MySQL Summit

Top 10 tips for MySQL Performance Tuning

Configuration, best practices and tracking the ugly duckling

Mike Frank
Product Management Director
MySQL

Urvashi Oswal
Principal Member Technical Staff
MySQL

Agenda

Top Ten Tips

Agenda

Importing the data

1. Use MySQL Shell Utility
2. Speeding up import

Schema Design

3. Primary Keys
4. Indexes
5. Parallel Index Creation

Configuration

6. The right config for the workload

Memory

7. Consumption
8. Linux memory allocator

All about queries

9. Workload
10. Ugly duckling

Using Machine Learning to Solve #4 –

Autopilot Indexing


1

Importing Data at speed of light !

Importing Data

For logical dumps, MySQL Shell Dump & Load Utility should be preferred over the old and single threaded mysqldump !

MySQL Shell Dump & Load can dump a full instance, one or multiple schemas or tables. You can also add a where clause.

This tool dumps and load the data in parallel !

The data can be stored on filesystem, OCI Object Storage, S3 and Azure Blob Storage.

```
JS > util.dumpInstance("/opt/dump/", {threads: 32})
```

Importing Data (2)

The dump can be imported into MySQL using util.loadDump().
loadDump() is the method used to load dumps created by:

- util.dumpInstance()
- util.dumpSchemas()
- util.dumpTables()

```
JS > util.loadDump("/opt/dump/", {threads: 32})
```


2

Importing Data – High Speed

We can speed up the process even more ! During an initial load, & the durability is **not a problem**, if there is a crash, the process can be restarted. Therefore, if the durability is not important, we can reduce it to speed up the loading even more.

We can disable binary logs, disable redo logs and tune InnoDB by altering a few settings.

Pay attention that disabling and enabling binary logs require a restart of MySQL.

```
start mysqld with --disable-log-bin

MySQL > ALTER INSTANCE DISABLE INNODB REDO_LOG;
MySQL > set global innodb_extend_and_initialize=OFF;
MySQL > set global innodb_max_dirty_pages_pct=10;
MySQL > set global innodb_max_dirty_pages_pct_lwm=10;
```


3

Schema Design

primary keys

indexes, not too little, not too much

Primary Keys

For InnoDB, a Primary Key is required and a good one is even better !

Some theory

InnoDB stores data in table spaces.

The records are stored and sorted using the clustered index (PK).

All secondary indexes also contain the primary key as the right-most column in the index (even if this is not exposed). That means when a secondary index is used to retrieve a record, two indexes are used: first the secondary one pointing to the primary key that will be used to finally retrieve the record.

InnoDB Primary Key – Non-sequential = many pages accessed

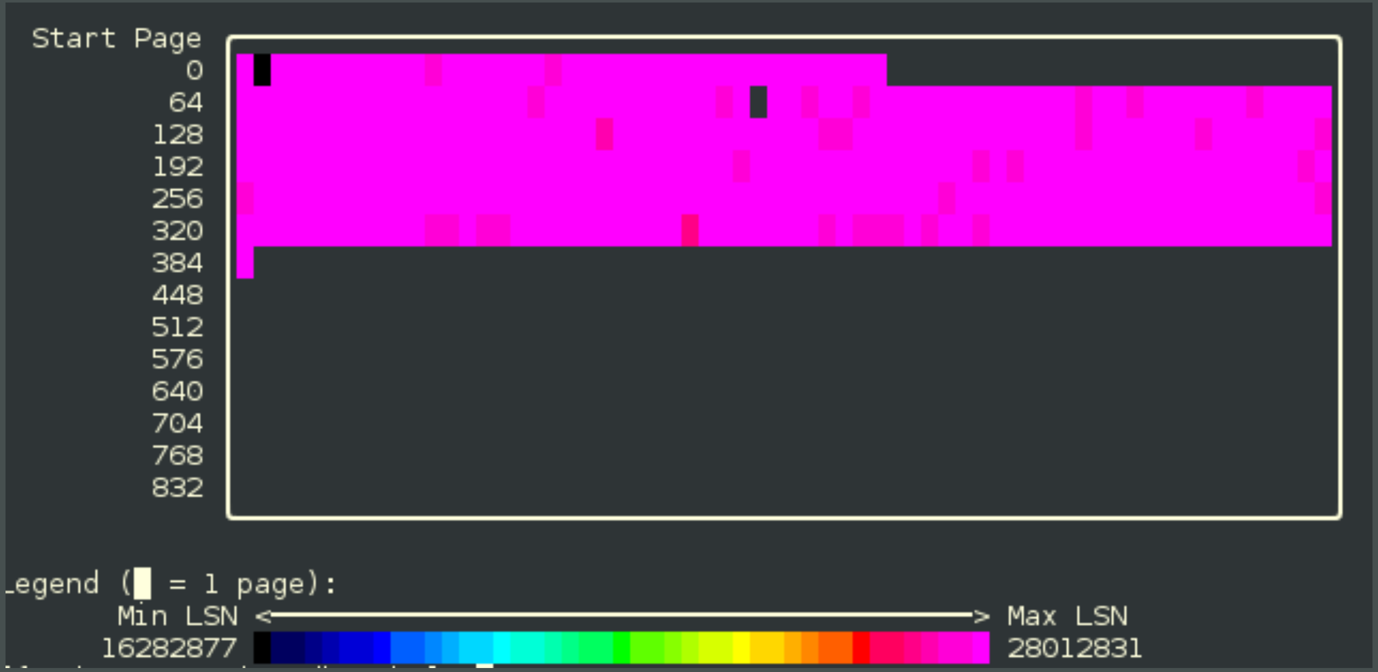
The primary key impact how the values are inserted and the size of the secondary indexes.

A non sequential PK can lead to many random IOPS.

Also, it's more and more common to use application that generates completely random primary keys

...that means if the Primary Key is not sequential,

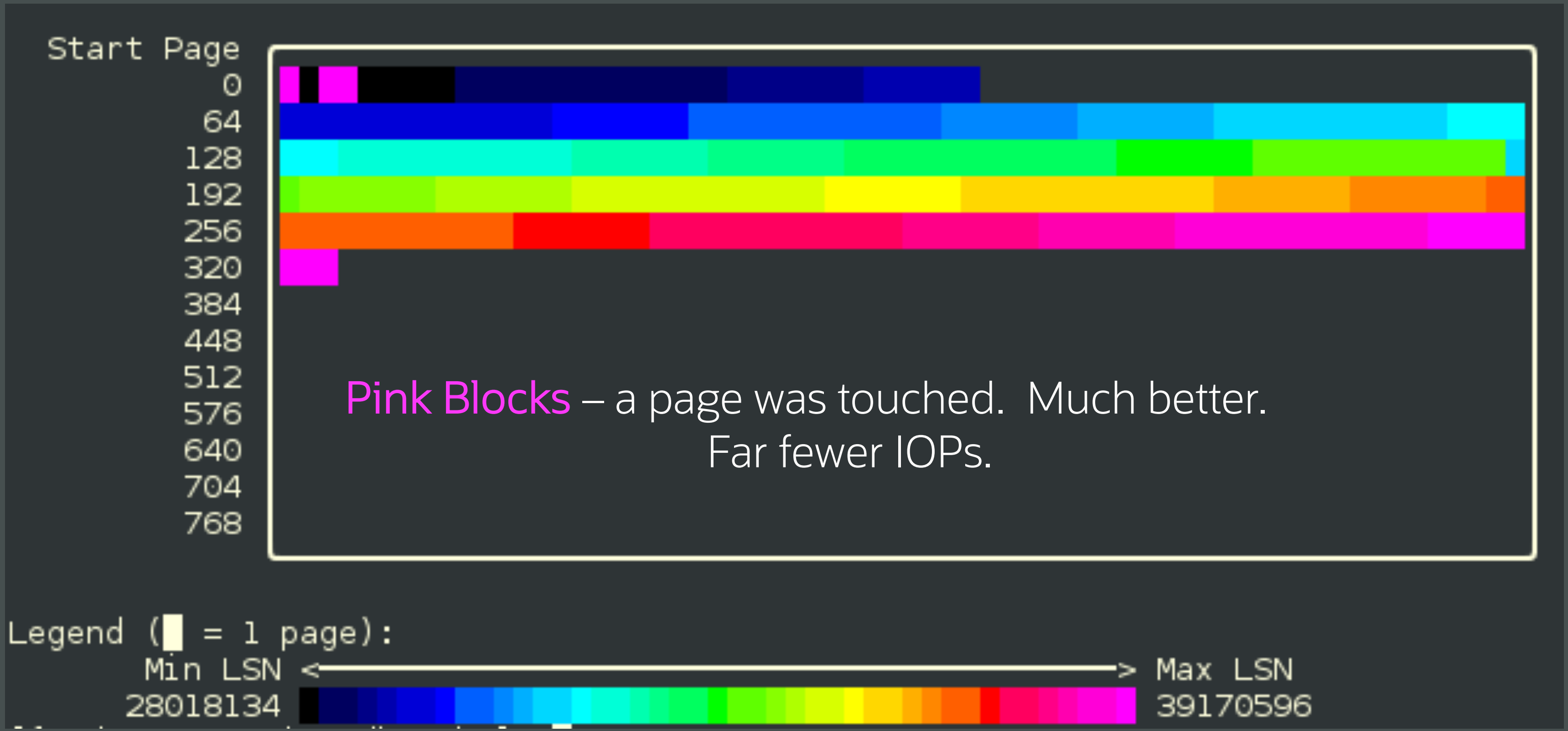
InnoDB will have to heavily re-balance all the pages on inserts.



