Overview

Partitioning refers to splitting a large table into smaller, more manageable pieces. It offers several benefits:

- **Improved Query Performance**: Partitioning can significantly boost performance, particularly when most heavily accessed rows are contained within one or a few partitions.
- **Efficient Indexing**: Partitioning can act as an upper-level index, improving memory efficiency by allowing the most frequently accessed parts of the data to be in memory.
- **Better Scan Performance**: When a query or update accesses a large portion of a single partition, it can benefit from a sequential scan, rather than the slower random-access reads needed for indexes across the entire table.
- **Cost-Effective Storage**: Infrequently used data can be moved to cheaper, slower storage media.

Partitioning is most beneficial for very large tables. A general rule of thumb is that the table should exceed the database server's physical memory capacity.

For more details, refer to the official PostgreSQL documentation on partitioning.

Approach

Our partitioning approach includes the following steps:

1. **Partitioning a non-partitioned table**: We support range-based partitioning based on a date attribute, with options for daily, weekly, and monthly partitions.
2. A table that is being partitioned can go for fixed retention period (meaning only those records are brought to the partition and going forward the retention period will be maintained) or indefinite partition (meaning no expiry - but the table is partitioned)
3. **Creating a scheduled partitioning strategy**: We use a cron scheduler to manage partition creation on a daily basis.
4. **Reconciliation between partitioned and non-partitioned tables**: This ensures data consistency between the original and partitioned tables.
5. **Logging**: Logs are created for analysis and debugging.

We currently only support range-based partitioning based on a date attribute. Users must determine the appropriate retention period for data, ensuring that older records beyond the retention period are excluded from partitioning (and potentially archived to external storage like S3).

Example:

- **Table Name**: Orders
- **Total Rows**: 45 Million
- **Retention Period**: 6 Months
- **Records in Retention Period**: 10 Million
- **Partition Type**: Weekly

The script will create 24 weekly partitions, containing 10 million records, and partition the table accordingly. This script is designed to be run once to partition a non-partitioned table.

Partition Management

After a table is partitioned, partitions must be created on an ongoing basis. A cron job is used to execute partition management scripts daily. More details on this process are outlined below.

Base Data Setup

1. **manage_PK_for_partition.sql -** Partition key of the table partitioned should be a primary key or a part of the composite primary key - This function will build the new primary key and associated indexes.
2. **List_and_drop_FK.sql** - This function scans the child tables that are dependent on the partitioned table using foreign key references and drops those references. A partitioned table should not have a foreign key enforced.
3. **Check_partition_continuity.sql -** This script is invoked on management partitions and looks at any gaps that may exist in the ongoing partitions.
4. **part_tbl.sql**: Creates the `part_tbl` table, which drives the partition management script.
5. **reconcile_partitioning.sql**: Used internally to reconcile the original table and partitioned table. These are stored in partition_logs table.
6. **create_logs.sql**: Logs and records activities across the partition creation and management scripts.
7. **TBL_PART_DTLS table is the meta table** table that will have the details of the partitions done. Please ensure that this table is not inserted / updated / deleted manually.

Scripts

Script 1 - partition_table.sql

Purpose: To partition a non-partitioned table into partitions.

Signature: `partition_table(<Table Name>, <Partition Key>, <Partition Type>, <Retention Period>, <Add to Management Strategy>,<Drop Existing Indexes>>, <<Array of excluded indexes>>, <<Drop FK Constraints>>)`

Attribute Name	Purpose	Default Values	Mandatory/ Optional	Data Type
p_table_name	Table name to partition	None	Mandatory	TEXT
p_partition_key	Attribute name for partitioning	None	Mandatory	TEXT
p_partition_type	Partition frequency: Daily (D), Weekly (W), Monthly (M)	None	Mandatory	CHAR(1)
p_retention_period	Records to retain before partitioning	None	Optional (Partition the entire table)	INTERVAL
p_add_to_reg_Maint	Add to ongoing partition management strategy	None	Optional (Default: TRUE) - Passover the metadata to manage partition scripts.	BOOLEAN
p_drop_avbl_indexes	If you want the existing indexes to be recreated then	NONE	Optional (Default: FALSE) -	BOOLEAN

	set this parameter to TRUE ** also refer to the next attribute.. If you intend doing manually set to FALSE			
p_exclude_indexes	When p_drop_avbl_indexes is set to TRUE and you need few indexes to be untouched then pass those indexes as an array	NONE	Optional default {}	TEXT[]
p_drop_fK_constraints	If the partitioned table is in a referential integrity with child tables, setting this attribute will cascade and drop all the referential integrity constraint on this table	NONE	BOOLEAN	BOOLEAN

