Designing Scalable Database ORM

Maghead Database Framework

Yo-An Lin
@c9s

Outline

- Maghead Overview
    - Why? And the history
    - The look of Maghead
    - Performance
- Sharding
    - When do you Shard?
    - What's Sharding?
    - How to Shard?
    - Consistent Hashing
    - Chunks and Chunk Migration
    - Shard Balancing
    - Commands and the APIs

Why **ORM**?

Actually when we mention **ORM**

Usually it refers to **tools**
including **ORM** and many
other things

DBAL | ORM | SQL

Builder | Migration | Database Management

Some people say ...

DON'T USE **ORM**

Because of

Performance

Use Hand-Written

SQL Queries…

Actually if you ....

Actually if you

- need to finish 2+ site projects per month
- have different modules shared between different projects with some customization
- need to maintain legacy projects without db query tests
- have frequently schema changes from different team members
- need to remember more than the changes from more than 10, 20, 30 tables

Trust me, You Won't like it

Writing basic SQL in every projects is really painful

ORM helps you …

- Have the domain logics well-organized.
- Perform lightweight automatic migration when you have schema changes.
- Generate the common, basic SQL queries for the most use cases without the need of manually updating the SQL queries.
- Manage the database connections with a centralized simple config file.
- !! Actually you can still do hand-written query if you want

And it saves your **Life**

What's

Maghead

Maghead
Database
Framework

Object
Relation
Mapper

Data**B**ase **A**bstraction **L**ayer

SQL Builder

Connection Manager

Database Manager

Table Definition

Parser

Automatic Migration

Migration Manager

And many components

The components are designed to be used separately.

The project was started since 2010

7 years

The project was created to have the **dynamic schema** in different projects

How does it looks like?

YAML configuration

```
---
cli:
  bootstrap: vendor/autoload.php
schema:
  auto_id: true
  finders:
  - { name: ComposerSchemaFinder, args: ["composer.json"] }
```

```
---
...
instance:
  local:
    dsn: 'mysql:host=localhost'
    user: root
    driver: mysql
    host: localhost
    password: null
    query_options: {  }
    connection_options:
      1002: 'SET NAMES utf8'
...
```

```
---
...
databases:
  master:
    dsn: 'mysql:host=localhost;dbname=testing'
    host: localhost
    user: root
    driver: mysql
    database: testing
    password: null
    query_options: {  }
    connection_options:
      1002: 'SET NAMES utf8'
```

Simple bootstrapping

Bootstrap Code

```
require 'vendor/autoload.php';

use Maghead\Runtime\Config\FileConfigLoader;
use Maghead\Runtime\Bootstrap;

$config = FileConfigLoader::load( __DIR__ . '/db/config/
database.yml');
Bootstrap::setup($config);
```

Bootstrap Code

```php
require 'vendor/autoload.php';
use Maghead\Runtime\Config\FileConfigLoader;
use Maghead\Runtime\Bootstrap;

$config = FileConfigLoader::load('db/config/database.yml');
Bootstrap::setup($config);
```

2 lines only!

ActiveRecord pattern

Most of **ActiveRecord**
patterns are implemented
in this way

```php
$record = new User;
$record->account = "c9s";
$record->password = sha1("mypassword");
$record->save();
```

Different States:
1. existing record
2. non-existing record (before inserting the row)

```php
function updatePassword($user) {
    // check the existence because the object
    // might be not able to update the
    // password
    if (!$user->hasKey()) {
        throw new LogicException;
    }
    // update ...
}
```

```php
function updatePassword($user) {
    // check the existence because the object
    // might be not able
    // password
    if (!$user->hasKey()) {
        throw new LogicException;
    }
    // update ...
}
```

The user record might be not created yet.

```php
function updatePassword($user) {
    // check the existence because the object
    // might be not able to update the
    // password
    if (!$user->hasKey()) {
        throw new LogicException;
    }
    // update ...
}
```

Check the key to verify the existence

The **Maghead** Way

```php
$record = User::load([ "account" => "timcook" ]);
```

If the returned $record is not false, then it must be an existing row.

```php
$user = User::createAndLoad([ "account" => "timcook" ]);
```

If the returned $record is not false, then it must be an existing row.

```php
$user->update([ "password" => sha1("newpw") ]);
```

Safely Update the row without the concern

Repository pattern

```php
$repo = new ProductRepo($write, $read);
```

```php
$repo = Product::repo('master', 'slave');
```

```php
$repo = Product::masterRepo();
```

Dynamic **Schema**

```php
namespace Todos\Model;

use Maghead\Schema\DeclareSchema;

class TodoSchema extends DeclareSchema
{
    public function schema()
    {

    }
}
```

Column Definition Goes Here

```php
namespace Todos\Model;

use Maghead\Schema\DeclareSchema;

class TodoSchema extends DeclareSchema
{
    public function schema()
    {
        $this->column('title')
            ->varchar(128)
            ->required()
            ;
    }
}
```

```
$this->column('done')
    ->boolean()
    ->default(false);
```

```
$this->column('description')
    ->text();
```

```
$this->column('created_at')
    ->timestamp()
    ->default(function() {
        return date('c');
});
```

Dynamic Default Value

```
$this->column('extra')
     ->text()
     ->isa('json');
```

Automatically Define JSON
Inflator and Deflator

```
$this->column('address')
    ->varchar(64)
    ->validator(function ($val, $args) {
        if (strlen($val) < 6) {
            return [false , “Invalid address”];
        }
        return [true , "Good address"];
})
```

Validation Integration

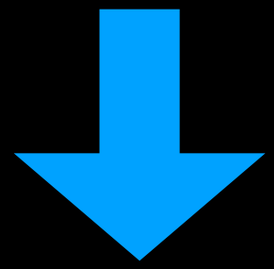
```
$this->column('id', 'ai-pk');
    // AutoIncrementPrimaryKeyColumn
```

```
$this->column('id', 'uuid-pk');
// UUIDPrimaryKeyColumn
```

And can be compiled into the static schema class

- Maghead\Schema**DeclareSchema**
- Maghead\Schema**DeclareColumn**

declarative implementation



Compile

- Maghead\Schema**RuntimeSchema**
- Maghead\Schema**RuntimeColumn**

lightweight implementation

maghead schema build

```php
namespace Todos\Model;

class TodoSchemaProxy
    extends RuntimeSchema
{

    const SCHEMA_CLASS = 'Todos\\Model\\TodoSchema';

    const LABEL = 'Todo';

    const MODEL_NAME = 'Todo';

    const MODEL_NAMESPACE = 'Todos\\Model';

    const MODEL_CLASS = 'Todos\\Model\\Todo';

    const REPO_CLASS = 'Todos\\Model\\TodoRepoBase';

    const COLLECTION_CLASS = 'Todos\\Model\\TodoCollection';

….
```

And This Reduces The Runtime Overhead

Pre-Generated SQL Queries

```
class ProductRepoBase extends Repo {

const FIND_BY_PRIMARY_KEY_SQL = 'SELECT * FROM products WHERE id = ? LIMIT 1';

const DELETE_BY_PRIMARY_KEY_SQL = 'DELETE FROM products WHERE id = ?';

const FETCH_CREATED_BY_SQL = 'SELECT * FROM users WHERE id = ? LIMIT 1';

const FETCH_UPDATED_BY_SQL = 'SELECT * FROM users WHERE id = ? LIMIT 1';

const FETCH_PRODUCT_FEATURES_SQL = 'SELECT * FROM product_feature_junction WHERE product_id =
?';

const FETCH_PRODUCT_PRODUCTS_SQL = 'SELECT * FROM product_products WHERE product_id = ?';

const FETCH_IMAGES_SQL = 'SELECT * FROM product_images WHERE product_id = ?’;

….
```

Pre-Generated Query Methods

```php
class ProductRepoBase
    extends Repo
{
    public function fetchCategoryOf(Model $record)
    {
        if (!$this->fetchCategoryStm) {
            $this->fetchCategoryStm =
                $this->read->prepare(self::FETCH_CATEGORY_SQL);
            $this->fetchCategoryStm->setFetchMode(
                PDO::FETCH_CLASS,
                \ProductBundle\Model\Category::class, [$this]);
        }
        $this->fetchCategoryStm->execute([$record->category_id]);
        $obj = $this->fetchCategoryStm->fetch();
        $this->fetchCategoryStm->closeCursor();
        return $obj;
    }
```

```php
class ProductBase
    extends Model
{
    public function getProductTags()
    {
        $collection = new \ProductBundle\Model\ProductTagCollection;
        $collection->where()->equal("product_id", $this->id);
        $collection->setPresetVars([ "product_id" => $this->id ]);
        return $collection;
    }
```

```php
class ProductBase
    extends Model
{
    public function getTags()
    {
        $collection = new \ProductBundle\Model\TagCollection;
        $collection->joinTable('product_tag_junction', 'j', 'INNER')
            ->on("j.tag_id = {$collection->getAlias()}.id");
        $collection->where()->equal('j.product_id', $this->id);
        $parent = $this;
        $collection->setAfterCreate(function($record, $args) use ($parent) {
            $a = [
                'tag_id' => $record->get("id"),
                'product_id' => $parent->id,
            ];
            if (isset($args['product_tags'])) {
                $a = array_merge($args['product_tags'], $a);
            }
            return \ProductBundle\Model\ProductTag::createAndLoad($a);
        });
        return $collection;
    }
```

Statements are prepared
and cached in the
Repository of each **Model**

Same as

your hand-written queries

Building **Table Schema**

```
maghead sql --rebuild node1
```

One Schema to Rule Them All

```php
$create = $product->asCreateAction();
```

```
$create = $product->asCreateAction();
```

```
{{ RecordAction.renderSignatureWidget|raw }}
{{ RecordAction.renderCSRFTokenWidget|raw }}

{% if Record.hasKey %}
    {{RecordAction.renderKeyWidget|raw}}
{% endif %}

{{RecordAction.renderField('account') |raw }}
{{RecordAction.renderField('password') |raw }}
```

Render Web Form Fields

Relationships

```
$this->belongsTo('book', BookSchema::class, 'id', 'book_id')
    ->onDelete('CASCADE')
    ->onUpdate('CASCADE')
    ;
```

```
$this->many('author_books',
    AuthorBookSchema::class, 'author_id', 'id');

$this->manyToMany('books', 'author_books', 'book');
```

```
$this->many('author_books',
    AuthorBookSchema::class, 'author_id', 'id');
$this->manyToMany('books', 'author_books', 'book');


    foreach ($author->books as $book) {
       ...
    }
```

Database Management Commands

```
maghead db create master
```

```
maghead db recreate master
```

```
maghead db create node1
```

```
maghead db drop node1
```

Repository pattern for **Multiple** Database Connections

The **Doctrine** Way

```php
use Doctrine\DBAL\DriverManager;
$conn = DriverManager::getConnection(array(
        'wrapperClass' =>
'Doctrine\DBAL\Connections\MasterSlaveConnection',
        'driver' => 'pdo_mysql',
        'keepSlave' => true,
        'master' => array(
            'user'     => 'ideato',
            'password' => 'ideato',
            'dbname'   => 'db_ideato'
        ),
        'slaves' => array(
            array(
                'user' => 'ideato',
                'password' => 'ideato',
                'dbname' => 'db_ideato_slave'
            )
        )
    )
);
$entityManager = EntityManager::create($conn, $config);
```

```php
// https://gist.github.com/ricfrank/d6f6317a1a1434cdc364
$entityManager = EntityManager::create($conn, $config);
$productRepository = $entityManager->getRepository('Product');
```

```php
$masterSlaveConn->connect('slave');
$productRepository = $entityManager->getRepository('Product');

$masterSlaveConn->connect('master');
$productRepository = $entityManager->getRepository('Product');
```

```php
$entityManager = EntityManager::create($conn, $config);
$productRepository = $entityManager->getRepository('Product');

$masterSlaveConn->connect('slave');

$product = new Product();
$product->setName("nuovo_prod1");
$product->setCategory("oeeooeoe");
$entityManager->persist($product);
$entityManager->flush();

$productRepository = $entityManager->getRepository('Product');

$masterSlaveConn->connect('master');
$productRepository = $entityManager->getRepository('Product');
```

