Building Relational 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, the 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**

HELIOS VIRTUALHEALTH

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

HELIOS VIRTUALHEALTH

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

HELIOS VIRTUALHEALTH

Outline and Credits

- We describe the data pipeline and share tips and tricks we have learned, such as
  - Why our OLAP queries were slow in the OLTP environment?
  - What type of queries benefit most from the MariaDB ColumnStore architecture?
  - How we transfer OLTP data to the MariaDB ColumnStore?

- This presentation is inspired by the VirtualHealth presentation by Alik Rubin

HELIOS VIRTUALHEALTH

The Problem

- Running analytical (OLAP) queries on the OLTP Relational Database can be slow and painful
  - To address this problem, a special storage format - columnar - can significantly improve performance of such analytical queries

HELIOS VIRTUALHEALTH

The Open Source Solution

- Although there are several open source columnar databases,
  - in this talk, we will focus on the MariaDB ColumnStore
- We will show representative use cases, and demonstrate how MariaDB ColumnStore can be used for typical OLAP queries

Slow Queries

Row-oriented RDBMS

Query 1: Ranking

- 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)
```

Why are we so interested in visit locations?

- Specific to transportation, HELIOS is changing the paradigm through a module that enables care managers to schedule recurring trips for patients without leaving their daily platform
- One managed health plan is using the HELIOS platform to forecast transportation usage trends across months, time of day and geography to help its team optimize operations and predict expenditures
- For example, the company can determine which patients are frequent transportation users, which can alert care managers to **book multiple provider appointments for a member in one day versus multiple round-trips across several days**

HELIOS VIRTUALHEALTH

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`)
```

HELIOS VIRTUALHEALTH

Index Improvements: Using Covered Index

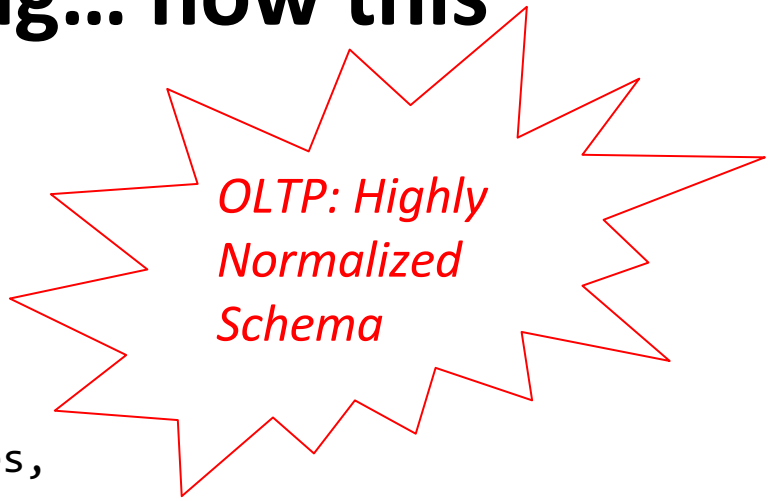
```
mysql> alter table client_visit add key comb (client_id, date,
service_location_name);
Query OK, 0 rows affected (2 min 31.424 sec)
Records: 0  Duplicates: 0  Warnings: 0

        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)
```

