Building Data Lake with MariaDB ColumnStore

Sasha Vaniachine

HELIOS SaaS Platform

- Ranked among fastest growing companies in North America by Deloitte for two years in a row, VirtualHealth empowers healthcare organizations to achieve enhanced outcomes, while maximizing efficiency and lowering costs
- Our SaaS platform HELIOS is utilized by largest and most innovative US health plans to manage about ten million members

HELIOS VIRTUALHEALTH®

Relational Data Lake

- In a course of daily operations, VirtualHealth clients accumulate a growing volume of transactional data in relational OLTP databases
  - With age, these operational data became less relevant to daily operations
  - In contrast, as historical volumes grow, these data grow in value for analytics
- VirtualHealth needs to provide data scientists and developers with on-demand access to de-identified patient data increasing in volume and complexity
  - We chose a relational data lake approach, storing daily, read-only snapshots of OLTP databases
  - To lower the costs, we chose MariaDB ColumnStore because of its inherent data compression **and S3 storage support**

Data Lake

- A data lake is a storage repository that holds a large amount of data in its native, raw format
  - James Dixon introduced this concept as: *“If you think of a Data Mart as a store of bottled water – cleansed and packaged and structured for easy consumption – the Data Lake is a large body of water in a more natural state.”*

Implementing one of the Data Warehouse rules:

- Store snapshot data captured at a given point in time
  - We store daily, read-only snapshots of OLTP databases

Bridging the Gap

- Healthcare operational data originate from relational database systems that are not directly suitable for analytics and/or machine learning algorithms
- We describe here VirtualHealth experience in building the data pipeline between the operational data in relational database systems, that are row-oriented and machine learning tools that prefer data in columnar formats
- We chose to build a data pipeline using MariaDB ColumnStore since it already provides open source examples of integration with Jupyter Notebooks and Apache Zeppelin used for data exploration and analysis by data scientists

Rationale

- Analytical queries are slow on a transactional database
  - A special storage format - columnar - improves performance of such queries
- Although there are several open source columnar databases,
  - in this talk, we will focus on the MariaDB ColumnStore

HELIOS
VIRTUALHEALTH®

Slow Queries

Row-oriented RDBMS

Query 1: Ranking

- A ranking query: top ten clients who visited doctors most often
  - data from 2017-2020

```
mysql> SELECT
    ->   client_id,
    ->   min(date) as first_visit,
    ->   max(date) as last_visit,
    ->   count(distinct date) as days_visited,
    ->   count(cv.id) as visits,
    ->   count(distinct cv.service_location_name) as locations
    -> FROM client_visit cv
    -> GROUP BY client_id
    -> ORDER by visits desc
    -> LIMIT 10;
+-----------+-------------+------------+--------------+--------+-----------+
| client_id | first_visit | last_visit | days_visited | visits | locations |
+-----------+-------------+------------+--------------+--------+-----------+
| ......... | 2017-08-07  | 2020-03-13 |           ..  |    ... |        .. |

10 rows in set (10 min 53.826 sec)
```

Ranking Query Speedup: Using index

```
  select_type: SIMPLE
        table: cv
   partitions: NULL
         type: index
possible_keys: FK_client_visit_author_id
          key: FK_client_visit_author_id
      key_len: 5
          ref: NULL
         rows: 26847507
     filtered: 100.00
        Extra: Using temporary; Using filesort

  PRIMARY KEY (`id`),
  KEY `FK_client_visit_author_id` (`client_id`)
```

Adding Covered Index

```
mysql> alter table client_visit add key comb (client_id, date, service_location_name);
Query OK, 0 rows affected (2 min 31.424 sec)
Records: 0  Duplicates: 0  Warnings: 0

        table: cv
   partitions: NULL
         type: index
possible_keys: FK_client_visit_author_id,comb
          key: comb
      key_len: 776
          ref: NULL
         rows: 26847507
     filtered: 100.00
        Extra: Using index; Using temporary; Using filesort

10 rows in set (21.096 sec)
```

Still slow!

That was only the beginning… now Query 2

```
SELECT
  cv.client_id as client_id,
  min(date) as first_visit,
  max(date) as last_visit,
  count(distinct date) as days_visited,
  count(distinct cv.id) as visits,
  count(distinct cp.cpt_code) as procedures,
  count(distinct cv.service_location_name) as locations,
  sum(billed_amount) as total_billed,
  max(billed_amount) as max_price,
  avg(billed_amount) as avg_price
FROM
     client_visit cv
     join client_procedure cp on cp.encounter_id = cv.encounter_id
     join client_procedure_claim cpc on cp.id = cpc.client_procedure_id
     join client_claim cc on cc.id = cpc.client_claim_id
GROUP BY client_id
ORDER BY total_billed desc
LIMIT 10
```

OLTP: Highly normalized schema

Query 2: Four table JOINs, all tables large

```
+-----------+------------+-----------+-------------+--------+------------+-----------+--------------+------------+------------+
| client_id | first_visit | last_visit | days_visited | visits | procedures | locations | total_billed | max_price | avg_price   |
+-----------+------------+-----------+-------------+--------+------------+-----------+--------------+------------+------------+
|   ....... | 2018-02-14  | 2019-09-04 |          154 |    161 |         .. |        .. |    724K      |  12K      |  355.49     |
...
10 rows in set (9 hours 22 min 28.387 sec)
```