The **Maghead** Way

```php
$ret = Product::repo("master")
          ->create([ "name" => "Samsung S3" ]);
```

```php
$ret = Product::masterRepo()
          ->create([ "name" => "Samsung S3" ]);
```

```php
$ret = Product::repo("node2")
    ->create([ "name" => "iPhone 7" ]);
```

```php
$ret = Product::repo(new Connection("mysql:host=db1"))
    ->create([ "name" => "Samsung S3" ]);
```

Automatic Migration

Known as lightweight migration in iOS app development

maghead diff

```
Performing comparison...
+ table 'metric_values'      examples/metric/Model/MetricValueSchema.php
  A val                 double
  A published_at        timestamp
  A unit                varchar
+ table 'Edm'                      tests/apps/TestApp/Model/EdmSchema.php
  D id                  int
  A edmNo               int
  A edmTitle            varchar
  A edmStart            date
  A edmEnd              date
  A edmContent          text
  A edmUpdatedOn        timestamp
+ table 'i_d_numbers'       tests/apps/TestApp/Model/IDNumberSchema.php
  A id_number           varchar
+ table 'posts'                    tests/apps/TestApp/Model/PostSchema.php
  A title               varchar
  A content             text
  A status              varchar
  A created_at          timestamp
  A created_by          int
+ table 'tables'                   tests/apps/TestApp/Model/TableSchema.php
  M id                  int
  A title               varchar
  A columns             text
  A rows                text
```

```
Performing comparison...
+ table 'metric_values'        examples/metric/Model/MetricValueSchema.php
  A val                  double
  A published_at         timestamp
  A unit                 varchar
+ table 'Edm'                  [illegible]stApp/Model/EdmSchema.php
  D id                   int
  A edmNo                int
  A edmTitle             varchar
  A edmStart             date
  A edmEnd               date
  A edmContent           text
  A edmUpdatedOn         timestamp
+ table 'i_d_numbers'          tests/apps/TestApp/Model/IDNumberSchema.php
  A id_number            varchar
+ table 'posts'                tests/apps/TestApp/Model/PostSchema.php
  A title                varchar
  A content              text
  A status               varchar
  A created_at           timestamp
  A created_by           int
+ table 'tables'               tests/apps/TestApp/Model/TableSchema.php
  M id                   int
  A title                varchar
  A columns              text
  A rows                 text
```

D: 欄位刪除

```
Performing comparison...
+ table 'metric_values'        examples/metric/Model/MetricValueSchema.php
  A val                  double
  A published_at         timestamp
  A unit                 varchar
+ table 'Edm'                  tests/apps/TestApp/Model/EdmSchema.php
  D id                   int
  A edmNo                int
  A edmTitle             varchar
  A edmStart             date
  A edmEnd               date
  A edmContent           text
  A edmUpdatedOn         timestamp
+ table 'i_d_numbers'          tests/apps/TestApp/Model/IDNumberSchema.php
  A id_number            varchar
+ table 'posts'                tests/apps/TestApp/Model/PostSchema.php
  A title                varchar
  A content              text
  A status               varchar
  A created_at           timestamp
  A created_by           int
+ table 'tables'               tests/apps/TestApp/Model/TableSchema.php
  M id                   int
  A title                varchar
  A columns              text
  A rows                 text
```

A: 欄位新增

```
Performing comparison...
+ table 'metric_values'      examples/metric/Model/MetricValueSchema.php
  A val                 double
  A published_at        timestamp
  A unit                varchar
+ table 'Edm'                      tests/apps/TestApp/Model/EdmSchema.php
  D id                  int
  A edmNo               int
  A edmTitle            varchar
  A edmStart            date
  A edmEnd              date
  A edmContent          text
  A edmUpdatedOn        timestamp
+ table 'i_d_numbers'          tests/apps/TestApp/Model/IDNumberSchema.php
  A id_number           varchar
+ table 'posts'                     tests/apps/TestApp/Model/PostSchema.php
  A title               varchar
  A content             text
  A status              varchar
  A created_at          timestamp
  A created_by          int
+ table 'tables'                    tests/apps/TestApp/Model/TableSchema.php
  M id                  int
  A title               varchar
  A columns             text
  A rows                text
```

M: 欄位修改

maghead m auto

```
Performing automatic upgrade over data source: master
Begining transaction...
Performing Query: ALTER TABLE `addresses` DROP FOREIGN KEY `addresses_ibfk_1`
Performing Query: ALTER TABLE `books` DROP FOREIGN KEY `books_ibfk_1`
Performing Query: ALTER TABLE `books` DROP FOREIGN KEY `books_ibfk_2`
Performing Query: ALTER TABLE `author_books` DROP FOREIGN KEY `author_books_ibfk_1`
Performing Query: ALTER TABLE `author_books` DROP FOREIGN KEY `author_books_ibfk_2`
Performing Query: ALTER TABLE `book_tags` DROP FOREIGN KEY `book_tags_ibfk_1`
Importing schema: MetricApp\Model\MetricValueSchema
Performing Query:
CREATE TABLE  IF NOT EXISTS `metric_values` (
  `id` int UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,
  `val` double(5,3) NOT NULL DEFAULT 0,
  `published_at` timestamp NOT NULL,
  `unit` varchar(3)
) ENGINE=InnoDB;
```