Pink Blocks – a page was touched. Lots of IOPs here.

InnoDB Primary Key – Sequential Key – Few Page accessed

If we compare the same load (inserts) when using an auto_increment integer as Primary Key, we can see that only the latest pages are recently touched:



Generated with https://github.com/jeremycole/innodb_ruby from @jeremycole

InnoDB Primary Key ? No Key !

Another common mistake when using InnoDB is to not define any Primary Key.

When no primary key is defined, the first unique not null key is used.

And if none is available, InnoDB will create an hidden primary key (6 bytes).

The problem with such key is that you don't have any control of it and worse, this value is global to all tables without primary keys and can be a contention problem if you perform multiple simultaneous writes on such tables (dict_sys->mutex).

And if you plan for High Availability, tables without Primary Key are **NOT supported** !

InnoDB Primary Key ? No Key !

To identify those tables, run the following SQL statement, to lookup GEN_CLUST_INDEX:

```
SELECT i.TABLE_ID,
       t.NAME
FROM INFORMATION_SCHEMA.INNODB_INDEXES i
JOIN
     INFORMATION_SCHEMA.INNODB_TABLES t ON (i.TABLE_ID = t.TABLE_ID)
WHERE
    i.NAME='GEN_CLUST_INDEX';
```

see https://elephantdolphin.blogspot.com/2021/08/finding-your-hidden-innodb-primary.html

InnoDB Primary Key ? No Key ! (2)

```
+----------+---------------------------+
| TABLE_ID | NAME                      |
+----------+---------------------------+
| 1198     | slack/some_table          |
| 1472     | test/default_test         |
| 1492     | test/t1                   |
| 2018     | world/orders              |
| 2019     | world/sales               |
| 2459     | dbt3/time_statistics      |
+----------+---------------------------+
```


InnoDB GIPK mode

Since MySQL 8.0.30, MySQL supports generated invisible primary keys when running in GIPK mode !

GIPK mode is controlled by the sql_generate_invisible_primary_key server system variable.

When MySQL is running in GIPK mode, a primary key is added to a table by the server, the column and key name is always **my_row_id**.


4

Indexes, not too little, not too much - unused indexes

Having to maintain indexes that are not used can be costly and increase unnecessary iops.
Using sys Schema and innodb_index_stats it's possible to identify those unused indexes:

```
select database_name, table_name, t1.index_name,
format_bytes(stat_value * @@innodb_page_size) size
from mysql.innodb_index_stats t1
join sys.schema_unused_indexes t2 on
object_schema=database_name
and object_name=table_name and
t2.index_name=t1.index_name
where stat_name='size' order by stat_value desc;
```

Indexes, not too little, not too much - unused indexes

```
select database_name, table_name, t1.index_name,
format_bytes(stat_value * @@innodb_page_size) size
from mysql.innodb_index_stats t1
join sys.schema_unused_indexes t2 on object_schema=database_name
and object_name=table_name and t2.index_name=t1.index_name
where stat_name='size' and database_name="employees" order by stat_value
desc;
+---------------+--------------+-------------------+-----------+
| database_name | table_name   | index_name        | size      |
+---------------+--------------+-------------------+-----------+
| employees     | employees    | hash_bin_names2   | 9.52 MiB  |
| employees     | employees    | month_year_hire_idx | 6.52 MiB  |
| employees     | dept_emp     | dept_no           | 5.52 MiB  |
| employees     | dept_manager | dept_no           | 16.00 KiB |
+---------------+--------------+-------------------+-----------+
4 rows in set (0.0252 sec)
```

Drop Unused

Indexes, not too little, not too much - unused indexes

And this is the same behaviour for duplicate indexes.

There is no reason to keep maintaining them:

```
select t2.*, format_bytes(stat_value * @@innodb_page_size) size
from mysql.innodb_index_stats t1
join sys.schema_redundant_indexes t2
on table_schema=database_name and t2.table_name=t1.table_name
and t2.redundant_index_name=t1.index_name
where stat_name='size' order by stat_value desc\G
```

Duplicate Indexes

```
*************************** 1. row ***************************
table_schema: world
table_name: city
redundant_index_name: part_of_name
redundant_index_columns: Name
redundant_index_non_unique: 1
dominant_index_name: name_idx
dominant_index_columns: Name
dominant_index_non_unique: 1
subpart_exists: 1
sql_drop_index: ALTER TABLE `world`.`city` DROP INDEX `part_of_name`
size: 112.00 KiB
*************************** 2. row ***************************
table_schema: world
table_name: countrylanguage
redundant_index_name: CountryCode
redundant_index_columns: CountryCode
redundant_index_non_unique: 1
dominant_index_name: PRIMARY
dominant_index_columns: CountryCode,Language
dominant_index_non_unique: 0
subpart_exists: 0
sql_drop_index: ALTER TABLE `world`.`countrylanguage` DROP INDEX `CountryCode`
size: 64.00 KiB
2 rows in set (0.0330 sec)
```

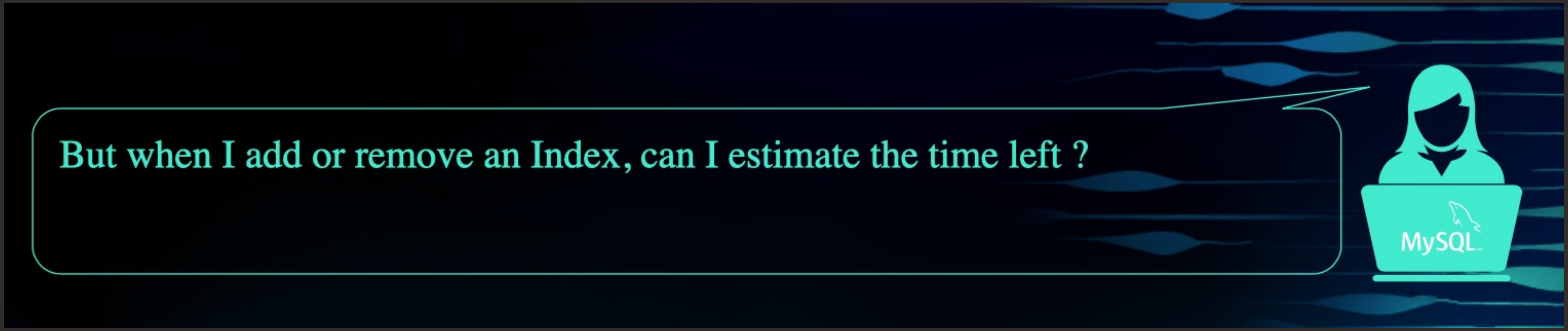
Drop the duplicate indexes

Drop the duplicate indexes

Don't forget !

Do not take recommendations at face value, check before deleting an index.

Do not delete an index immediately, but first set it as INVISIBLE for some time. Once in a while this index might be used, like for a monthly report.