HELIOS
VIRTUALHEALTH®

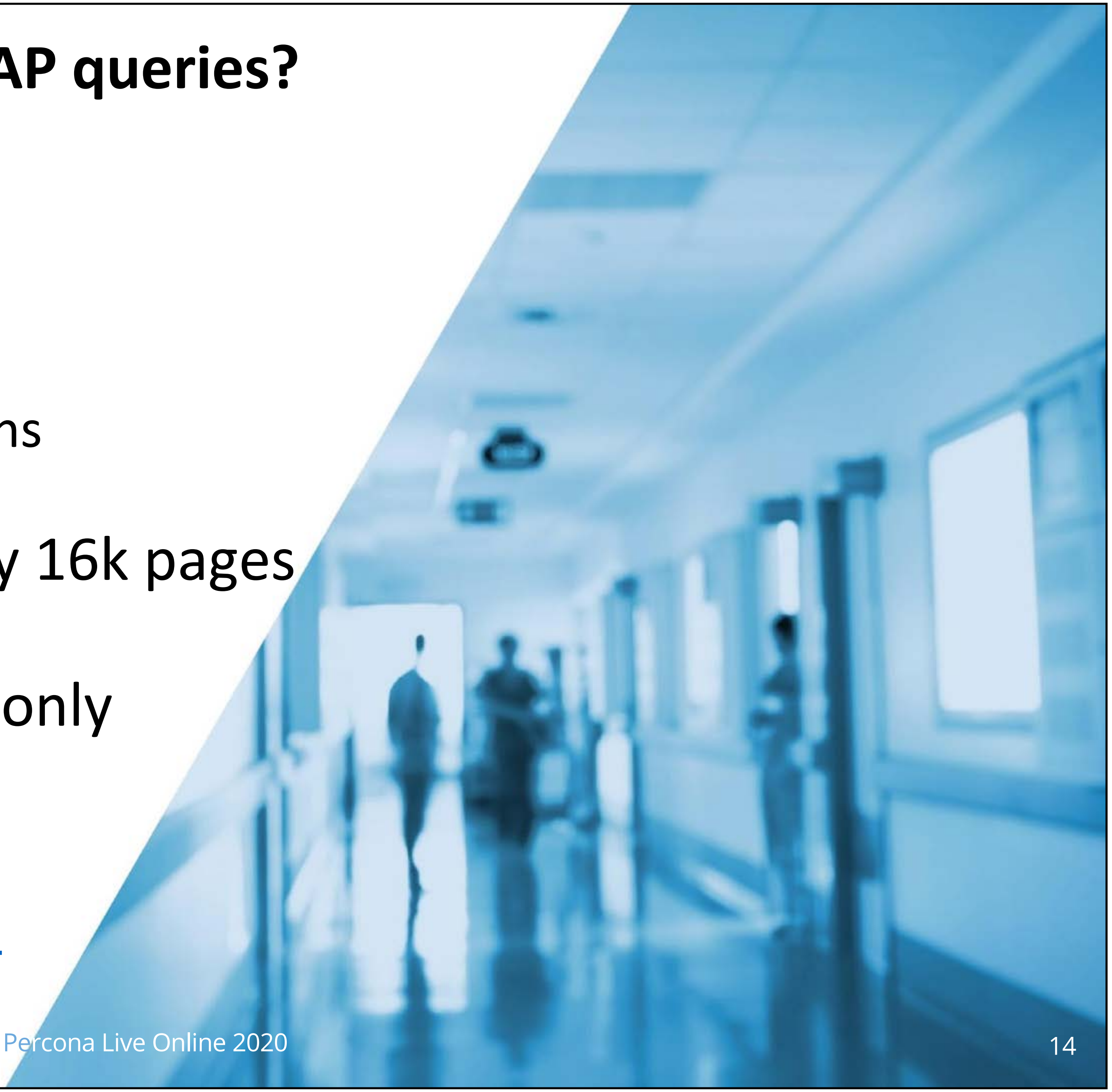
Why our OLAP queries were slow in the OLTP environment?

Rows vs. Columns

Why MariaDB is slow for OLAP queries?

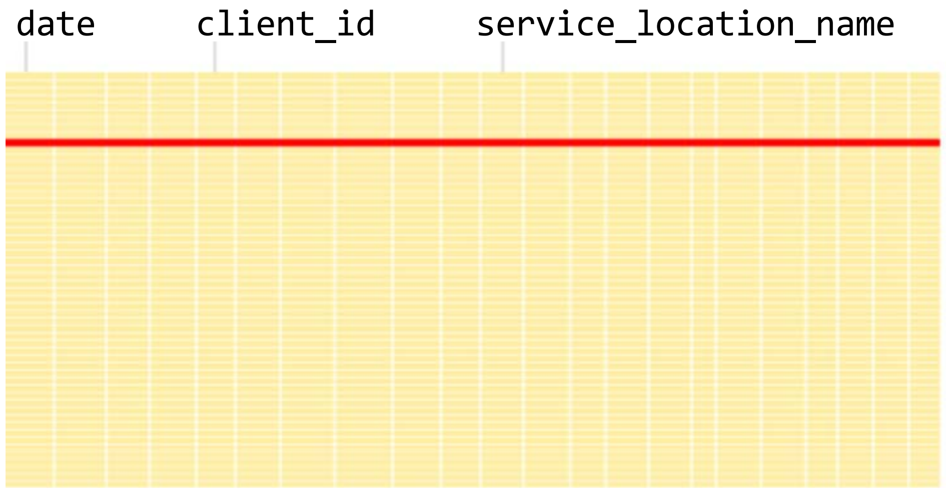
- It is row-oriented
    - if query needs two columns
        - it will read the entire row
- InnoDB organizes table by 16k pages
    - will read even more
- MariaDB/MySQL will use only one CPU-core per query
    - not utilizing all cores

