MySQL Schema Best Practices

Mike Benshoof - Technical Account Manager, Percona

PERCONA
LIVE · USA
SANTA CLARA

Agenda

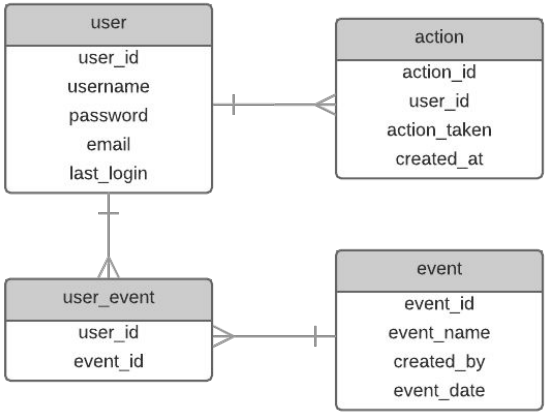
- Introduction
- Key things to consider and review
- Tools/Queries to isolate issues
- Common problems caused by schema design

Introduction

- What comprises a “schema”?
  - Formal definition of how data is organized
  - Collection of tables, indexes, functions

Introduction - Sample Schema

Sample schema to highlight and refine:



PERCONA

Key Considerations

- Data Types
- Indexing

Data Types

- Key Principle
  - Choose the smallest type you can
- Why does it matter?
  - Disk space concerns
  - Memory buffer allocation

Data Types

- Example:
  - ~4 million rows, 4 columns, 3 indexes
  - Identical data, one with bigint PK, one with unsigned int PK
  - **Over 16% overhead!**

```
[root@plive-2017-demo plive_2017]# ls -alh action*.ibd
-rw-r-----. 1 mysql mysql 740M Apr 13 13:20 action_bigint.ibd
-rw-r-----. 1 mysql mysql 636M Apr 13 13:20 action.ibd
```

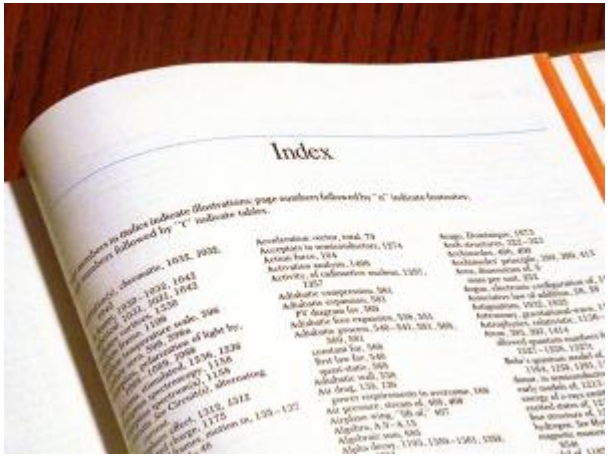
PERCONA

Data Types - Extra Considerations

- Numeric Types
  - signed vs unsigned
- Date Types
  - Timestamp for tracking (i.e. last update)
  - Datetime for historical (i.e. birthday)

Indexing - High Level

Indexes serve to speed up data retrieval by using a secondary reference, just like the index in a normal book...



PERCONA

Indexing

- Key Principle
  - Use fewest indexes needed to ensure optimal query performance
- Why does it matter?
  - Indexes are needed for optimized query execution
  - Each index has an additional cost
    - *Space (both disk and memory footprint)*
    - *Write performance hit (see InnoDB Change Buffer)*

Indexing - Extra Considerations

- PRIMARY KEY is appended to secondary indexes (InnoDB)
- Composite indexes move from left to right
  - Consider this to avoid duplicate indexes
  - Combine columns based on queries
  - Try to use more selective columns on the left
- Use EXPLAIN to verify which indexes are used

