MySQL and Innodb for the rest of us

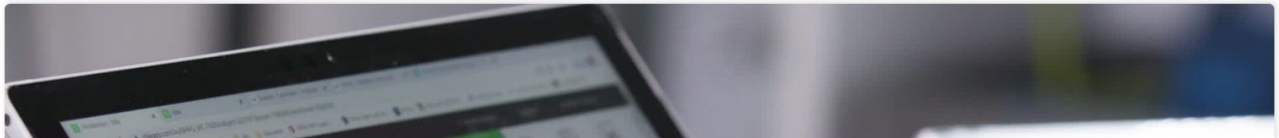
Less MySQL. More OurSQL

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.

Set up a trial



@yomilk

@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 (
  uuid='62722494-8a07-4aee-9515-05-a608dc413d,
  first_name='Badger',
  last_name='McTavish'
)
```

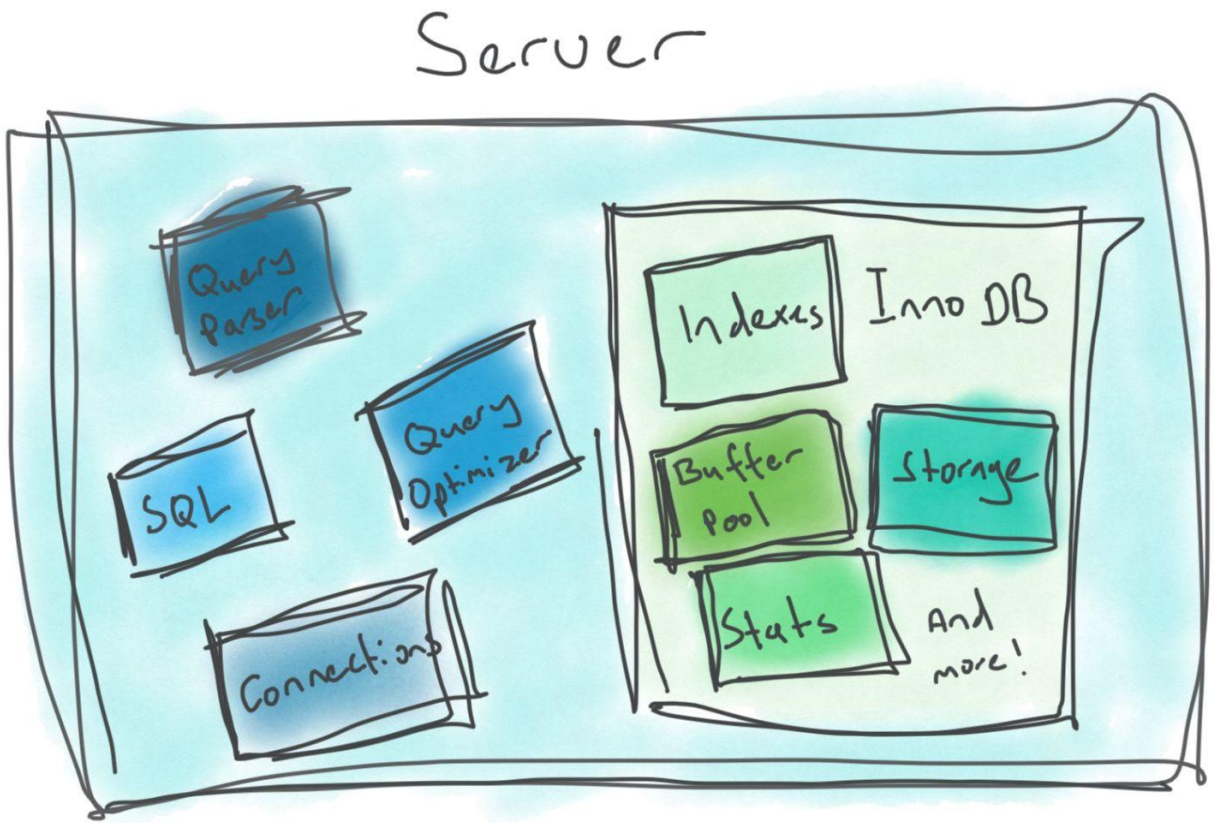
```
