Pivotal

Petabyte Scale Data Warehousing Greenplum

Storage Considerations

Postgres Conf 2018

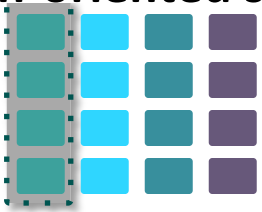
Marshall Presser
Craig Sylvester
Andreas Scherbaum
17 April 2018

Polymorphic Storage
Row-Oriented and Column-Oriented Tables

Row-oriented storage

- Supports mixed workloads (INSERT, UPDATE, DELETE, SELECT)
- Is supported with on both heap and append-optimized storage

Column-oriented storage



- Works well with data warehouse workloads
- Works well for data where you aggregate over a small number of columns
- Efficient for data where you modify a single column
- Supported on append-optimized storage

Pivotal

Table Storage Models

Customer	
PK	Customer_ID
	Customer_Name
	Customer_Street_Addr
	Customer_City
	Customer_State
	Customer_Country
	Customer_Postal_Code

Heap storage

- Default storage model
- Supports `INSERT`, `UPDATE`, `DELETE`
- Best for:
  - Data that is often modified
  - Smaller dimension tables
- Supports row-oriented tables
- Uses MVCC to support transactions

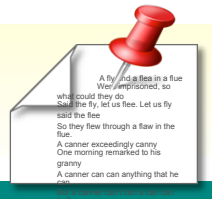
Append-optimized storage

- Append-optimized storage model:
- Optimized for data warehouses
- Works best with denormalized data
- Supports `UPDATE` and `DELETE`
- Best for:
  - Older data
  - Large fact tables
- Supports row and column-oriented tables
- Supports in-database compression
- Uses a Visibility Map (visimap) to hide outdated rows

Pivotal

Creating Heap and Append-Optimized Tables

Action	Example
Creating a heap, row-oriented table	`CREATE TABLE tc_heap (id int, descr text) DISTRIBUTED BY (id);`
Creating an append-optimized, row-oriented table	`CREATE TABLE tc_ao (id int, sales float) WITH (appendonly=true) DISTRIBUTED BY (id);`
Creating an append-optimized, column-oriented table	`CREATE TABLE tc_ao_c (id int, sales float) WITH (appendonly=true, orientation=column) DISTRIBUTED BY (id);`



Note: You cannot modify the storage or orientation of a table once defined. You can create a new table with the desired options and migrate your data.

Defining Append-Optimized Compression Tables

Action	Example

Creating a zlib compressed table with compression level 5	`CREATE TABLE tc_ao_zlib5 (id int, sales float)`
	`WITH (appendonly=true, compresstype=zlib, compresslevel=5)`
	`DISTRIBUTED BY (id);`

Creating a quicklz compressed table	`CREATE TABLE tc_ao_quicklz (id int, sales float)`
	`WITH (appendonly=true, compresstype=quicklz)`
	`DISTRIBUTED BY (id);`

Creating an AO table with an RLE compressed column and a zlib compressed column	`CREATE TABLE tc_ao_rletype (`
	`id int,`
	`sales float ENCODING (compresstype=zlib,`
	`compresslevel=3),`
	`salesdate date ENCODING (compresstype=rle_type))`
	`WITH (appendonly=true, orientation=column)`
	`DISTRIBUTED BY (id);`

Pivotal

Compressing Table Data

Compression Algorithm	Compression Levels	Description	Table-Level Compression	Row-Level Compression
ZLIB	1 - 9	Offers the most compact ratio with a potential impact to CPU performance	Supported	Supported
QUICKLZ	1	Offers faster, but lower, data compression	Supported	Supported
RLE_TYPE Delta Compression (specific data types)	1 - 4	Offers run-length encoding compression for columns based on repeated values	Unsupported	Supported

Question: What type of data do you think would work well with the different offerings of compression?

Pivotal

Defining Default Table Storage Options

gp_default_storage_options

Options	Level	Command
APPENDONLY BLOCKSIZE CHECKSUM COMPRESSTYPE COMPRESSLEVEL ORIENTATION	Object level	CREATE TABLE … WITH (…)
	Role level	ALTER ROLE … SET …
	Database level	ALTER DATABASE … SET …
	System level	gpconfig …

Highest priority

Lowest priority

Usage: Update default storage options at role level

```
names=> alter role student set
gp_default_storage_options='appendonly=true,compresstype=zlib';
Names=> set role student;
```

Pivotal

Additional Table Types

Temporary table

Temporary tables can be used for:

- Storing transient results needed for other session queries
- Perform transformations on data

Readable external table

Writable external table

External tables:

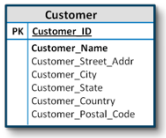
- Facilitate parallel data loading
- Stream data in from external sources
- Push data out of the database, in parallel

Temporary Tables – Overview

- Session-specific
- Dropped at the end of the session
- Take precedence over permanent tables of the same name
- Created in a special schema created on connection to a session
- Are distributed
- Can be indexed and analyzed

Pivotal

Temporary Tables – Two Use Cases



FILTE
R

Large table with billions of rows

Subset of larger table(s)

Working with smaller tables

Raw Source Data
(Serial Data Stream)

ETL Server

Raw Data - Block Loaded
(Parallel Data Streams - gpfdist)

TRANSFORMATION
PROCESS
(In Database Transforms
using SETs of data)

Data transformation

Creating a Temporary Table

Example: Creating a temporary

```
gpadmin=# CREATE TEMP[ORARY] TABLE monthlytranssummary (
  storeid   INTEGER,
  customerid INTEGER,
  transmonth SMALLINT,
  salesamttot DECIMAL(10,2)
)
ON COMMIT PRESERVE ROWS
DISTRIBUTED BY (storeid, customerid);
```

You ca

The following options to the `ON COMMIT` clause let you define how a temporary table is handled:

- `PRESERVE ROWS` – No action is taken on the table
- `DELETE ROWS` – The table is truncated
- `DROP` – The table is dropped

Pivotal