Profiling your Schema

- Quick Review: *pt-mysql-summary --databases=plive_2017*

```
# Schema ################################################

  Database   Tables Views SPs Trigs Funcs   FKs Partn
  plive_2017      5

  Database   InnoDB
  plive_2017      5

  Database   BTREE
  plive_2017    17

                i   v   t   b   c
                n   a   i   i   h
                t   r   m   g   a
                    c   e   i   r
                    h   s   n
                    a   t   t
                    r   a
                        m
                        p
  Database   === === === === ===
  plive_2017   6   6   4   1   2
```


PERCONA

Profiling your Schema

- Information_Schema
  - Query directly
  - Tables (estimates for InnoDB!)
  - Indexes
  - Stats_on_metadata (=0)

Finding Largest Schema

- Step 1:
  - Find the largest schemas

```
SELECT table_schema, engine,
count(*) tables,
concat(round(sum(table_rows)/1000000,2),'M') rows,
concat(round(sum(data_length)/(1024*1024*1024),2),'G') data,
concat(round(sum(index_length)/(1024*1024*1024),2),'G') idx,
concat(round(sum(data_length+index_length)/(1024*1024*1024),2),'G') total_size,
round(sum(index_length)/sum(data_length),2) idxfrac
FROM information_schema.TABLES
WHERE table_schema not in ("information_schema", "mysql", "performance_schema", "sys")
GROUP BY table_schema, engine
ORDER BY sum(data_length+index_length) DESC LIMIT 20;
```

Finding Largest Schema

table_schema	engine	tables	rows	data	idx	total_size	idxfrac
plive_2017	InnoDB	5	10.81M	0.94G	0.84G	1.78G	0.89
test	InnoDB	2	0.79M	0.03G	0.00G	0.03G	0.00

PERCONA

Find Largest Tables in Schema

- Step 2:
  - Isolate the largest tables:

```
SELECT engine,
table_name,
concat(round(table_rows/1000000,2),'M') rows,
concat(round(data_length/(1024*1024*1024),2),'G') data,
concat(round(index_length/(1024*1024*1024),2),'G') idx,
concat(round((data_length+index_length)/(1024*1024*1024),2),'G') total_size,
round(index_length/data_length,2) idxfrac
FROM information_schema.TABLES
WHERE table_schema in ("plive_2017")
ORDER BY data_length+index_length DESC LIMIT 10;
```

PERCONA

Find Largest Tables in Schema

```
+--------+---------------+-------+-------+-------+------------+---------+
| engine | table_name    | rows  | data  | idx   | total_size | idxfrac |
+--------+---------------+-------+-------+-------+------------+---------+
| InnoDB | action        | 4.08M | 0.30G | 0.28G | 0.59G      |    0.93 |
| InnoDB | user_event    | 1.91M | 0.22G | 0.08G | 0.30G      |    0.37 |
| InnoDB | event         | 0.98M | 0.11G | 0.12G | 0.23G      |    1.12 |
| InnoDB | user          | 0.01M | 0.00G | 0.00G | 0.00G      |    0.70 |
+--------+---------------+-------+-------+-------+------------+---------+
```

- engine, table_name
  - Just what you would think :)
- rows
  - Estimate of the number of rows
- data, idx, total_size
  - Estimate of the data/index size
- idxfrac
  - **Ratio of index size : data size (ideally < 1)**
  - Change ORDER BY to target these tables:
    - ***ORDER BY idxfrac DESC LIMIT 10;***

PERCONA

What can you learn from this?

- (idxfrac) - which tables likely have:
  - Poor choices for PK data type
  - Excess indexes
- (rows/size) - which tables you should focus on for:
  - Archiving
  - Data retention
  - Partitioning

Finding duplicate keys...

The information_schema is a good way to potentially spot indexing issues at a high level…

Enter ***pt-duplicate-key-checker*** to quickly identify those problem indexes...