      Parallel Query Execution

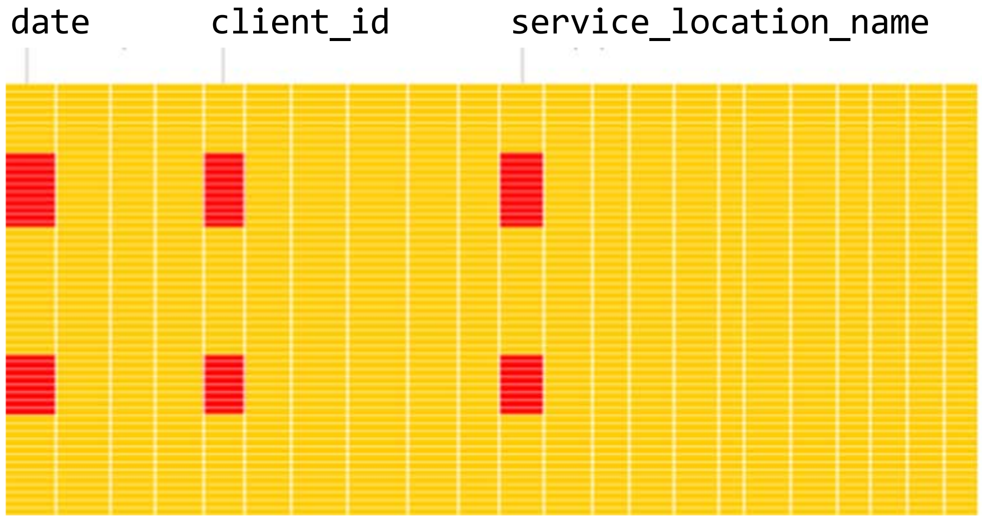


Benefits of the ColumnStore Approach

Row-oriented MariaDB



Column-oriented MariaDB



Databases comparison by ClickHouse

HELIOS VIRTUALHEALTH®

What type of queries benefited most from MariaDB ColumnStore architecture?

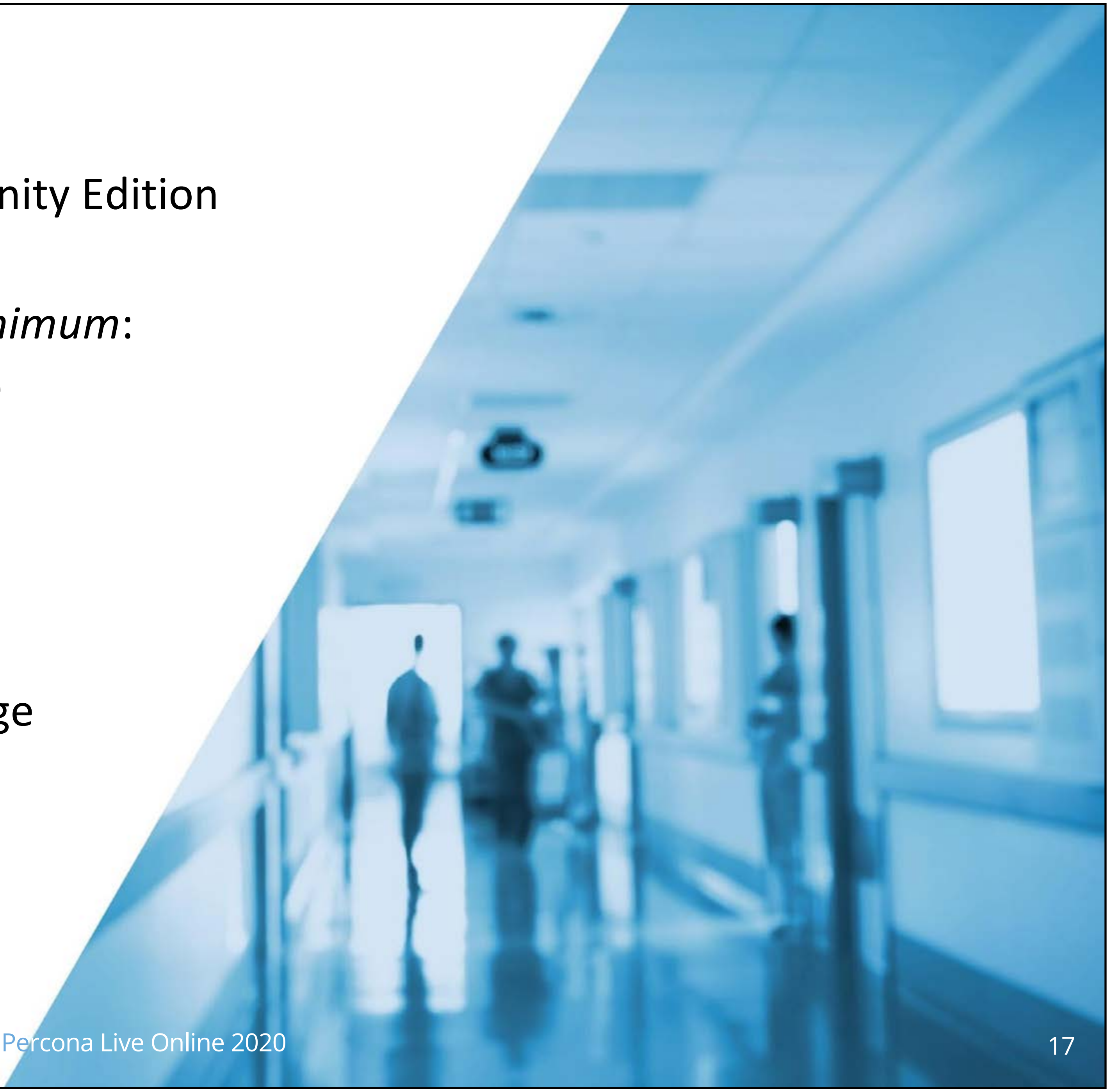
InnoDB vs. ColumnStore

MariaDB ColumnStore Tests

MariaDB ColumnStore: 1.2.5 Community Edition

  - single-node distributed install

- Testing box 1 – *recommended minimum*:
  - AWS EC2 instance: m4.4xlarge
  - RAM: 64.0 GiB
  - vCPU: 16
  - Disk: gp2 SSD
- Testing box 2:
  - AWS EC2 instance: c5d.18xlarge
  - RAM: 144.0 GiB
  - vCPU: 72
  - Disk: gp2 SSD



HELIOS VIRTUALHEALTH®

Query 1: Is it worth using MariaDB ColumnStore?