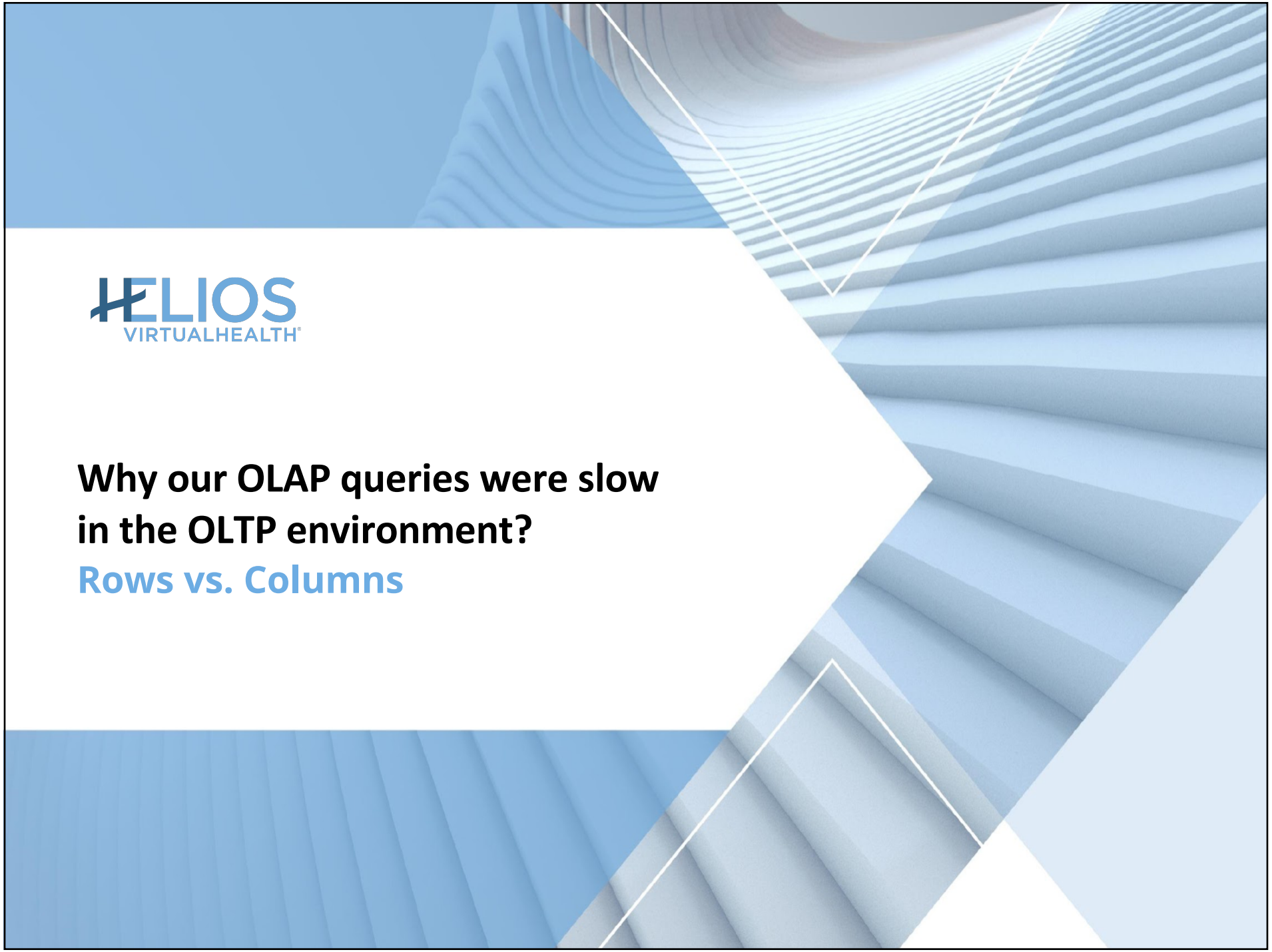
That was only the beginning… now this

```
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
```



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

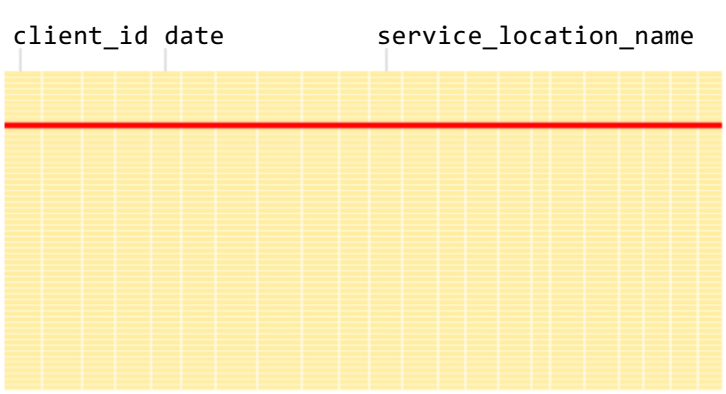
Why MariaDB is slow for OLAP queries?

