Percona XtraDB: Compressed Columns with Dictionaries – an alternative to InnoDB table compression

Yura Sorokin, Senior Software Engineer at Percona
Existing compression methods
Existing compression methods for MySQL

• InnoDB Table Compression (starting from 5.1)

• InnoDB Page Compression (starting from 5.7)

• Filesystems with transparent compression (BTRFS, ZFS, NTFS, etc.)
  will not be covered in this talk

• COMPRESS() / UNCOMPRESS() MySQL functions
  https://dev.mysql.com/doc/refman/5.5/en/encryption-functions.html#function_compress
  https://dev.mysql.com/doc/refman/5.5/en/encryption-functions.html#function_uncompress

• Something intriguing from Percona Server :)
  something you might have already noticed in the title of this talk
InnoDB Table Compression

ROW_FORMAT=COMPRESSED
InnoDB Table Compression

```
SET GLOBAL innodb_file_per_table=1;
SET GLOBAL innodb_file_format=Barracuda;

CREATE TABLE tbl (
    id SERIAL PRIMARY KEY,
    value BLOB NOT NULL
) ENGINE=InnoDB ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE=N;
```

Based on zlib compression library (LZ77 compression algorithm).
InnoDB Table Compression

• Uncompressed “modification log” records changes made to the page (to avoid unnecessary uncompression and recompression)
• When “modification log” is full, recompression is needed.
• Recompression may fail as compressed data no longer fit and in this case B-tree nodes are split.
• To avoid frequent compression failures (splitting), there is reserved empty space (padding)
InnoDB Table Compression

- BLOB, VARCHAR, and TEXT columns that are not part of the primary key may be stored on separately allocated overflow pages.
- All data written to overflow pages is compressed “as is” (MySQL applies the zlib compression algorithm to the entire data item)
InnoDB Table Compression

Tuning:

- **KEY_BLOCK_SIZE=(1|2|4|8|16) combined with innodb_page_size**
  (KEY_BLOCK_SIZE must be \(<=\) innodb_page_size)
  (Table compression does not support 32k and 64k innodb_page_size)

- **innodb_compression_level = (0..9)**

- **innodb_compression_pad_pct_max**

- **innodb_compression_failure_threshold_pct**
InnoDB Table Compression

“…In general, compression works best on tables that include a reasonable number of character string columns and where the data is read far more often than it is written…”

Real world data

JSON documents
Real world data

JSON data sets for JSON Studio tutorials
http://jsonstudio.com/resources/

• JSON data set of projects funded by the World Bank (436K compressed).
• JSON data set for US zip (postal) codes (656K compressed).
• JSON data set of listed stocks (1.6M compressed).
• JSON data set for Enron emails (3.9M compressed).
• JSON data set of startup company information (14.8M compressed).

‘companies.json’ repeated 8 times
Real world data

{ "_id": { "$oid": "52cdef7c4bab8bd675297d8b" },
  "name": "AdventNet",
  "permalink": "http://www.crunchbase.com/company/adventnet",
  "crunchbase_url": "http://adventnet.com",
  "homepage_url": "",
  "blog_url": "",
  "blog_feed_url": "",
  "twitter_username": "manageengine",
  "category_code": "enterprise",
  "number_of_employees": 600,
  "founded_year": 1996,
  "deadpooled_year": 2,
  "email_address": "pr@adventnet.com",
  "phone_number": "925-924-9500",
  "description": "Server Management Software",
  "created_at": { "$date": 1180121062000 },
  "updated_at": "Wed Oct 31 18:26:09 UTC 2012",
  "overview": "AdventNet is now <a href="/company/zoho-manageengine" title="Zoho ManageEngine" rel="nofollow">Zoho ManageEngine</a>.<p>Founded in 1996, AdventNet has served a diverse range of enterprise IT networking and telecom customers.</p>AdventNet supplies server and network management software."
}

...
Real world data

Loading data from the file

```
SET max_heap_table_size = 1024 * 1048576;
CREATE TABLE companies_src (  
id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,  
value LONGTEXT NOT NULL,  
PRIMARY KEY(id)
) ENGINE=Memory;
LOAD DATA INFILE 'companies.json' INTO TABLE companies_src FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '' (value);
```
Real world data