CREATE TABLE users (
  uuid VARCHAR[36] NOT NULL,
  first_name VARCHAR[100]
  last_name VARCHAR[100]
  PRIMARY KEY (uuid)
);
```

MySQL

Innodeebee?

InnoDB

Storage engine

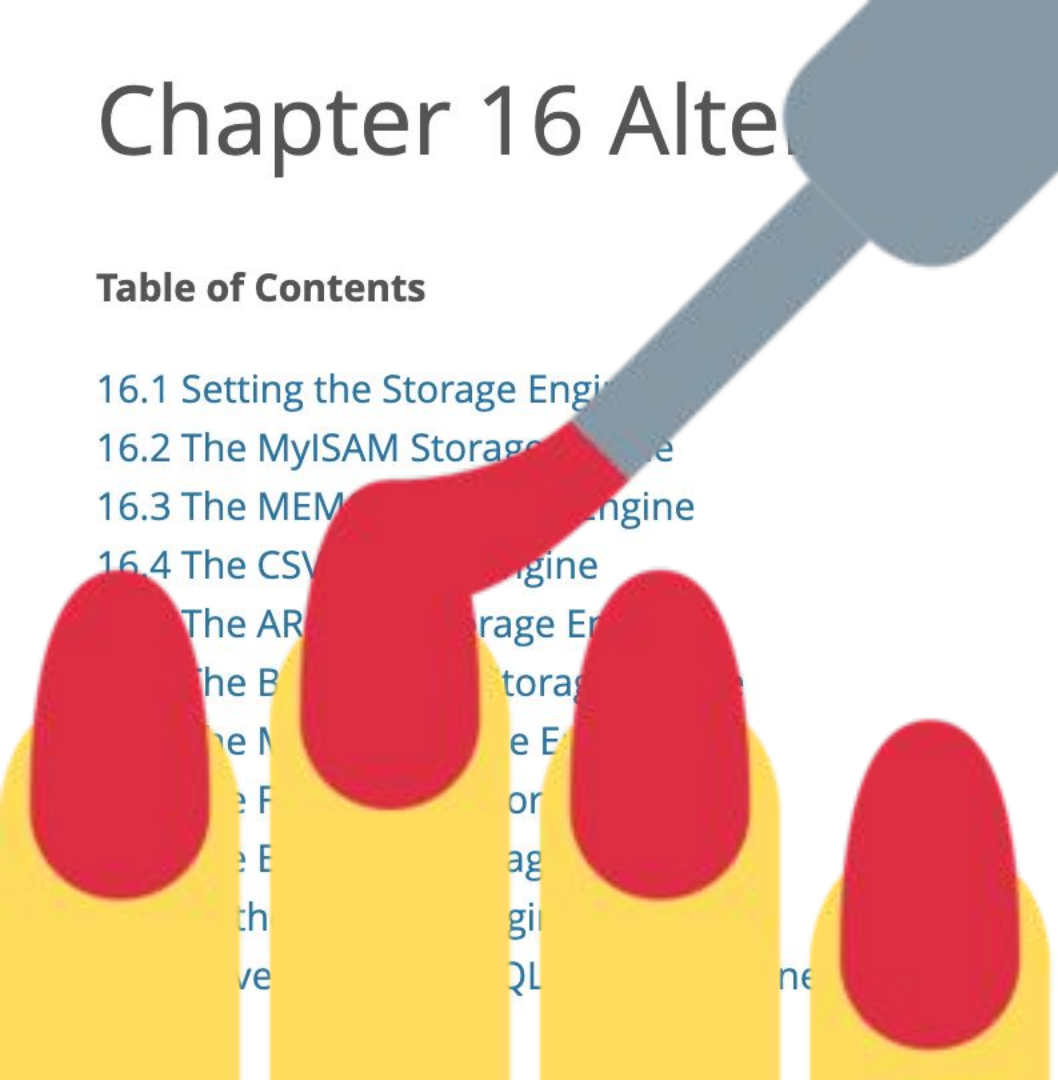

Server
Query Parser
Query Optimizer
SQL
Connections
Indexes
InnoDB
Buffer Pool
Storage
Stats
And more!

Chapter 16 Alternative Storage Engines

Table of Contents

Chapter 16 Alte Storage Engines

