Implementing System Versioned Temporal Table

Surafel Temesgen Mamo
Pgcon 2020

About me

- Surafel Temesgen
- I am a dba
- I contribute to PostgreSQL
- @surafelTem

Agenda

- Definition
- Use Case
- Implementation Options
- Temporal Query

Temporal Table(1)

- System versioned temporal table is SQL standard.
- It is about retaining of past record alone with current record automatically by database management system and ability of queering both current and history record

Temporal Table(2)

- There are also an application time period which are for meeting the requirements of applications. It is based on application specific time periods which is valid in the business world
- A table can also be both a system versioned and an application time period table

Usage

- Recovery
- Auditing
- Can be used in place of application time period in same use case
- Trend analysis

Implementation Options

- System versioned temporal table can be implemented in two ways depends on the location of old record
- There are a wiki page describing the design for implement it using two tables
- But I go with one table approach

Two Tables Approach

- Involves two tables
- One is current table for current data storage
- The other is history table for historical data storage
- Old row inserts to history table using trigger
- Temporal queries satisfy by the union of the two tables
- Multiple technical columns have to be created implicitly

One Table Approach

- Both current and history record stores in one table
- Uses row end time column for record classification
- System versioning columns treat like a kind of generated column
- History data filter clause adds to non-temporal query

CREATE TABLE(1)

Can be specifies like

CREATE TABLE t (a integer PRIMARY KEY, start_timestamptz timestamp with time zone GENERATED ALWAYS AS ROW START, end_timestamptz timestamp with time zone GENERATED ALWAYS AS ROW END, PERIOD FOR SYSTEM_TIME (start_timestamptz, end_timestamptz)) WITH SYSTEM VERSIONING;

or

CREATE TABLE t (a integer PRIMARY KEY) WITH SYSTEM VERSIONING;

CREATE TABLE(2)

- Both current and history data stores in one table
- Row end time column adds to primary and unique key constraint to avoid conflict between current and history data
- Table partitioning can be used to minimize performance impact to non temporal query

INSERT

- System versioning columns values set automatically
- Row start time column fills with current transaction time and row end time column fills with infinity

UPDATE

- For non-system versioned table, update operation performed by marking a tuple to be updated to delete, and then insert the update tuple into the table
- In system versioning, to be deleted tuple will be inserts with row end time column sets to current transaction time
- For updated tuple row start and end time columns set to current transaction time and infinity respectively

DELETE

- Delete became logical
- Row end time column sets to current transaction time and inserts

SELECT

- System version table have to be upward compatible
- Filter condition adds to non-temporal query to filter out history record

Advantages Of One Table Approach

- Alter table is simple
- No need of primary key or technical columns
- No union operation
- It have optimization opportunity
- It can benefits from partition pruning

AS OF <tp>

- Its used to see the current records in specified point in time
- All the records that have row start time column value less than specified point in time and row end time column value greater or equal to specified point in time will be returned

e.g SELECT * FROM t FOR system_time AS OF 'ts' ORDER BY start_timestamp, a;

FROM <tp1> TO <tp2>

- Its return all the records that were current at any point between tp1 and tp2, including tp1, but excluding tp2

  e.g SELECT * FROM t FOR system_time FROM 'ts1' TO 'ts2' ORDER BY start_timestamp, a;

BETWEEN <tp1> AND <tp2>

- It returns all the records that were current at any point between tp1 and tp2, row visible exactly at tp1 or exactly at tp2 will be returned .

  e.g SELECT * FROM t FOR system_time BETWEEN ASYMMETRIC 'ts1' AND 'ts2' ORDER BY start_timestamp, a;

BETWEEN ASYMMETRIC <tp1>AND<tp2>

- It is the same as BETWEEN <tp1> and <tp2>

  e,g SELECT * FROM t FOR system_time BETWEEN ASYMMETRIC 'ts1' AND 'ts2' ORDER BY start_timestamp, a;

BETWEEN SYMMETRIC <tp1>AND<tp2>

- Returns all the records that were current at any point between the least and greatest time point between tp1 and tp2
- Row current exactly at least time point or exactly at greater time point will be returned

  e.g SELECT a FROM t FOR system_time BETWEEN SYMMETRIC 'ts1' AND 'ts2' ORDER BY start_timestamp, a;

Add System Versioning(1)

```
ALTER TABLE t ADD SYSTEM VERSIONING;
```

- Adds system versioning to table
- Uses default system versioning columns
- If the table is not empty the records will be set to current data

Add System Versioning(2)

- System versioning can also enabled by issuing ADD COLUMN statement to system versioning columns in one command

  e.g ALTER TABLE t ADD COLUMN start timestamptz GENERATED ALWAYS AS ROW START, ADD COLUMN end timestamptz GENERATED ALWAYS AS ROW END;

Remove System Versioning(1)

```
ALTER TABLE t DROP SYSTEM
VERSIONING;
```

- It removes system versioning from the table
- System versioning columns will be dropped too
- If the table contain history record, it removed together as per the standard

Remove System Versioning(2)

- System versioning can also be disabled by issuing DROP column statement to system time columns in one command

  e.g ALTER TABLE t DROP COLUMN start , DROP COLUMN end;

Thank You