Monitoring an ALTER statements progress

```
select stmt.thread_id, stmt.sql_text, stage.event_name as state,
stage.work_completed, stage.work_estimated,
lpad(concat(round(100*stage.work_completed/stage.work_estimated, 2),"%"),10," ")
as completed_at,
lpad(format_pico_time(stmt.timer_wait), 10, " ") as started_ago,
lpad(format_pico_time(stmt.timer_wait/round(100*stage.work_completed/stage.work_estimated,2)*10
0),
10, " ") as estimated_full_time,
lpad(format_pico_time((stmt.timer_wait/round(100*stage.work_completed/stage.work_estimated,2)*1
00)
-stmt.timer_wait), 10, " ") as estimated_remaining_time,
current_allocated memory
from performance_schema.events_statements_current stmt
inner join sys.memory_by_thread_by_current_bytes mt
on mt.thread_id = stmt.thread_id
inner join performance_schema.events_stages_current stage
on stage.thread_id = stmt.thread_id\G
```

Monitoring an ALTER statements progress

For example

SQL
```
select stmt.thread_id, stmt.sql_text, stage.event_name as state,
                   stage.work_completed, stage.work_estimated,
                   lpad(concat(round(100*stage.work_completed/stage.work_estimated, 2),"%"),10," ")
                   as completed_at,
                   lpad(format_pico_time(stmt.timer_wait), 10, " ") as started_ago,
                   lpad(format_pico_time(stmt.timer_wait/round(100*stage.work_completed/stage.work_estimated,2)*100),
                        10, " ") as estimated_full_time,
                   lpad(format_pico_time((stmt.timer_wait/round(100*stage.work_completed/stage.work_estimated,2)*100)
                        -stmt.timer_wait), 10, " ") as estimated_remaining_time,
                   current_allocated memory
            from performance_schema.events_statements_current stmt
            inner join sys.memory_by_thread_by_current_bytes mt
                    on mt.thread_id = stmt.thread_id
            inner join performance_schema.events_stages_current stage
                    on stage.thread_id = stmt.thread_id\G
*************************** 1. row ***************************
               thread_id: 5169
                sql_text: alter table temperature_history add index time_idx(time_stamp)
                   state: stage/innodb/alter table (read PK and internal sort)
          work_completed: 607064
          work_estimated: 1303965
            completed_at:     46.56%
             started_ago:   4.30 min
     estimated_full_time:   9.24 min
estimated_remaining_time:   4.94 min
                  memory: 3.78 MiB
```

Missing indexes

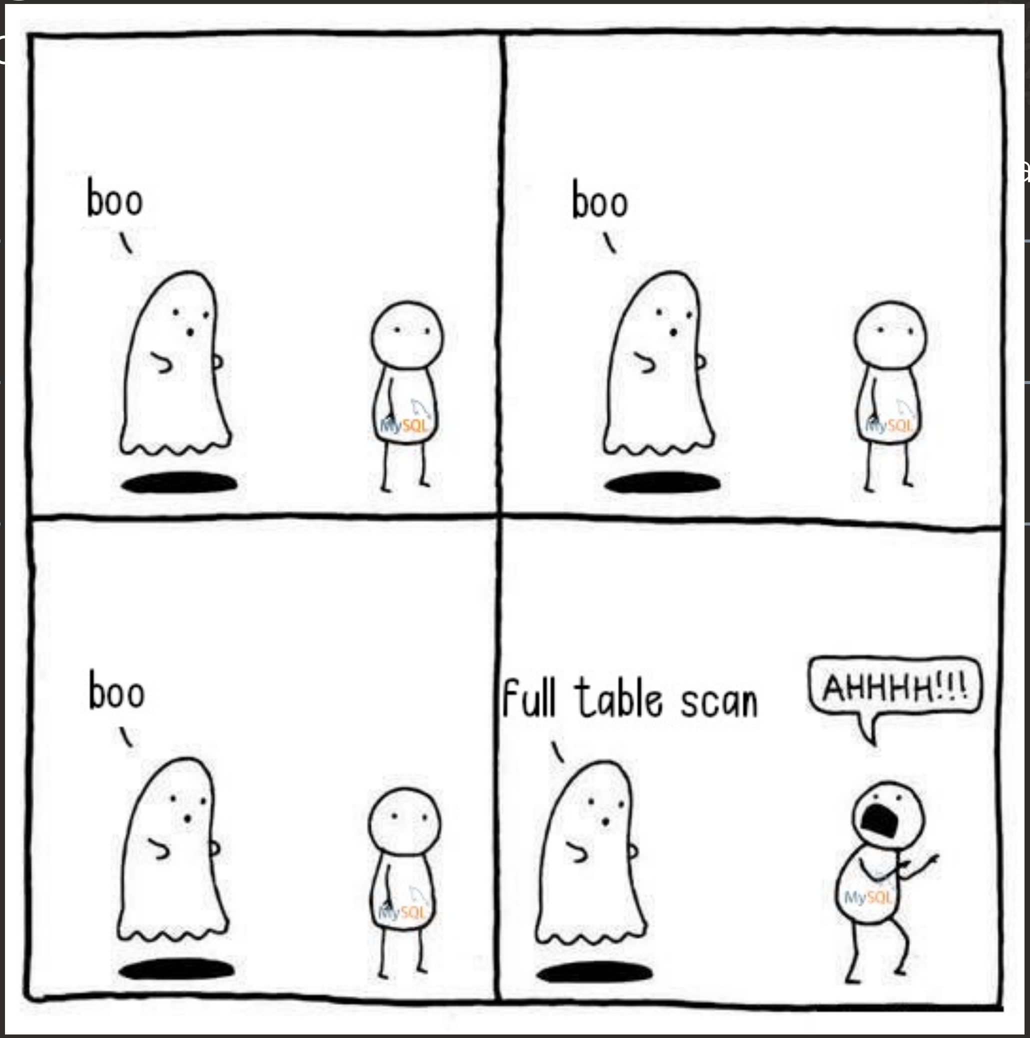
We also need to find which indexes might be missing:

```
MySQL > select * from sys.schema_tables_with_full_table_scans;
+---------------+-------------+-------------------+-----------+
| object_schema | object_name | rows_full_scanned | latency   |
+---------------+-------------+-------------------+-----------+
| students      | Customers   | 12210858800       | 41.28 min |
+---------------+-------------+-------------------+-----------+
```

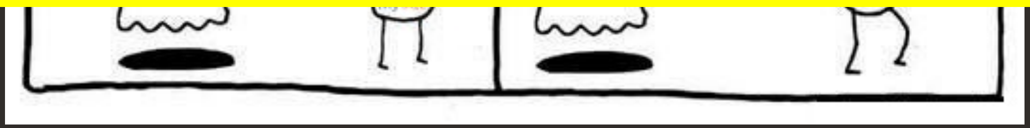
Missing indexes

We also need to finc

```
MySQL > select *                                          able_scans;
+---------------                                          -------------+------------+
| object_schema                                             latency                 |
+---------------                                          -------------+------------+
| students                                                  41.28 min               |
+---------------                                          -------------+------------+
```




```
MySQL > select * from sys.statements_with_full_table_scans where db='students' and query
like '%customers%'\G
*************************** 1. row ***************************
query: SELECT * FROM `Customers` WHERE `age` > ?
db: students
exec_count: 140
total_latency: 17.97s
no_index_used_count: 137
no_good_index_used_count: 0
no_index_used_pct: 100
rows_sent: 87220420
rows_examined: 12210858800
rows_sent_avg: 623003
rows_examined_avg: 2505942
first_seen: 23-01-27 14:34:12.66877
last_seen: 2023-02-23 17:44:47.738911
digest: 4396a7fc5d8f2cdc157b04bbd0543facaeaa5d4bb0ab02734b101ab5018a9b18
```