Sample input

```
SELECT * FROM partition_table(
    'orders',                -- Table name
    'created_at',            -- Partition key
    'D',                     -- Partition type (e.g., 'D' for daily,
'M' for monthly)
    30,                      -- Retention period (e.g., 30 days)
    TRUE,                    -- Add to registry maintenance
```

```
    TRUE,                    -- Drop available indexes
    ARRAY['idx_orders_user', 'idx_orders_status'],  -- Excluded
indexes
    TRUE                      -- Drop foreign key constraints
);
```

Output:

- **Activity Name**: `Orders_Part_Maint`
- **Total Partitions**: Number of partitions created
- **Records Partitioned**: Number of records moved to partitions
- **Migration Status**: Success/Failure
- **Error Logs**: If any

This script performs the following tasks:

- Renames the original table by appending `_old` (e.g., `orders_old`).
- Modifies the primary key of the table to include the partition key. If the primary key did not exist then the partition key becomes the primary key.
- Drops old indexes associated with the previous primary key and creates new one - Also drops other indexes unless and until explicitly passed as an array as an input to this script.
- Cascades all child tables - and removes the foreign key constraints.
- Creates a new partitioned table with the same name as the original (e.g., `orders`).
- Creates the appropriate daily, weekly, or monthly partitions and attaches them to the partitioned table.
- When retention period is specified it only extracts that fall within the retention period. If the retention period is unspecified (NULL) then it partitions all the records in the source table.
- Creates default partition for any records that have futuristic dates (outliers)
- Inserts/updates `part_tbl` for future partition creation.
- Maintains the partition details in a meta table tbl_part_dtls

IMPORTANT: Retention frequency and Retention period used in the partition table is carried over as metadata and the same will be used for managing future partitions. So Please ensure that good due diligence is put onto determining these attributes, and should not be modified after the partition has been created.

Also ensure that tbl_part_dtls and part_tbl are not inserted / updated / deleted manually

```
SELECT * FROM partition_logs WHERE activity_name =
'orders_part_maint' ORDER BY log_time;
```

Script 2 - manage_partitions.sql

Purpose: Creates upcoming partitions based on daily, weekly, or monthly schedules.

Signature: `manage_partitions()`

This script does not take arguments. It is scheduled to run daily at 23:00 using cron. It uses the entries in the `part_tbl` to create partitions for the respective tables.

- **Daily Partition**: Creates a new daily partition and detaches the oldest partition .
- **Weekly Partition**: Creates a new weekly partition and detaches the oldest weekly partition (only if it's the end of the week).
- **Monthly Partition**: Creates a new monthly partition and detaches the oldest monthly partition (only if it's the end of the month).

(Please refer to the section on detached partitions)

If there were any referential integrity between this table and any other tables, then those referential integrity constraints will be dropped.

This script also looks at the continuity or gaps in partition - if there were any gaps then this function errors out - and manual intervention will be required.

This script will create one additional partition - to give some time for recovery in case any manual intervention may be required.

Also please note that if the retention period is NULL then no partitions are detached and only newer partitions are created. So in such scenarios if you want to maintain partitions you will have to manually detach them.

part_Tbl - This is a key table for managing partitions. An entry into this table should either be manually created (less desirable) or enabled through the partition_table.sql. This table has the following attributes.

Table_name	TEXT	Name of the table that gets through partition management.	Mandatory	This attribute is the PK.
partition_type	Char	Daily (D) / Weekly (W) / Monthly (M)	Mandatory	
retention_duration	TEXT	Interval of data retention period	Optonal	When Null - NEw partitions are created and no older partitions are detached

---

Detach Partitions

As partitions are managed - newer partitions get added - and older partitions based on the configuration of retention_period in the part_tbl will be detached from the existing tables. If the retention_period in the part_tbl is NULL then it implies that no partition needs to be detached.

When the partitions are detached - the object will no longer be linked to the parent table, but will appear as an individual table in the schema. Any fetch to the original table on the detached rows will not fetch any data. Since the detached partition is available in the schema - Based on the need you may want to export the data and move them to S3 to claim the detached partition space.

Ex: Table: Orders

Retention period: 4 weeks

Partition type: Weekly

So you will have 5 partitions as below

- Orders_Weekly_13012025
- Orders_Weekly_20012025
- Orders_Weekly_27012025
- Orders_Weekly_03022025
- Orders_Weekly_10022025 (New Partitions that was added)
- Orders_weekly_17022025 (Additional partition)

As managed partition runs on 9th feb 2025 this will create a new partition for the next week beginning 10th Feb and as well create one additional partition for 17th Feb. The oldest partition for 13012025 (in this example) will be detached.

- Any fetch to orders table will only fetch data on the existing partition and not the detached partition.
- Orders_Weekly_13012025 will be directly now attached to the schema.
- We can create a new script to export data from the detached partitions and store them in S3 - and import them back in case of any need.
- Detaching a partition will not claim any space till the data is moved and the object being dropped.