Table of Contents



Performance

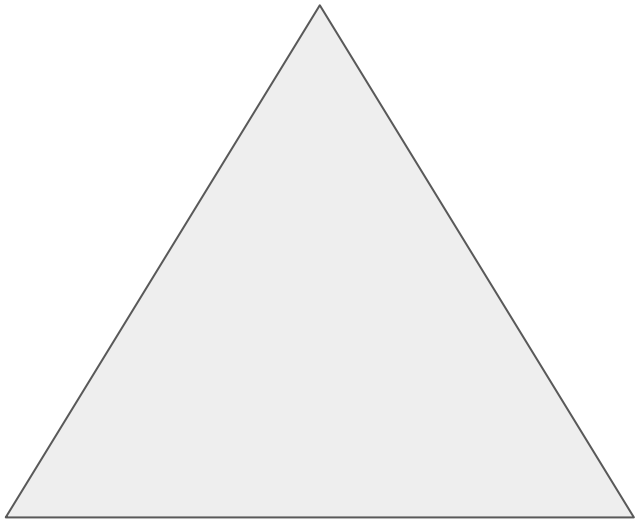
1. Query time
2. Query time
3. Query time

1. Query time,
2. Query time,
3. Query time

...and memory and disk

query time

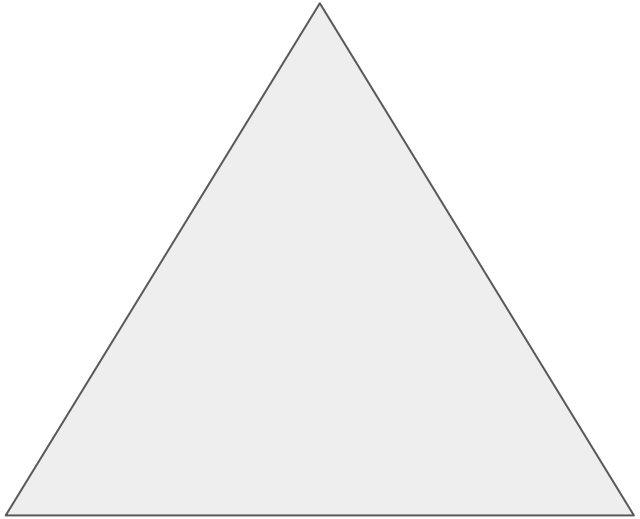
?



memory

disk

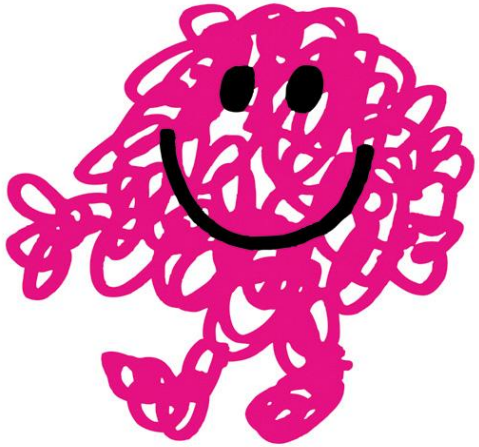
query time



memory

disk

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)

);
```