Parsing JSON documents

```sql
CREATE TABLE companies_src_parsed(
    id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
    value JSON NOT NULL,
    PRIMARY KEY(id)
) ENGINE=Memory AS SELECT * FROM companies_src;
```
Real world data

Statistics:

```sql
SELECT COUNT(*) FROM companies_src_parsed;
150408

SELECT COUNT(*) FROM companies_src_parsed
  WHERE JSON_VALID(value) <> 1;
0

SELECT MIN(LENGTH(value)), ROUND(AVG(LENGTH(value))),
  MAX(LENGTH(value))
  FROM companies_src_parsed;
1027 4160 278016
```
Real world data

More statistics:

```sql
SELECT JSON_KEYS(value) AS keys, JSON_LENGTH(keys)
    FROM companies_src_parsed ORDER BY id LIMIT 1;
```

```
```

38

```sql
SELECT MIN(JSON_DEPTH(value)), MAX(JSON_DEPTH(value))
    FROM companies_src_parsed;
```

3 7
InnoDB Table Compression Experiments

Compression ratio and insertion time
InnoDB Table Compression Experiments

Copying data to a new table:
SET GLOBAL innodb_compression_level = LEVEL;
CREATE TABLE companies_compressed(
  id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
  value LONGTEXT NOT NULL,
  PRIMARY KEY(id)
) ENGINE=InnoDB ROW_FORMAT=COMPRESSED
KEY_BLOCK_SIZE=KBS;
INSERT INTO companies_compressed
  SELECT * FROM companies_src;
LEVEL = [0, 1, 6, 9]   KBS = [1, 2, 4, 8, 16]
InnoDB Table Compression Experiments

Calculating compressed data size:

```sql
SELECT data_length FROM information_schema.tables
WHERE table_schema = DATABASE()
AND table_name = 'companies_compressed';
```
InnoDB Table Compression Experiments

Identity: Data size, MB and query time, sec

![Graph showing data length and query time for LONGTEXT and JSON formats.]

- **Data Length, MB**
  - LONGTEXT: 898 MB
  - JSON: 880 MB

- **Query Time, sec**
  - LONGTEXT: 33 sec
  - JSON: 36 sec
InnoDB Table Compression Experiments
Identity: Data size, % and query time, %

- **data_length_relative, %**
  - LONGTEXT: 100,00%
  - JSON: 98,00%

- **query_time_relative, %**
  - LONGTEXT: 100,00%
  - JSON: 109,09%
InnoDB Table Compression Experiments

Data size, MB

<table>
<thead>
<tr>
<th></th>
<th>LONGTEXT</th>
<th>JSON</th>
<th>KBS1</th>
<th>KBS2</th>
<th>KBS4</th>
<th>KBS8</th>
<th>KBS16</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_length(LVL0), MB</td>
<td>898</td>
<td>880</td>
<td>749</td>
<td>873</td>
<td>938</td>
<td>917</td>
<td>854</td>
</tr>
<tr>
<td>data_length(LVL1), MB</td>
<td>898</td>
<td>880</td>
<td>749</td>
<td>873</td>
<td>938</td>
<td>917</td>
<td>854</td>
</tr>
<tr>
<td>data_length(LVL6), MB</td>
<td>898</td>
<td>880</td>
<td>749</td>
<td>873</td>
<td>938</td>
<td>917</td>
<td>854</td>
</tr>
<tr>
<td>data_length(LVL9), MB</td>
<td>898</td>
<td>880</td>
<td>749</td>
<td>873</td>
<td>938</td>
<td>917</td>
<td>854</td>
</tr>
</tbody>
</table>
InnoDB Table Compression Experiments
Query time, sec

![Bar chart showing query time for different LVL levels and data types.](chart.png)
InnoDB Table Compression Experiments

Data size, %
InnoDB Table Compression Experiments

Query time, %

![Bar chart](image.png)

1. Compare the performance of different table compression levels on query time.
2. Identify the level that offers the best balance between compression and performance.
InnoDB Page Compression

COMPRESSSION=\("zlib\" \| \"lz4\")
InnoDB Page Compression

SET GLOBAL innodb_file_per_table=1;

CREATE TABLE tbl ( id SERIAL PRIMARY KEY, value BLOB NOT NULL ) ENGINE=InnoDB COMPRESSION=("zlib" | "lz4");

