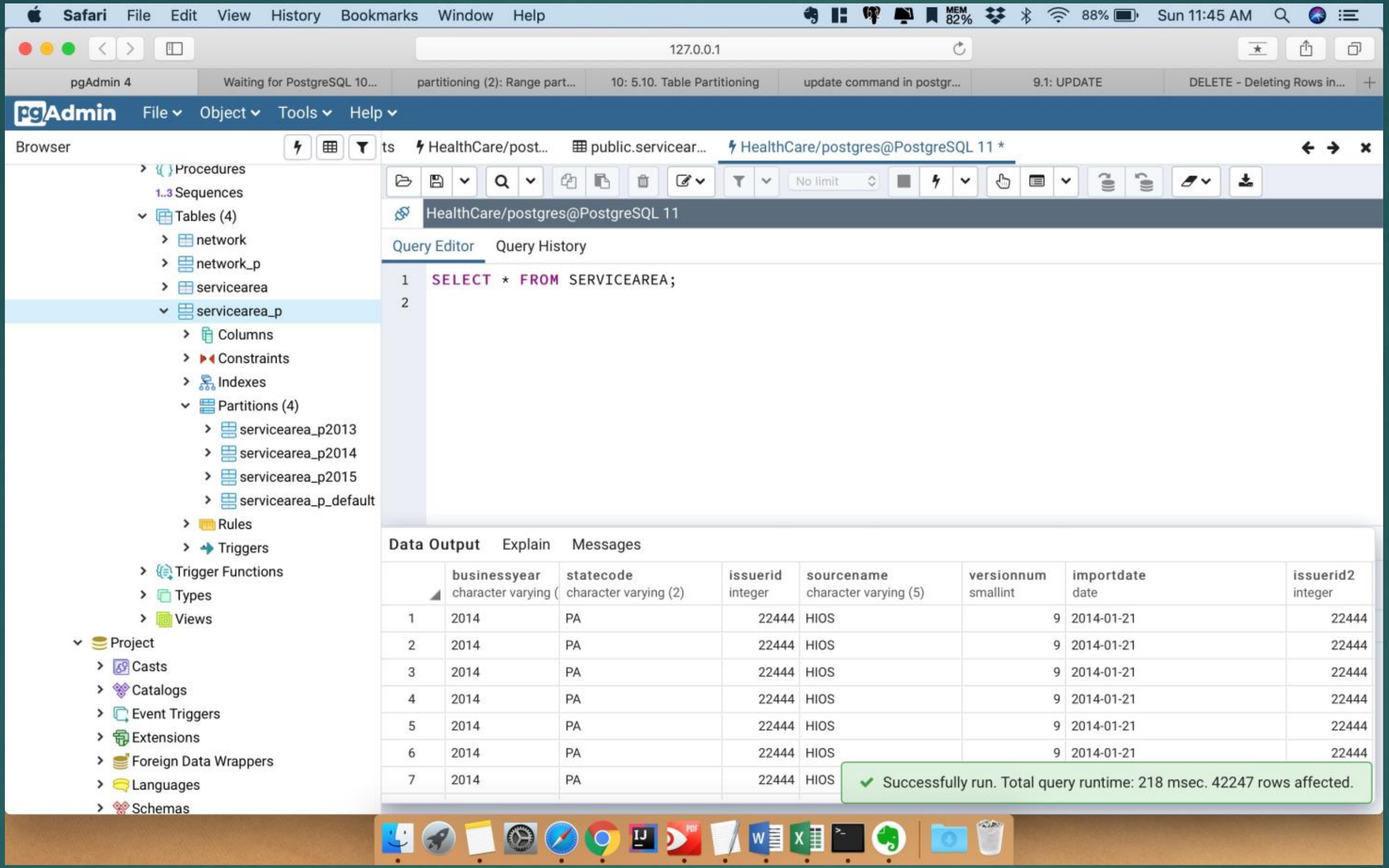
Range Partition:



```
tatecode, importdate from network where statecode = 'AK';

ANALYSE
tatecode, BusinessYear, StateCode, count(IssuerId) FROM servicearea_p
oortDate >= '2014-01-01' and ImportDate < '2014-12-31'
 SourceName, BusinessYear, StateCode;


analyse
etwork  set statecode = 'fl' where statecode ='ak'and  importdate < '2014-01-01' returning statecode;
```

	QUERY PLAN text
1	Update on network (cost=0.00..141.03 rows=6 wi...
2	-> Seq Scan on network (cost=0.00..141.03 rows...
3	Filter: ((importdate < '2014-01-01'::date) AND (...
4	Rows Removed by Filter: 3822
5	Planning Time: 0.448 ms
6	Execution Time: 0.588 ms

Select Operation: Partitioning:

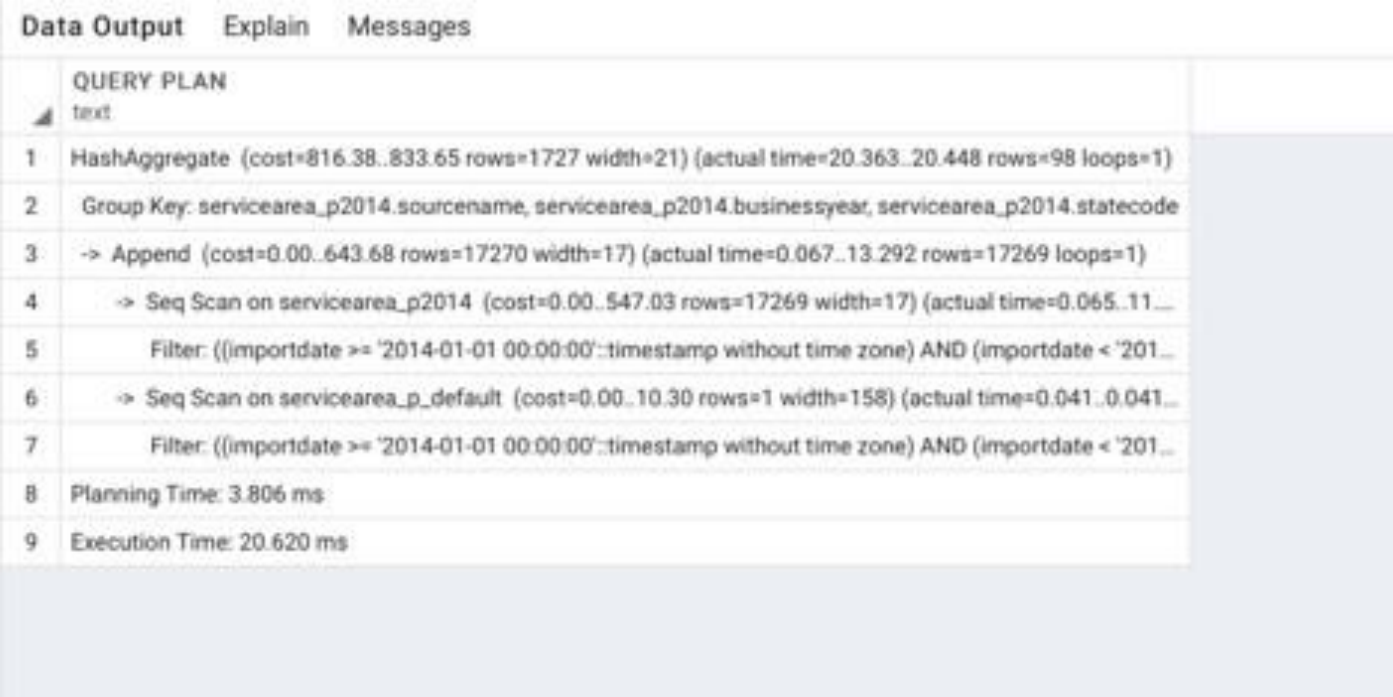


```
SELECT * FROM SERVICEAREA;
```

	businessyear character varying (	statecode character varying (2)	issuerid integer	sourcename character varying (5)	versionnum smallint	importdate date	issuerid2 integer
1	2014	PA	22444	HIOS	9	2014-01-21	22444
2	2014	PA	22444	HIOS	9	2014-01-21	22444
3	2014	PA	22444	HIOS	9	2014-01-21	22444
4	2014	PA	22444	HIOS	9	2014-01-21	22444
5	2014	PA	22444	HIOS	9	2014-01-21	22444
6	2014	PA	22444	HIOS	9	2014-01-21	22444
7	2014	PA	22444	HIOS			

✓ Successfully run. Total query runtime: 218 msec. 42247 rows affected.

Select Operation: Non-Partition:



```
   QUERY PLAN
   text
1  HashAggregate  (cost=816.38..833.65 rows=1727 width=21) (actual time=20.363..20.448 rows=98 loops=1)
2    Group Key: servicearea_p2014.sourcename, servicearea_p2014.businessyear, servicearea_p2014.statecode
3    ->  Append  (cost=0.00..643.68 rows=17270 width=17) (actual time=0.067..13.292 rows=17269 loops=1)
4          ->  Seq Scan on servicearea_p2014  (cost=0.00..547.03 rows=17269 width=17) (actual time=0.065..11....
5                Filter: ((importdate >= '2014-01-01 00:00:00'::timestamp without time zone) AND (importdate < '201...
6          ->  Seq Scan on servicearea_p_default  (cost=0.00..10.30 rows=1 width=158) (actual time=0.041..0.041...
7                Filter: ((importdate >= '2014-01-01 00:00:00'::timestamp without time zone) AND (importdate < '201...
8  Planning Time: 3.806 ms
9  Execution Time: 20.620 ms
```

ANALYSIS OF RESULTS:

- There is a significant improvement in the query performance of different operations.
- performed such as select, update, delete and inner join on partitioned tables compared to a non-partitioned table.

CONCLUSION:

- Partitioning can provide tremendous benefits to a wide variety of applications by improving performance, manageability, and availability.

- Partitioning improves the performance of certain queries by an order of magnitude.
- Significant improvement in performance can be achieved on tables that are significantly very large.

Pros & Cons of Partitioning :

Pros :

* Manageability
* Fast Data Deletion and Data Load
* Piecemeal backup / restore of historical data
* Performance querying Large Tables
* Join efficiency

Cons:

* Cannot span multiple DBs or instances
* Potentially use PV or DPVs a top Partitioned Tables

REFERENCES:

- Data set chosen https://www.kaggle.com/hhs/health-insurance-marketplace/ home.
- “MySQL 5.7 Reference Manual,” docs.oracle.com. [Online]. Available: https:// docs.oracle.com/cd/E17952_01/mysql-5.7-en/partititoning -range.html [Accessed: Nov. 26, 2019].

- “Database VLDB and partitioning Guide,”

docs.oracle.com. [Online]. Available: https://docs.oracle.com [Online]. Available: https://docs.oracle.com/database/121/VLDBG/GUID-BE424ACC-F746-4CA8-973C-F578CF98FF10.htm#VLDBG00225 [Accessed: Oct. 26, 2019].

- Eleni Markou, “SQL Database, Table and data Partitioning: When and How to Do It,” November 3, 2017. [Online]. Available: https://www.blendo.co/blog/sql-tableand-data-partitioning-howto/ [Accessed: Sep. 13, 2019].

[redacted]