PARTITION TECHNIQUES ON HEALTH INSURANCE MARKETPLACE

ABSTRACT:
Partitioning is the database process where very large tables are divided into multiple smaller parts. By splitting a large table into smaller, individual tables, queries that access only a fraction of the data can run faster because there is less data to scan. The main of goal of partitioning is to aid in maintenance of large tables and to reduce the overall response time to read and load data for particular SQL operations. Partitioning enhances the performance, manageability, and availability of a wide variety of applications and helps reduce the total cost of ownership for storing large amounts of data. However, proper database is a much more complex issue than described here. We analyzed the performance of Health insurance marketplace data by performing some simple queries. These improvisations of performance are discussed in detail in this document.

Keywords: Health Insurance, Health Benefits, Query Performance, Partitioning, Data tables, Medicare, SSMS, SSDT.

I. INTRODUCTION

The term "health insurance "is used to describe any form of insurance providing protection against the costs of medical services.This usage includes private insurance and social insurance programs such as Medicare, which pools resources and reduces the financial risk associated with major medical expenses across the entire population to protect everyone. In addition to that, social welfare programs like Medicaid and the Children's Health Insurance Program provides assistance to people who cannot afford health coverage.

For such applications there are millions of transactions and umpteen numbers of records in each table. In order to view best plans or any monthly reports, the processing time would be more and is tedious. The optimum and feasible solution to retrieve the data in such cases can be achieved by performing partitioning[3]-[5] on table.

Different partitioning techniques considered[5]

Vertical table partitioning

Vertical partitioning is mainly used to increase SQL Server performance where a query retrieves all columns from a table that contains a number of very wide text or BLOB columns that can be split to its own table with distinct subset of columns along with the same number of rows ,different columns

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

A. *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.

B. *Range Partitioning*

A table that is partitioned by range is partitioned in such a way that each partition contains rows for which the partitioning expression value lies within a given range. Ranges should be contiguous but not overlapping [3]. Range partitioning is widely used when we plan to access medium data aggregations/dumps on a frequent basis based on dates (partition key column).

II. METHODOLOGY

A. *Dataset Chosen*

Open Source Dataset: **https://www.kaggle.com/hhs/health-insurance-marketplace**[1]

Health Insurance Market place dataset encompasses of information pertaining to the below areas:

- How do plan rates and benefits vary across states?
- How do plan benefits relate to plan rates?
- How do plan rates vary by age?
- How do plans vary across insurance network providers?

2.1 Files to be used in the dataset[1]

2.1.1.BenefitsCostSharing.csv: 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.

2.1.2.BusinessRules.csv:
Rules associated with each plan are distinguished based on TIN Number. This helps in determining the price for the user.

2.1.3.Network.csv:
It contains details of all the network coverage areas for each plan. **2.1.4.PlanAttributes.csv*:***
It contains details of each plan with all the covered benefits under each plan. Columns under this are of different plan attributes.

2.1.5.Rate.csv:
It describes the variables contained in the RatePUF. 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, 2015, and 2016.

2.1.6.ServiceArea.csv*:*
It describes each plan and their corresponding service areas associated with it.

2.1.7.Cross walk 2014:
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 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.

2.1.8. 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 2016 to plans that will be offered through the Marketplaces in 2017. 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.

To implement partitioning (Vertical,list and Range) on the tables, above open source dataset will be used. Main focus of the project will be to analyze the real-world scenarios using insurance tables and also compare the results with and without partitioning tables.

B. 2.2. Data Import:[5]

Data is imported from above mentioned .csv files into SQL server using SSIS tool. Data Flow Task is used for loading data from above mentioned csv files (Source) to SQL Server databases (target). Preliminary analysis on the input fields that were causing issue for data extraction i.e., (fields having blank data and fields having data like paragraphs) was completed. The properties of the output columns of the source files were modified such that data can be loaded into tables from source files. In Data flow task, "flat file source" is used as source component and "OLE DB Destination" as destination component. Connection managers utilized for the data flow task:

- ➢ Flat file Connection manager
- ➢ OLE DB connection using SQL Server Native Client 11.0(Provider)

Select the CSV file, which needs to be imported and give the destination path correctly with the required parameters.