Can be configured to use either zlib or lz4 compression.
InnoDB Page Compression

Also referred to as Transparent Page Compression


Requirements:
- Sparse file and hole punching support on Linux
- NTFS on Windows
InnoDB Page Compression

`ls -l tablespace_name.ibd` – will show uncompressed file size. Use `du --block-size=1 tablespace_name.ibd` instead.

```
SELECT FS_BLOCK_SIZE, FILE_SIZE, AlLOCATED_SIZE
    FROM INFORMATION_SCHEMA.INNODB_SYS_TABLESPACES;
```
InnoDB Page Compression Experiments

Compression ratio and insertion time
InnoDB Page Compression Experiments

Copying data to a new table:

```sql
SET GLOBAL innodb_compression_level = LEVEL;
CREATE TABLE companies_compressed(
    id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
    value LONGTEXT NOT NULL,
    PRIMARY KEY(id)
) ENGINE=InnoDB COMPRESSION='ALG';
INSERT INTO companies_compressed
SELECT * FROM companies_src;
LEVEL = [0, 1, 6, 9]   ALG = [zlib, lz4]
```
InnoDB Page Compression Experiments

Calculating compressed data size:

```
SELECT allocated_size
FROM information_schema.innodb_sys_tablespaces
WHERE name = CONCAT(DATABASE(), '/companies_compressed');
```
InnoDB Page Compression Experiments

Data size, MB

<table>
<thead>
<tr>
<th></th>
<th>LONGTEXT</th>
<th>JSON</th>
<th>ZLIB</th>
<th>LZ4</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_length(LVL0), MB</td>
<td>898</td>
<td>898</td>
<td>898</td>
<td>898</td>
</tr>
<tr>
<td>data_length(LVL1), MB</td>
<td>880</td>
<td>880</td>
<td>880</td>
<td>880</td>
</tr>
<tr>
<td>data_length(LVL6), MB</td>
<td>920</td>
<td>920</td>
<td>920</td>
<td>920</td>
</tr>
<tr>
<td>data_length(LVL9), MB</td>
<td>434</td>
<td>373</td>
<td>372</td>
<td>507</td>
</tr>
</tbody>
</table>
InnoDB Page Compression Experiments

Query time, sec

<table>
<thead>
<tr>
<th>Format</th>
<th>query_time(LVL0), sec</th>
<th>query_time(LVL1), sec</th>
<th>query_time(LVL6), sec</th>
<th>query_time(LVL9), sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>LONGTEXT</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>JSON</td>
<td>36</td>
<td>36</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>ZLIB</td>
<td>18</td>
<td>22</td>
<td>28</td>
<td>30</td>
</tr>
<tr>
<td>LZ4</td>
<td>20</td>
<td>19</td>
<td>19</td>
<td>19</td>
</tr>
</tbody>
</table>
InnoDB Page Compression Experiments

Data size, %
# InnoDB Page Compression Experiments

## Query time, %

<table>
<thead>
<tr>
<th>Data Type</th>
<th>0.00%</th>
<th>20.00%</th>
<th>40.00%</th>
<th>60.00%</th>
<th>80.00%</th>
<th>100.00%</th>
</tr>
</thead>
<tbody>
<tr>
<td>LONGTEXT</td>
<td>100.00%</td>
<td>109.09%</td>
<td>109.09%</td>
<td>109.09%</td>
<td>109.09%</td>
<td>109.09%</td>
</tr>
<tr>
<td>JSON</td>
<td>100.00%</td>
<td>109.09%</td>
<td>109.09%</td>
<td>109.09%</td>
<td>109.09%</td>
<td>109.09%</td>
</tr>
<tr>
<td>ZLIB</td>
<td>54.55%</td>
<td>66.67%</td>
<td>84.85%</td>
<td>90.91%</td>
<td>93.16%</td>
<td>96.15%</td>
</tr>
<tr>
<td>LZ4</td>
<td>60.61%</td>
<td>67.78%</td>
<td>73.38%</td>
<td>73.38%</td>
<td>73.38%</td>
<td>73.38%</td>
</tr>
</tbody>
</table>

- **query_time_relative(LVL0), %**
- **query_time_relative(LVL1), %**
- **query_time_relative(LVL6), %**
- **query_time_relative(LVL9), %**
COMPRESS() MySQL function