PERCONA

Finding duplicate keys...

```
[root@plive-2017-demo plive_2017]# pt-duplicate-key-checker --databases=plive_2017
# ########################################################################
# plive_2017.action
# ########################################################################

# idx_user is a left-prefix of idx_user_created
# Key definitions:
#   KEY `idx_user` (`user_id`),
#   KEY `idx_user_created` (`user_id`,`created_at`)
# Column types:
#         `user_id` int(11) default null
#         `created_at` timestamp not null default current_timestamp on update current_timestamp
# To remove this duplicate index, execute:
ALTER TABLE `plive_2017`.`action` DROP INDEX `idx_user`;

# ########################################################################
# Summary of indexes
# ########################################################################

# Size Duplicate Indexes   10205240
# Total Duplicate Indexes  1
# Total Indexes            13
```

PERCONA

Finding Unused Indexes

- Leverage the Performance Schema
  - Tracked since last restart

```
SELECT object_name, index_name
FROM
performance_schema.table_io_waits_summary_by_index_usage
WHERE index_name IS NOT NULL
AND count_star = 0
AND object_schema = "plive_2017"
ORDER BY object_name;
```

```
+-------------+------------------+
| object_name | index_name       |
+-------------+------------------+
| action      | PRIMARY          |
| action      | idx_user         |
| action      | idx_created      |
| action      | idx_user_created |
| event       | PRIMARY          |
| event       | idx_created_by   |
| event       | idx_date         |
| user        | PRIMARY          |
| user        | idx_user         |
| user        | idx_email        |
| user        | idx_most_recent  |
| user_event  | PRIMARY          |
| user_event  | idx_event        |
+-------------+------------------+
13 rows in set (0.00 sec)
```

PERCONA

Finding Unused Indexes

- Let’s run some queries against our schema...

```
# Fetch all actions for user
SELECT *
FROM action
WHERE user_id = 104;


# Fetch all events a user is attending
SELECT username, event_id, event_name, event_date
FROM user JOIN user_event USING (user_id)
JOIN event USING (event_id)
WHERE user_id = 104;


# Find all users attending an event
SELECT username
FROM user
JOIN user_event USING (user_id)
WHERE event_id = "eb602e39-fe0d-11e5-a8b2-080027a5bc34";
```

PERCONA

Check Performance Schema...

```
SELECT object_name, index_name
FROM performance_schema.table_io_waits_summary_by_index_usage
WHERE index_name IS NOT NULL
AND count_star = 0
AND object_schema = "plive_2017"
ORDER BY object_name;
```

```
+-------------+------------------+
| object_name | index_name       |
+-------------+------------------+
| action      | PRIMARY          |
| action      | idx_created      |
| action      | idx_user_created |
| event       | idx_created_by   |
| event       | idx_date         |
| user        | idx_user         |
| user        | idx_email        |
| user        | idx_most_recent  |
+-------------+------------------+
8 rows in set (0.00 sec)
```

Great… now we can see the indexes that STILL haven't been touched!

PERCONA

Let’s make some revisions...

- Drop duplicate index on **action** table
  - Found via pt-duplicate-key-checker
- Investigate data types / indexes in **event**
  - **Based on the idxfrac > 1**
  - I.e. indexes are larger than actual data

Drop duplicate keys

```
mysql> ALTER TABLE `plive_2017`.`action` DROP INDEX `idx_user`;
```

Before: `-rw-r-----. 1 mysql mysql` **`636M`** `Apr 13 13:20 action.ibd`

After: `-rw-r-----. 1 mysql mysql` **`524M`** `Apr 13 13:53 action.ibd`

That's an **18%** reduction in space!