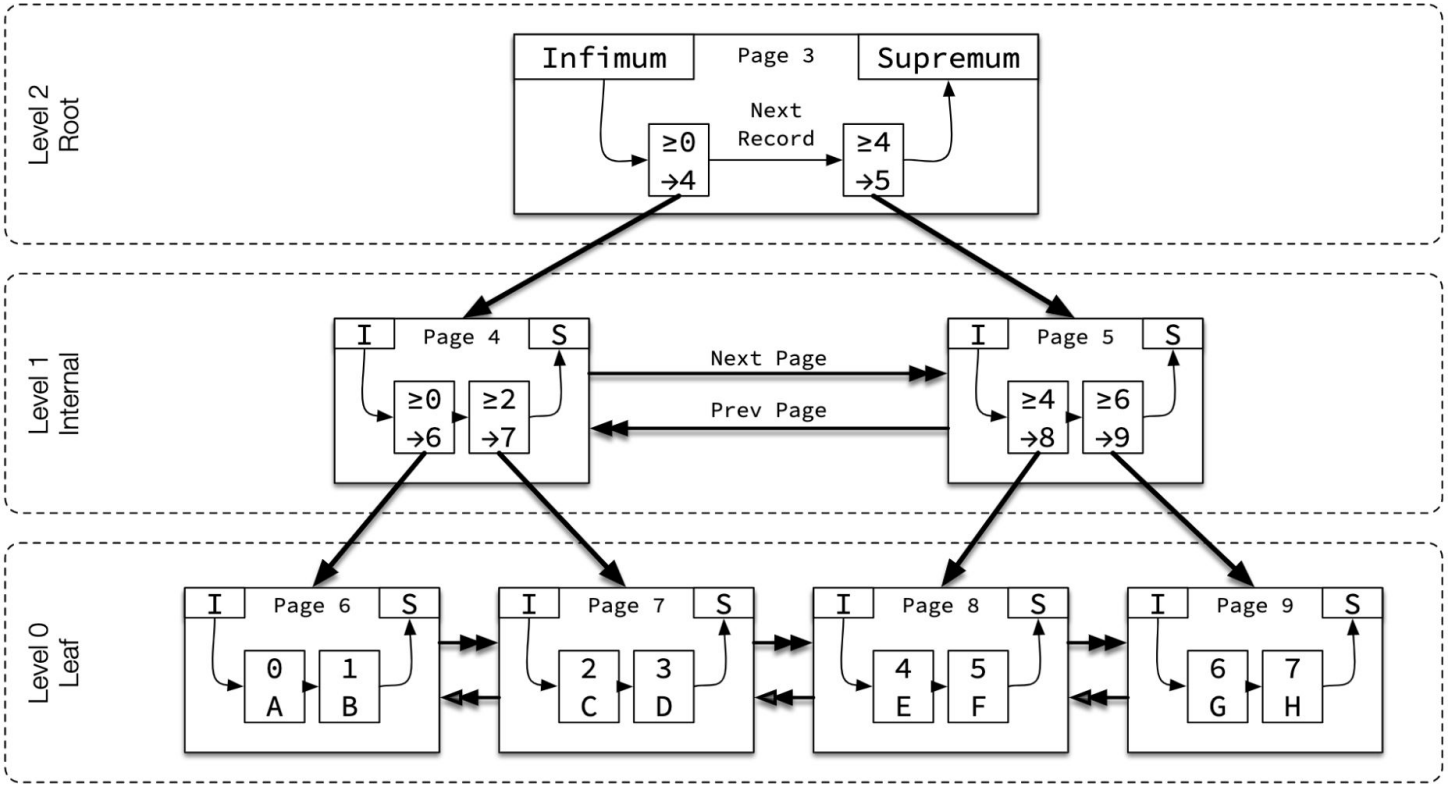
Find data quickly, in order

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



What's a secondary index key + value look like?

```
INDEX(
    eye_colour,
    lastname,
    firstname,
    age,
    date_joined
)
```

blue-okeefe-shaun-36-2012-12-12	brown-okeefe-mum-61-2015-01-02	blue-okeefe-dad-62-2010-11-01
25	32	10