Data Source	Response time	Improvement (times)
InnoDB: no index	`10 min 53.826 sec`	1
InnoDB: *Using index*	`21 sec`	31
ColumnStore	`26 sec`	25

AWS EC2 instance: m4.4xlarge

Query 2: Using MariaDB ColumnStore

Data Source	Response time	Improvement (times)
InnoDB	9 hours 22 min 28.387 sec	
ColumnStore	**?**	

MariaDB ColumnStore: 1.2.5
AWS EC2 instance: m4.4xlarge

Query 2: Using MariaDB ColumnStore

Data Source	Response time	Improvement (times)
InnoDB	9 hours 22 min 28.387 sec	
ColumnStore	**1st attempt**	

MariaDB ColumnStore: 1.2.5
AWS EC2 instance: m4.4xlarge

```
ERROR 1815 (HY000): Internal
error: IDB-2001: Join or
subselect exceeds memory limit.
```

Query 2: Using MariaDB ColumnStore

Data Source	Response time	Improvement (times)
InnoDB	9 hours 22 min 28.387 sec	
ColumnStore	**Allow SSD Based Joins**	

MariaDB ColumnStore: 1.2.5

AWS EC2 instance: m4.4xlarge

ERROR 1815 (HY000): Internal error: IDB-2001: Join or subselect exceeds memory limit.

```
mcsadmin shutdownSystem y
/usr/local/mariadb/columnstore/bin/setConfig HashJoin AllowDiskBasedJoin Y
mcsadmin startSystem
```

Query 2: Using MariaDB ColumnStore

Data Source	Response time	Improvement (times)
InnoDB	9 hours 22 min 28.387 sec	
ColumnStore	3 min 50.772 sec	146.2

MariaDB ColumnStore: 1.2.5
AWS EC2 instance: m4.4xlarge

146 times faster!

Even with disk-based joins (using gp2 SSD volume)

Query 2: Using MariaDB ColumnStore

Data Source	Response time	Improvement (times)
InnoDB	9 hours 22 min 28.387 sec	
ColumnStore	2 min 32.626 sec	221.1

MariaDB ColumnStore: 1.2.5

AWS EC2 instance: **c5d.18xlarge**

221 times faster!

No disk-based joins

Table Sizes on Disk

Table	InnoDB (GB)	Columnstore (GB)	Improvement
client_visit	11	4.2	2.6
client_procedure	30	7.1	4.2
client_procedure_claim	5.7	0.68	8.4
client_claim	26	7.9	3.3
Total	**73**	**19.9**	**3.7**

