



PERCONA

TECH DAYS

MySQL Schema Review 101

Mike Benshoof, Technical Account Manager

Agenda

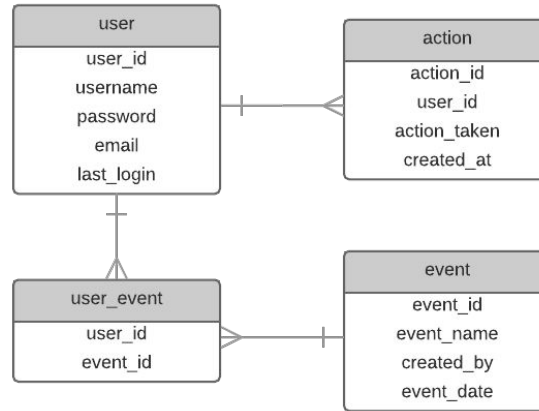
- .Introduction
- .Key things to consider and review
- .Tools to isolate issues
- .Common problems caused by schema design

Introduction

- What comprises a “schema”
 - Formal definition of how data is organized
 - Collection of tables, indexes, functions
- Schema is “owned” by the overall database instance
- Permissions can be granted at schema or table level

Introduction - Sample Schema

Sample schema to highlight and refine:



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 40% overhead!**

```
[root@sample-host tech_days]# ls -alh action*.ibd
-rw-rw---- 1 mysql mysql 908M Sep  7 16:22 action_bigint.ibd
-rw-rw---- 1 mysql mysql 636M Sep  7 16:23 action.ibd
```

Data Types - Extra Considerations

- Numeric Types

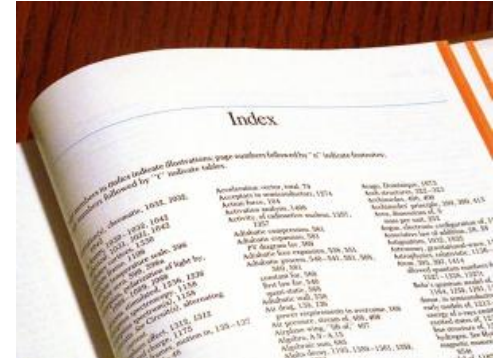
- signed vs unsigned

- 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...



Indexing

•Key Principle

- Use fewest indexes needed to ensure optimal query performance

•Why does it matter?

- Indexes are needed for optimal query execution
- Each index has an additional cost
 - *Space (both disk and memory footprint)*
 - *Write performance hit (to maintain)*

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=tech_days*

```
# Schema #####  
  
Database  Tables Views SPs Trigs Funcs  FKs Partn  
tech_days      6  
  
Database  InnoDB  
tech_days      6  
  
Database  BTREE  
tech_days      17  
  
          i   v   t   c  
          n   a   i   h  
          t   r   m   a  
          c   e   r  
          h   s  
          a   t  
          r   a  
          m  
          p  
Database  ===  ===  ===  ===  
tech_days  9   6   4   2
```



Profiling your Schema

•Information_Schema

- Wealth of information
- Query directly
- Tables
- 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 10;
```

Finding Largest Schema

table_schema	engine	tables	rows	data	idx	total_size	idxfrac
tech_days	InnoDB	4	8.03M	0.60G	0.62G	1.22G	1.04
test	InnoDB	8	0.00M	0.00G	0.00G	0.00G	0.13
percona	InnoDB	2	0.00M	0.00G	0.00G	0.00G	0.50

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 ("tech_days")  
GROUP BY table_name, table_schema, engine  
ORDER BY data_length+index_length DESC LIMIT 10;
```

Find Largest Tables in Schema

engine	table_name	rows	data	idx	total_size	idxfrac
InnoDB	action	3.85M	0.29G	0.26G	0.55G	0.91
InnoDB	user_event	1.93M	0.19G	0.15G	0.34G	0.76
InnoDB	event	1.02M	0.11G	0.12G	0.23G	1.12
InnoDB	user	0.01M	0.00G	0.00G	0.00G	0.80

Breaking it down...

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

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...