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
    )
)
```

blue-okeefe-shaun-36-2012-12-12	blue-okeefe-dad-62-2010-11-01	brown-okeefe-mum-61-2015-01-02
25	10	32

32	25	10
okeefe	okeefe	okeefe
mum	shaun	dad
g	b	b
green	blue	red
61	36	62
2015-01-02	2012-12-12	2010-11-01

Table scan

Index my data

Find my data using the index

My data is an index

What a PK wants

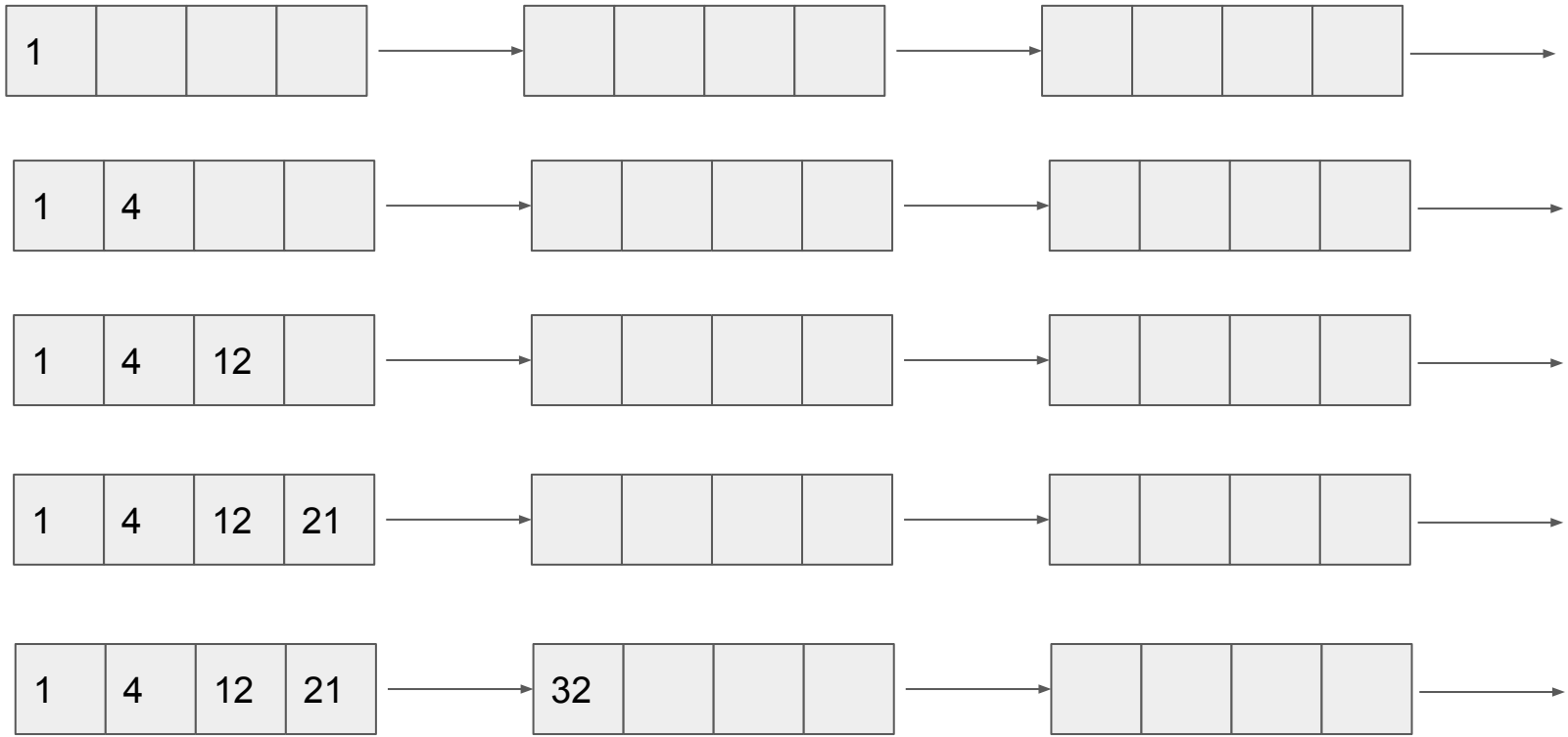
Smaller, simpler data

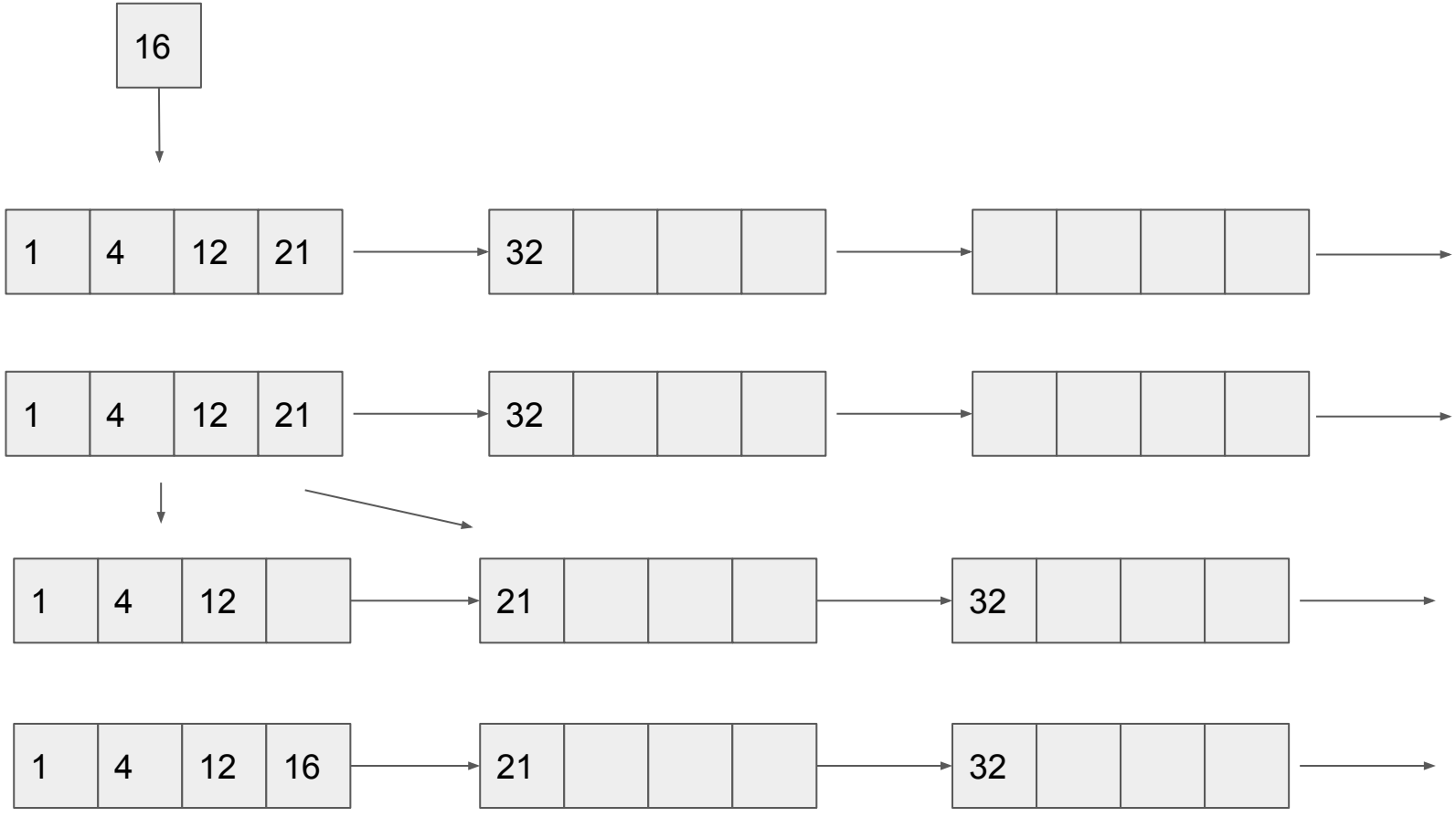
Matching data across join columns