- It is row-oriented
  - if query needs two columns
    - it will read the whole 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

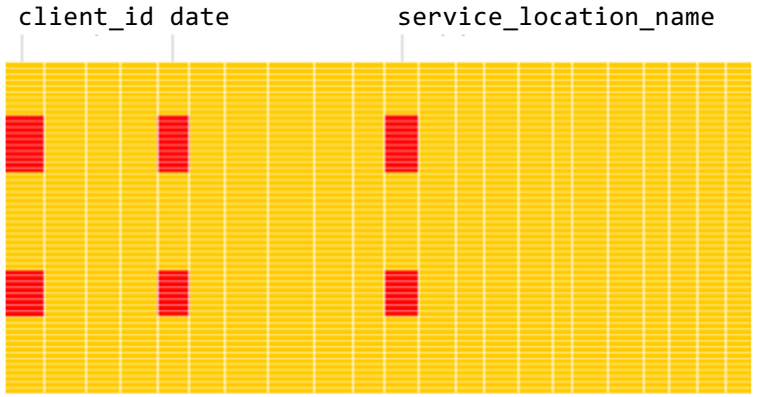
HELIOS VIRTUALHEALTH

Benefits of the ColumnStore Approach

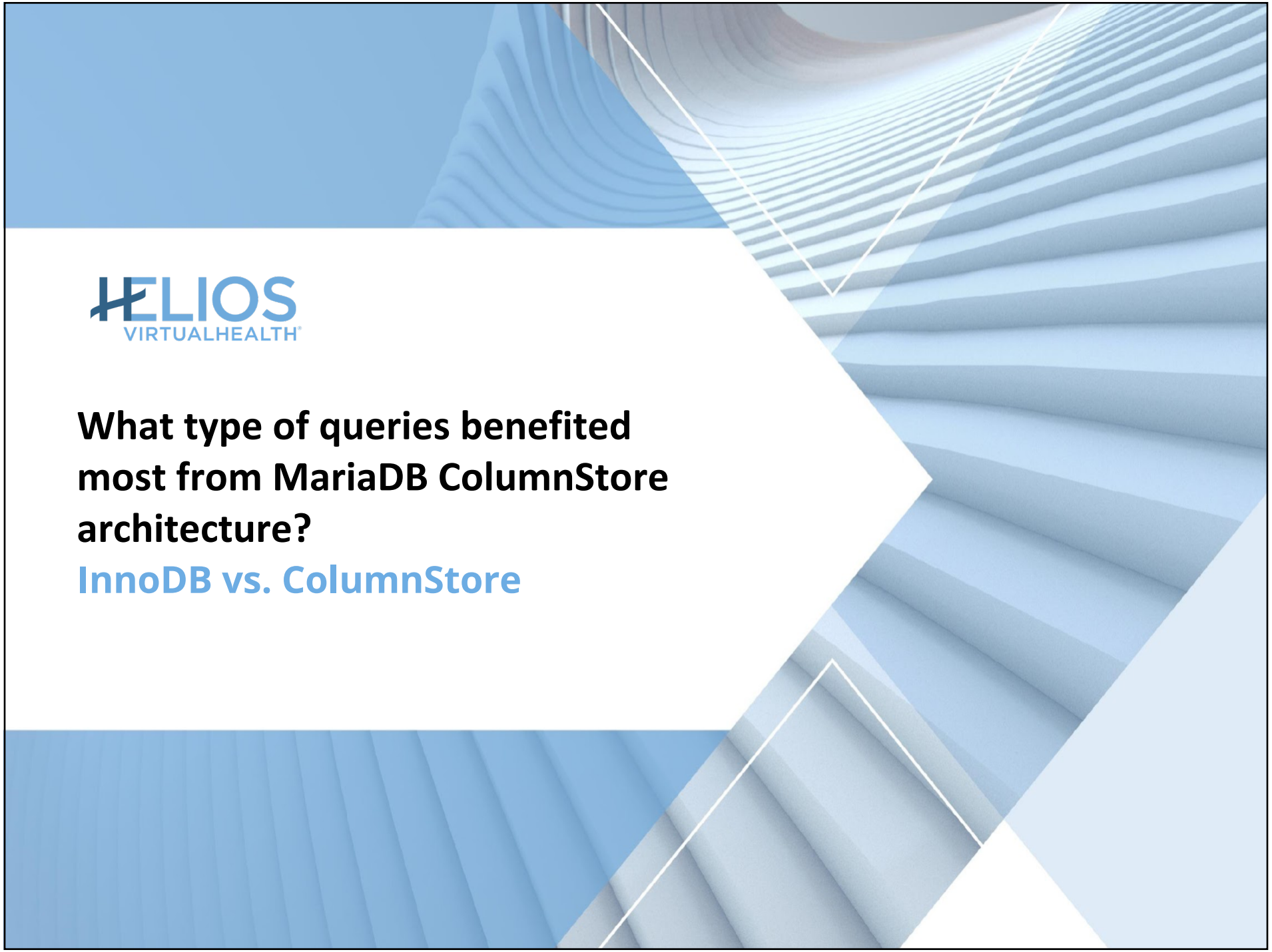
Row-oriented MariaDB