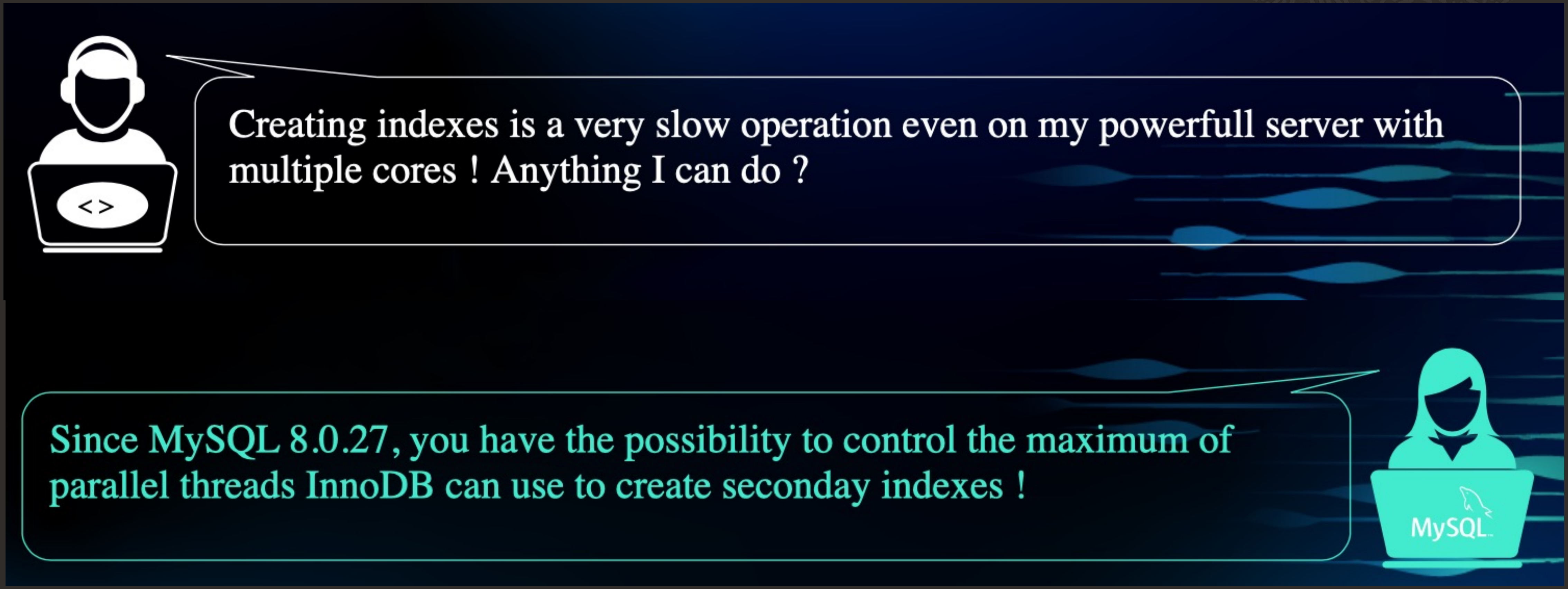


Looks like – Machine Learning could automate this process

Autopilot Indexing – demo


5

Index Creation is slow



Parallel Index Creation - example

```
MySQL > alter table booking
add index idx_2(flight_id, seat, passenger_id);
Query OK, 0 rows affected (9 min 0.6838 sec)
```

The default settings are:

```
innodb_ddl_threads = 4
innodb_ddl_buffer_size = 1048576
innodb_parallel_read_threads = 4
```

The **innodb_ddl_buffer_size** is shared between all **innodb_ddl_threads** defined.
If you increase the amount of threads, we recommend that you also increase the buffer size.

Parallel Index Creation - example (2)

To find the best values for these variables, let's have a look at the amount of CPU cores:

```
MySQL > select count from information_schema.INNODB_METRICS
where name = 'cpu_n';
+--------+
| count |
+--------+
| 16     |
+--------+
```

So we have **16 cores** to share.

As this machine has plenty of memory, we can allocate 1GB for the InnoDB DDL buffer.

Parallel Index Creation - example (3)

```
MySQL > SET innodb_ddl_threads = 8;
MySQL > SET innodb_parallel_read_threads = 8;
MySQL > SET innodb_ddl_buffer_size = 1048576000;
```

We can now retry the same index creation as previously:

```
MySQL > alter table booking add index idx_2(flight_id, seat, passenger_id);
Query OK, 0 rows affected (2 min 43.1862 sec)
```

Parallel Index Creation - example (4)

Best to run tests to define the optimal settings for your database, your hardware and data.

For example, here we got the best result setting the buffer size to 2GB
and both ddl threads and parallel read threads to 4.

It took 2 min 43 sec, much better than the initial 9 minutes !

For more information, go to
https://lefred.be/content/mysql-8-0-innodb-parallel-threads-for-online-ddl-operations/


6

Configuration

when MySQL is configured to match the workload

The secret #1 is the size of InnoDB Buffer Pool

It's important to have the working set in memory.

The size of the InnoDB Buffer Pool is important:

```
MySQL > SELECT format_bytes(@@innodb_buffer_pool_size) BufferPoolSize,
FORMAT(A.num * 100.0 / B.num,2) BufferPoolFullPct,
FORMAT(C.num * 100.0 / D.num,2) BufferPollDirtyPct
FROM
(SELECT variable_value num FROM performance_schema.global_status
WHERE variable_name = 'Innodb_buffer_pool_pages_data') A,
(SELECT variable_value num FROM performance_schema.global_status
WHERE variable_name = 'Innodb_buffer_pool_pages_total') B,
(SELECT variable_value num FROM performance_schema.global_status
WHERE variable_name='Innodb_buffer_pool_pages_dirty') C,
(SELECT variable_value num FROM performance_schema.global_status
WHERE variable_name='Innodb_buffer_pool_pages_total') D;
```

The secret #1 is the size of InnoDB Buffer Pool

It's important to have the working set in memory.

The size of the InnoDB Buffer Pool is important:

```
MySQL > SELECT format_bytes(@@innodb_buffer_pool_size) BufferPoolSize,
FORMAT(A.num * 100.0 / B.num,2) BufferPoolFullPct,
FORMAT(C.num * 100.0 / D.num,2) BufferPollDirtyPct
FROM
(SELECT vari
WHERE varia
(SELECT vari
WHERE variable_name = 'Innodb_buffer_pool_pages_total') B,
(SELECT variable_value num FROM performance_schema.global_status
WHERE variable_name='Innodb_buffer_pool_pages_dirty') C,
(SELECT variable_value num FROM performance_schema.global_status
WHERE variable_name='Innodb_buffer_pool_pages_total') D;
```

```
+----------------+-------------------+--------------------+
| BufferPoolSize | BufferPoolFullPct | BufferPollDirtyPct |
+----------------+-------------------+--------------------+
| 128.00 MiB     | 87.12             | 0.36               |
+----------------+-------------------+--------------------+
1 row in set (0.0012 sec)
```

The secret #1 is the size of InnoDB Buffer Pool (2)

We can also verify the Ratio of pages requested and read from disk:

```
MySQL > SELECT FORMAT(A.num * 100 / B.num,2) DiskReadRatioPct
FROM
(SELECT variable_value num FROM performance_schema.global_status
WHERE variable_name = 'Innodb_buffer_pool_reads') A,
(SELECT variable_value num FROM performance_schema.global_status
WHERE variable_name = 'Innodb_buffer_pool_read_requests') B;
+------------------+
| DiskReadRatioPct |
+------------------+
| 3.53             |
+------------------+
```


Secret #2: InnoDB Redo Log

Too big or too small can affect perform

It's not recommended to oversize the Redo Log Capacity.

Redo Log files consume disk space and increases the recovery time in case of a restart (innodb_fast_shutdown=1) or a sudden crash.

And it also slows down shutdown when innodb_fast_shutdown=0.

Secret #2: InnoDB Redo Log - Recommendations

During peak traffic time, you can get an estimation of the required amount for the Redo Log Capacity by running the query below (all in one single line):

```
MySQL > SELECT VARIABLE_VALUE from performance_schema.global_status
WHERE VARIABLE_NAME='Innodb_redo_log_current_lsn' INTO @a;SELECT sleep(60)
INTO @garb ;SELECT VARIABLE_VALUE FROM performance_schema.global_status
WHERE VARIABLE_NAME='Innodb_redo_log_current_lsn' INTO @b;select
format_bytes(abs(@a - @b)) per_min, format_bytes(abs(@a - @b)*60) per_hour;
+-----------+-----------+
| per_min   | per_hour  |
+-----------+-----------+
| 21.18 MiB | 1.24 GiB  |
+-----------+-----------+
```

Secret #2: InnoDB Redo Log - Recommendations

During peak traffic time, you can get an estimation of the required amount for the Redo Log Capacity by running the query below (all in one single line):

```
MySQL > SELECT VARIABLE_VALUE from performance_schema.global_status
WHERE VARIABLE_NAME='Innodb_redo_log_current_lsn' INTO @a;SELECT sleep(60)
INTO @garb ;SELECT VARIABLE_VALUE FROM performance_schema.global_status
WHERE
format_
+-------
| per_min | per_hour |
+-----------+----------+
| 21.18 MiB | 1.24 GiB |
+-----------+----------+
```

```
MySQL > SET persist innodb_redo_log_capacity=1.24*1024*1024*1024;
```

Optimal InnoDB Configuration to start

On a dedicated MySQL Server,
the best is to let InnoDB decide the size of the Buffer Pool and the Redo Log Capacity.