Most used lookups

Auto inc is the best case

Random insertion order is the worst


1
1 4
1 4 12
1 4 12 21
1 4 12 21 32


16
1
4
12
21
32
1
4
12
21
32
1
4
12
21
32
1
4
12
16
21
32

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 eye_colour 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

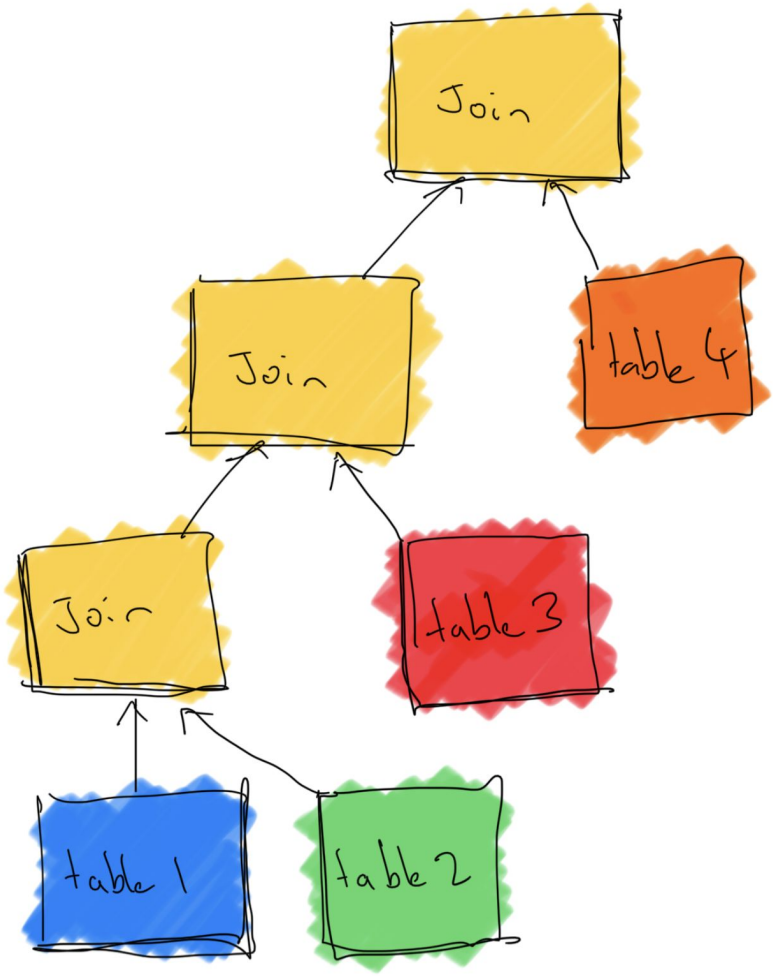
```
SELECT
  film.film_id,
  film.title,
  film.release_year,
  actor.actor_id,
  actor.first_name,
  actor.last_name
FROM sakila.film
INNER JOIN sakila.film_actor USING(film_id)
INNER JOIN sakila.actor USING(actor_id);
```

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: actor
         type: ALL
possible_keys: PRIMARY
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 200
        Extra:
*************************** 2. row ***************************
           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
*************************** 3. row ***************************
           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:
```