全自動產生 Alter Table Query

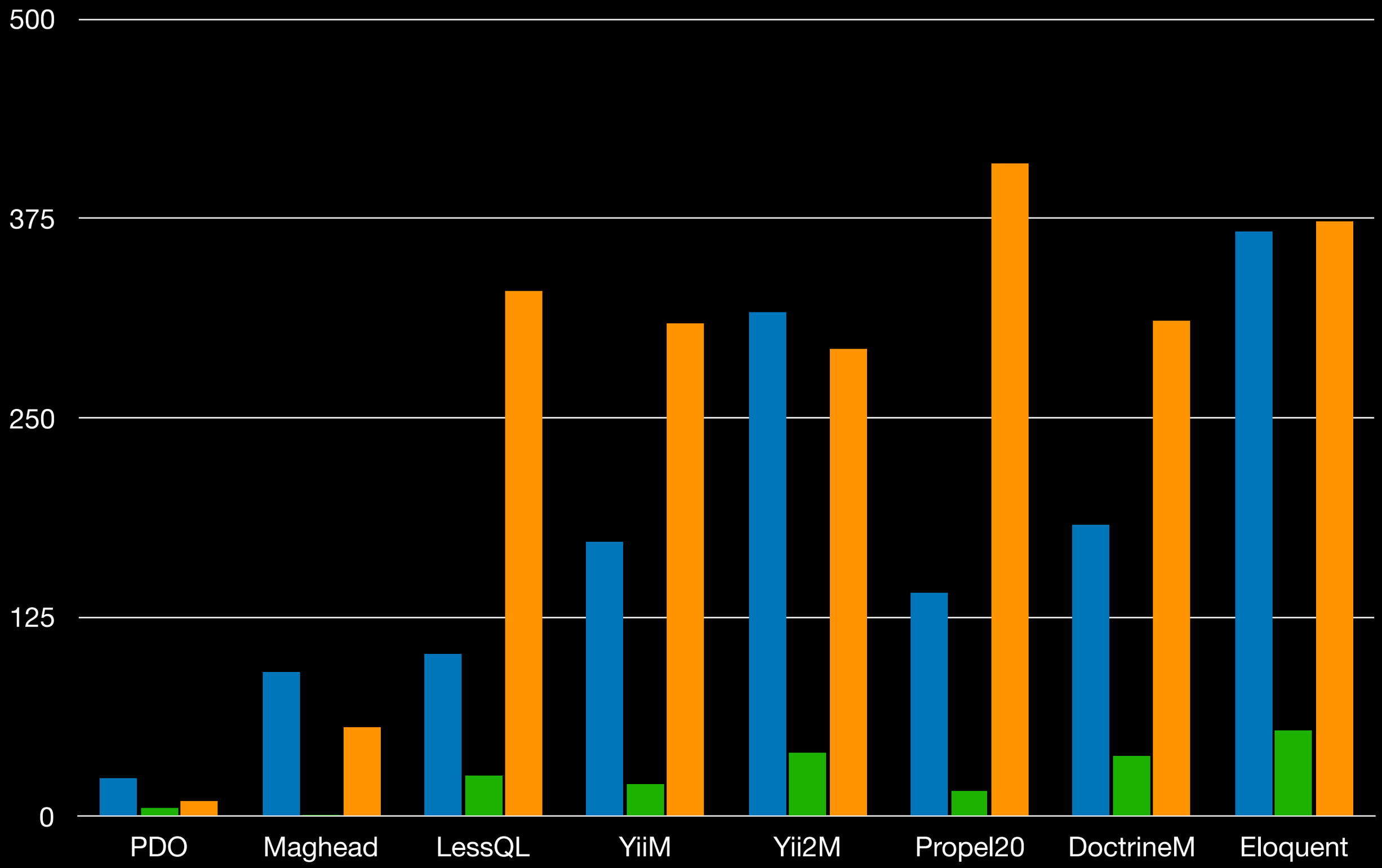
Database Support

SQLite

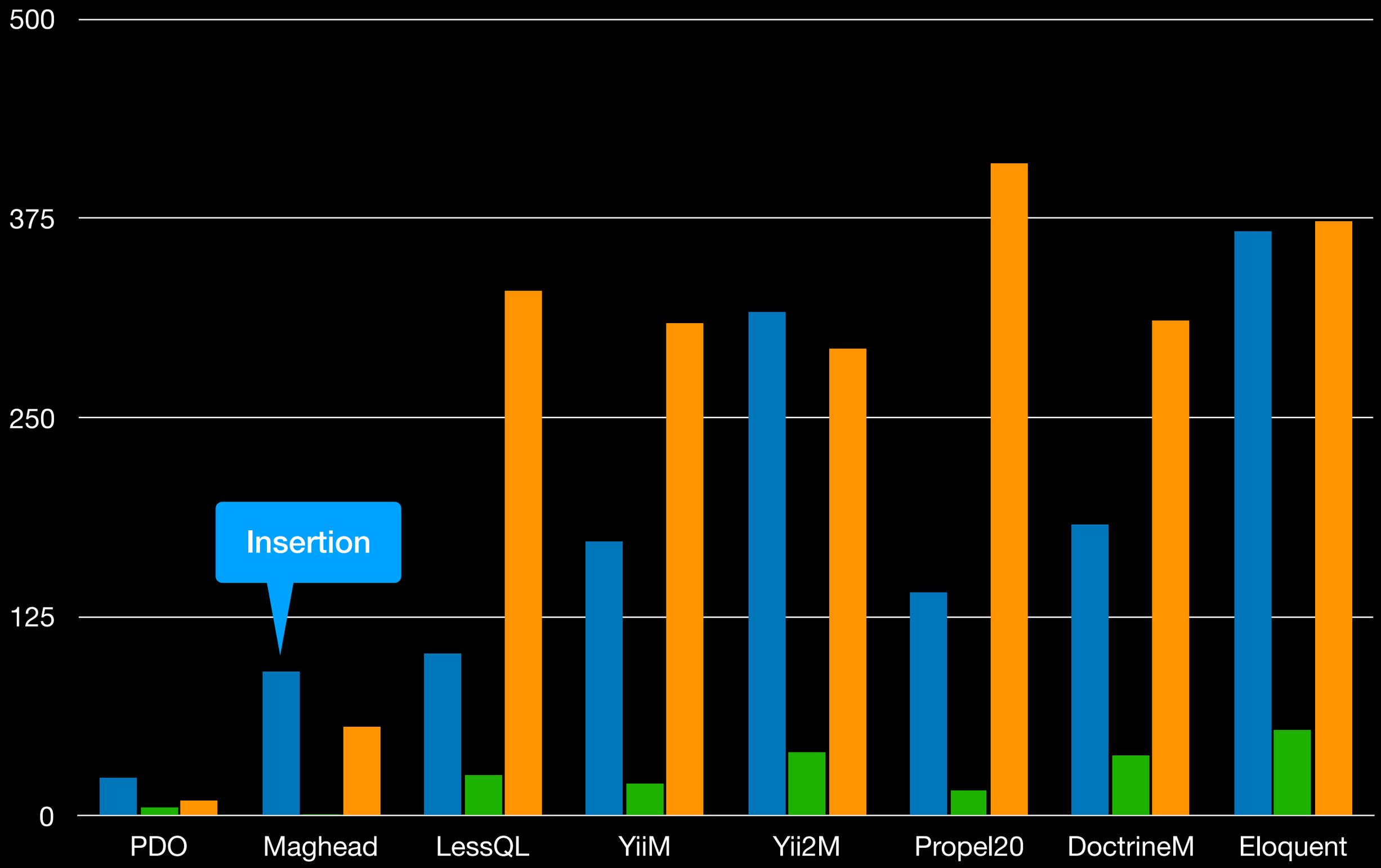
MySQL

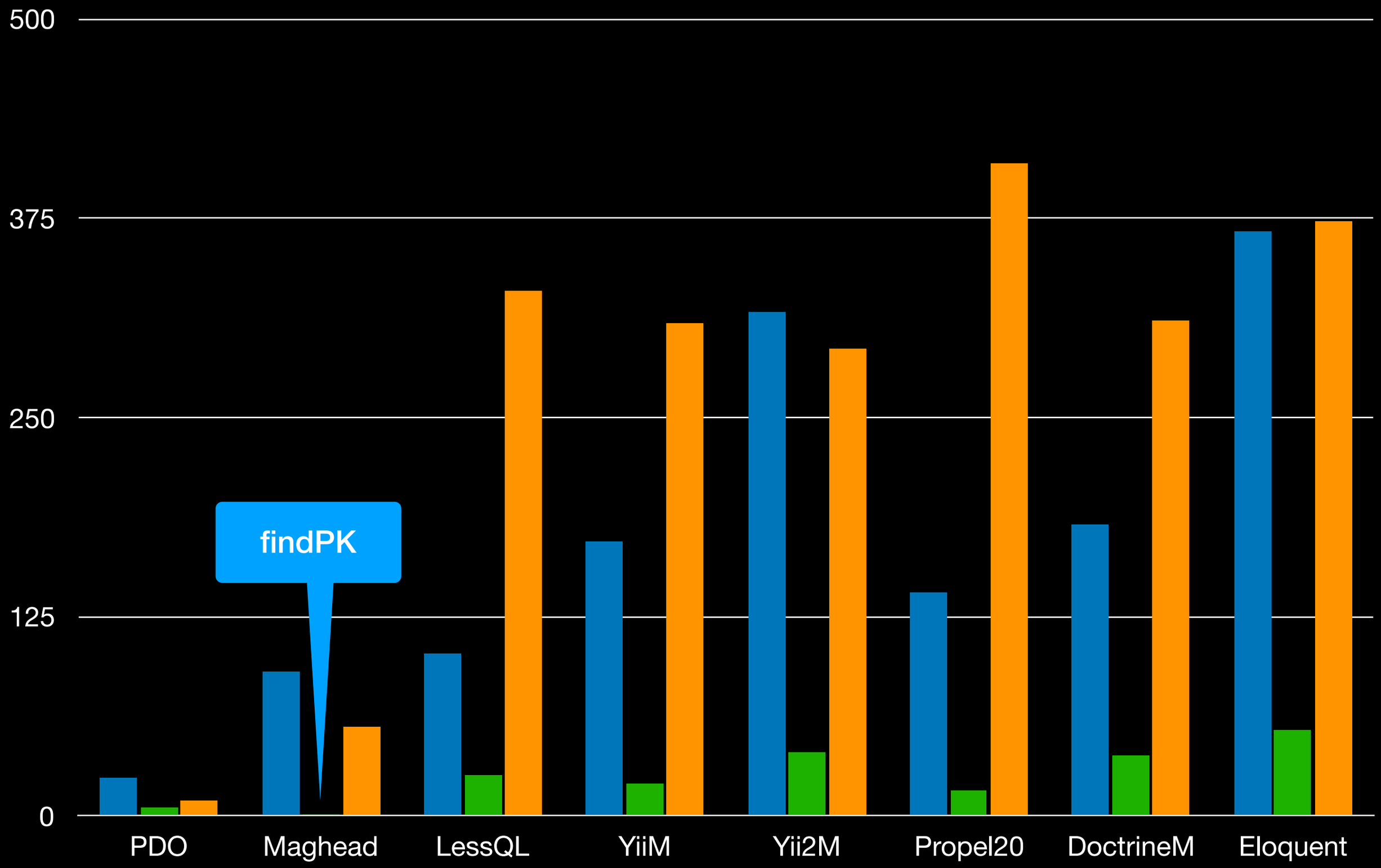
PostgreSQL

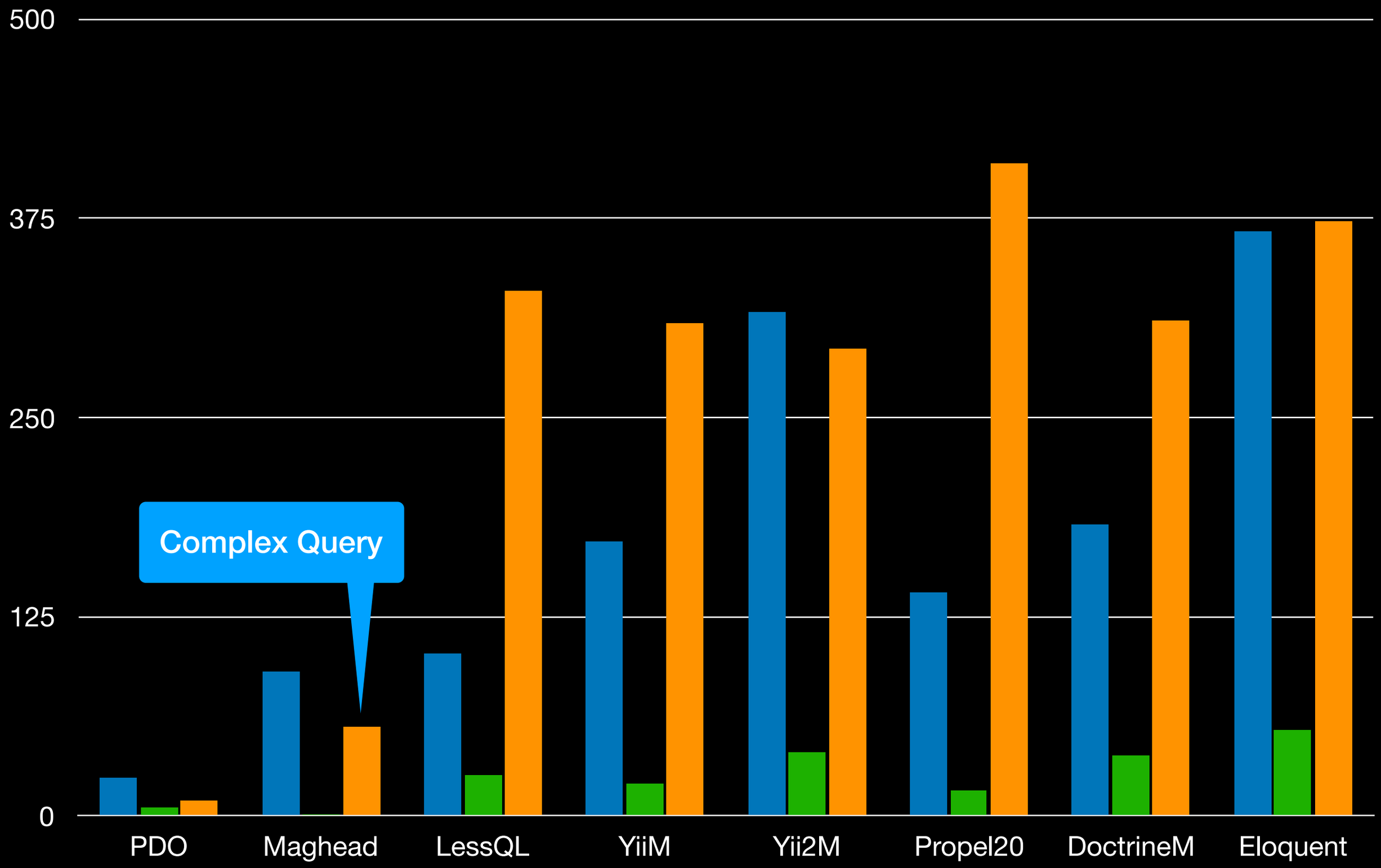
How Fast?