In my.cnf:
innodb_dedicated_server=1

See https://dev.mysql.com/doc/refman/8.0/en/innodb-dedicated-server.html

Auto SHAPE selection ...


7

Memory Consumption

How much memory and how to limit it

Memory - InnoDB

The secret is to always run a production server with a warm Buffer Pool.

If you need to restart MySQL for any reason (maintenance, updgrade, crash), it's recommended to dump the content of the InnoDB Buffer Pool to disk and load it at startup:

```
innodb_buffer_pool_dump_at_shutdown=1
innodb_buffer_pool_load_at_startup=1
```

Memory - InnoDB (2)

We can get the InnoDB Buffer Pool memory allocation usage with the following query:

```
MySQL > SELECT * FROM sys.memory_global_by_current_bytes
WHERE event_name LIKE 'memory/innodb/buf_buf_pool'\G
*************************** 1. row ***************************
event_name: memory/innodb/buf_buf_pool
current_count: 1
current_alloc: 130.88 MiB
current_avg_alloc: 130.88 MiB
high_count: 1
high_alloc: 130.88 MiB
high_avg_alloc: 130.88 MiB
1 row in set (0.0010 sec)
```

Memory - MySQL

From Performance_Schema (and sys) we can get information about the Memory consumption of MySQL, this instrumentation has been extended in MySQL 8.0:

```
SELECT * FROM sys.memory_global_total;
```

And you can have details related to the code area:

```
SELECT SUBSTRING_INDEX(event_name,'/',2) AS code_area,
format_bytes(SUM(current_alloc)) AS current_alloc
FROM sys.x$memory_global_by_current_bytes
GROUP BY SUBSTRING_INDEX(event_name,'/',2)
ORDER BY SUM(current_alloc) DESC;
```

```
+------------------+
| total_allocated |
+------------------+
| 4.28 GiB |
+------------------+
+----------------------------+----------------+
| code_area                  | current_alloc |
+----------------------------+----------------+
| memory/innodb              | 2.30 GiB       |
| memory/group_rpl           | 1024.00 MiB.   |
| memory/performance_schema | 916.88 MiB     |
| memory/sql                 | 75.80 MiB      |
| memory/mysys               | 9.13 MiB       |
| memory/temptable           | 3.00 MiB       |
| memory/mysqlx              | 22.42 KiB      |
| memory/vio                 | 3.16 KiB       |
+----------------------------+----------------+
```


8

Memory: better allocation = better performance !

To have better performance choosing the right memory allocator (Linux) is important !

The default memory allocator in Linux distribution (glibc-malloc) doesn't perform well in high concurrency environments and should be avoided !

Fortunately we have 2 other choices:

- jemalloc (good for perf, but less RAM management efficiency)
- tcmalloc (recommended choice)

Memory: better allocation = better performance ! (2)

Install tcmalloc:

```
$ sudo yum -y install gperftools-libs
```

And in systemd service file you need to add:

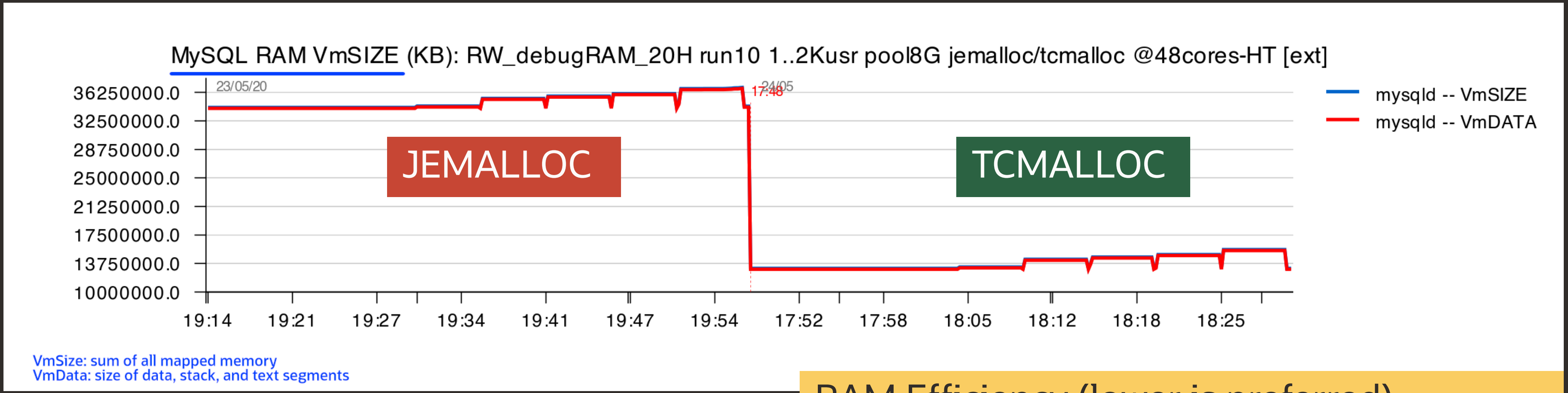
```
$ sudo EDITOR=vi systemctl edit mysqld
[Service]
Environment="LD_PRELOAD=/usr/lib64/libtcmalloc_minimal.so.4"
```

Memory: better allocation = better performance ! (3)

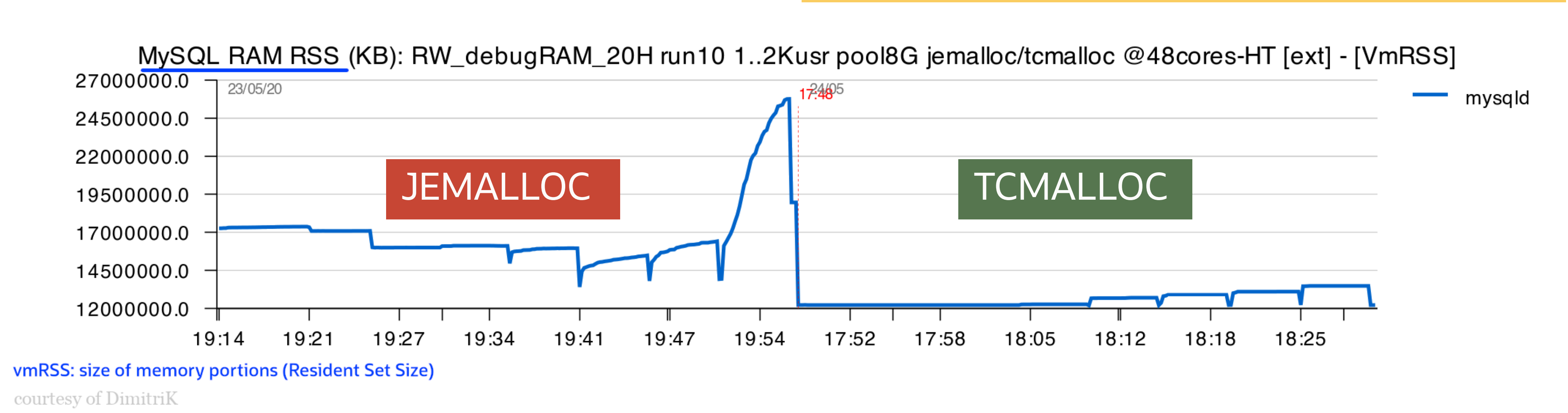
Reload the service and restart MySQL:
Memory Allocator: jemalloc vs tcmalloc:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart mysqld
```


Memory Allocator: jemalloc vs tcmalloc:



RAM Efficiency (lower is preferred)




9

All about queries

everything you need to know about your queries

Know your workload ! Overall

It's important to know what type of workload your database is performing.
Most of the time, people are surprised with the result !