COMPRESS() / UNCOMPRESS()
COMPRESS() MySQL function

CREATE TABLE tbl (  
id SERIAL PRIMARY KEY,  
value BLOB NOT NULL  
) ENGINE=<ANY_ENGINE>;

INSERT INTO tbl VALUES  
(DEFAULT, COMPRESS(REPEAT('a', 2048)));

SELECT id, UNCOMPRESS(value) FROM tbl;
COMPRESS() MySQL function

- Based on zlib algorithm
- Inconvenient – need to explicitly wrap field names into COMPRESS() / UNCOMPRESS().
- Losing actual field type – compressed fields should be VARBINARY or BLOB.
- No control over compression level.
COMPRESS() MySQL function

Experiments

Compression ratio and insertion time
COMPRESS MySQL function Experiments

Copying data to a new table:

CREATE TABLE companies_compressed(
    id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
    value LONGBLOB NOT NULL,
    PRIMARY KEY(id)
) ENGINE=InnoDB;

INSERT INTO companies_compressed
SELECT id, COMPRESS(value) FROM companies_src;
COMPRESS MySQL function Experiments
Data size, MB and query time, sec

- **data_length, MB**
  - LONGTEXT: 898 MB
  - JSON: 880 MB
  - COMPRESS(): 283 MB

- **query_time, sec**
  - LONGTEXT: 33 sec
  - JSON: 36 sec
  - COMPRESS(): 27 sec
COMPRESS MySQL function Experiments
Data size, % and query time, %

**data_length_relative, %**
- LONGTEXT: 100.00%
- JSON: 98.00%
- COMPRESS(): 31.49%

**query_time_relative, %**
- LONGTEXT: 100.00%
- JSON: 109.09%
- COMPRESS(): 81.82%
XtraDB Compressed Columns

COLUMN_FORMAT COMPRESSED
XtraDB Compressed Columns

Availability:
• Percona Server 5.6.33-79.0
  https://www.percona.com/doc/percona-server/5.6/flexibility/compressed_columns.html
• Percona Server 5.7.17-11
  https://www.percona.com/doc/percona-server/5.7/flexibility/compressed_columns.html
Compressed Columns

Key characteristics:

• It is a data type modifier, independent from user-level SQL and InnoDB data compression, that causes the data stored in the column to be compressed on writing to storage and decompressed on reading.

• For all other purposes, the data type is identical to the one without the modifier, i.e. no new data types are created.

• Compression is done by using the zlib library.
XtraDB Compressed Columns

CREATE TABLE tbl (  
id SERIAL PRIMARY KEY,  
value TEXT COLUMN_FORMAT COMPRESSED NOT NULL
) ENGINE=InnoDB;

INSERT INTO tbl VALUES  
(DEFAULT, REPEAT('a', 2048));

SELECT id, value FROM tbl;
XtraDB Compressed Columns

Supported data types:

- BLOB (including TINYBLOB, MEDIUMBLOB, LONGBLOB)
- TEXT (including TINYTEXT, MEDIUMTEXT, LONGTEXT)
- VARCHAR (including NATIONAL VARCHAR)
- VARBINARY
XtraDB Compressed Columns

Compressing/uncompressing existing data:

```
ALTER TABLE tbl CHANGE value
    value TEXT COLUMN_FORMAT DEFAULT NOT NULL;

ALTER TABLE tbl MODIFY
    value TEXT COLUMN_FORMAT DEFAULT NOT NULL;

ALTER TABLE tbl CHANGE value
    value TEXT COLUMN_FORMAT COMPRESSED NOT NULL;

ALTER TABLE tbl MODIFY
    value TEXT COLUMN_FORMAT COMPRESSED NOT NULL;
```
XtraDB Compressed Columns

5.7-specific features:
- Support for compressed JSON data type
- Support for compressed generated stored columns

```sql
CREATE TABLE tbl(
    id SERIAL PRIMARY KEY,
    value JSON COLUMN_FORMAT COMPRESSED NOT NULL
    gen TEXT GENERATED ALWAYS AS (value->"$.bio")
    STORED COLUMN_FORMAT COMPRESSED
) ENGINE=InnoDB;
```
XtraDB Compressed Columns

Tuning:

- `innodb_compressed_columns_zip_level`
  https://www.percona.com/doc/percona-server/5.7/flexibility/compressed_columns.html#innodb_compressed_columns_zip_level

