<https://www.sqlshack.com/sql-server-partitioned-views/#:~:text=A%20Distributed%20Partitioned%20View%20contains,processing%20load%20across%20multiple%20servers>.

SQL Server Partitioned Views

November 18, 2016 by [Ahmad Yaseen](https://www.sqlshack.com/author/ahmad-yaseen/)

[](https://www.quest.com/solutions/sql-server/?utm_source=sql_shack&utm_medium=Direct-Outbound%20Marketing&utm_campaign=FY2024_Q2_AMER_SQLServer_SQLShack_ADV&utm_term=&utm_content=banner)

SQL Server table partitioning is a great feature that can be used to split large tables into multiple smaller tables, transparently. It allows you to store your data in many filegroups and keep the database files in different disk drives, with the ability to move the data in and out the partitioned tables easily. A common example for tables partitioning is archiving old data to slow disk drives and use the fast ones to store the frequently accessed data. Table partitioning improves query performance by excluding the partitions that are not needed in the result set. But table partitioning is available only in the Enterprise SQL Server Edition, which is not easy to upgrade to for most of small and medium companies due to its expensive license cost.

Fortunately, SQL Server allows you to design your own partitioning solution without the need to upgrade your current SQL Server instance to Enterprise Edition. This new option is called **Partitioned Views**. Although this new solution is not as flexible as table partitioning, Partitioned Views will give you a good result if you design it properly. It is up to you to manually design the tables that will work as partitions and combine it together using the UNION ALL operator in the Partitioned View that will work like table partitioning.

SQL Server Partitioned Views enable you to logically split a huge amount of data that exist in large tables into smaller pieces of data ranges, based on specific column values, and store this data ranges in the participating tables. To achieve this, a CHECK constraint should be defined on the partitioning column to divide the data into data ranges. Then, a partitioned view that combines SELECTs from all participating tables as one result set using UNION ALL operator, will be created. The CHECK constraint is used to specify which table contains the requested data when selecting data from the view, which is similar to defining the portioning function in the table partitioning feature. The check constraint is used also to improve query performance, If the CHECK constraint is not defined in the participating tables, the SQL Server Query Optimizer will search in all participating tables within the view to return the result.

Partitioned Views don’t require defining a partitioning schema or function, have no specific syntax to swap the data in and out the tables and no need to figure out the RIGHT and LEFT data boundaries. In the same way as table partitioning, you can create each table in the Partitioned View in a separate filegroup on a different disk drive, where you can locate the old archived data in a slow drive and locate the most active data in a fast drive such as the solid state disks. But in an opposite approach to table partitioning, each table that participates in the Partitioned View is an individual story that you can ALTER or CREATE INDEX on individually and the tables can have different columns. Adding a new table to the Partitioned view can be done easily by modifying the view definition to include that new table.

Partitioned Views allow you to perform data changes on the partitioned view itself which will handle any changes in participating tables. To be able to update from the view, the view should combine the SELECTs from the participating tables using the UNION ALL operator, where each SELECT references to one of these local or linked tables and the partitioning column should be a part of the primary key on each participating table. If all the Partitioned View participating tables are located on the same SQL Server, the view is referred to as a **Local Partitioned View**. A **Distributed Partitioned View** contains participating tables from multiple SQL Server instances, which can be used to distribute the data processing load across multiple servers. Another advantage for the SQL Server Partitioned Views is that the underlying tables can participate in more than one Partitioned View, which could be helpful in some implementations.