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

HELIOS VIRTUALHEALTH

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

HELIOS VIRTUALHEALTH

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

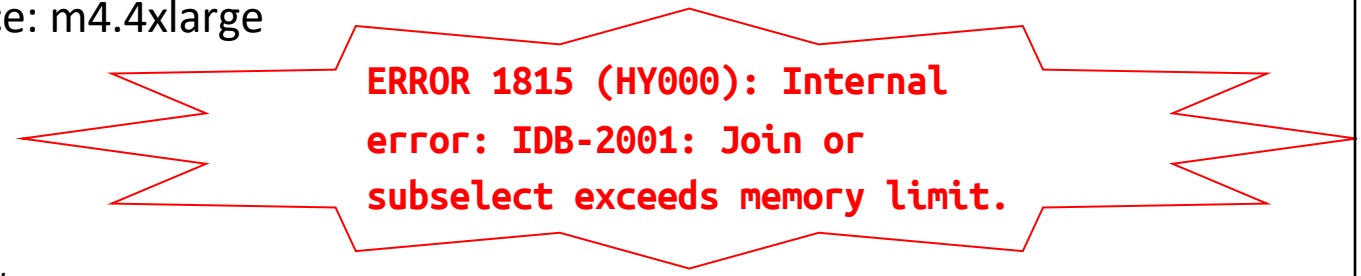
```
ERROR 1815 (HY000): Internal
error: IDB-2001: Join or
subselect exceeds memory limit.
```