```
MySQL > SELECT SUM(count_read) `tot reads`,
CONCAT(ROUND((SUM(count_read)/SUM(count_star))*100, 2),"%") `reads`,
SUM(count_write) `tot writes`,
CONCAT(ROUND((SUM(count_write)/sum(count_star))*100, 2),"%") `writes`
FROM performance_schema.table_io_waits_summary_by_table
WHERE count_star > 0 ;
+-----------+---------+------------+---------+
| tot reads | reads   | tot writes | writes  |
+-----------+---------+------------+---------+
| 16676217  | 99.11% | 149104     | 0.89%   |
+-----------+---------+------------+---------+
```

Know your workload ! (2) – Per schema

```
MySQL > SELECT object_schema,
CONCAT(ROUND((SUM(count_read)/SUM(count_star))*100, 2),"%") `reads`,
CONCAT(ROUND((SUM(count_write)/SUM(count_star))*100, 2),"%") `writes`
FROM performance_schema.table_io_waits_summary_by_table
WHERE count_star > 0 GROUP BY object_schema;
+-----------------------------+----------+---------+
| object_schema               | reads    | writes  |
+-----------------------------+----------+---------+
| sys                         | 100.00% | 0.00%.  |
| mydb                        | 100.00% | 0.00%   |
| test                        | 100.00% | 0.00%   |
| docstore                    | 100.00% | 0.00%   |
| sbtest                      | 99.09%  | 0.91%   |
+-----------------------------+----------+---------+
```

Know your workload ! (3) – Per Table

And we can check the statistics per table:

```
MySQL > SELECT object_schema, object_name,
CONCAT(ROUND((count_read/count_star)*100, 2),"%") `reads`,
CONCAT(ROUND((count_write/count_star)*100, 2),"%") `writes`
FROM performance_schema.table_io_waits_summary_by_table
WHERE count_star > 0 and object_schema='sbtest' ;
+---------------+-------------+--------+--------+
| object_schema | object_name | reads  | writes |
+---------------+-------------+--------+--------+
| sbtest        | sbtest1     | 99.67% | 0.33%  |
| sbtest        | sbtest2     | 97.71% | 2.29%  |
| sbtest        | sbtest3     | 97.71% | 2.29%  |
| sbtest        | sbtest4     | 97.73% | 2.27%  |
+---------------+-------------+--------+--------+
```


10

Finding the Ugly Duckling

We can define bad queries in two different categories:

- Queries called too often
- Queries that are too slow
  - Full table scan
  - Use filesort
  - Use temporary tables

If there could be only one?

If you should optimize only one query, the best candidate should be
the query that consumes the most of the execution time (seen as latency in PFS, aka *"response time"*).

sys Schema contains all the necessary info to find that Ugly Duckling:

```
SELECT schema_name, format_pico_time(total_latency) tot_lat,
exec_count, format_pico_time(total_latency/exec_count) latency_per_call,
query_sample_text
FROM sys.x$statements_with_runtimes_in_95th_percentile AS t1
JOIN performance_schema.events_statements_summary_by_digest AS t2
ON t2.digest=t1.digest
WHERE schema_name NOT in ('performance_schema', 'sys')
ORDER BY (total_latency/exec_count) desc LIMIT 1\G
```

If there could be only one? And we have the biggest loser.