Query parser

Query optimiser

Join optimiser


Join
Join
table 4
Join
table 3
table 1
table 2

Join execution (nested loops)

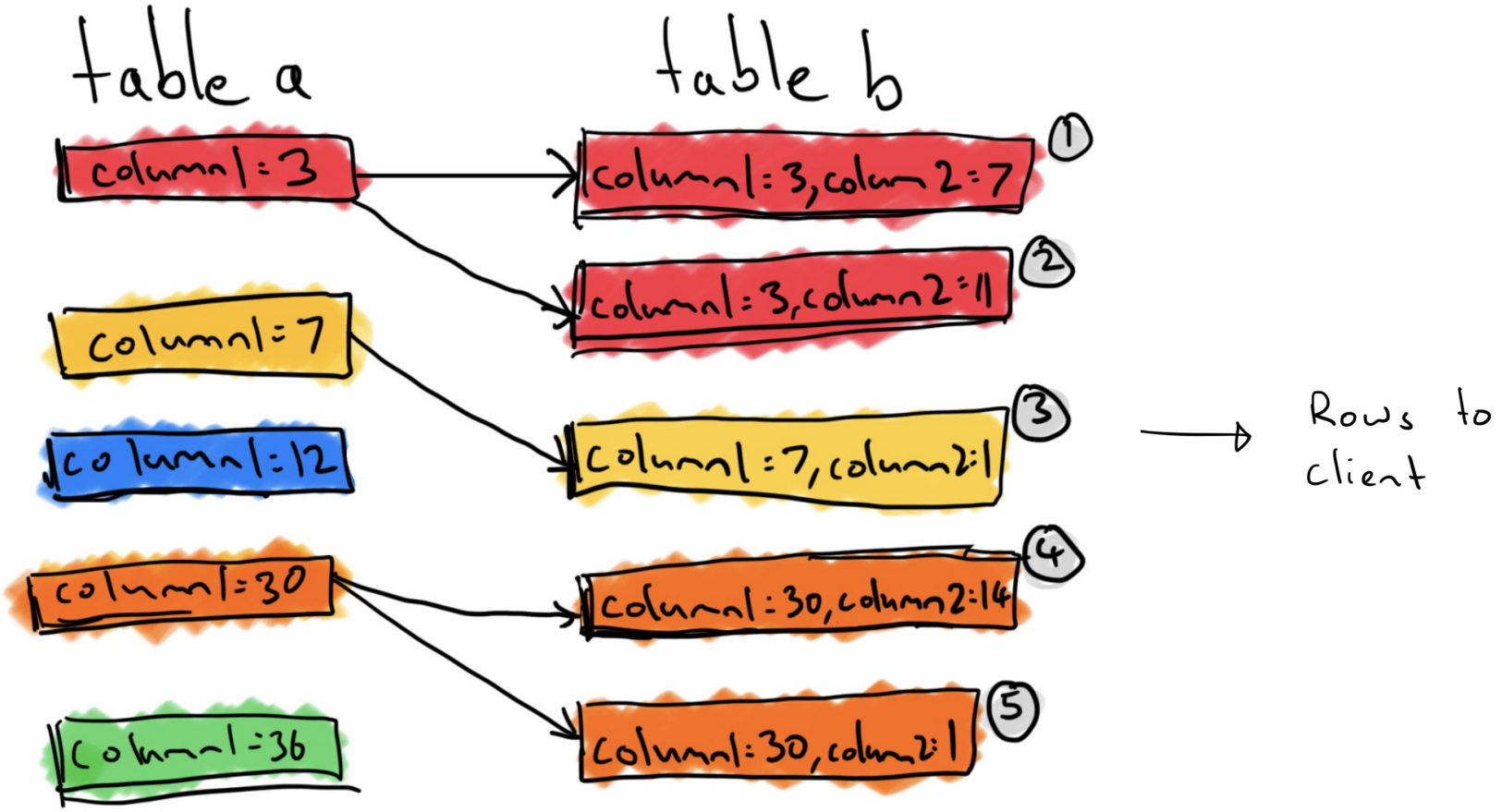


table a
table b
column1=3
column1=7
column1=12
column1=30
column1=36
column1=3, colum 2=7
column1=3, column2=11
column1=7, column2=1
column1=30, column2=14
column1=30, colum2=1
1
2
3
4
5
Rows to client

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);
```

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: actor
         type: ALL
possible_keys: PRIMARY
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 200
        Extra:
*************************** 2. row ***************************
           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
*************************** 3. row ***************************
           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:
```

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: actor
         type: ALL
possible_keys: PRIMARY
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 200
        Extra:
```

```
*************************** 2. row ***************************
           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
```

```
*************************** 3. row ***************************
           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://learning.oreilly.com/library/view/high-performance-mysql
- https://dev.mysql.com/doc/refman/8.0/en
- https://dev.mysql.com/doc/internals/en/
- https://github.com/jeremycole/innodb_diagrams
- https://blog.jcole.us/innodb