- `innodb_compressed_columns_threshold = (0..9)`
  https://www.percona.com/doc/percona-server/5.7/flexibility/compressed_columns.html#innodb_compressed_columns_threshold
XtraDB Compressed Columns

Limitations (both for 5.6 and 5.7):

- Compressed columns **cannot be used in indices** (neither on their own nor as parts of composite keys).
- **ALTER TABLE ... DISCARD/IMPORT TABLESPACE** is not supported for tables with compressed columns.
XtraDB Compressed Columns

2-bytes compressed blob header:

<table>
<thead>
<tr>
<th>R</th>
<th>W</th>
<th>ALGORITHM</th>
<th>LEN-LEN</th>
<th>C</th>
<th>UNUSED</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>

- **R** – reserved for future versions (currently must always be 0)
- **ALGORITHM** - identifies which algorithm was used to compress this BLOB. Currently, the only value 0 (meaning zlib) is supported
- **W** – 'wrap' identifies if compression algorithm calculated a checksum (adler32 in case of zlib) and appended it to the compressed data
- **LEN-LEN** – identifies the length of the column length data portion followed by this header
XtraDB Compressed Columns

2-bytes compressed blob header:

<table>
<thead>
<tr>
<th>R</th>
<th>W</th>
<th>ALGORITHM</th>
<th>LEN-LEN</th>
<th>C</th>
<th>UNUSED</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>

- C – 'compressed'. If set to 1, then this header is immediately followed by 1..8 bytes (depending on the value of LEN-LEN bitfield) which determine original (uncompressed) block size. These LEN-LEN bytes are followed by compressed representation of the original data.
- If 'compressed' bit is set to 0, every other bitfield must be ignored. In this case the header is immediately followed by uncompressed (original) data.
XtraDB Compressed Columns Experiments

Compression ratio and insertion time
XtraDB Compressed Columns
Experiments

Copying data to a new table:

```sql
CREATE TABLE companies_compressed(
    id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
    value [ LONGTEXT | JSON ] COLUMN_FORMAT COMPRESSED NOT NULL,
    PRIMARY KEY(id)
) ENGINE=InnoDB;

INSERT INTO companies_compressed
SELECT * FROM companies_src;
```
XtraDB Compressed Columns Experiments

Data size, MB

<table>
<thead>
<tr>
<th>Column Type</th>
<th>data_length(LVL0), MB</th>
<th>data_length(LVL1), MB</th>
<th>data_length(LVL6), MB</th>
<th>data_length(LVL9), MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>LONGTEXT</td>
<td>898</td>
<td>898</td>
<td>880</td>
<td>880</td>
</tr>
<tr>
<td>JSON</td>
<td>898</td>
<td>898</td>
<td>880</td>
<td>880</td>
</tr>
<tr>
<td>COMPRESSED TEXT</td>
<td>304</td>
<td>284</td>
<td>283</td>
<td></td>
</tr>
<tr>
<td>COMPRESSED JSON</td>
<td>387</td>
<td>377</td>
<td>377</td>
<td></td>
</tr>
</tbody>
</table>
XtraDB Compressed Columns Experiments

Query time, sec

- LONGTEXT: 33 sec
- JSON: 36 sec
- COMPRESSED TEXT: 22 sec
- COMPRESSED JSON: 52 sec
XtraDB Compressed Columns Experiments
Data size, %

![Bar chart showing data size percentage for different data types and compression levels.]
XtraDB Compressed Columns Experiments
Query time, %
zlib Compression Algorithm

Overview
zlib Compression Algorithm

zlib overview
https://en.wikipedia.org/wiki/Zlib

• Only supports one algorithm **DEFLATE**, that is a variation of **LZ77**
• LZ77 is a theoretically dictionary coder
• When processing data also keeps track of the most commonly used byte sequences in a dictionary so that those sequences can be referred when occur again in the stream.
zlib Compression Algorithm

zlib compression examples (oversimplified)
Example 1 – no repeating byte sequences

Input : “Sunday Monday Tuesday”
OutDict: {“Sunday”, “Monday”, “Tuesday”}
Output : “Sunday Monday Tuesday”

Very bad compression ratio (100%).
Just put some words into the dictionary.
zlib Compression Algorithm

zlib compression examples (oversimplified)
Example 2 – a lot of byte sequences are repeated