Compression
Indexing

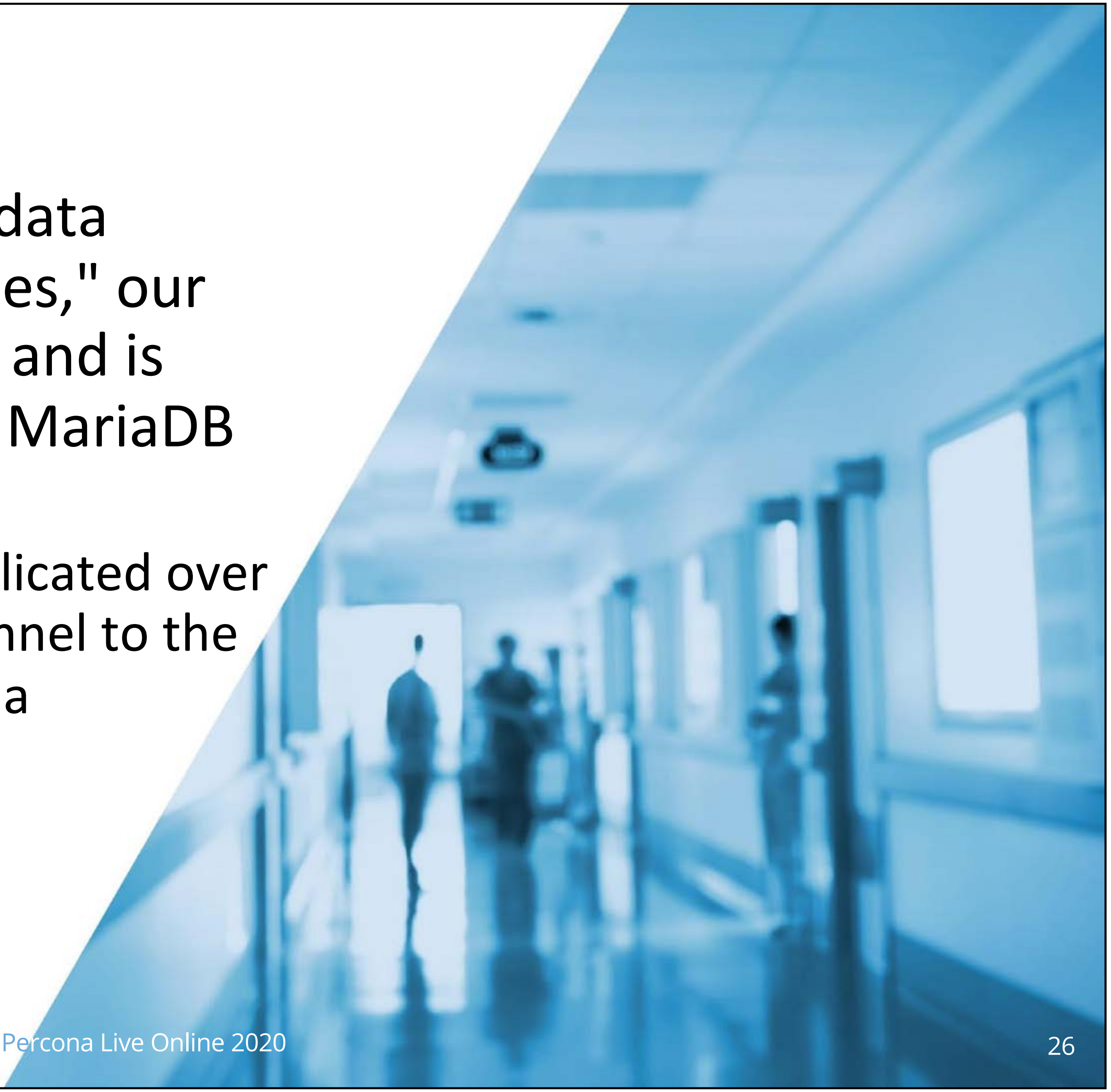
HELIOS
VIRTUALHEALTH®

How we transfer OLTP data to MariaDB ColumnStore?

0. Extract-Transform-Load

Extract

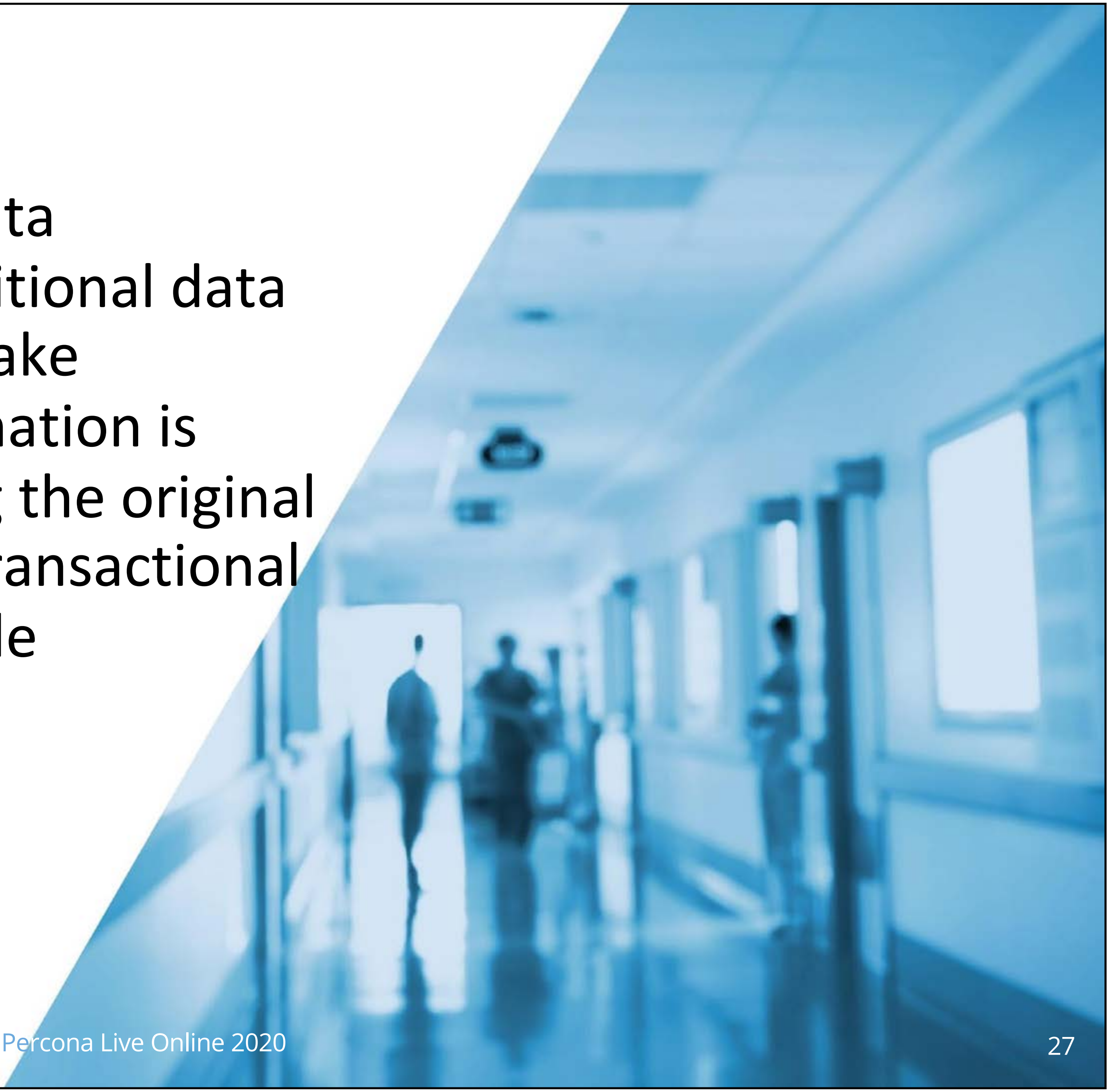
- In contrast to traditional data extraction done in "batches," our Staging Area is persistent and is implemented as a secure MariaDB slave replica
    - Data are continuously replicated over the secure encrypted channel to the same OLTP InnoDB schema



HELIOS VIRTUALHEALTH®

Transform

- In contrast to complex data transformations in a traditional data warehouse, in the Data Lake approach, data transformation is minimized, thus retaining the original form and format of our transactional data to the extent possible



Load

- We load daily data snapshots to the MariaDB ColumnStore schema like **HELIOS_ColumnStore** using a simple but elegant approach:

```
1. STOP SLAVE;
```

2. Perform efficient parallel transfer of the binary data (encrypted PHI) via multiple queries like:

```
Insert into HELIOS_ColumnStore.client_visit select * from HELIOS.client_visit;
```

```
3. START SLAVE;
```

ELT

- By minimizing complex data transformation step, we are implementing the big data ELT paradigm that avoids significant business analysis and modeling before storing data in our Data Lake
- Essentially, we are flipping the order ETL with ELT, where data transformation happens later - at the point where it is needed, such as during analysis

HELIOS
VIRTUALHEALTH®

How we transfer OLTP data to MariaDB ColumnStore?