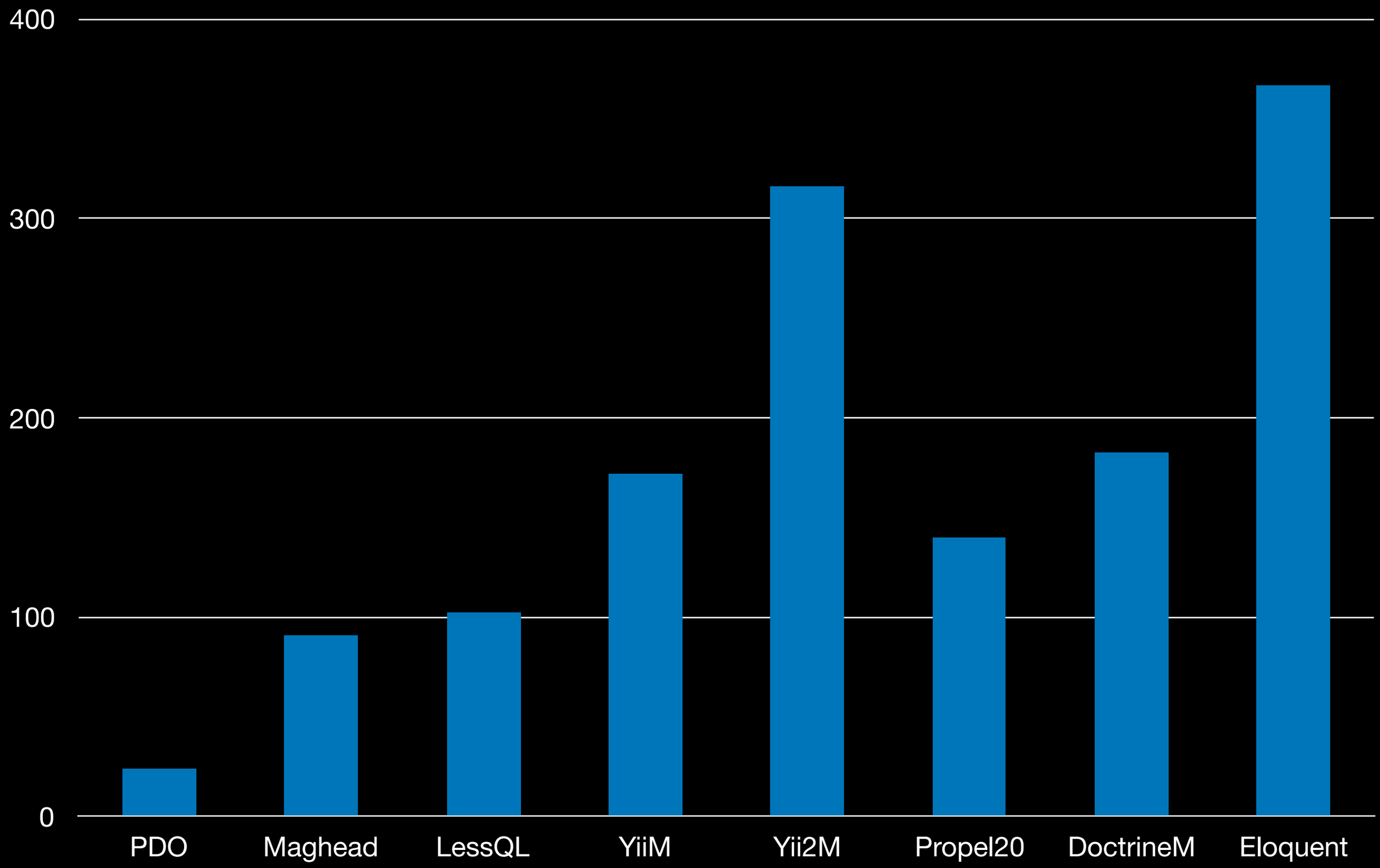


Benchmark: score -> lower is better

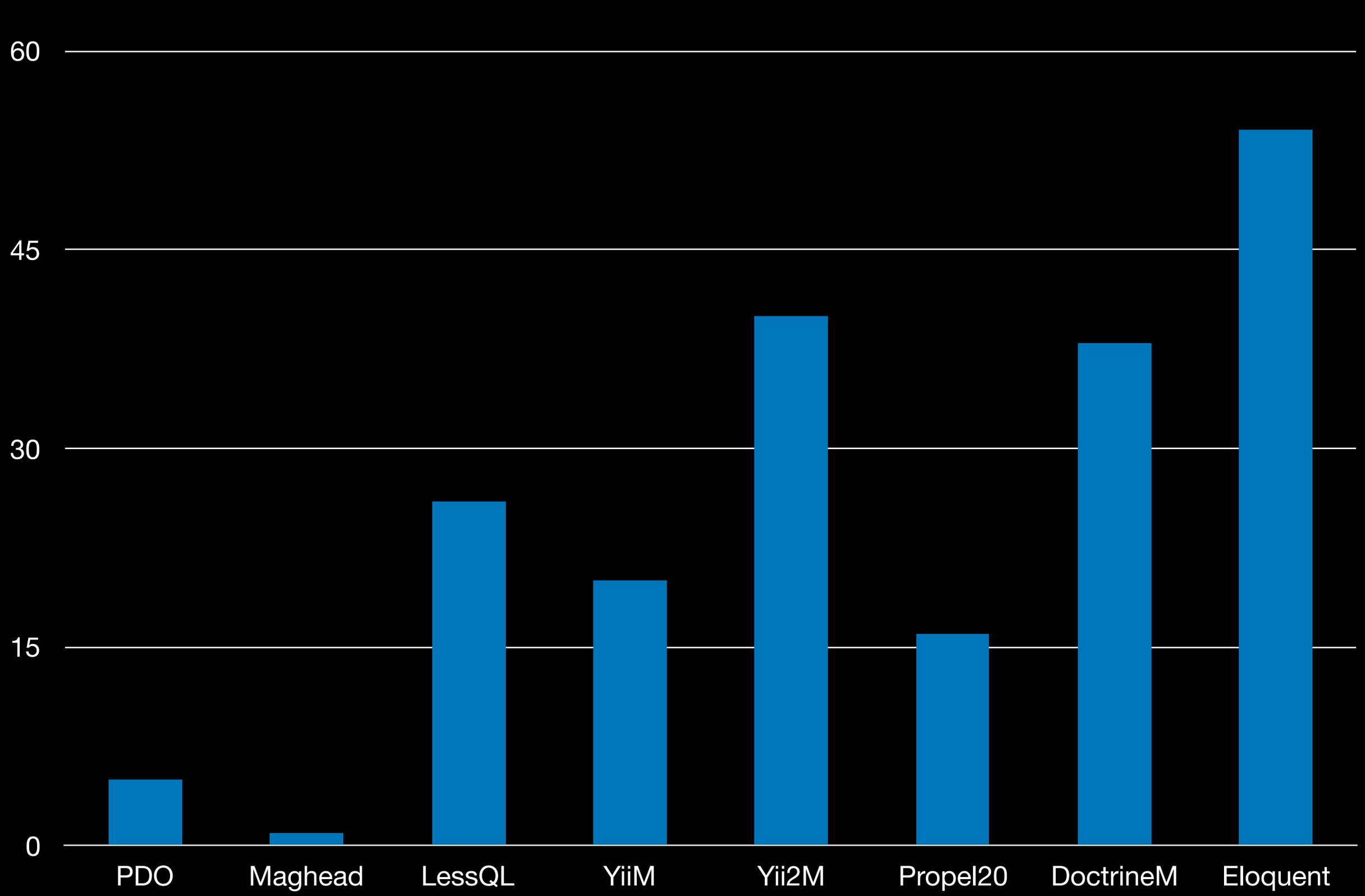

500
375
250
125
0
Insertion
PDO
Maghead
LessQL
YiiM
Yii2M
Propel20
DoctrineM
Eloquent


500
375
250
125
0
findPK
PDO
Maghead
LessQL
YiiM
Yii2M
Propel20
DoctrineM
Eloquent

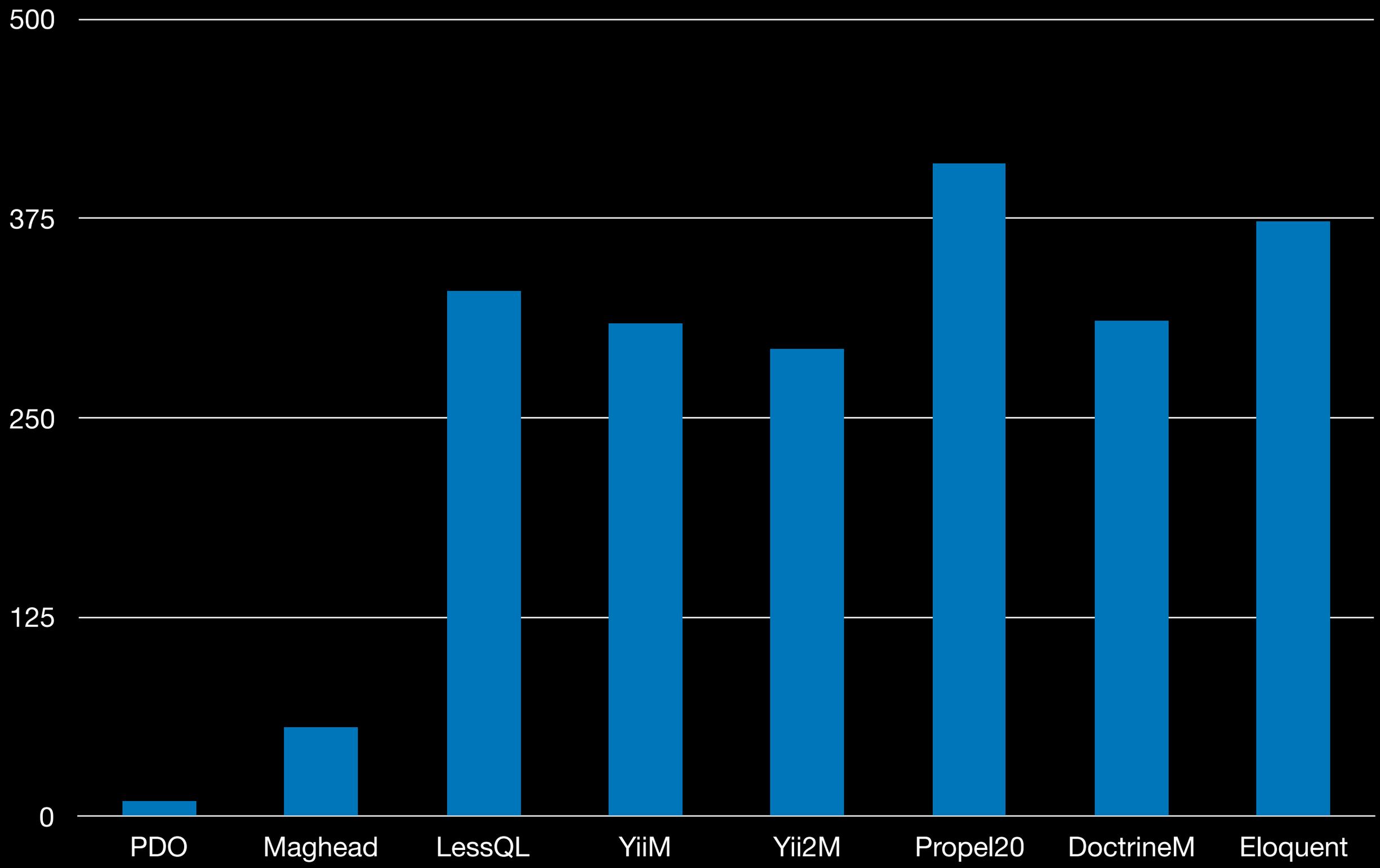

500
375
250
125
0
Complex Query
PDO
Maghead
LessQL
YiiM
Yii2M
Propel20
DoctrineM
Eloquent