HELIOS VIRTUALHEALTH

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




```
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

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

HELIOS VIRTUALHEALTH

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**

No disk-based joins

HELIOS VIRTUALHEALTH

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

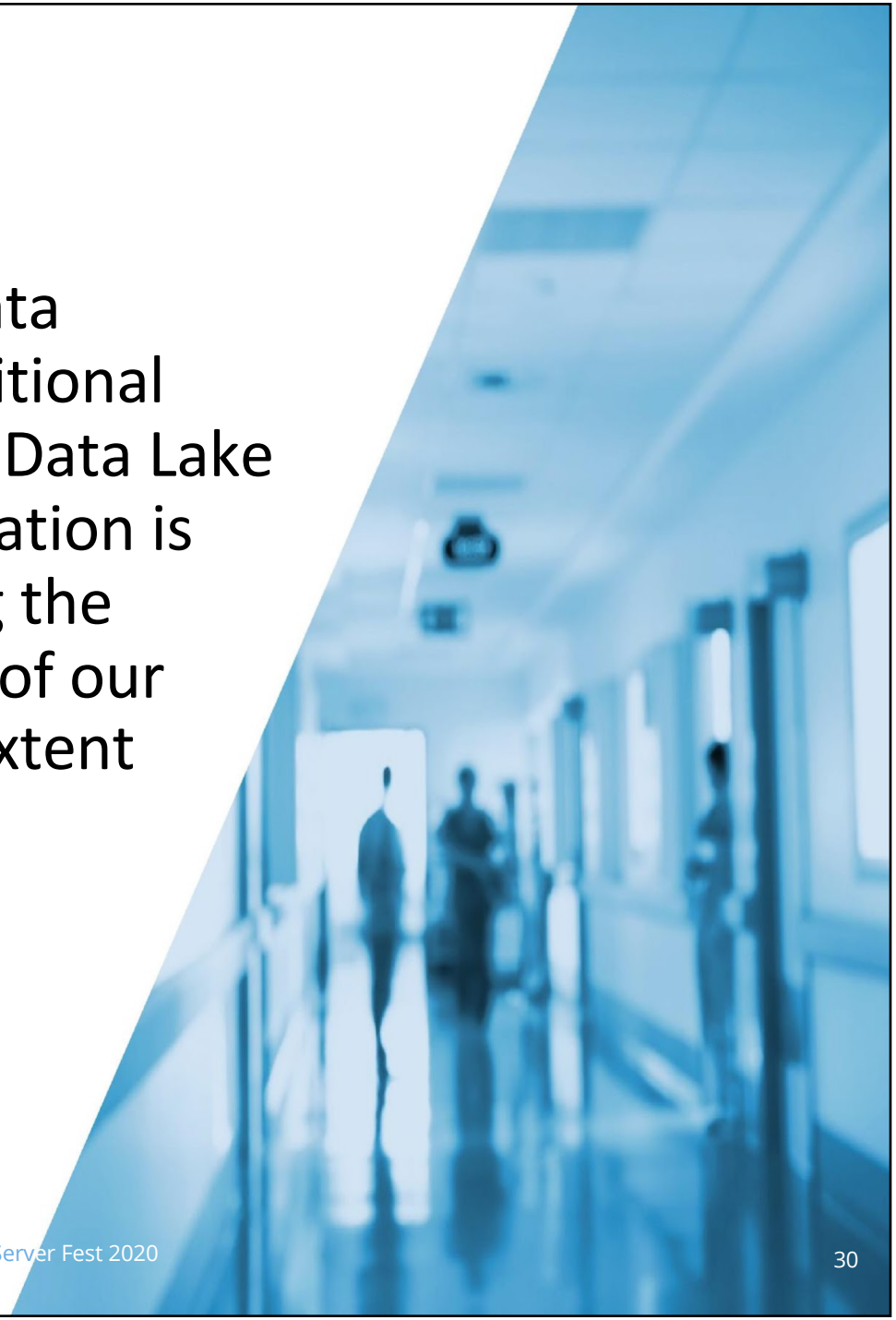

HELIOS
VIRTUALHEALTH
How we transfer OLTP data to MariaDB ColumnStore?
0. Extract-Transform-Load

Extract

- In contrast to typical 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
    - with MariaDB system-versioning enabled

Transform

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



HELIOS VIRTUALHEALTH

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. Import Schema

Schema Export from MariaDB

Customizing schema

`mysqldump --no-data`

… *change* `ENGINE=InnoDB`

to `ENGINE=Columnstore`

Schema Import from MariaDB

Customizing schema

`mysqldump --no-data`

```
$ 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 MariaDB/MySQL 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 the column definition like

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

ColumnStore Unsupported Data Types

We used the following replacements:

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

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'
```

HELIOS VIRTUALHEALTH

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
```