Let’s go through our demo to understand practically how to implement the SQL Server Partitioned View and its benefits. We will start with creating four new tables under SQLShackDemo database with identical schema and each table keeps the shipments information for a specific quarter of the year. The partitioning column is the Ship\_Quarter column in which the constraint that specifies the four quarters is defined. The Ship\_Quarter column is also included in the Primary Key constraint. The T-SQL script to create the four quarters tables will be like:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | USE SQLShackDemo  GO  CREATE TABLE Shipments\_Q1 (  Ship\_Num INT NOT NULL,  Ship\_CountryCode CHAR(3) NOT NULL,  Ship\_Date DATETIME NULL,  Ship\_Quarter SMALLINT NOT NULL CONSTRAINT CK\_Ship\_Q1 CHECK (Ship\_Quarter = 1),  CONSTRAINT PK\_Shipments\_Q1 PRIMARY KEY (Ship\_Num, Ship\_Quarter)  );    GO  CREATE TABLE Shipments\_Q2 (  Ship\_Num INT NOT NULL,  Ship\_CountryCode CHAR(3) NOT NULL,  Ship\_Date DATETIME NULL,  Ship\_Quarter SMALLINT NOT NULL CONSTRAINT CK\_Ship\_Q2 CHECK (Ship\_Quarter = 2),  CONSTRAINT PK\_Shipments\_Q2 PRIMARY KEY (Ship\_Num, Ship\_Quarter)  );    GO  CREATE TABLE Shipments\_Q3 (  Ship\_Num INT NOT NULL,  Ship\_CountryCode CHAR(3) NOT NULL,  Ship\_Date DATETIME NULL,  Ship\_Quarter SMALLINT NOT NULL CONSTRAINT CK\_Ship\_Q3 CHECK (Ship\_Quarter = 3),  CONSTRAINT PK\_Shipments\_Q3 PRIMARY KEY (Ship\_Num, Ship\_Quarter)  );    GO  CREATE TABLE Shipments\_Q4 (  Ship\_Num INT NOT NULL,  Ship\_CountryCode CHAR(3) NOT NULL,  Ship\_Date DATETIME NULL,  Ship\_Quarter SMALLINT NOT NULL CONSTRAINT CK\_Ship\_Q4 CHECK (Ship\_Quarter = 4),  CONSTRAINT PK\_Shipments\_Q4 PRIMARY KEY (Ship\_Num, Ship\_Quarter)  ); |

Once the tables created successfully, we will create the Partitioned View that combines four the SELECT statement, one per each participating table, using the UNION ALL T-SQL statement as in the below CREATE VIEW statement:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | USE SQLShackDemo  GO  CREATE VIEW DBO.Shipments\_Info  WITH SCHEMABINDING  AS  SELECT [Ship\_Num],[Ship\_CountryCode],[Ship\_Date],[Ship\_Quarter] FROM DBO.Shipments\_Q1  UNION ALL  SELECT [Ship\_Num],[Ship\_CountryCode],[Ship\_Date],[Ship\_Quarter] FROM DBO.Shipments\_Q2  UNION ALL  SELECT [Ship\_Num],[Ship\_CountryCode],[Ship\_Date],[Ship\_Quarter] FROM DBO.Shipments\_Q3  UNION ALL  SELECT [Ship\_Num],[Ship\_CountryCode],[Ship\_Date],[Ship\_Quarter] FROM DBO.Shipments\_Q4 |

That’s it! Now the Partitioned View solution is ready to use. If we try to insert four values to the view directly, which is updatable in our case here as the view contains SELECT statement per each participant table, the tables combined by UNION ALL operator and the partitioning column is involved in the Primary Key constraint as described previously. The INSERT INTO T-SQL statement to the view will handle spreading the data through the participating tables depending on the CHECK constraint. If we run the below INSERT INTO statements:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | INSERT INTO DBO.Shipments\_Info VALUES(1117,'JOR',GETDATE(),1)  INSERT INTO DBO.Shipments\_Info VALUES(1118,'JFK',GETDATE(),2)  INSERT INTO DBO.Shipments\_Info VALUES(1119,'CAS',GETDATE(),3)  INSERT INTO DBO.Shipments\_Info VALUES(1120,'BEY',GETDATE(),4) |

And try to retrieve the data from the four participating tables:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | SELECT \* FROM DBO.Shipments\_Q1  GO  SELECT \* FROM DBO.Shipments\_Q2  GO  SELECT \* FROM DBO.Shipments\_Q3  GO  SELECT \* FROM DBO.Shipments\_Q4  GO |

The result will show us that one row will be inserted on each participating table depending on the Ship\_Quarter column value as follows:

A screenshot of a computer

Description automatically generated

Also, running a direct SELECT statement from the view itself:

|  |  |
| --- | --- |
| 1  2  3 | SELECT \* FROM DBO.Shipments\_Info |

Will combine each table’s result into one result set, showing the four inserted columns as below:

A screenshot of a computer

Description automatically generated

It is clear from the execution plan generated using the APEXSQL PLAN application that the SQL Server Query Optimizer will scan each participating table and finally concatenate all results into one result set using the UNION ALL operator as in the below execution plan:

A screenshot of a computer screen

Description automatically generated

If we plan to retrieve the third quarter data using a filter on the Ship\_Num column:

|  |  |
| --- | --- |
| 1  2  3 | SELECT \* FROM DBO.Shipments\_Info WHERE [Ship\_Num] = 1119 |

The SQL Server Query Optimizer will seek all participating tables for this value, which is part of the Primary Key on all tables, in order to retrieve the requested data as in the execution plan below generated using the APEXSQL PLAN application:

A screenshot of a computer

Description automatically generated

But if we search for the same data by filtering the Ship\_Quarter partitioning column:

|  |  |
| --- | --- |
| 1  2  3 | SELECT \* FROM DBO.Shipments\_Info WHERE [Ship\_Quarter] = 3 |

The SQL Server Query Optimizer will identify directly in which table it can find that data without the need to touch all participating tables as shown in the below execution plan generated using the APEXSQL PLAN application:

A screenshot of a computer program

Description automatically generated

Comparing the two queries performance by running it at the same time and enabling the STATISTICS TIME counter:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | SET STATISTICS TIME ON  SELECT \* FROM DBO.Shipments\_Info WHERE [Ship\_Num] = 1119  GO  SELECT \* FROM DBO.Shipments\_Info WHERE [Ship\_Quarter] = 3  SET STATISTICS TIME OFF |

It is clear from the result that using the partitioning column included in the CHECK constraint to filter the retrieved data will enhance the query performance four times in our situation here, as it will visit only the table that contains the requested data, which is become easier by defining the CHECK constraint in the participating tables. The below figure compares the performance of both queries execution plans:



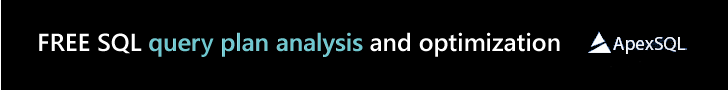
Also the time required to execute the first query is four to five times the time required to execute the second query searching on the partitioning column:



And the real difference can be touched when dealing with tables that contain millions of rows.

See more

Check out ApexSQL Plan, to [view SQL execution plans](https://www.apexsql.com/sql-tools-plan.aspx?utm_source=sqlshack&utm_campaign=plan&utm_medium=native_link&utm_content=sql-shack), including comparing plans, stored procedure performance profiling, missing index details, lazy profiling, wait times, plan execution history and more

[](https://www.apexsql.com/sql-tools-plan.aspx?utm_source=sqlshack&utm_campaign=plan&utm_medium=banner&utm_content=sql-shack)

[**Ahmad Yaseen**](https://www.sqlshack.com/author/ahmad-yaseen/)

Ahmad Yaseen is a Microsoft Big Data engineer with deep knowledge and experience in SQL BI, SQL Server Database Administration and Development fields.  
  
He is a Microsoft Certified Solution Expert in Data Management and Analytics, Microsoft Certified Solution Associate in SQL Database Administration and Development, Azure Developer Associate and Microsoft Certified Trainer.  
  
Also, he is contributing with his SQL tips in many blogs.  
  
[View all posts by Ahmad Yaseen](https://www.sqlshack.com/author/ahmad-yaseen/)

**Related posts:**

1. [**The SQL Server system views/tables/functions. Common questions and solutions to real life problems**](https://www.sqlshack.com/sql-server-system-viewstablesfunctions-common-questions-solutions-real-life-problems/)
2. [**CREATE VIEW SQL: Creating views in SQL Server**](https://www.sqlshack.com/create-view-sql-creating-views-in-sql-server/)
3. [**Introducing SQL Server Incremental Statistics for Partitioned Tables**](https://www.sqlshack.com/introducing-sql-server-incremental-statistics-for-partitioned-tables/)
4. [**FORCESCAN and Partitioned table in SQL Server**](https://www.sqlshack.com/forcescan-hint-and-partitioned-table/)
5. [**Options for Partitioned Tables and Indexes in SQL Server**](https://www.sqlshack.com/options-for-partitioned-tables-and-indexes-in-sql-server/)

[Partitioning](https://www.sqlshack.com/category/database-design/partitioning/), [Query analysis](https://www.sqlshack.com/category/sql-server-performance-tuning/query-analysis/), [Views](https://www.sqlshack.com/category/database-design/views/)