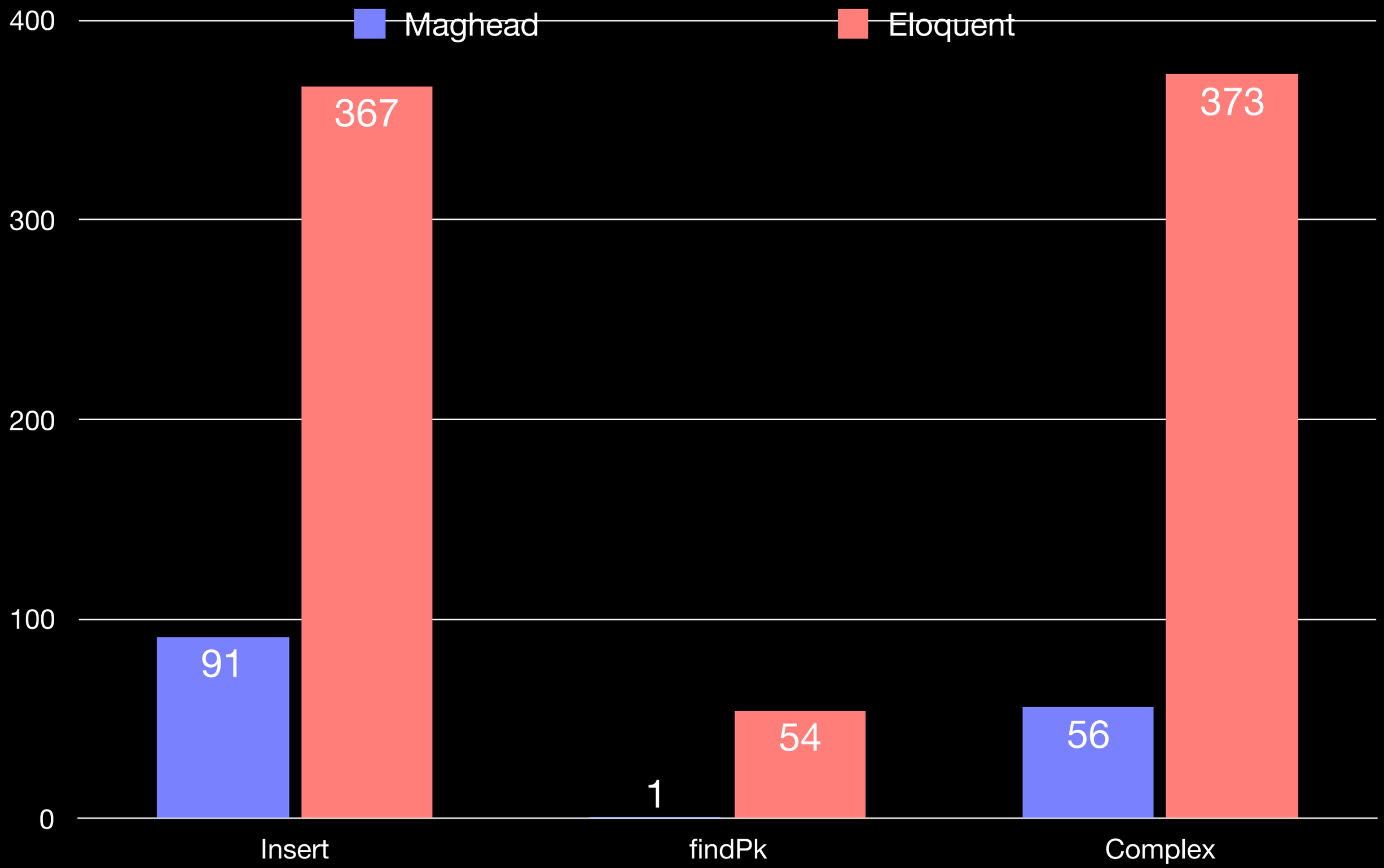
Benchmark: Insert **lower is better**



Benchmark: Find PK **lower is better**



Benchmark: Complex Query **lower is better**


Maghead
Eloquent
400
300
200
100
0
367
373
91
1
54
56
Insert
findPk
Complex

The fastest pure PHP ORM implementation

Sharding

When do you **Shard**?

You started from one

single database server

When rows grows up

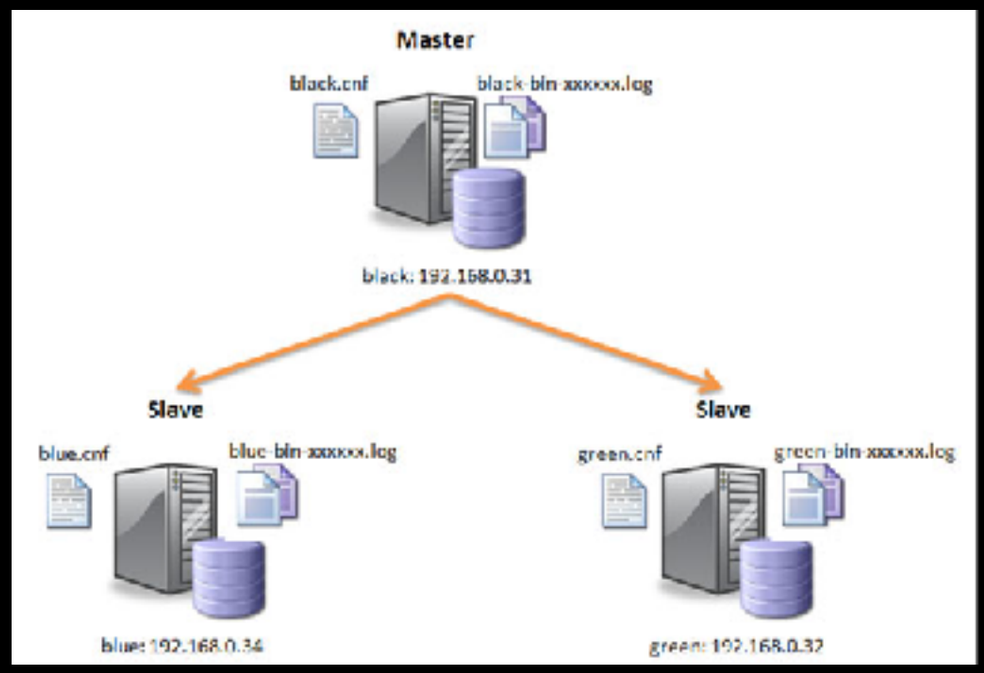
to **million**...

queries become

slow …

more read queries start hitting on the database

then you split **read/write**
to
master/replicas



finally the write load increases, **master** is crying

And rows **grows up** to 10M+...

Covering index doesn't work well anymore

Any query without
covering index won't
come back in 2-5 minutes

What **NOW**?

Upgrade the hardware?

$$$$

What's **Sharding**?

Vertical Partitioning

Horizontal Partitioning

10,000,000 records

10,000,000 records

100,000 records
100,000 records
100,000 records
100,000 records
100,000 records
100,000 records
100,000 records
100,000 records
100,000 records
100,000 records
100,000 records
100,000 records

ID	Name
1	Shaun
2	Tao
3	Ray
4	Jesse
5	Robin

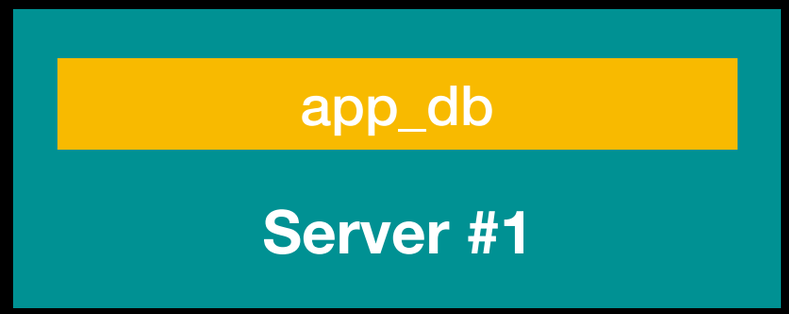
ID	Name
1	Shaun
2	Tao
3	Ray

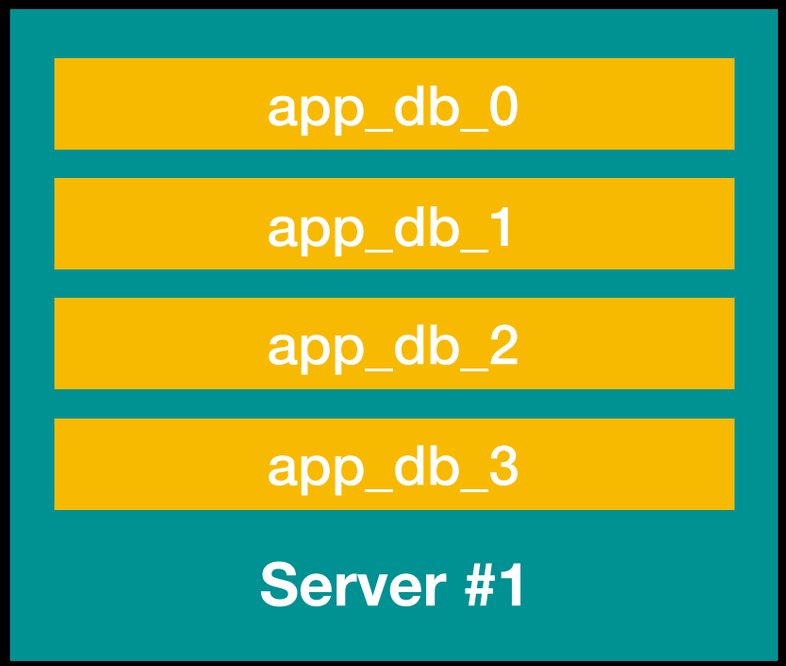
ID	Name
4	Jesse
5	Robin

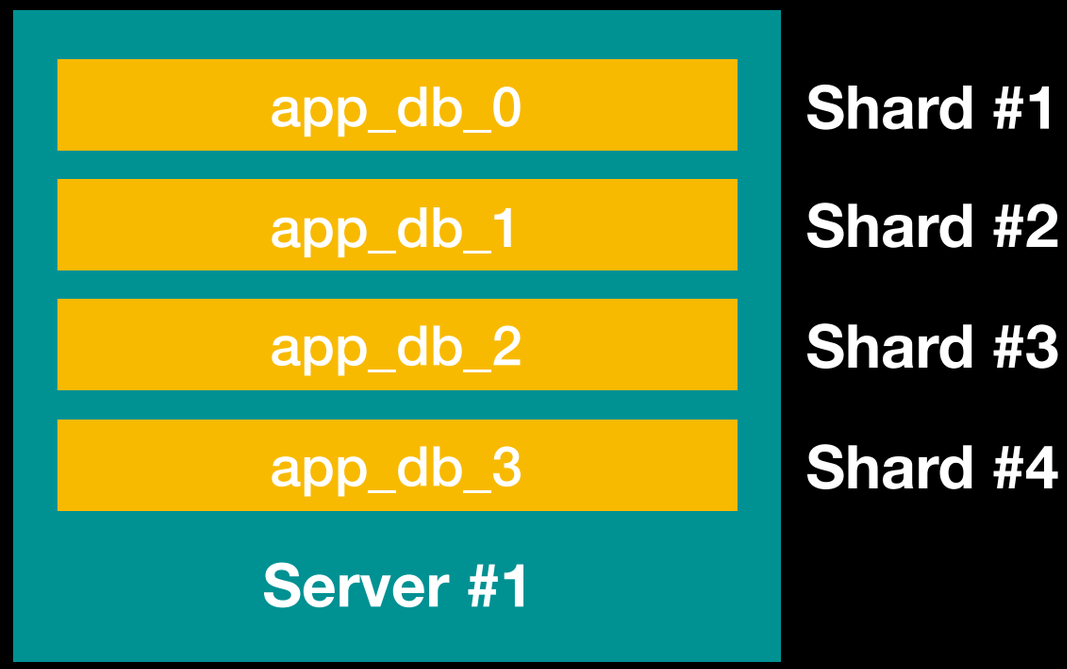
Sounds Easy?