Input : “Sunday Monday Tuesday Tuesday Monday Sunday”
OutDict: {“Sunday”, “Monday”, “Tuesday”}
Output: “Sunday Monday Tuesday \ref[2] \ref[1] \ref[0]”

Good compression ratio (50% + overhead).
For the second half of the input stream dictionary words were used.
zlib Compression Algorithm

zlib compression examples (oversimplified)
Example 3 – an initial state of the dictionary (not ideal) is provided by the user

InDict : {“Sunday”, “Monday”}
Input  : “Sunday Monday Tuesday Tuesday Monday Sunday”
OutDict: {“Sunday”, “Monday”, “Tuesday”}
Output : “\ref[0] \ref[1] Tuesday \ref[2] \ref[1] \ref[0]”

Good compression ratio (16% + overhead).
For 5 out of 6 words from the input sequence dictionary references were used. Still not perfect though.
zlib Compression Algorithm

zlib compression examples (oversimplified)
Example 3 – an initial state of the dictionary (ideal) is provided by the user

InDict : {“Sunday”, “Monday”, “Tuesday”}
Input : “Sunday Monday Tuesday Tuesday Monday Sunday”
OutDict: {“Sunday”, “Monday”, “Tuesday”}
Output : “\ref[0] \ref[1] \ref[2] \ref[2] \ref[1] \ref[0]”

Good compression ratio (0% + overhead).
For all the words from the input sequence dictionary references were used. The best results.
Dictionary usage conclusions:

• Compression ratio can significantly depend on whether the initial compression dictionary is provided or not.
• Compression ratio depends on the quality of that dictionary.
Compressed Columns with Dictionaries

COLUMN_FORMAT COMPRESSED WITH COMPRESSION_DICTIONARY
XtraDB Compressed Columns with Dictionaries

CREATE COMPRESSION_DICTIONARY dict('SundayMondayTuesday');

SET @dict_content = 'SundayMondayTuesday';
CREATE COMPRESSION_DICTIONARY dict(@dict_content);

DROP COMPRESSION_DICTIONARY dict;
XtraDB Compressed Columns with Dictionaries

CREATE TABLE tbl (  
id SERIAL PRIMARY KEY,  
value TEXT COLUMN_FORMAT COMPRESSED  
  WITH COMPRESSION_DICTIONARY dict NOT NULL  
) ENGINE=InnoDB;

INSERT INTO tbl VALUES  
(DEFAULT, REPEAT('a', 2048));

SELECT id, value FROM tbl;
XtraDB Compressed Columns with Dictionaries

INFORMATION_SCHEMA.XTRADB_ZIP_DICT

- id BIGINT UNSIGNED – compression dictionary ID
- name VARCHAR(64) – compression dictionary name (’dict’ in the example)
- zip_dict BLOB – compression dictionary content (’SundayMondayTuesday’ in the example)
XtraDB Compressed Columns with Dictionaries

INFORMATION_SCHEMA.XTRADB_ZIP_DICT_COLS

- `table_id` BIGINT UNSIGNED – table ID from INFORMATION_SCHEMA.INNODB_SYS_TABLES
- `column_pos` BIGINT UNSIGNED – column position (starts from 0 as in INFORMATION_SCHEMA.INNODB_SYS_COLUMNS)
- `dict_id` BIGINT UNSIGNED – compression dictionary ID
Generating Compression Dictionary

Keyword histogram
Generating Compression Dictionary

zlib 1.2.11 Manual
http://www.zlib.net/manual.html

“…The dictionary should consist of strings (byte sequences) that are likely to be encountered later in the data to be compressed, with the most commonly used strings preferably put towards the end of the dictionary…”
Generating Compression Dictionary

A table for storing JSON keys

CREATE TABLE extracted_words(
    level BIGINT UNSIGNED NOT NULL,
    word VARCHAR(255) NOT NULL,
    weight BIGINT UNSIGNED NOT NULL,
    PRIMARY KEY(level, word)
) ENGINE=InnoDB;
Generating Compression Dictionary