1. ETL for Schema

Extract-Transform-Load InnoDB Schema to ColumnStore

Extract:

```
mysqldump --no-data
```

Transform:

… *change* `ENGINE=InnoDB`

to `ENGINE=Columnstore`

Schema Load

```
mcsmysql test < client_visit.sql
ERROR 1069 (42000) at line 25: Too many keys specified;
max 0 keys allowed

mcsmysql test < client_visit.sql
ERROR 1075 (42000) at line 25: Incorrect table
definition; there can be only one auto column and it
must be defined as a key
```

ColumnStore DDL Syntax Differences

You can not load InnoDB table schema to ColumnStore as is

- Remove all lines with word KEY like

```
PRIMARY KEY (`id`),
UNIQUE KEY `uuid` (`uuid`),
KEY `type` (`type`),
CONSTRAINT FK_city_id FOREIGN KEY (city_id) REFERENCES city (id)
```

- Remove AUTO_INCREMENT from column definitions like

```
`id` int unsigned NOT NULL AUTO_INCREMENT,
```

- Remove CHECK from column definitions like

```
CHECK (json_valid(`json_data`))
```

ColumnStore Unsupported Data Types

	InnoDB	ColumnStore
	binary	tinyblob
	bit	tinyint
	set	char(N)
	enum	char(N)
	year	date
	varbinary	tinyblob or blob
In 1.2.5	timestamp	datetime
In 1.2.5	mediumint	int

Other Unsupported ColumnStore DDL Syntax

- Replace ENGINE name InnoDB to ColumnStore
- Remove legacy InnoDB table definitions like

  `ROW_FORMAT=COMPACT | ROW_FORMAT=DYNAMIC`
- Remove not supported definitions like

  `DEFAULT CURRENT_TIMESTAMP | ON UPDATE CURRENT_TIMESTAMP`
- Remove unsupported collations like

  `COLLATE utf8_unicode_ci`
- Remove escaped apostrophe in possessives like

  `COMMENT 'Submitter''s ID’`
- Three-byte ZIP Code

  `mediumint(5) unsigned zerofill` replaced with `char(5)`

NULL Values vs Empty Strings

Consider string type columns like:

```
CREATE TABLE test (
    `empty_string` varchar(10) NOT NULL
) ENGINE=InnoDB;
```

Note: The implicit default for string types is an empty string

```
CREATE TABLE test_cs (
    `empty_string` varchar(10) NOT NULL
) ENGINE=Columnstore;

insert into test_cs select * from test;
```

Note: ColumnStore treats a zero-length string as a NULL value

```
Line number 1;  Error: Data violates NOT NULL constraint with no default; field 1
```

ColumnStore DDL: NOT NULL constraint with no default

Remove NOT NULL for columns with string data types

- `CHAR`
- `VARCHAR`
- `TINYTEXT/MEDIUMTEXT/TEXT/LONGTEXT`
- `TINYBLOB/MEDIUMBLOB/BLOB/LONGBLOB`

Otherwise you will be unable to load InnoDB data with empty strings

To reduce confusion, remove `DEFAULT ''`

HELIOS
VIRTUALHEALTH®

How we transfer OLTP data to MariaDB ColumnStore?

2. ETL for Data

ETL from InnoDB to ColumnStore

- Execute

  `insert into columstore_table select * from innodb_table`

  - Injects the **binary** row data from MariaDB into cpimport
  - During import, you may see two subprocesses:

```
 1300 ?        Sl    14:31  \_ /usr/local/mariadb/columnstore/mysql//bin/mysqld
 9958 ?        Sl     0:44      \_ /usr/local/mariadb/columnstore/bin/cpimport -m 1 -N -s ? -e 0 -E ? HELIOS VirtualHealth
...
 1663 ?        Sl     2:07      \_ [WriteEngineServ]
 9982 ?        S<l    2:38      |   \_ /usr/local/mariadb/columnstore/bin/cpimport.bin -e 0 -s ? -E ?
-R /tmp/columnstore_tmp_files/BrmRpt03051540539958.rpt -m 1 -P pm1-9958 -u98e45db5-41b0-42aa-8616-4c1d6e2c35f2 HELIOS VirtualHealth
```

  - Note the undocumented option -R for the BrmReport file about import
    - BRM = Block Resolution Manager

Another way to import data from InnoDB to ColumnStore

- Due to MCOL-3933, during

  ```
  insert into columstore_table select * from innodb_table
  ```

  a row with the backslash character \ results in

  ```
  ERROR 1030 (HY000) at line 1: Got error -1 "Internal error < 0 (Not system error)"
  from storage engine Columnstore
  ```

- To debug, look in your mysql datadir for files like:

```
-rw-rw---- 1 mysql mysql          83 Apr  1 20:04 VirtualHealth.tbl.Job_14171_30475.err_1
-rw-rw---- 1 mysql mysql         115 Apr  1 20:04 VirtualHealth.tbl.Job_14171_30475.bad_1
```

- To retry with a different escape (^Q) and/or separator (^G), execute:

  ```
  mcsmysql -q -e 'select * from client_memo' -N HELIOS \
  | cpimport -s '\t' HELIOS_ColumnStore VirtualHealth
  ```

Configuring data import from InnoDB to ColumnStore

- During

```
insert into columstore_table select * from innodb_table
```

you may encounter an error like:

```
ERR  : Error reading import file VirtualHealth.tbl; near line 18;  Single row fills read
buffer; try larger read buffer. [1456]
```