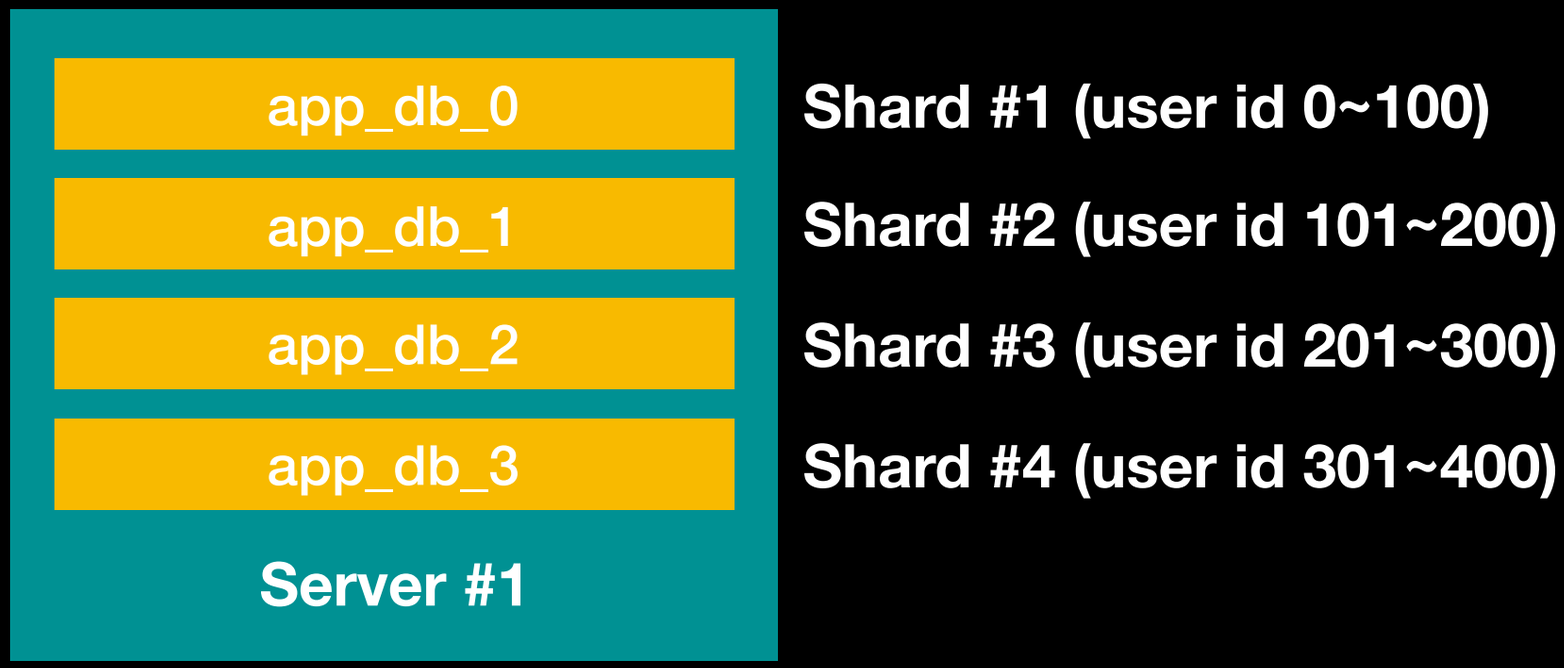
Then, How to **Shard** ?

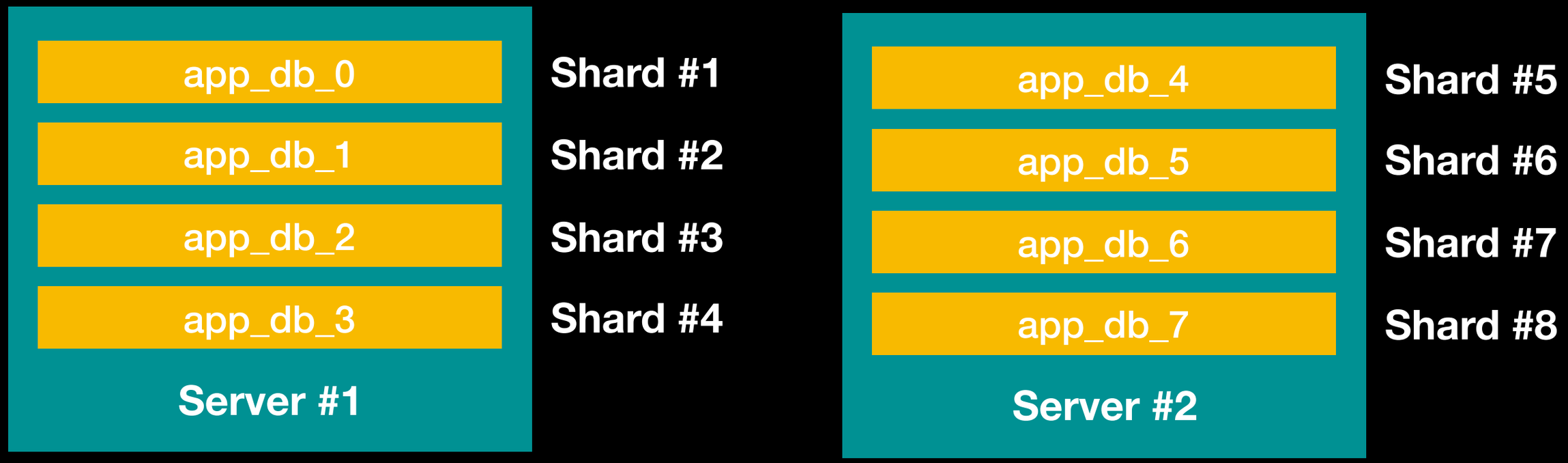
Shards


app_db
Server #1


app_db_0
app_db_1
app_db_2
app_db_3
Server #1


app_db_0
app_db_1
app_db_2
app_db_3
Server #1
Shard #1
Shard #2
Shard #3
Shard #4


app_db_0
app_db_1
app_db_2
app_db_3
Server #1
Shard #1 (user id 0~100)
Shard #2 (user id 101~200)
Shard #3 (user id 201~300)
Shard #4 (user id 301~400)



Multiple Shards per Server

app_db_0

Server #1

Shard #1

app_db_1

Server #2

Shard #2

app_db_2

Server #3

Shard #3

app_db_3

Server #4

Shard #4

One Shard per Server

Shard Key

Shard Key is the key you use to split the data

Key

Key -> Compute Shard

-> Shard #

user_id -> hash -> shard #5

store_id -> hash -> shard #2

Types of Shard Keys

- Hash
- Range
- User-defined (Tags in MongoDB)

How to choose the

Shard Key ?

It depends on how you **find** / **query** the data

Example #1

- Company (id)
- Store (id, company_id)
- Order (id, store_id, amount ...)
- OrderItem (order_id, quantity, ...)
- OrderPayment (order_id...)

Then you need to analysis across different stores

SELECT SUM(amount) FROM orders
WHERE **store_id IN (2,3,4,5)**
GROUP BY **store_id**

Your key is
the ID of the stores

Example #1

- Company (id)
- Store (id, company_id)
- Order (id, **company_id**, **store_id**, amount ...)
- OrderItem (**company_id**, **store_id**, order_id, quantity, ...)
- OrderPayment (**company_id**, **store_id**, order_id ...)

And then you can split the tables by **store_id**

The **store_id** could also be a part of the **UUID** of the order

And so you can extract the **store_id** from the **UUID** to lookup the **shard**

Different Sharding Approaches

- Database level sharding
- SQL level sharding
- Middleware-based sharding
- Application level sharding

RDBMS Solutions

- Vitas (from YouTube, Written in Go)
- MySQL Cluster (NDB)
- MySQL Fabric (Middleware-based sharding in Python)
- MariaDB Spider Engine
- GPDB (PostgreSQL)

Vitas

- Written in Go
- Use sql parser to distribute the queries
- Only for MySQL
- protocol gRPC
- Used by YouTube

MySQL Cluster

- Limited to 48 hosts
- Shard Key MUST BE in primary key and can't be changed. Similar to MySQL partitions.
- Primary key could be compound primary key
- NDB Engine
- When query doesn't include shard key, query will be scattered query.

MySQL Fabric

- Middleware-based sharding
- Written in Python
- Has it's own connector API binding for different languages (Java, Python)
- Manually set the host to operate from the application code.
- Bottleneck will be the middleware.

MySQL 上 sharding 的方案

Posted on February 2, 2017 by Gea-Suan Lin

Percona 的人剛好針對 database sharding 的事情整理了一篇文章，專門講 SaaS 服務時對 sharding 的規劃：「MySQL Sharding Models for SaaS Applications」。

