PostgreSQL as a Columnar Store

DCPUG May 2014
Reston, VA

Stephen Frost
sfrost@snowman.net

Stephen Frost

- PostgreSQL
  - Major Contributor, Committer
  - Implemented Roles in 8.3
  - Column-Level Privileges in 8.4
  - Contributions to PL/pgSQL, PostGIS
- Resonate, Inc.
  - Principal Database Engineer
  - Online Digital Media Company
  - We're Hiring! - techjobs@resonateinsights.com

resonate

Do you read...

- planet.postgresql.org

Why Columnar?

- Reduced overhead
- High compressability
- Min/Max indexes

PG Columnar Challenges

- Large per-row overhead
  - Page header
  - Line pointers
  - 24 bytes per row of visibility information
  - 226 rows/page with single 4-byte integer
- No Min/Max indexes
- No compression

Page Header specifics

- pd_lsn - Identifies xlog record for last change
- pd_checksum - Page checksum, if set
- pd_flags - Set of flags
- pd_lower - Offset to start of free space
- pd_upper - Offset to end of free space
- pd_special - Offset to special area
- pd_pagesize_version - Size (bytes) & layout version
- pd_prune_xid - Oldest prunable XID in tuples

Page Header Sizes

- pd_lsn - 8 bytes
- pd_checksum - 2 bytes
- pd_flags - 2 bytes
- pd_lower - 2 bytes
- pd_upper - 2 bytes
- pd_special - 2 bytes
- pd_pagesize_version - 2 bytes
- pd_prunce_xid - 4 bytes
-
- Total: 24 bytes

Line pointers

- One pointer per tuple
- 4 bytes each
- 15 bits for offset to start of tuple
- 2 bits for flags (tuple state)
- 15 bits for length of tuple (in bytes)

Tuple Header

- xmin - inserting XID - 4 bytes
- xmax - deleteing XID - 4 bytes
- ctid - command ID - 4 bytes
- tid - current TID - 4 bytes
- infomask - 4 bytes
- header size / offset - 1 byte
- variable length NULL bitmap - 1+ bytes
-
- Total: 24 bytes

Tuples

- Data: Single integer - 4 bytes
- Padding: 4 bytes (would get bigint for free...)
- Tuple length total: 32 bytes
- Stored from the end of the page
- First tuple at 8160 (8192-32)
- Last tuple at 960 (960 - 991)

Overall Page

- Page header
  - 0 - 24
- Line pointers
  - 24 - 927 (226x 4 bytes each)
- Tuples
  - 960 - 8192 (226x 32 bytes each)
- Free space
  - 928 - 960 (32 bytes)
  - Can't fit another line # + tuple
- Density: 11% (904 bytes of data in 8192 page)

Why PG?

- ACID hotness
- Checksums (tho we don't use them, yet)
- Already use it for many other things
- Great platform for development
- Can "hide" the columnar reality

Arrays in PG

- An array adds another header to the mix
- After the header, tightly packed integers
- Arrays can be compressed via TOAST
- Can store mixed arrays / non-arrays in a tuple

Array Header

- vl_len - Array length (req'd for var-length types)
- ndims - Number of dimensions
- dataoffset - Offset to array data (0 if no NULLs)
- elemtype - Oid for array element (eg: integer)
- (per dimension)
  - ARR_DIMS - Number of entries in the dimension
  - ARR_LBOUND - Lower bound of the dimension

Array Header Sizes

- vl_len - 4 bytes
- ndims - 4 bytes
- dataoffset - 4 bytes
- elemtype - 4 bytes
- ARR_DIMS - 4 bytes
- ARR_LBOUND - 4 bytes
-
- Total: 24
- Tuple + Array headers: 24 + 24 = 48 bytes

Array Tuples

- Data: Single integer - 4 bytes
- Padding: 4 bytes (would get bigint for free...)
- Tuple length total: 56 bytes
- Stored from the end of the page
- First tuple at 8136 (8192-56)
- Last tuple at 576 (576 - 631)

Overall Page

- Page header
  - 0 - 24
- Line pointers
  - 24 - 567 (136x 4 bytes each)
- Tuples
  - 576 - 8192 (136x 56 bytes each)
- Free space
  - 568 - 576 (8 bytes)
  - Can't fit another line # + tuple
- Density: 7% (544 bytes of data in 8192 page)

Why Arrays?

- After array header, the values are *tightly packed*
- Consider arrays of 200 integers
- Big array header, but
- *Much* less overhead overall
- Far fewer line pointers
- Am I for real? Let's check it out-

Array Tuples x200

- Data: 200 integers - 800 bytes
- Padding: NONE
- Tuple length total: 848 bytes
- Stored from the end of the page
- First tuple at 7344 (8192-848)
- Last tuple at 560 (560 - 1407)

Overall Page

- Page header
  - 0 - 24
- Line pointers
  - 24 - 59 (9x 4 bytes each)
- Tuples
  - 560 - 8192 (9x 848 bytes each)
- Free space
  - 60 - 560 (500 bytes)
  - Could fit another array if we tried..
- Density: 88% (7200 bytes of data in 8192 page)

Array Tuples x212

- Squeeze it out...
- Data: 212 integers - 848 bytes
- Padding: NONE
- Tuple length total: 896 bytes
- Stored from the end of the page
- First tuple at 7296 (8192-896)
- Last tuple at 128 (128 - 1023)

Overall Page

- Page header
  - 0 - 24
- Line pointers
  - 24 - 59 (9x 4 bytes each)
- Tuples
  - 128 - 8192 (9x 896 bytes each)
- Free space
  - 60 - 128 (68 bytes) - Lots of room...
  - Not that much lost
- Density: 93% (7632 bytes of data in 8192 page)

Caveats

- Have to use unnest()
- PG may have more difficulty planning
- Have to unnest() an entire array to extract data
- No visibility info on each value
- Stored in individual tables, can be awkward
- Requires mapping tables

Columnar Advantages?

- Compression via TOAST
  - Will try to compress arrays > 2k
  - May be unable to
  - Requires more CPU to decompress
- Min/Max
  - Add 'min_value' and 'max_value columns
  - Use regular btree indexes
  - Include in queries

Questions?

resonate

Thank you!

Stephen Frost
sfrost@snowman.net
@net_snow

resonate