MySQL and InnoDB
for the rest of us

Less MySQL. More OurSQL

Shaun O’Keefe
Head of Platform Engineering @ Stile Education
A world-class science education for every student

Stile helps teachers bring their science classes to life with beautiful lessons based on real-world science and global issues.
@yomilk
What’s the plan?
MySQL
Innodeebee?
Performance
Why is this query slow?

SELECT username FROM users
WHERE join_date > '2019-02-26'

CREATE TABLE users (  
id INT NOT NULL AUTO_INCREMENT,  
first_name VARCHAR[100]  
last_name VARCHAR[100]  
age DATE NOT NULL,  
join_date DATE NOT NULL  
PRIMARY KEY (id)  
INDEX(age, join_date)  
);
And this one?

```
SELECT count(*)
FROM blockchain_enthusiasts
where
gender='Male'

CREATE TABLE people (  
  id INT NOT NULL AUTO_INCREMENT,
  first_name VARCHAR[100]
  last_name VARCHAR[100]
  gender ENUM(<all the genders here>)
 PRIMARY KEY (id)
 INDEX(gender)
);
```
Where did all my disk space go?

INSERT INTO users (Uuid, first_name, Last_name)
VALUES ('62722494-8a07-4aee-9515-05-a608dc413d', 'Badger', 'McTavish')

CREATE TABLE users (uuid VARCHAR[36] NOT NULL, first_name VARCHAR[100], last_name VARCHAR[100] PRIMARY KEY (uuid));
MySQL
InnoDB
Storage engine
Chapter 16 Alternative Storage Engines

Table of Contents

16.1 Setting the Storage Engine
16.2 The MyISAM Storage Engine
16.3 The MEMORY Storage Engine
16.4 The CSV Storage Engine
16.5 The ARCHIVE Storage Engine
16.6 The BLACKHOLE Storage Engine
16.7 The MERGE Storage Engine
16.8 The FEDERATED Storage Engine
16.9 The EXAMPLE Storage Engine
16.10 Other Storage Engines
16.11 Overview of MySQL Storage Engine Architecture
Chapter 16 Alternative Storage Engines

Table of Contents

16.1 Setting the Storage Engine
16.2 The MyISAM Storage Engine
16.3 The MEMORY Storage Engine
16.4 The CSV Storage Engine
16.5 The ARCHIVE Storage Engine
16.6 The BLACKHOLE Storage Engine
16.7 The NDB Storage Engine
16.8 The BDB Storage Engine
16.9 The TREE Storage Engine
16.10 The File Storage Engine
16.11 The Memory Storage Engine
16.12 The Engine API for Custom Storage Engines
16.13 Examples of Custom Storage Engines
16.14 How to Write a Custom Storage Engine
Performance
1. Query time
2. Query time
3. Query time
1. Query time,
2. Query time,
3. Query time
...and memory and disk
Memory

Disk

Query time

?
query time

memory
disk
What takes time in queries?

- Planning how to run a query
- Executing a query
- Waiting to get a lock
- Finding rows
- Moving data between memory and disk
- Sorting and grouping results
- Sending data back to the client
• Are you fetching too much data?
• Are you looking at too much data
• Are you fetching too much data?
• Are you looking at too much data
Indexes
This is a secondary index

CREATE TABLE my_table (  
  [...],  
  column1 VARCHAR[100],  
  Column2 DATE,  
  INDEX (column1, column2)  
);
This is a primary index

CREATE TABLE my_table (  
id INT NOT NULL AUTO_INCREMENT,  
[...],  
PRIMARY KEY (id)  
);
Find data quickly, in order
Find data quickly, in order

- Quickly find rows specified in where clauses
- Quickly fetch adjacent values for ranges
- Quickly find value for joins
- For group, distinct, data is in order
What do they REALLY look like?
B+Tree Structure

Copyright (c) 2013, Jeremy Cole <jeremy@jcole.us>
Copyright (c) 2013, Davi Arnaut <darnaut@gmail.com>
What’s a secondary index key + value look like?

INDEX(
    eye_colour,
    lastname,
    firstname,
    age,
    date_joined
)
So what’s the go with the PKs?
Clustered index
CREATE TABLE okeefes (  
id INT NOT NULL AUTO_INCREMENT  
firstname VARCHAR(128),  
lastname VARCHAR(128),  
eye_colour ENUM,  
favourite_colour VARCHAR(128),  
join_date DATE,  
PRIMARY KEY (id)  
INDEX(  
eye_colour,  
lastname,  
firstname,  
age,  
date Joined   )  )
Table scan