HELIOS VIRTUALHEALTH

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 ''`

Be Careful with Reserved Words in MariaDB ColumnStore

- Our schema has table `user` like in `mysql`.`user`
  - Whose does not?
    - Those who have table `users`

MariaDB ColumnStore / MCOL-3799

Missing backticks for reserved table name during DROP DATABASE

Details

Type:	Bug	Status:	CONFIRMED
Priority:	Minor		(View Workflow)
Affects Version/s:	1.2.5	Resolution:	Unresolved
Component/s:	DDLProc	Fix Version/s:	None
Labels:	None		
Environment:	mariadb-community-columnstore-docker Server version: 10.3.16-MariaDB-log Columnstore 1.2.5-1		
Sprint:	2020-4, 2020-5, 2020-6, 2020-7		

People

Assignee: Roman

Reporter: Sasha

Votes: 0 Vote fo

Watchers: 1 Start w

Description

ColumnStore has additional reserved words that cannot be used as table names without backticks, like USER. For example:

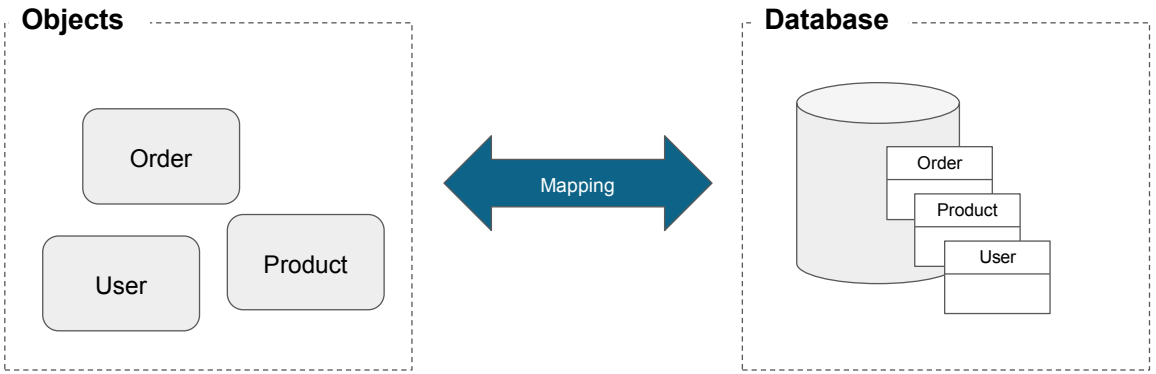
```
CREATE DATABASE test;
USE test;
CREATE TABLE user ( id int ) ENGINE=ColumnStore;
ERROR 1178 (42000): The storage engine for the table doesn't support The
```