PERCONA

Changing data types on event tables

```
mysql> show create table event\G
*************************** 1. row ***************************
       Table: event
Create Table: CREATE TABLE `event` (
  `event_id` char(36) NOT NULL DEFAULT '',
  `event_name` varchar(255) DEFAULT NULL,
  `created_by` int(10) unsigned DEFAULT NULL,
  `event_date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`event_id`),
  KEY `idx_created_by` (`created_by`),
  KEY `idx_date` (`event_date`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1

mysql> show create table user_event\G
*************************** 1. row ***************************
       Table: user_event
Create Table: CREATE TABLE `user_event` (
  `user_id` int(10) unsigned NOT NULL DEFAULT '0',
  `event_id` char(36) NOT NULL DEFAULT '',
  PRIMARY KEY (`user_id`,`event_id`),
  KEY `idx_event` (`event_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

PERCONA

Changing data types on event tables

Transitional tables:

```
CREATE TABLE `event_new` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `event_id` char(36) NOT NULL DEFAULT '',
  `event_name` varchar(255) DEFAULT NULL,
  `created_by` int(10) unsigned DEFAULT NULL,
  `event_date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  KEY `idx_created_by` (`created_by`),
  KEY `idx_date` (`event_date`)
) ENGINE=InnoDB

CREATE TABLE `user_event_new` (
  `user_id` int(10) unsigned NOT NULL DEFAULT '0',
  `event_id` int(10) unsigned NOT NULL DEFAULT '0',
  PRIMARY KEY (`user_id`,`event_id`),
  KEY `event_id` (`event_id`)
) ENGINE=InnoDB
```

PERCONA

Changing data types on event tables

Populating Transitional Tables & Cleanup:

```
# Generate an integer PK for each row
mysql> INSERT INTO event_new SELECT NULL, event.* FROM event;
Query OK, 1048576 rows affected (12.25 sec)
Records: 1048576  Duplicates: 0  Warnings: 0

# Replace each char-based event_id with the new integer
mysql> INSERT INTO user_event_new SELECT user_id, event_new.id FROM user_event JOIN event_new USING
(event_id);
Query OK, 2048431 rows affected (22.20 sec)
Records: 2048431  Duplicates: 0  Warnings: 0

# Remove the old character event_id field
mysql> ALTER TABLE event_new DROP COLUMN event_id;
Query OK, 0 rows affected (19.22 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

PERCONA

Changing data types on event tables

```
Review the Impact:

-rw-r-----. 1 mysql mysql 252M Apr 13 13:17 event.ibd
-rw-r-----. 1 mysql mysql 132M Apr 13 14:04 event_new.ibd

-rw-r-----. 1 mysql mysql 332M Apr 13 13:18 user_event.ibd
-rw-r-----. 1 mysql mysql 172M Apr 13 14:02 user_event_new.ibd
```

That is a 48% reduction in space!

Schema breakdown - revisited...

engine	table_name	rows	data	idx	total_size	idxfrac
InnoDB	action	4.17M	0.35G	0.14G	0.50G	0.41
InnoDB	user_event	1.91M	0.22G	0.08G	0.30G	0.37
InnoDB	**event**	**0.98M**	**0.11G**	**0.12G**	**0.23G**	**1.12**
InnoDB	user_event_new	1.91M	0.12G	0.03G	0.15G	0.21
InnoDB	**event_new**	**1.04M**	**0.09G**	**0.03G**	**0.12G**	**0.38**
InnoDB	user	0.01M	0.00G	0.00G	0.00G	0.70

PERCONA

Common Issues

- Wasted resources
  - Memory (buffer pool, per-session buffers, etc)
  - Disk
- Sub-optimal performance
  - Primarily write operations
- Additional overhead for optimizer
  - More execution paths to analyze
  - Higher likelihood for wrong path to be chosen

The times, they are a changin'...

- Schema review is an iterative process, not a one time event!
- Index utilization isn't remotely static
  - Periodically refresh P_S, especially after code and/or index changes
    - *TRUNCATE TABLE performance_schema.table_io_waits_summary_by_index_usage*
- Track size over time for trending
  - Help to answer the common question: "When do we need more disk space?"
  - Determine the frequency of archiving/pruning

PERCONA

MySQL Schema Review 101

Questions?

PERCONA

MySQL Schema Review 101

Thanks!

PERCONA