```
*************************** 1. row ***************************
schema_name: piday
tot_lat: 4.29 h
exec_count: 5
latency_per_call: 51.51 min
query_sample_text: select a.device_id, max(a.value) as `max temp`,
min(a.value) as `min temp`, avg(a.value) as `avg temp`,
max(b.value) as `max humidity`, min(b.value) as `min humidity`,
avg(b.value) as `avg humidity`
from temperature_history a
join humidity_history b on b.device_id=a.device_id
where date(a.time_stamp) = date(now())
and date(b.time_stamp)=date(now()) group by device_id
```

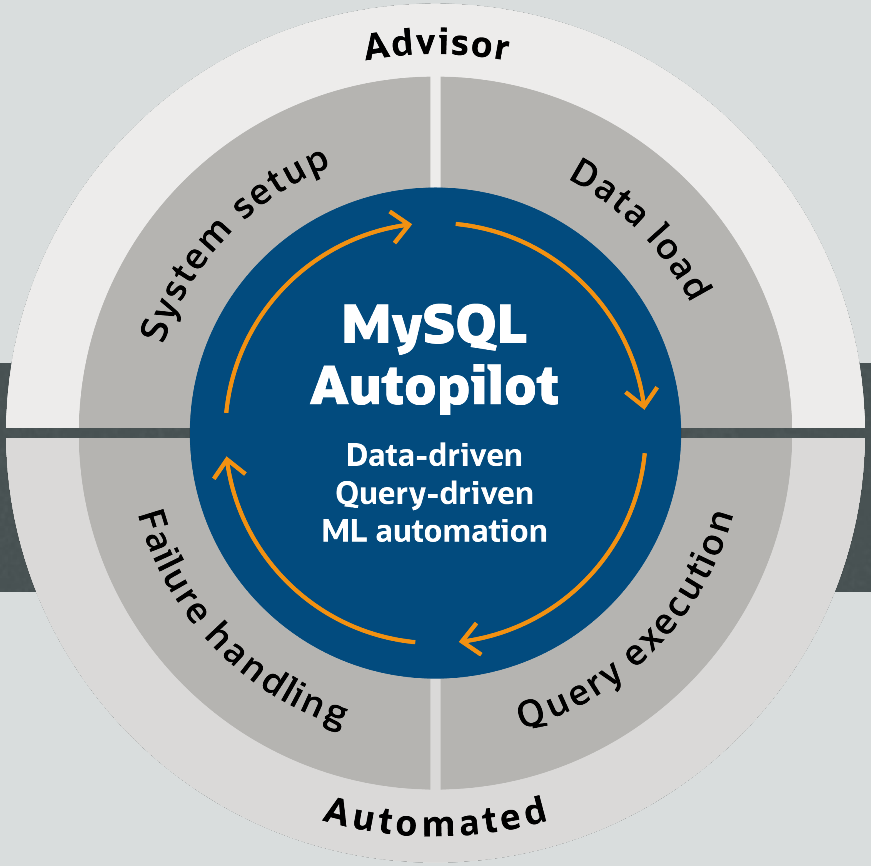
Oh, so now you want that –

AUTOMATED Using Machine Learning

Workload-aware ML-powered automation

INCREASES PRODUCTIVITY AND HELPS ELIMINATE HUMAN ERRORS | CAPABILITIES FOR ANALYTICS AND OLTP

Auto Provisioning
Auto Shape Prediction
Auto Schema Inference
Adaptive Data Sampling



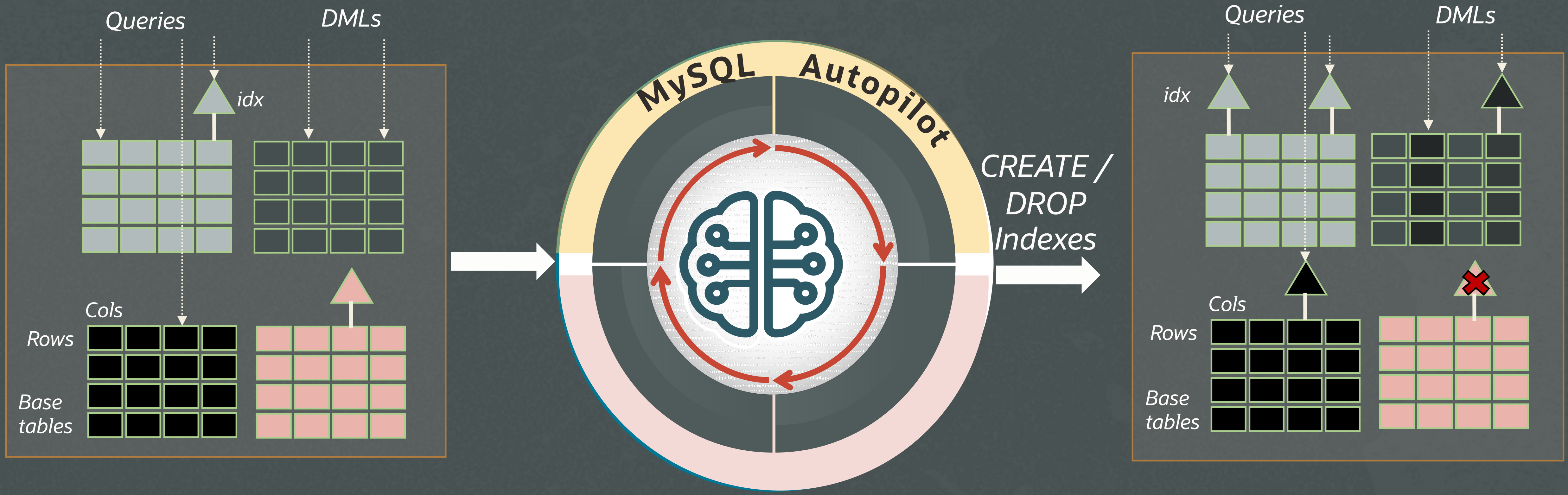
(In LA) **Autopilot indexing**
Auto Parallel Load
Auto Data Placement
Auto Encoding
Auto Unload
Auto Compression
Adaptive Data Flow

Auto Error Recovery

Auto Scheduling
Auto Change Propagation
Auto Query Time Estimation
Auto Query Plan Improvement
Adaptive Query Execution
Auto Thread Pooling

MySQL Autopilot Indexing (Limited Availability)

RECOMMENDS SECONDARY INDEXES FOR OLTP WORKLOADS



- Create & Drop suggestions