Dates

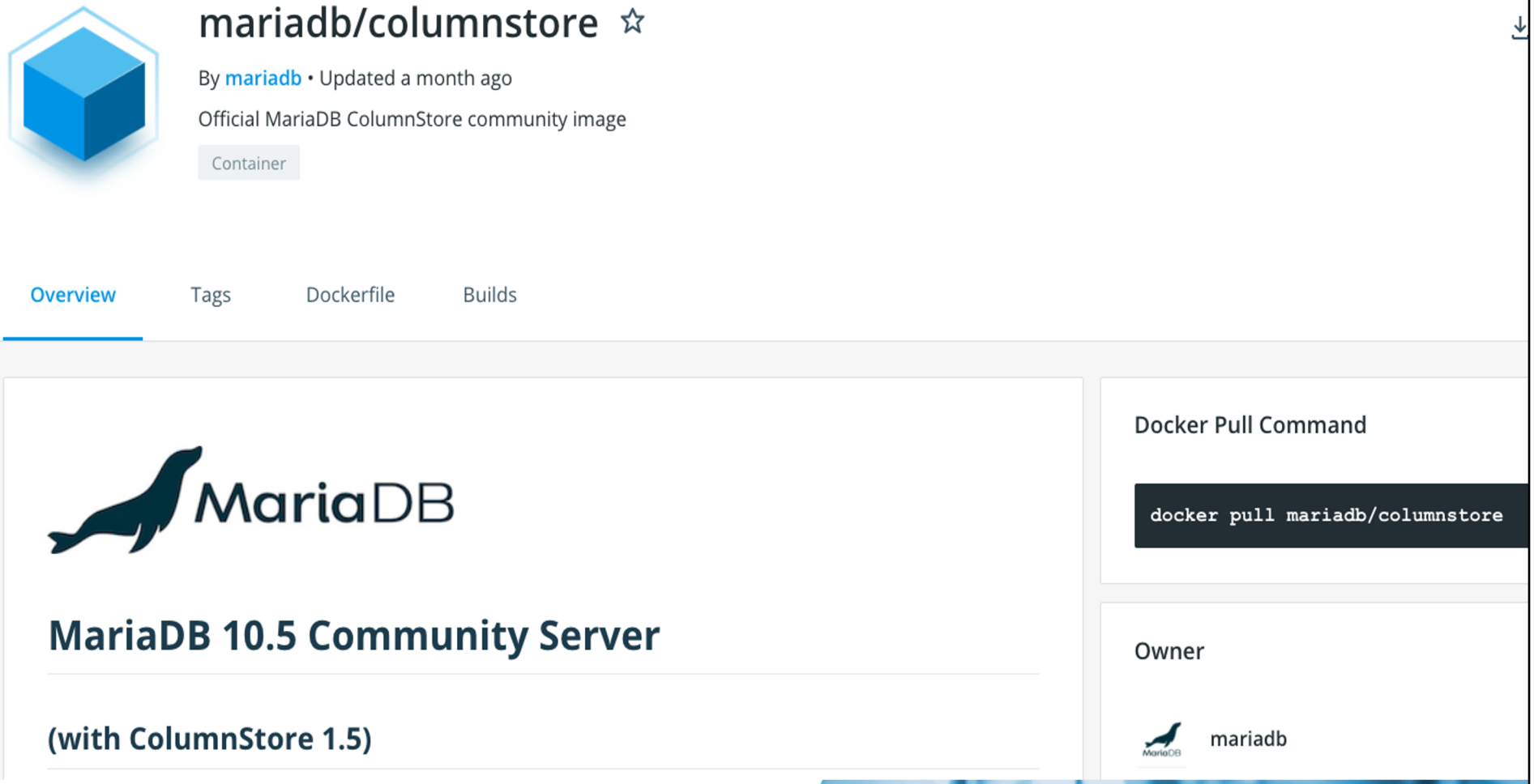
Created: 2020-02-1

Updated: 2020-04-1

What is Object-Relational Mapping?



Hint: Use ColumnStore Docker to Migrate Your Schema



How we transfer OLTP data to MariaDB ColumnStore?

2. Import Data

Import data from InnoDB to ColumnStore

- Load data into InnoDB locally

  MariaDB ColumnStore includes MariaDB server

- 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 for the files in your datadir 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
```

Importing data from InnoDB

- 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

HELIOS VIRTUALHEALTH

Conclusions

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

HELIOS VIRTUALHEALTH

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

HELIOS VIRTUALHEALTH

Next Steps

- We look forward to evaluate latest MariaDB Community Server 10.5 that includes ColumnStore plugin with S3 storage support
- The Data Lake approach looks attractive for data archival
  - As OLTP data became less relevant to daily operations with age, we must archive the old data while retaining full access via application UI
  - Data Lake retention of the original schema simplifies application data access

HELIOS VIRTUALHEALTH

HELIOS VIRTUALHEALTH

Thank you!

Any Questions?

Extra: MariaDB ColumnStore Versions

Community Edition

MariaDB	ColumnStore	Multi-node
10.5.5-GA	1.5.4-Gamma	No
10.5.4-GA	1.5.2-Beta	No
10.3.16-GA	1.2.5-GA	Yes

Docker