Index my data

Find my data using the index

My data is an index
What a PK wants
Smaller, simpler data
Matching data across join columns
Most used lookups
Auto inc is the best case
Random insertion order is the worst
Convincing MySQL to use your index
MySQL literally only wants one thing and it’s disgusting
- full table scan
- Full index scan
- Index range scan / lookup
- Unique index lookups
- Constant
So... what do I do?
One index ideally services many different query types
One index ideally covers as much of your query as possible
We can only use the leftmost part of the index
These will use the index

INDEX (eye_colour, last_name, first_name, age, join_date)

SELECT * FROM my_table where eye_colour='blue';

SELECT * FROM my_table where eye_colour='blue' AND last_name='McTavish';

SELECT * FROM where eye_colour IN ['blue', 'green', 'brown'] AND last_name='McTavish' AND first_name='Badger' AND age > 25;

SELECT * FROM where eye_colour IN ['blue', 'green', 'brown'] AND last_name LIKE 'Mc%'
These will partially use the index

INDEX (eye_colour, last_name, first_name, age, join_date)

SELECT * FROM my_table where eye_colour='blue' AND last_name LIKE '%Tavish';

SELECT * FROM where eyeColour in ['blue', 'green', 'brown'] AND last_name='McTavish' AND first_name='Badger' AND age > 25 AND join_date < '2016-10-12';
It’s not not using your index
Also, sometimes it’s not using your index
Covering indexes
Asking questions about how my query went
EXPLAIN
SELECT
  film.film_id,
  film.title,
  film.release_year,
  actor.actor_id,
  actor.first_name,
  actor.last_name
FROM sakila.film
INNER JOIN sakila.film_actor USING(film_id)
INNER JOIN sakila.actor USING(actor_id);
<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>actor</td>
<td>ALL</td>
<td>PRIMARY</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>film_actor</td>
<td>ref</td>
<td>PRIMARY,idx_fk_film_id</td>
<td>PRIMARY</td>
<td>2</td>
<td>sakila.actor.actor_id</td>
<td>1</td>
<td>Using index</td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>film</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>2</td>
<td>sakila.film_actor.film_id</td>
<td>1</td>
<td>Extra:</td>
</tr>
</tbody>
</table>
Query parser
Query optimiser
Join optimiser
Join execution (nested loops)
table a

<table>
<thead>
<tr>
<th>column 1</th>
<th>column 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
</tbody>
</table>

table b

<table>
<thead>
<tr>
<th>column 1</th>
<th>column 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>14</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
</tr>
</tbody>
</table>

Row 1:
- Table A: column 1 = 3
- Table B: column 1 = 3, column 2 = 7

Row 2:
- Table A: column 1 = 7
- Table B: column 1 = 3, column 2 = 11

Row 3:
- Table A: column 1 = 12
- Table B: column 1 = 7, column 2 = 1

Row 4:
- Table A: column 1 = 30
- Table B: column 1 = 30, column 2 = 14

Row 5:
- Table A: column 1 = 36
- Table B: column 1 = 30, column 2 = 1

Rows to client
After nested loops
SELECT
  film.film_id,
  film.title,
  film.release_year,
  actor.actor_id,
  actor.first_name,
  actor.last_name
FROM sakila.film
INNER JOIN sakila.film_actor USING(film_id)
INNER JOIN sakila.actor USING(actor_id);
<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>table</th>
<th>type</th>
<th>possible_keys</th>
<th>key</th>
<th>key_len</th>
<th>ref</th>
<th>rows</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>actor</td>
<td>ALL</td>
<td>PRIMARY</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>film_actor</td>
<td>ref</td>
<td>PRIMARY,idx_fk_film_id</td>
<td>PRIMARY</td>
<td>2</td>
<td>sakila.actor.actor_id</td>
<td>1</td>
<td>Using index</td>
</tr>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>film</td>
<td>eq_ref</td>
<td>PRIMARY</td>
<td>PRIMARY</td>
<td>2</td>
<td>sakila.film_actor.film_id</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
id: 1
select_type: SIMPLE
table: actor
type: ALL
possible_keys: PRIMARY
  key: NULL
  key_len: NULL
  ref: NULL
  rows: 200
Extra:
id: 1
select_type: SIMPLE
table: film_actor
type: ref
possible_keys: PRIMARY, idx_fk_film_id
    key: PRIMARY
    key_len: 2
    ref: sakila.actor.actor_id
rows: 1
Extra: Using index
id: 1
select_type: SIMPLE
table: film
type: eq_ref
possible_keys: PRIMARY
key: PRIMARY
key_len: 2
ref: sakila.film_actor.film_id
rows: 1
Extra:
EXPLAIN EXTENDED SHOW WARNINGS
Thanks!

Shaun O’Keefe - Stile Education

@yomilk
Sources

- https://www.percona.com/blog
- https://github.com/jeremycole/ innodb_diagrams
- https://blog.jcole.us/innodb