Test Scenarios

Test Case 1: Daily Partition with Retention and Maintenance

- **Signature**: `partition_table('Orders', 'Order_Date', 'D', '180 Days', TRUE,TRUE,'{}',TRUE)`
- **Expected Output**:
  - **Activity**: Orders_Part_Maint
  - **Partitions Created**: 180
  - **Records in Partitions**: 2,345,678
  - **Records in Original Table**: 12,345,678
  - **Status**: SUCCESS
  - **Error Count**: 0

Also, a record should be created in `part_tbl` with partition type `D` (Daily) and retention duration `180 Days`.

Test Case 2: Daily Partition with No Retention Period

- **Signature**: `partition_table('Orders', 'Order_Date', 'D', NULL, TRUE, FALSE,'{}',TRUE)`
- **Expected Output**:
  - **Partitions Created**: (current date - Min(Order_date)
  - **Records in Partitions**: 45,678
  - **Records in Original Table**: 12,345,678
  - **Status**: SUCCESS
  - **Error Count**: 0

Manage Partitions Test Script

Given the following entries in `part_tbl`:

Table Name	Partition Type	Retention Duration
Orders	D	60
Customer	M	6
Shipping	W	9

The `manage_partitions` script runs daily. Below are example test cases for different scenarios:

1. **Test Case 1**: Run day is a weekday and not the end of the month/Week. Only the daily partitions for `Orders` will be created and one additional partitions will be created for the future date and the oldest daily partition detached.
2. **Test Case 2**: Run day is a Sunday, and it's not the end of the month. Weekly partitions for `Shipping` will be created and daily partitions for `Orders` will also be created.Oldest monthly partitions for `Shipping` and `Orders` will be detached.
3. **Test Case 3**: Run day is a weekday, and it's the end of the month. Monthly partitions for `Customers` will be updated and daily partitions for `Orders` will also be created.Oldest partitions for `Customers` and `Orders` will be detached.
4. **Test Case 4**: Run day is a Sunday, and it's the end of the month. All partition types for `order` , `customer` and `Shipping` will be created. Oldest partitions for `Customers, Shipping` and `Orders` will be detached.

Recommendations

We strongly recommend the following steps before you partition an existing table.

1) Please expect some downtime - You may want to get a comfortable window to do the performance maintenance.
2) Though the script will backup the original table suffixing with _old - We would recommend a full backup of the database (or at least the table at play) before the partition is run.
3) Please ensure that, there is no foreign key constraint defined on any other table that is referring to the primary key /Unique constraint of the partitioned table.
4) Please ensure to set enable_partition_pruning = “on” - this is important for the planner to enable partition pruning, resulting in better performance.
5) Partition is meant to improve performance - Please ensure that Select statements use the partition key in the where clause. This will enable the query planner to exclude unnecessary partitions from being scanned . **You might need to rewrite the query to make it more conducive to partition pruning.**
6) Please ensure to remove foreign key relationships on the child-tables as otherwise detaching partitions will result in errors.
7) There is a link on the partition script to the managed partitions script - Partition script will populate the meta data like partition_type, retention period for partition management. Please ensure that you have thought through these. These metadata are populated in the part_tbl, please do not edit these attributes.
8) Managed partitions use the cron.schedule to execute managed partitions - Please ensure that necessary extensions and pgcron are enabled for execution. Please follow instructions at https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/PostgreSQL_pg_cron.html
9) Please watch out the logs in the partiton_logs table..
    - Activity_name for partition enablement will appear as <<Table_name>>_part_maint
    - Activity_name for manage partitions will appear as <<Table_name>>_Manage_partitions

```
SELECT * FROM partition_logs WHERE activity_name =
'orders_part_maint' ORDER BY log_time;

SELECT * FROM partition_logs WHERE activity_name =
'orders_Manage_partitions' ORDER BY log_time;
```

10) Ensure that PostgreSQL has up-to-date statistics for the table and partitions. Run the ANALYZE command on your partitioned table to collect statistics:

    ANALYZE partitioned_table;

Conclusion

This documentation covers partitioning strategy implementation, setup, and maintenance. It provides details on partition creation, management, and error logging, ensuring the partitioned tables are efficiently managed while maintaining the data's integrity.

Appendix

https://www.google.com/url?sa=t&source=web&cd=&ved=2ahUKEwj5iuDppqqLAxUYzzgGHUfJEP8QFnoECBAQBQ&url=https%3A%2F%2Fpostgresqlblog.hashnode.dev%2Fhow-to-fix-and-improve-partition-pruning-in-postgresql-for-faster-queries&usg=AOvVaw1W7LckBvmB4GxP6TJGE0OA&opi=89978449

https://sematext.com/blog/postgresql-performance-tuning/

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/PostgreSQL_pg_cron.html

Back pocket Scripts

If you want to reconstruct a partitioned table to an non-partitioned table then you can use the non_partionted_table.sql

If you want to export a detached partition you can use export_table_to_file.sql - at this time the file is generated. You still have to drop the table manually and move them to S3.