Stored procedure for extracting keys from a JSON value recursively

```sql
CREATE PROCEDURE extract_json_keys_from_tree(IN tree JSON, IN path VARCHAR(255), IN level INT) BEGIN
    DECLARE i, n INT;
    DECLARE tree_keys JSON;
    DECLARE word VARCHAR(255);
    SET tree_keys = JSON_KEYS(tree, path);
    IF tree_keys IS NOT NULL THEN
        SET i = 0;
        SET n = JSON_LENGTH(tree_keys);
        WHILE i < n DO
            SET word = JSON_EXTRACT(tree_keys, CONCAT('$[', i, ']' ));
            INSERT INTO extracted_words
            VALUES(level, JSON_UNQUOTE(word), 1) ON DUPLICATE KEY UPDATE weight = weight + 1;
            CALL extract_json_keys_from_tree(tree, CONCAT(path, '.', word), level + 1);
            SET i = i + 1;
        END WHILE;
    ELSEIF JSON_CONTAINS_PATH(tree, 'one', CONCAT(path, '[*]')) THEN
        SET i = 0;
        SET n = JSON_LENGTH(tree, path);
        WHILE i < n DO
            CALL extract_json_keys_from_tree(tree, CONCAT(path, '[', i, ']'), level + 1);
            SET i = i + 1;
        END WHILE;
    END IF;
END
```
Generating Compression Dictionary

Stored procedure for extracting keys from JSON values stored in a table

```sql
CREATE PROCEDURE extract_json_keys_from_table()
BEGIN
    DECLARE done BOOLEAN DEFAULT FALSE;
    DECLARE tree JSON;
    DECLARE cur CURSOR FOR SELECT value FROM companies_src_parsed ORDER BY id;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO tree;
        IF done THEN
            LEAVE read_loop;
        END IF;
        CALL extract_json_keys_from_tree(tree, '\$', 0);
    END LOOP;
    CLOSE cur;
END
```
Generating Compression Dictionary

Filling `extracted_words` with JSON keys from `companies_src_parsed`

```
SET max_sp_recursion_depth = 255;
CALL extract_json_keys_from_table();
```
Generating Compression Dictionary

```
SELECT * FROM extracted_words;
...
1  source_url     2677
1  stock_symbol   390
1  term_code      2677
1  valuation_amount 390
1  valuation_currency_code 390
2  acquired_day   4412
2  acquired_month 4412
2  acquired_year  4412
2  address1      16705
...
Generating Compression Dictionary

Simplified `extracted_words` view

```sql
SET group_concat_max_len = 32768;

CREATE VIEW extracted_words_simple AS
SELECT word, SUM(weight) AS weight,
    BIT_OR(POWER(2, level)) AS level_mask
FROM extracted_words
GROUP BY word;
```
Generating Compression Dictionary

Creating compression dictionaries from keywords of the zero level

```sql
SELECT GROUP_CONCAT(word ORDER BY weight,
  word SEPARATOR ' ' ) INTO @res
FROM extracted_words_simple WHERE (level_mask & 1) != 0;
CREATE COMPRESSION_DICTIONARY dict_zero_level (@res);

SELECT GROUP_CONCAT(word ORDER BY weight DESC,
  word SEPARATOR ' ' ) INTO @res
FROM extracted_words_simple WHERE (level_mask & 1) != 0;
CREATE COMPRESSION_DICTIONARY dict_zero_level_reversed (@res);
```
Generating Compression Dictionary

Creating compression dictionaries from keywords of all levels

```
SELECT GROUP_CONCAT(word ORDER BY weight,
    word SEPARATOR ' ') INTO @res
FROM extracted_words_simple;
CREATE COMPRESSION_DICTIONARY dict_all_levels(@res);