Due to MCOL-1234 this error is silent - but you will get as a result:

```
The following tables are locked:
LockID Name                              Process  PID   Session  CreationTime            State     DBRoots
50     HELIOS_ColumnStore.VirtualHealth  cpimport 8593  BulkLoad 2020-04-05 11:49:42 PM  Abandoned  1
```

As a workaround, use cpimport command with increased buffer, like:

```
mcsmysql -q -e 'select * from VirtualHealth' -N HELIOS |
/usr/local/mariadb/columnstore/bin/cpimport -s '\t' -c 4194304
HELIOS_ColumnStore VirtualHealth
```

cpimport default option for NULL values

- As documented, using default cpimport command, like:

```
mcsmysql -q -e 'select * from VirtualHealth' -N HELIOS |
/usr/local/mariadb/columnstore/bin/cpimport -s '\t' HELIOS_ColumnStore VirtualHealth
```

would result in replacement of NULL values with 0 for nullable INT or date/time columns, like:

```
2020-04-07 14:24:09 (14236) WARN : Column HELIOS_ColumnStore.VirtualHealth.updated_date;
Number of invalid date/times replaced with zero value : 6
```

- This is due to the default cpimport option:

```
cpimport -h
           -n      NullOption (0-treat the string NULL as data (default);
                                         1-treat the string NULL as a NULL value)
```

- To avoid that, change the default option by adding: **`cpimport -n 1`**

Big Data

- For very large tables, during
  `insert into columstore_table select * from innodb_table`
  you may experience

```
ERROR 1206 (HY000) at line 1: The total number of locks exceeds the lock table size
```

  - Increase MariaDB **`innodb_buffer_pool_size`** dynamically, then check:

```
SHOW STATUS LIKE 'Innodb_buffer_pool_resize_status';
+----------------------------------+----------------------------------------------------+
| Variable_name                    | Value                                              |
+----------------------------------+----------------------------------------------------+
| Innodb_buffer_pool_resize_status | Completed resizing buffer pool at 200403 17:13:33. |
+----------------------------------+----------------------------------------------------+
```

Binary logs during data import from InnoDB to ColumnStore

- You will accumulate huge binary logs volume during

```
insert into columstore_table select * from innodb_table
```

https://mariadb.com/kb/en/columnstore-storage-architecture/#transaction-log

- You could disable binary logging for the session

```
SET SESSION SQL_LOG_BIN=0
```

HELIOS
VIRTUALHEALTH®

Summary

Next Steps

Success

- The successful load of healthcare data to ColumnStore is attesting to its level of maturity
- A preview of healthcare systems complexity is provided by open source LibreHealthIO and OpenEMR database schemas, with about two hundred tables each
  - The VirtualHealth HELIOS database schema is on par with more comprehensive commercial electronic health records systems that have three times as much tables and thousands of columns

Summary

- Relational Data Lake built with MariaDB ColumnStore retains the source data in their original format
- We observed OLAP query speedup of more than two orders of magnitude
- “Native” MariaDB/MySQL protocol
  - easier to integrate
- Native shared nothing cluster
  - cluster version 1.5 requires Enterprise Edition

Getting Ready for Upgrade

From 1.2.5 to 1.5.4/1.5.5

MariaDB ColumnStore Versions

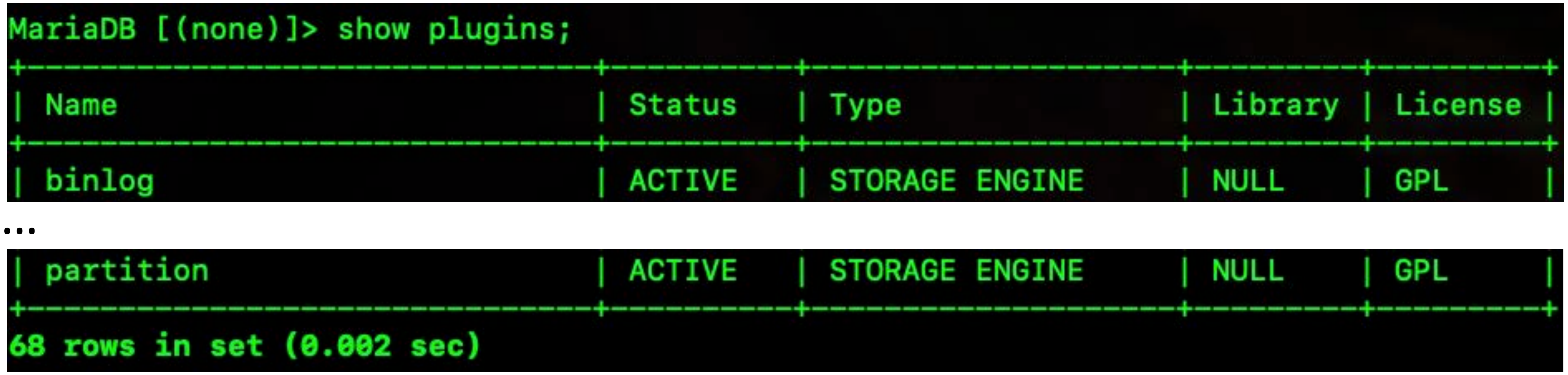
Community Edition

MariaDB	ColumnStore	Release Date
10.5.5-GA	1.5.4-Gamma	2020-08-10
10.5.4-GA	1.5.2-Beta	2020-06-24
10.3.16-GA	1.2.5-GA	2019-06-23