Once the data is loaded successfully, data is depicted as below

TABLE NAME	ROWS
BenefitsCostSharing	50578788
BusinessRules	27876
Network	2522
PlanAttributes	77853

Rate	12694445
ServiceArea	52570
Crosswalk2015	279300
Crosswalk2016	315000

C. **Softwares Used[6]**

- **SSMSTOOL:** SQL Server Management Studio (SSMS) is used to query, design, and manage your databases and data warehouses. SSMS tool can also be used to load data to SQL Server installed on machine.
- **Pgadmin4**

D. **Chosen Partition Tables**

We will be using the below mentioned attributes as the partitioning Tables for the respective partitions:

1. Vertical Partitioning: "Crosswalk2015" & "CrossWalk2016"
2. List Partitioning: "BenefitsCostSharing".
3. Range Partitioning: "Network" & "Service area" column.

III. Experiments and Results

Multiple columns to be Partitioned are identified based on testing scenarios. Partitioning are implemented on different columns. Partition is created on dbo.BenefitsCostsSharing table on IssuerID Column and dbo.BenefitsCostsSharing table on BenefitName, StateCode, PlanId, CoinsOutofNet columns. Below table represents the various columns, which are Partitioned for all the six tables of the data set. **Indexed Columns of Data:**

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,Market Coverage
CrossWalk2015	IssuerID	PlanId,Dentalcode, Fipscode,state
Crosswalk2016	IssuerID	PlanId,Dentalcode, Fipscode,state

Perform different partitions on the dataset and perform queries on the partitioned tables.

A. Vertical partitioning

Created and Performed vertical Partitioning on tables:

- Crosswalk2014_by vertical
- Crosswalk2015_by vertical

Queries implemented on vertical Partitioned Table:

- SELECT * FROM CROSSWALK2015$_PARTITION_201
  4_1 WHERE MetalLevel_2014 LIKE '%GOLD%'
  Query Execution Time: 0.02 Seconds
- UPDATE CROSSWALK2015$_PARTITION_201
  4_1 SET STATE_ID='233' WHERE STATE_ID=23

  Query Execution Time: 0.11 Seconds
- DELETE FROM CROSSWALK2015$
  Query Execution Time: 0.38 Seconds
- SELECT * FROM CROSSWALK2015$_PARTITION_201
  4_1 AS A INNER JOIN CROSSWALK2016$ AS B ON A.PlanID_2014=B.PLANID_2015 AND A.IssuerID_2014=B.ISSUERID_2015 AND A.FIPSCODE=B.FIPSCODE WHERE A.PlanID_2014='73231GA0070002'
  Query Execution Time = 0.10 Seconds

Queries implemented on Non-Partitioned Table:

- SELECT * FROM CROSSWALK2015$WHERE MetalLevel_2014 LIKE '%GOLD%'
  Query Execution Time: 0.114Seconds
- update CROSSWALK2015$ set state='Ix' where state='IA';
  Query Execution Time: 2.80 Seconds
- DELETE FROM CROSSWALK2015$
  Query Execution Time: 3.30 Seconds
- SELECT * FROM CROSSWALK2015$ AS A INNER JOIN CROSSWALK2016$ AS B ON A.PLANID_2015=B.PLANID_2015 AND A.IssuerID_2015=B.ISSUERID_2015 AND A.FIPSCODE=B.FIPSCODE WHERE A.PlanID_2015='73231GA0070002';
  Query Execution Time = 10.0 Seconds

B. *list Partitioning*

Created and Performed list Partitioning on tables:

- Benefits cost sharing _by list *Queries implemented on Range Partitioned Table:*
- Select ["StateCode"],* From dbo.Benefits_Cost_Sharing_PUF
  Query Execution Time = 0.15 Seconds
- update dbo.Benefits_Cost_Sharing_PUF set ["VersionNum"]=4 Where id< 10000;
  Query Execution Time = 0.97 Seconds
- Delete from dbo.Benefits_Cost_Sharing_PUF;
  Query Execution Time = 0.20 Seconds
- Select * From dbo.Benefits_Cost_Sharing_PUF) d inner join dbo.Network von d.["BusinessYear"]=v.[BusinessYear];
  Query Execution Time = 0.7 Seconds

Queries implemented on Non-Partitioned Table:

- Select ["StateCode"],* From dbo.Benefits_Cost_Sharing_PUF1;
  Query Execution Time = 1.23 Seconds
- update dbo.Benefits_Cost_Sharing_PUF set ["VersionNum"]=4 Where id< 10000;
  Query Execution Time = 5.501 Seconds
- Delete from dbo.Benefits_Cost_Sharing_PUF;
  Query Execution Time = 6.036 Seconds
- Select * From dbo.Benefits_Cost_Sharing_PUF) d inner join dbo.Network von d.["BusinessYear"]=v.[BusinessYear];
  Query Execution Time = 5.721 Seconds

C. *Range Partitioning*

Created and Performed Range Partitioning on tables:

- Network by Range
- Service area by Range *Queries implemented on Range Partitioned Table:*
- Select * from Service area;
  Query Execution Time = 0.20 Seconds.
- UPDATE service area SET market coverage = 'NOT AVAILABLE' WHERE market coverage IS NULL AND IMPORTDATE >= '2014-01-01' AND IMPORTDATE < '2014-12-31';
  Query Execution Time = 0.05 Seconds.
- Delete,Join cannot be performed on service area as it is a unique attribute.

Queries implemented on Non-Partitioned Table:

- Select * from Service area P;
  Query Execution Time = 0.64 Seconds.
- UPDATE service area_p SET market coverage = 'NOT AVAILABLE' WHERE market coverage IS NULL AND IMPORTDATE >= '2014-01-01' AND IMPORTDATE < '2014-12-31';
  Query Execution Time = 0.85 Seconds.
- Delete,Join cannot be performed on crime_id as it is a unique attribute.

D. *Approach to Partitioning:*

- Data in a partitioned table is partitioned based on a single column, the partition column, which is called as the partition key.
- It is important to select a partition column that is almost always used as a filter in queries. When the partition column is used as a filter in queries, SQL Server can access only the relevant partitions. This is called partition elimination and can greatly improve performance when querying large tables.
- If the partition column value is NULL, the rows are placed in the first partition.
- Partition Function: The partition function defines how to partition databased on the partition column. The partition function does not explicitly define the partitions and which rows are placed in each partition. Instead, the partition function specifies boundary values, the points between partitions. The total number of partitions is always the total number of boundary values +
1.
- Partition Scheme: The partition scheme maps the logical partitions to physical filegroups. It is possible to map each partition to its own filegroup or all partitions to one filegroup.

IV. ANALYSIS OF RESULTS

Table 1. shows the summary of the query execution time for different operations such as select, update, delete,and inner join performed on different Partitioned tables and a NonPartitioned table. There a significant improvement in the query performance of select, update, delete, inner join operations on partitioned tables compared to a non-partitioned table.

A. Advantages:

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

B. Disadvantages:

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

Table 1. Summary of Query Execution time on Partitioned tables and Non-Partitioned table

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
	Inner Join	0.10	10.0
List	Select	0.15	1.23
	Update	0.97	5.501
	Delete	0.20	6.036
	Inner Join	0.7	5.721
Range	Select	20.620ms	64.029ms
	Update	85.95ms	106.ms

V. CONCLUSION

The Health Insurance Marketplace Public Uses files that contain data on health and dental plans offered to individuals and small-scale businesses through the US Health Insurance Marketplace.
The Health Insurance Market Place uses the Partitioning techniques, which improves the query plan, query runtime and the performance of the application.
Overall partition performance is good for this data set when compared.
It seems like the query performance using partitioning is better when we have to deal with large tables involving complex queries while joining multiple tables.

REFERENCES [1]

Dataset: https://www.kaggle.com/hhs/ health-insurance-marketplace [2] Oracle, supported by Oracle representative, ”My SQL 5.7 Reference Manual,” in docs.oracle.com,[online], Nov 28, 2019. Available: https://docs.oracle.com/cd/ E17952_01/mysql-5.7-en/ [Accessed: Oct. 26,2019].
[3] Oracle and its affiliates, “Database VLDB and partititioning Guide,” in

docs.oracle.com, [online], 2018. Available: https://docs.oracle.com/database/121/VLDBG/ index.html [Accessed: Sep. 13,2019].

[4] Medic Milica, “Database table partitioning in SQL Server,” in sqlshack.com,
[online], April 4, 2014. Available: https://www.sqlshack.com/database-table-partitioningsql-server/ [Accessed: Sep 14,2019].

[5] Edgewood Solutions, MSSQL tips,
“SQL Server Partitioning Tips,” in
mssqltips.com, [online], 2006. Available:
https://www.mssqltips.com/sql-server-tipcategory/65/partitioning/ [Accessed: Oct.12, 2019].