SELECT GROUP_CONCAT(word ORDER BY weight DESC,
    word SEPARATOR ' ') INTO @res
FROM extracted_words_simple;
CREATE COMPRESSION_DICTIONARY dict_all_levels_reversed(@res);
```
Compression with Dictionaries
Experiments

Compression ratio and insertion time
Compression with Dictionaries
Experiments

Copying data to a new table:

CREATE TABLE companies_compressed(
    id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
    value LONGTEXT COLUMN_FORMAT COMPRESSED
    WITH COMPRESSION_DICTIONARY DICT
    NOT NULL,
    PRIMARY KEY(id)
) ENGINE=InnoDB;

INSERT INTO companies_compressed
SELECT * FROM companies_src;

DICT = [ dict_zero_level, dict_zero_level_reversed,
         dict_all_levels, dict_all_levels_reversed ]
Compression with Dictionaries Experiments
Data size, MB

LONGTEXT  COMpressed TEXT  DICT ZERO  DICT ZERO REV  DICT ALL  DICT ALL REV

data_length(LVL0), MB  data_length(LVL1), MB  data_length(LVL6), MB  data_length(LVL9), MB
Compression with Dictionaries Experiments
Query time, sec

<table>
<thead>
<tr>
<th></th>
<th>query_time(LVL0), sec</th>
<th>query_time(LVL1), sec</th>
<th>query_time(LVL6), sec</th>
<th>query_time(LVL9), sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>LONGTEXT</td>
<td>33 33 33 33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPRESSED TEXT</td>
<td>17 22 28 30</td>
<td>16 21 28 30</td>
<td>16 21 28 30</td>
<td>17 21 29 31</td>
</tr>
<tr>
<td>DICT ZERO</td>
<td>17 22 28 30</td>
<td>16 21 28 30</td>
<td>16 21 28 30</td>
<td>17 21 29 31</td>
</tr>
<tr>
<td>DICT ZERO REV</td>
<td>17 22 28 30</td>
<td>16 21 28 30</td>
<td>16 21 28 30</td>
<td>17 21 29 31</td>
</tr>
<tr>
<td>DICT ALL</td>
<td>17 22 28 30</td>
<td>16 21 28 30</td>
<td>16 21 28 30</td>
<td>17 21 29 31</td>
</tr>
<tr>
<td>DICT ALL REV</td>
<td>17 22 28 30</td>
<td>16 21 28 30</td>
<td>16 21 28 30</td>
<td>17 21 29 31</td>
</tr>
</tbody>
</table>
Compression with Dictionaries Experiments
Data size, %

- Longtext: 100.00%
- Compressed text: 100.00%
- Dict zero: 100.00%
- Dict zero rev: 100.00%
- Dict all: 100.00%
- Dict all rev: 100.00%

- Data length relative (LVL0), %: 100.00%
- Data length relative (LVL1), %: 100.00%
- Data length relative (LVL6), %: 100.00%
- Data length relative (LVL9), %: 100.00%
Compression with Dictionaries Experiments
Query time, %
The most important slide

The best from all worlds
The most important slide
Data size, MB
The most important slide
Data size, %

<table>
<thead>
<tr>
<th></th>
<th>data_length_relative(LVL0), %</th>
<th>data_length_relative(LVL1), %</th>
<th>data_length_relative(LVL6), %</th>
<th>data_length_relative(LVL9), %</th>
</tr>
</thead>
<tbody>
<tr>
<td>LONGTEXT</td>
<td>100.00%</td>
<td>102.45%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>PAGE ZLIB</td>
<td>48.36%</td>
<td>41.59%</td>
<td>37.61%</td>
<td>31.49%</td>
</tr>
<tr>
<td>TABLE KB54</td>
<td>41.37%</td>
<td>35.27%</td>
<td>35.15%</td>
<td>33.83%</td>
</tr>
<tr>
<td>COMPRESS()</td>
<td>31.49%</td>
<td>31.49%</td>
<td>31.49%</td>
<td>31.49%</td>
</tr>
<tr>
<td>COLUMN COMPRESS</td>
<td>33.43%</td>
<td>31.60%</td>
<td>31.49%</td>
<td>28.04%</td>
</tr>
<tr>
<td>DICT ALL</td>
<td>30.38%</td>
<td>29.92%</td>
<td>31.49%</td>
<td>27.92%</td>
</tr>
</tbody>
</table>
## The most important slide

**Summary**

<table>
<thead>
<tr>
<th></th>
<th>LVL1</th>
<th>LVL6</th>
<th>LVL9</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN COMPRESS</td>
<td>33.83%</td>
<td>31.60%</td>
<td>31.49%</td>
</tr>
<tr>
<td>DICT ALL</td>
<td>30.38%</td>
<td>28.04%</td>
<td>27.92%</td>
</tr>
<tr>
<td>RATIO</td>
<td>89.79%</td>
<td>88.71%</td>
<td>88.67%</td>
</tr>
</tbody>
</table>

-11.33%
Thank You Sponsors!
SAVE THE DATE!

April 23-25, 2018
Santa Clara Convention Center

CALL FOR PAPERS OPENING SOON!

www.perconalive.com
Thank you

You survived 90 slides