- Considers both query and DML perf

Autopilot Indexing

Workload–aware machine learning recommendations for adding and removing table indexes

- Considers both query and DML performance (index maintenance cost)
- Recommends CREATE and DROP of indexes
- Generates DDLs for index creation/drop
- Provides performance prediction (per query and total workload)
- Provides storage prediction
- Provides explanation for the recommendations

Why ML-based automation?

Works for individual workloads

- No guess work
- Interpretable

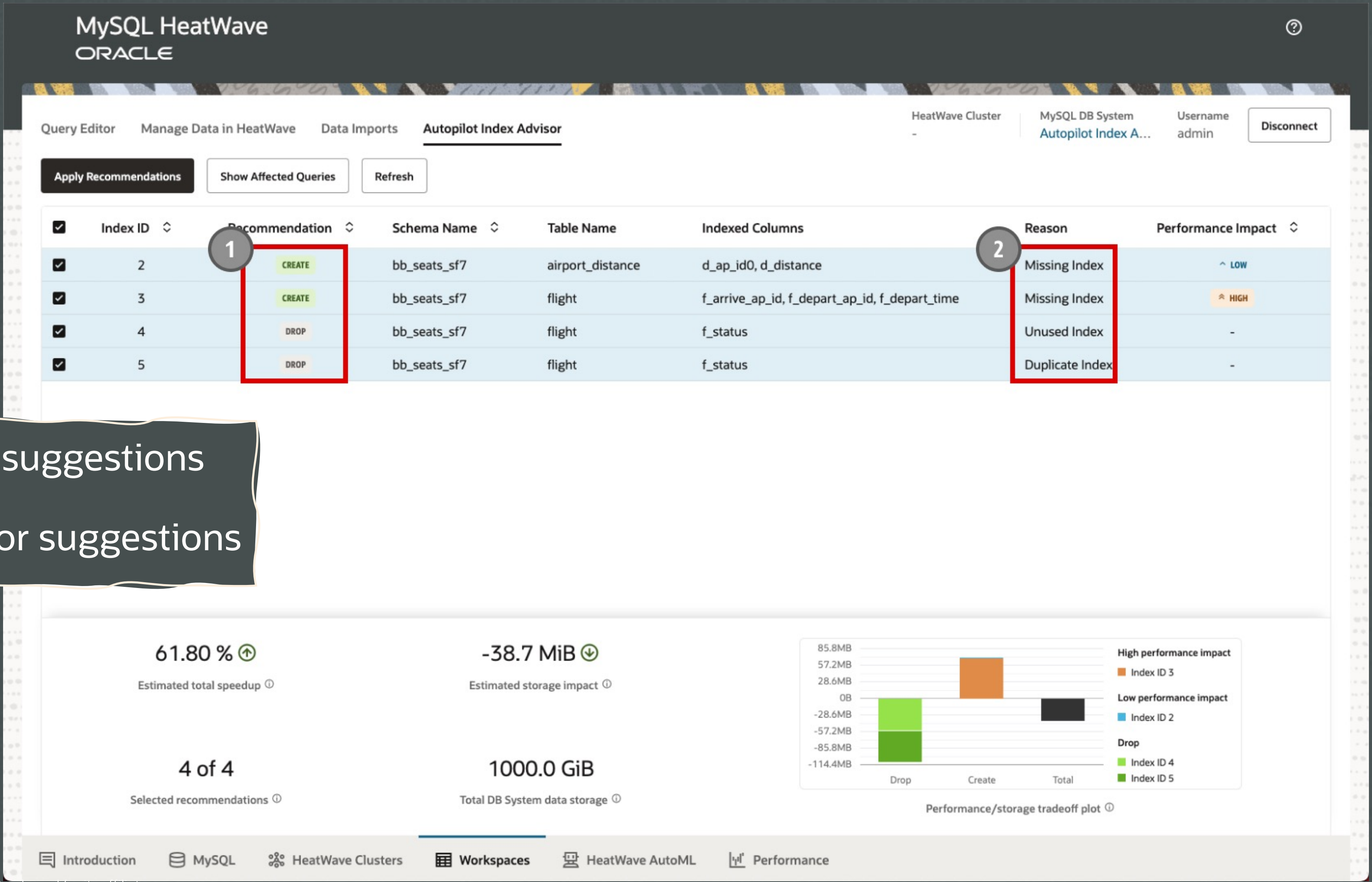
ML models are adaptable

- Ever-changing cloud env
- New server releases

Various optimization targets

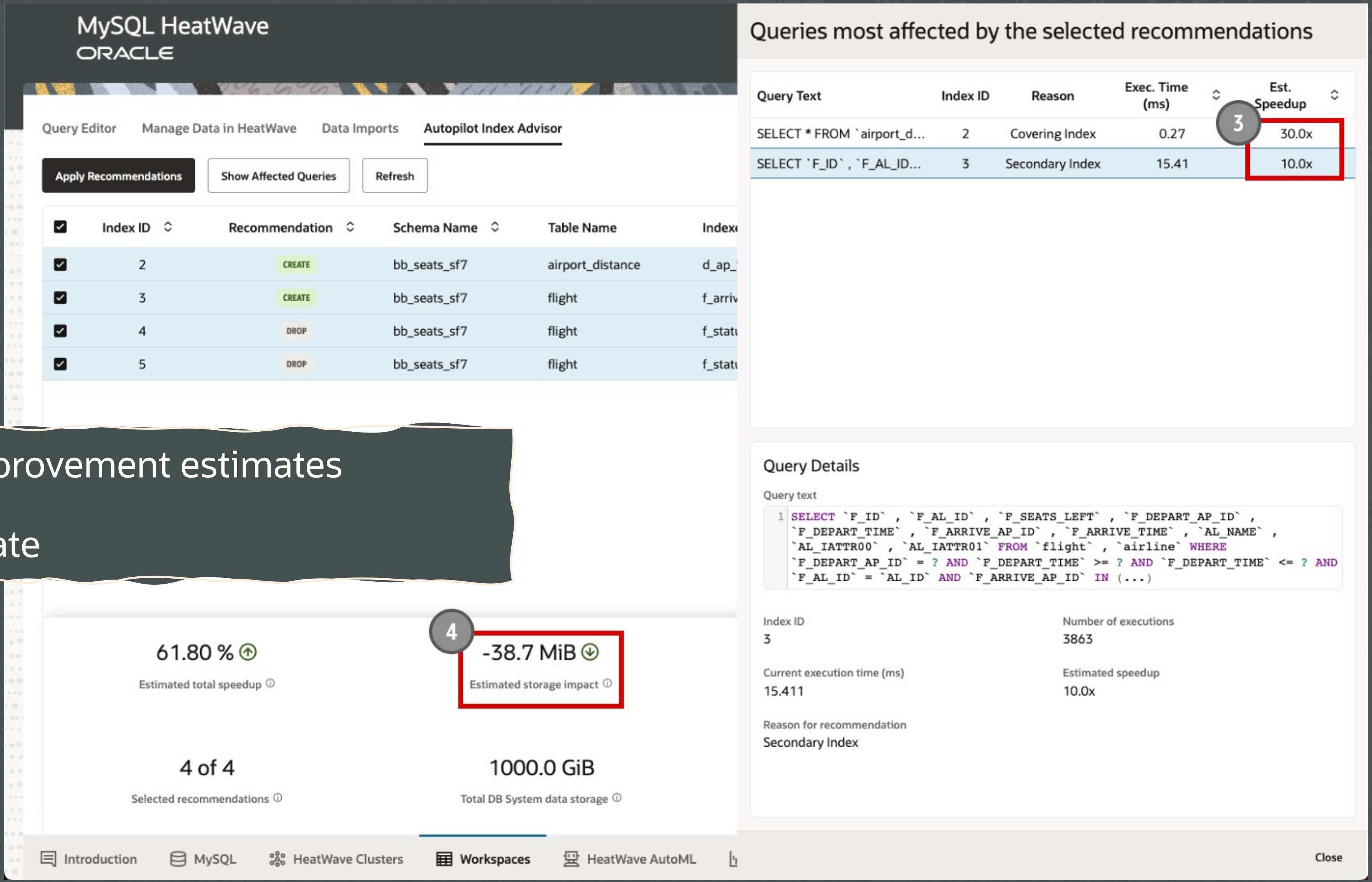
- Throughput
- Latency
- Storage

Autopilot Indexing console



1. Create & Drop suggestions
2. Explanations for suggestions

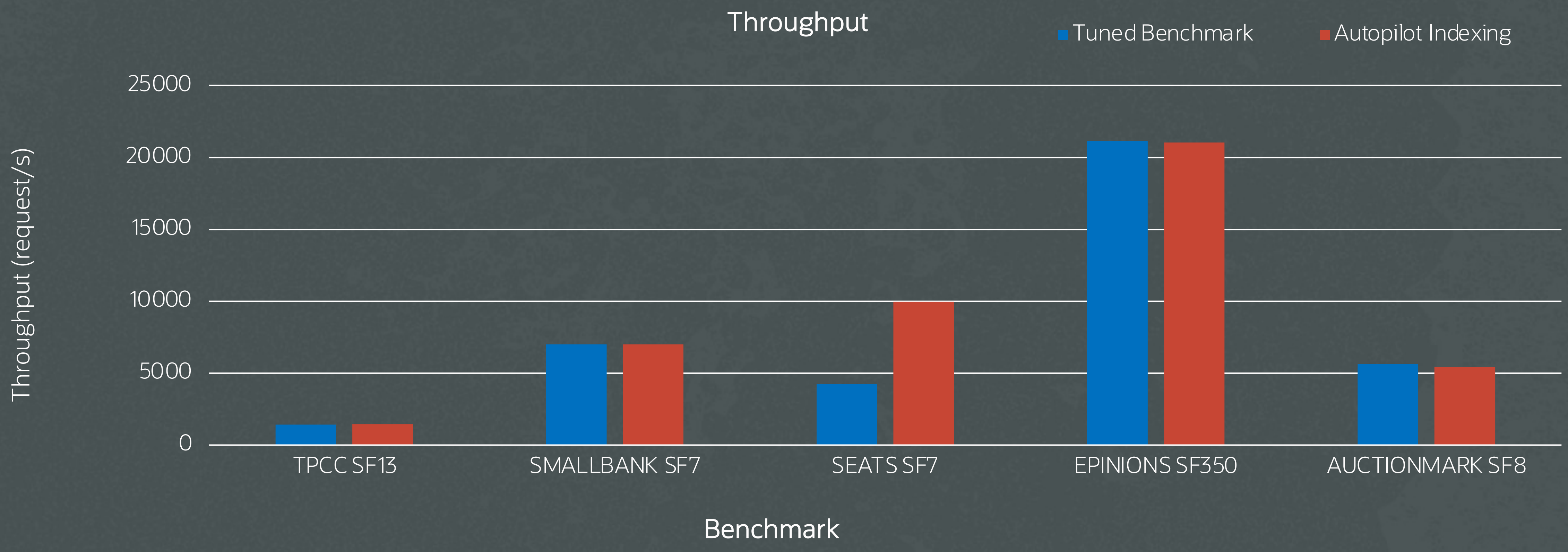
Autopilot Indexing console



3. Query perf improvement estimates
4. Storage estimate

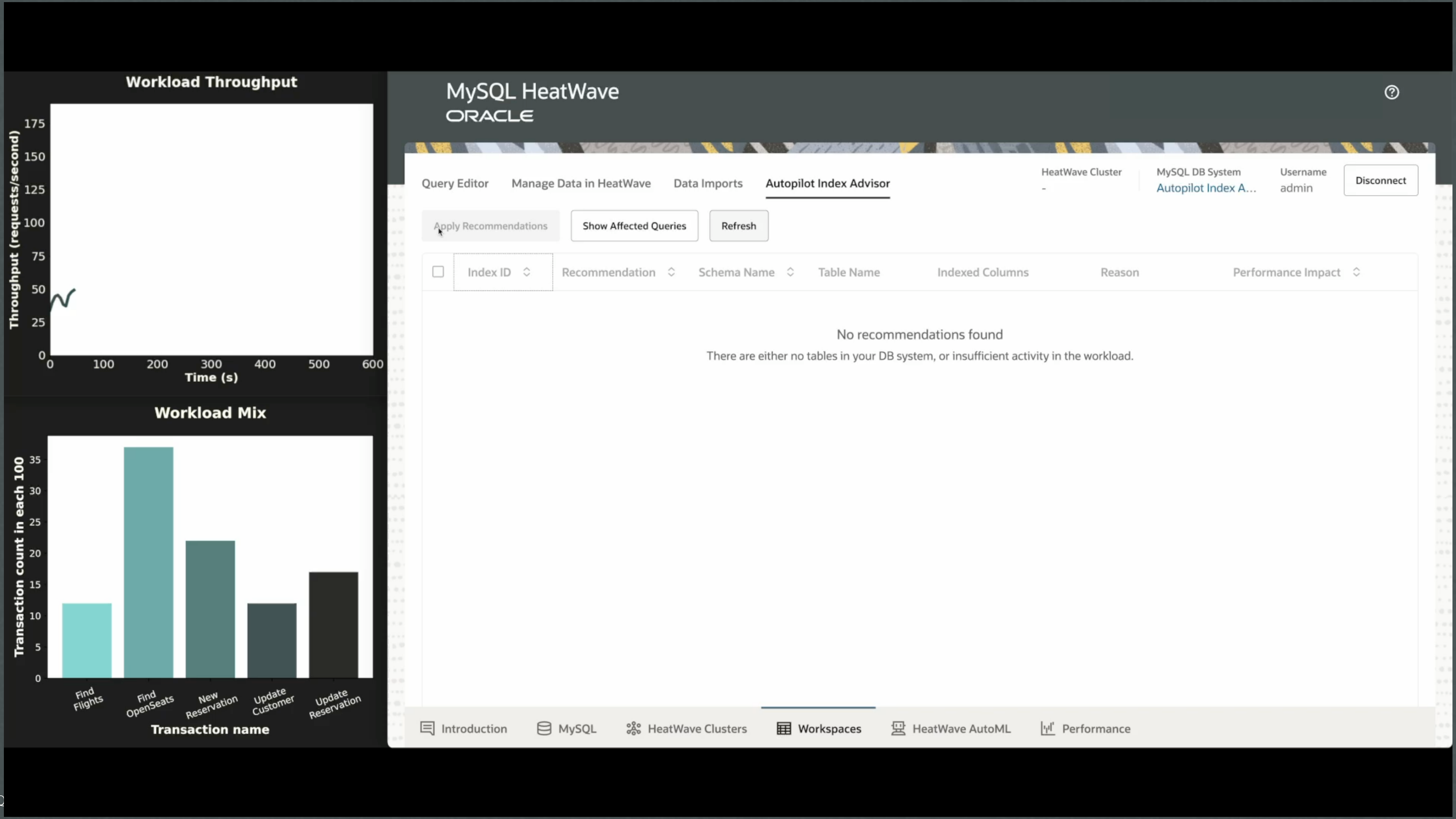
Results

THROUGHPUT AT PAR OR BETTER EVEN ON BENCHMARKS WHICH ARE TUNED



- Autopilot recommends indexes whose performance is at par or better than manually tuned benchmarks
- In some cases, Autopilot recommends fewer indexes which saves storage

MySQL Autopilot Indexing Demo



Thank you!