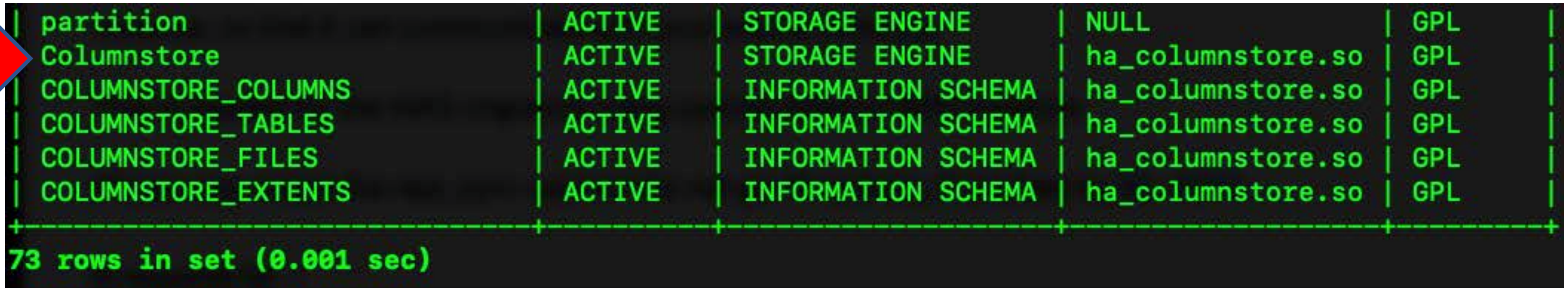
Docker

Caveat

- MariaDB 10.5.5 official docker image does not have ColumnStore



```
MariaDB [(none)]> show plugins;
+-------------------------------+----------+--------------------+---------+---------+
| Name                          | Status   | Type               | Library | License |
+-------------------------------+----------+--------------------+---------+---------+
| binlog                        | ACTIVE   | STORAGE ENGINE     | NULL    | GPL     |
...
| partition                     | ACTIVE   | STORAGE ENGINE     | NULL    | GPL     |
+-------------------------------+----------+--------------------+---------+---------+
68 rows in set (0.002 sec)
```



Column Store

```
| partition                   | ACTIVE   | STORAGE ENGINE     | NULL              | GPL     |
| Columnstore                 | ACTIVE   | STORAGE ENGINE     | ha_columnstore.so | GPL     |
| COLUMNSTORE_COLUMNS         | ACTIVE   | INFORMATION SCHEMA | ha_columnstore.so | GPL     |
| COLUMNSTORE_TABLES          | ACTIVE   | INFORMATION SCHEMA | ha_columnstore.so | GPL     |
| COLUMNSTORE_FILES           | ACTIVE   | INFORMATION SCHEMA | ha_columnstore.so | GPL     |
| COLUMNSTORE_EXTENTS         | ACTIVE   | INFORMATION SCHEMA | ha_columnstore.so | GPL     |
+-----------------------------+----------+--------------------+-------------------+---------+
73 rows in set (0.001 sec)
```

New Maturity in 10.5.5/1.5.4 and 10.5.4/1.5.2

```
MariaDB [test]> SELECT PLUGIN_DESCRIPTION, PLUGIN_AUTH_VERSION, PLUGIN_MATURITY
    -> FROM INFORMATION_SCHEMA.PLUGINS
    -> WHERE PLUGIN_TYPE='STORAGE ENGINE' AND PLUGIN_NAME='Columnstore';
+----------------------------+---------------------+-----------------+
| PLUGIN_DESCRIPTION         | PLUGIN_AUTH_VERSION | PLUGIN_MATURITY |
+----------------------------+---------------------+-----------------+
| ColumnStore storage engine | 1.5.4               | Gamma           |
+----------------------------+---------------------+-----------------+

MariaDB [test]> SELECT PLUGIN_DESCRIPTION, PLUGIN_AUTH_VERSION, PLUGIN_MATURITY
    -> FROM INFORMATION_SCHEMA.PLUGINS
    -> WHERE PLUGIN_TYPE='STORAGE ENGINE' AND PLUGIN_NAME='Columnstore';
+----------------------------+---------------------+-----------------+
| PLUGIN_DESCRIPTION         | PLUGIN_AUTH_VERSION | PLUGIN_MATURITY |
+----------------------------+---------------------+-----------------+
| ColumnStore storage engine | 1.5.2               | Beta            |
+----------------------------+---------------------+-----------------+
```

Steep Learning Curve

- MariaDB ColumnStore 1.5 underwent significant refactoring
  - It is now managed by systemd
  - infinidb_vtable is gone
- On the other hand, the systemd is absent in Docker
  - ColumnStore 1.5.2 docker image replaces systemd with tiny
  - Official ColumnStore 1.5.4 docker image has not been released yet
- As a result, you must
  - either use VirtualBox to install official 1.5.4 ColumnStore distribution
  - or build your own Docker image

to familiarize yourself with 1.5.4 ColumnStore version syntax

New Defaults in MariaDB 10.5 vs. MariaDB 10.3

```
MariaDB [test]> select @@version,@@sql_mode\G
*************************** 1. row ***************************
 @@version: 10.5.5-MariaDB-1:10.5.5+maria~stretch
@@sql_mode: STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION

MariaDB [test]> select @@version,@@sql_mode\G
*************************** 1. row ***************************
 @@version: 10.5.4-MariaDB
@@sql_mode: STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION

MariaDB [(none)]> select @@version,@@sql_mode\G
*************************** 1. row ***************************
 @@version: 10.3.16-MariaDB-log
@@sql_mode: ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
```

New Features and Behavior in 10.5.5/1.5.4

```
MariaDB [test]> alter table test engine=Columnstore;
ERROR 1815 (HY000): Internal error: CAL0001: Insert
Failed:  IDB-4015: Column 'empty_string' cannot be null.

MariaDB [test]> insert into test_cs select * from test;
ERROR 1815 (HY000): Internal error: IDB-2001: Join or
subselect exceeds memory limit.
```

Lesson Learned

- Do not wait for the new ColumnStore GA release
- Start evaluating Beta/Gamma releases now

HELIOS
VIRTUALHEALTH®

This presentation extended the VirtualHealth presentation by Alik Rubin at Percona Live 2019 in Austin

Any Questions?