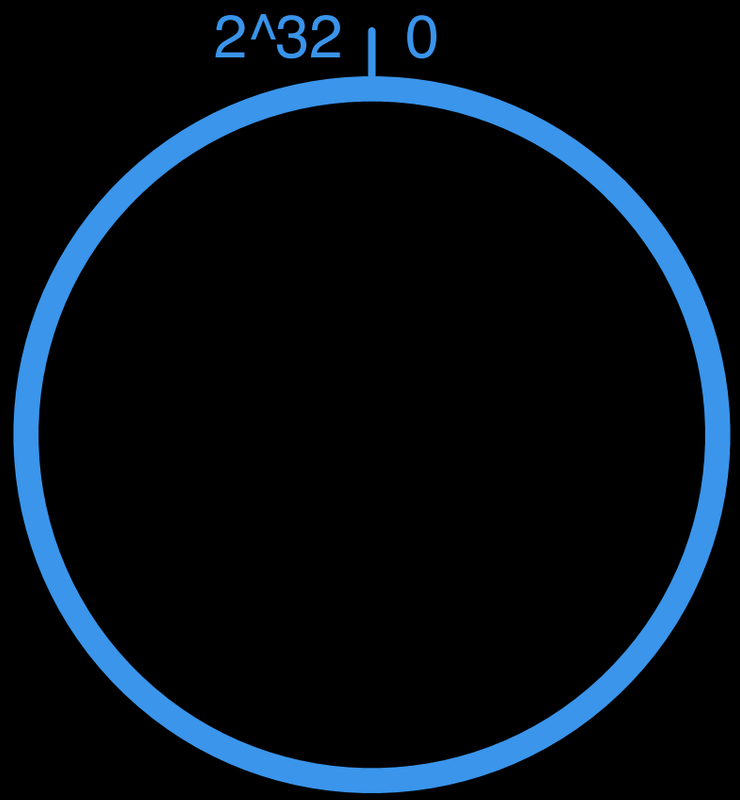
我主要是看這段：

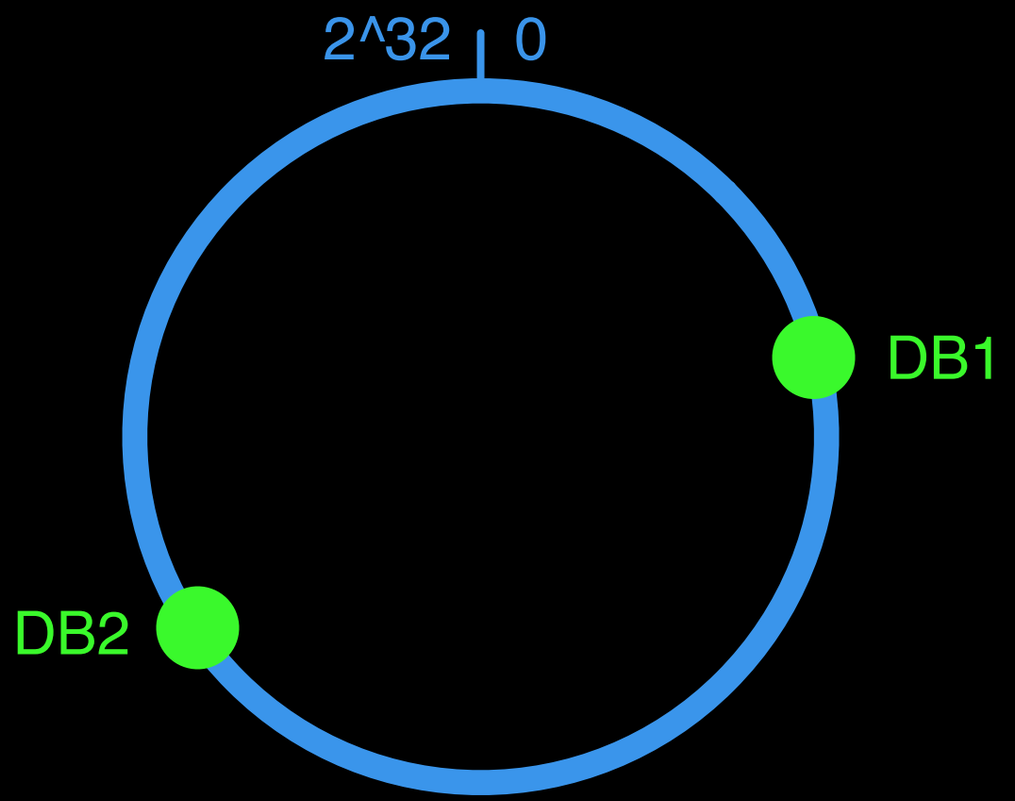
> When sharding today, you have a choice of rolling your own system from scratch, using comprehensive sharding platform such as Vitess or using a proxy solution to assist you with sharding. For proxy solutions, MySQL Router is the official solution. But in reality, third party solutions such as open source ProxySQL, commercial ScaleArc and semi-commercial (BSL) MariaDB MaxScale are widely used. Keep in mind, however, that traffic routing is only one of the problems that exist in large scale sharding implementations.

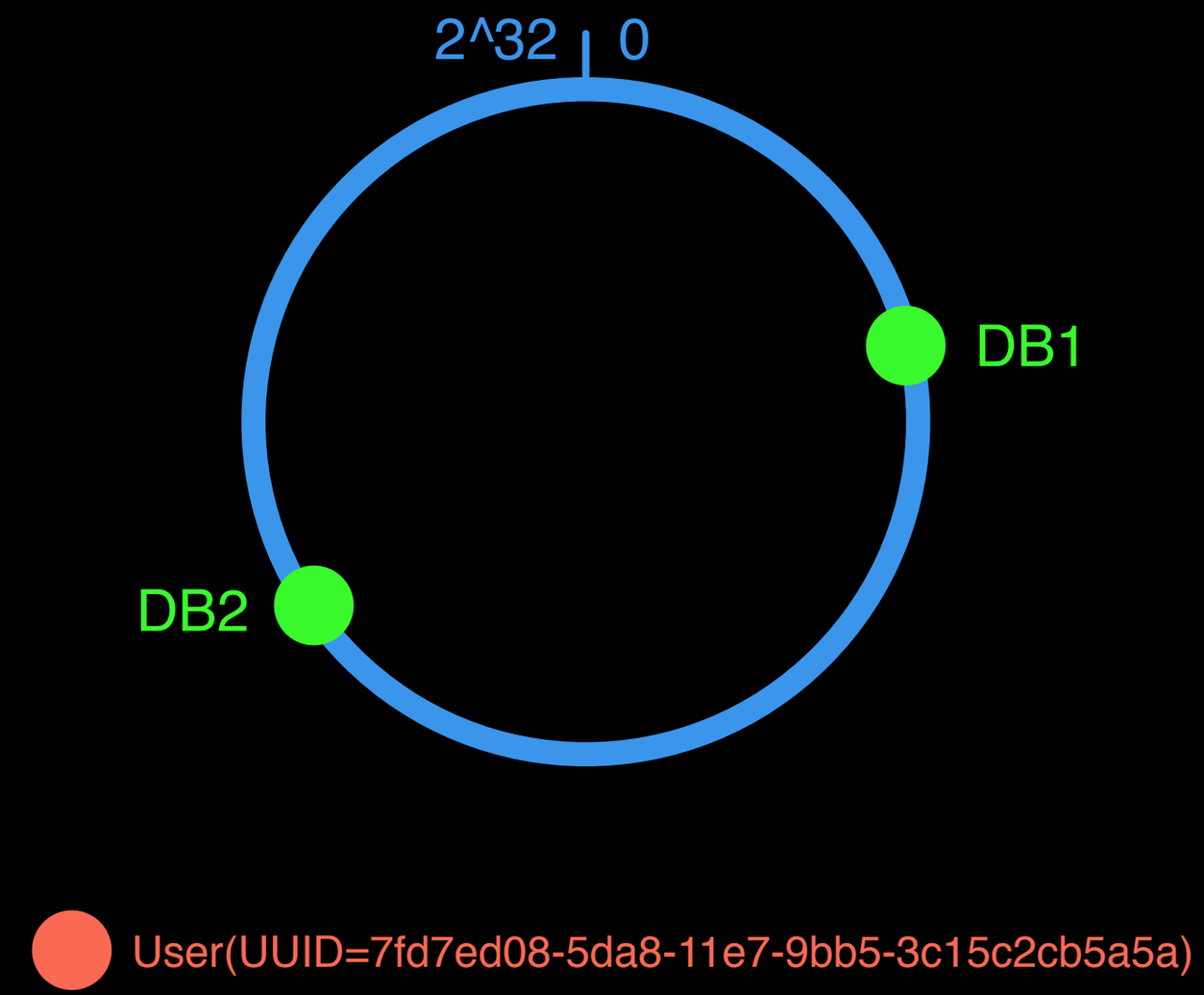
也就是說，除了在 application 上自己實作外，目前能用的方案 (堪用的方案) 大概就這些了，而且這些方案主要是解 routing 問題，對於跨 database server 的 join 都還是要自己小心實作。