Finding duplicate keys...

```
[root@cur-master tech_days]# pt-duplicate-key-checker
# #####
# tech_days.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 `tech_days`.`action` DROP INDEX `idx_user`;

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

# Size Duplicate Indexes    9627515
# Total Duplicate Indexes   1
# Total Indexes             53
```

Finding Unused Indexes

- userstat module (Percona Server)
 - SET @@global.userstat = 1
 - SELECT * FROM information_schema.INDEX_STATISTICS;
 - Shows rows read from each index
- performance_schema
 - SELECT * FROM sys.schema_unused_indexes
- How does this help?

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";
```

Now look at INDEX_STATISTICS

```
mysql> select * from information_schema.INDEX_STATISTICS;
```

TABLE_SCHEMA	TABLE_NAME	INDEX_NAME	ROWS_READ
tech_days	user_event	idx_event	3
tech_days	action	idx_user	385
tech_days	event	PRIMARY	182
tech_days	user	PRIMARY	4
tech_days	user_event	PRIMARY	182

Great... but I wanted to know which indexes aren't being used!

Combine with base index statistics... and voila!

```
SELECT `Table`, `Index`
FROM (
SELECT table_name AS `Table`, index_name AS `Index`
FROM information_schema.statistics
WHERE table_schema = 'tech_days'
GROUP BY 1,2) as all_indexes
LEFT JOIN INFORMATION_SCHEMA.index_statistics ON all_indexes.Table = index_statistics.TABLE_NAME
      AND all_indexes.Index=index_statistics.INDEX_NAME
WHERE ROWS_READ IS NULL
      AND `Index` != "PRIMARY";
```

```
+-----+-----+
| Table      | Index      |
+-----+-----+
| action     | idx_created |
| action     | idx_user_created |
| event      | idx_created_by |
| event      | idx_date   |
| user       | idx_email  |
| user       | idx_most_recent |
| user       | idx_user   |
+-----+-----+
```

...or just use performance_schema (and view)

```
SELECT * FROM sys.schema_unused_indexes where object_schema = "tech_days";
```

object_schema	object_name	index_name
tech_days	action	idx_created
tech_days	action	idx_user_created
tech_days	event	idx_created_by
tech_days	event	idx_date
tech_days	user	idx_email
tech_days	user	idx_most_recent
tech_days	user	idx_user

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 `tech_days`.`action` DROP INDEX `idx_user`;
```

Before: -rw-rw---- 1 mysql mysql **636M** Sep 6 16:23 action.ibd

After: -rw-rw---- 1 mysql mysql **452M** Sep 8 04:03 action.ibd

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
```

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
```

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, 1 warning (52.67 sec)
```

```
# 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 (5 min 2.00 sec)
```

```
# Remove the old character event_id field
```

```
mysql> ALTER TABLE event_new DROP COLUMN event_id;
```

```
Query OK, 0 rows affected (1 min 59.31 sec)
```

Changing data types on event tables

Review the Impact:

```
-rw-rw---- 1 mysql mysql 252M Sep  6 23:46 event.ibd
-rw-rw---- 1 mysql mysql 112M Sep  8 02:56 event_new.ibd

-rw-rw---- 1 mysql mysql 360M Sep  6 01:02 user_event.ibd
-rw-rw---- 1 mysql mysql 172M Sep  8 02:41 user_event_new.ibd
```

That is over 50% reduction in space!

Schema breakdown - revisited...

engine	table_name	rows	data	idx	total_size	idxfrac
InnoDB	action	4.17M	0.31G	0.13G	0.44G	0.41
InnoDB	user_event	1.93M	0.19G	0.15G	0.34G	0.76
InnoDB	event	1.02M	0.11G	0.12G	0.23G	1.12
InnoDB	user_event_new	2.14M	0.11G	0.02G	0.14G	0.22
InnoDB	event_new	1.04M	0.08G	0.03G	0.11G	0.38
InnoDB	user	0.01M	0.00G	0.00G	0.00G	0.80

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 run `FLUSH INDEX_STATISTICS` (with PS)
 - Especially after code and/or index changes
- 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

MySQL Schema Review 101

Questions?



PERCONA

LIVE ONLINE

20-21 OCTOBER

2020