by @gslin

Consistent Hashing

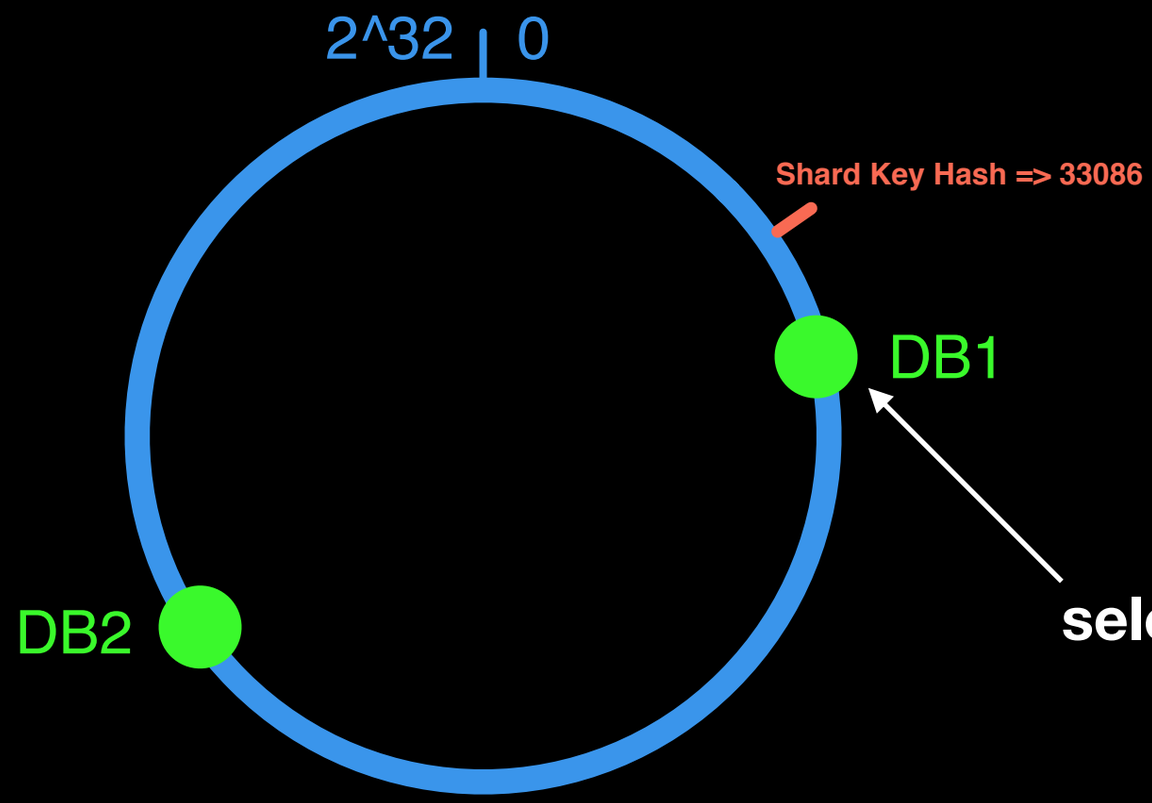

2^32
0

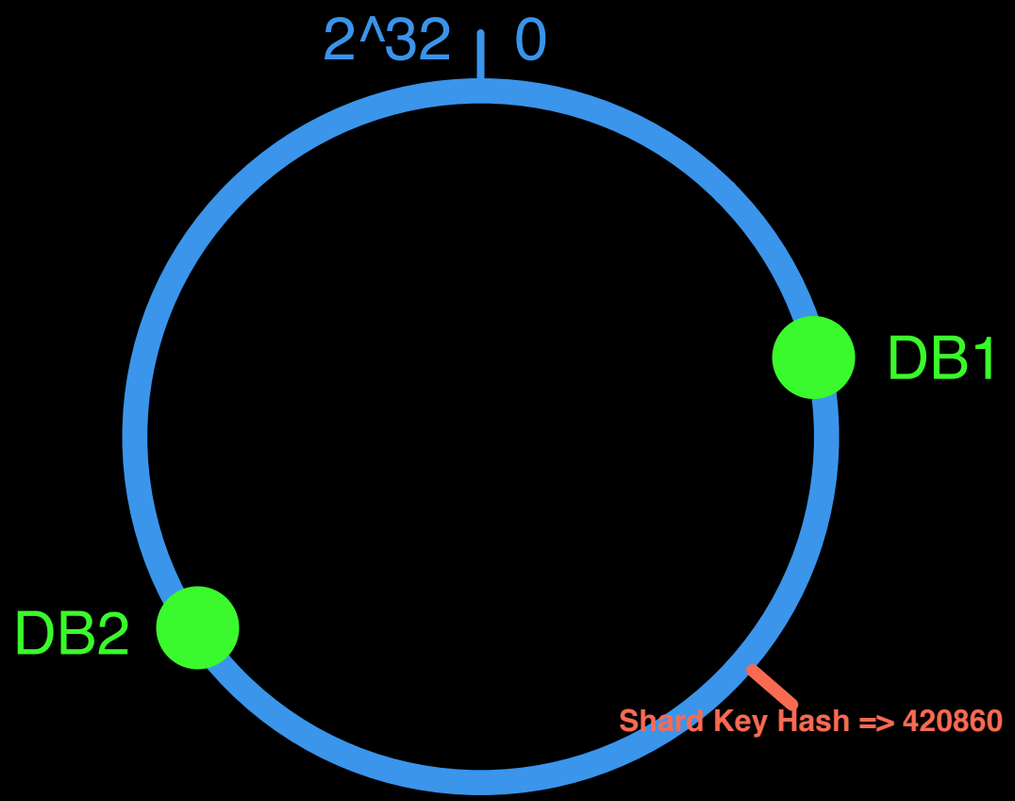

2^32
0
DB1
DB2

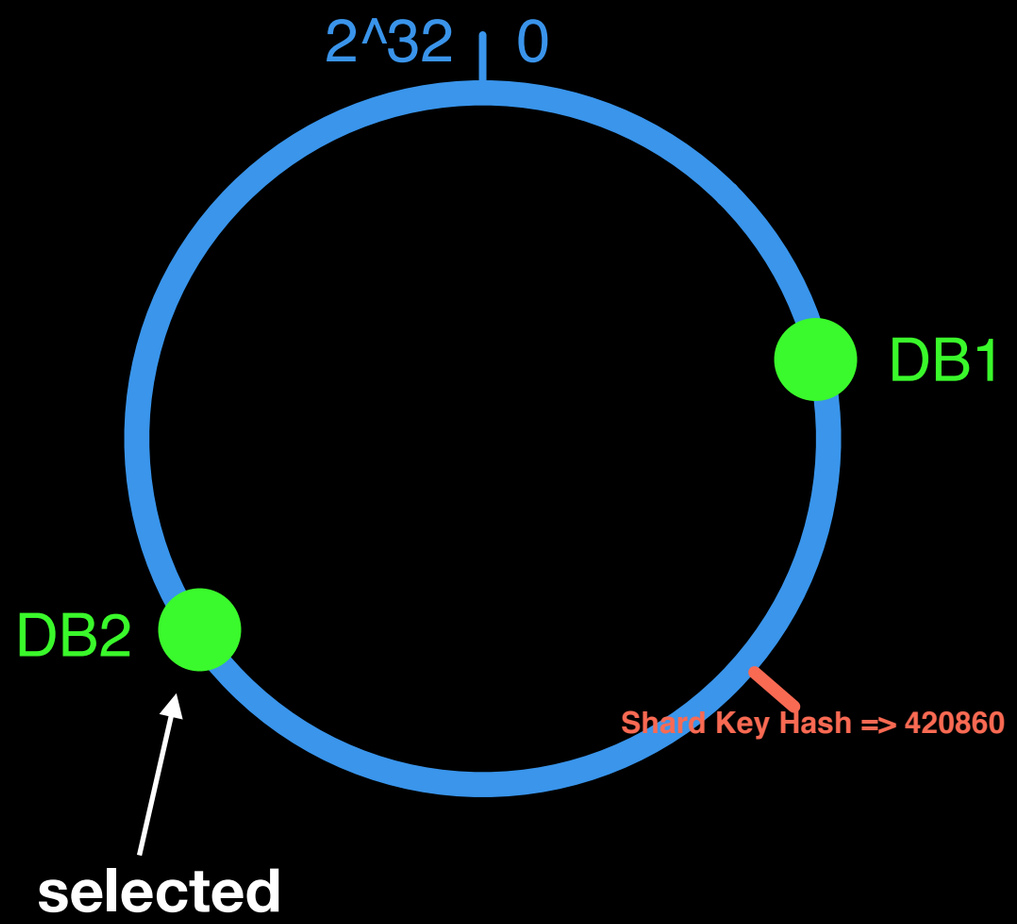


User(UUID=7fd7ed08-5da8-11e7-9bb5-3c15c2cb5a5a)

User(UUID=7fd7ed08-5da8-11e7-9bb5-3c15c2cb5a5a)




2^32
0
DB1
DB2
Shard Key Hash => 420860


2^32
0
DB1
DB2
Shard Key Hash => 420860
selected

PHP **CRC32** returns **2^32** positive integer on 64bit machine

And so we use the **native**

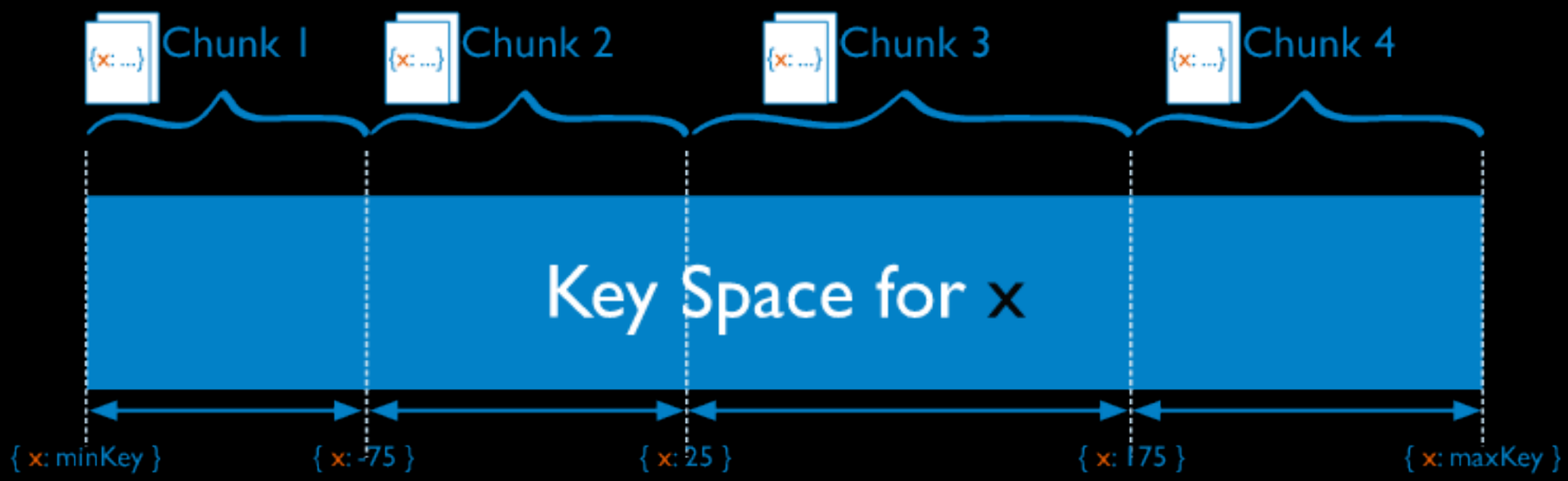
PHP array to record them

```php
foreach ($mapping->chunks as $i => $c) {
    $x = $c['index'];
    $this->buckets[$x] = $i;
    $this->targetIndexes[$i][] = $x;
}
ksort($this->buckets, SORT_REGULAR);
```

Chunks

Split the key space of the shard key into **chunks**

Chunks


Chunk 1
Chunk 2
Chunk 3
Chunk 4
{x: ...}
Key Space for x
{ x: minKey }
{ x: -75 }
{ x: 25 }
{ x: 175 }
{ x: maxKey }

And each **chunk** is assigned to a **shard**

- Chunk #1
- Chunk #2
- Chunk #3
- Chunk #4
- .....

- Key(0~100) Chunk #1
- Key(101~200) Chunk #2
- Key(201~300) Chunk #3
- Key(301~400) Chunk #4
- .....

- Key(0~100) Chunk #1 -> Shard #1
- Key(101~200) Chunk #2 -> Shard #1
- Key(201~300) Chunk #3 -> Shard #2
- Key(301~400) Chunk #4 -> Shard #2
- .....

```php
$mapping = new ShardMapping('store_keyspace', [
    'key' => 'store_id',
    'shards' => ['node1', 'node2', 'node3'],
    'chunks' => [
        ["from" => 0,          "index" => 536870912,  "shard" =>  "node1" ],
        ["from" => 536870912,  "index" => 1073741824, "shard" =>  "node1" ],
        ["from" => 1073741824, "index" => 1610612736, "shard" =>  "node1" ],
        ["from" => 1610612736, "index" => 2147483648, "shard" =>  "node2" ],
        ["from" => 2147483648, "index" => 2684354560, "shard" =>  "node2" ],
        ["from" => 2684354560, "index" => 3221225472, "shard" =>  "node2" ],
        ["from" => 3221225472, "index" => 3758096384, "shard" =>  "node3" ],
        ["from" => 3758096384, "index" => 4294967296, "shard" =>  "node3" ],
    ]
], $dataSourceManager);

$hasher = new FastHasher($mapping);
$hash = $hasher->hash($key); // return 3221225472
```

Shard Balancing

Shard Migration

- Server #1 (10 shards)
- Server #2 (10 shards)
- Server #3 (10 shards)

- Server #1 (10 shards)
- Server #2 (10 shards)
- Server #3 (10 shards)

Server load increases because server #1 has more rows than other servers.

- Server #1 (9 shards)
- Server #2 (10 shards)
- Server #3 (10 shards)
- Server #4 (1 shard)

Migrate the bottleneck shard from Server #1 to a new server.

The underlying tool

mysqldbcopy

```
mysqldbcopy --replication=slave
   --locking=lock-all
--source=root:pass@host1
--destination=root:pass@host2
database
```

replication mode=slave: copy a database from one slave to another attached to the same master

```
[root@localhost /]# mysqldbcopy --source=superadmin:root@192.168.1.10:3306 --destination=superadmin:root@192.168.1.20:3306
 --all --force --locking=snapshot --rpl master --rpl-user=repl:root
# Source on 192.168.1.10: ... connected.
# Destination on 192.168.1.20: ... connected.
# Including all databases.
# GTID operation: SET @MYSQLUTILS_TEMP_LOG_BIN = @@SESSION.SQL_LOG_BIN;
# GTID operation: SET @@SESSION.SQL_LOG_BIN = 0;
# GTID operation: SET @@GLOBAL.GTID_PURGED = 'f2e752b0-0a94-11e3-87dd-080027c9399e:1-608';
# Copying database db1
# Copying TABLE db1.foo
# Copying TABLE db1.gaga
# Copying TABLE db1.gtid
# Copying PROCEDURE db1.load_foo_test_data
# Copying PROCEDURE db1.one_load_foo_test_data
# Copying GRANTS from db1
# Copying database percona
# Copying TABLE percona.checksums
# Copying database test
# Copying TABLE test.heartbeat
# Copying GRANTS from test
# Copying data for TABLE db1.foo
# Copying data for TABLE db1.gaga
# Copying data for TABLE db1.gtid
# Copying data for TABLE percona.checksums
# Copying data for TABLE test.heartbeat
# GTID operation: SET @@SESSION.SQL_LOG_BIN = @MYSQLUTILS_TEMP_LOG_BIN;
# Connecting to the current server as master
#...done.
```

```php
use Maghead\Sharding\Operations\CloneShard;

$op = new CloneShard($config);

// public function clone($mappingId, $instanceId, $newNodeId, $srcNodeId)
$op->clone("store_keyspace", "server3", "shard1000", "shard0003");
```

```
maghead shard clone
    --drop-first
    --mapping store_key
    --instance server2
    a01 a11
```

clone shard "a01" to the a new shard on server2 named "a11"

```
maghead shard allocate
    --mapping store_key
    --instance server2
    a02
```

allocate an empty shard "a02" on server 2 and initialize all the tables

Shard Balancing API

```
$balancer = new ShardBalancer(
      new ConservativeShardBalancerPolicy(1, false, 1.3)
);
```

```
$migrateInfo = $balancer->balance($mapping,
                                  [new OrderSchema]);
```

```
$balancer = new ShardBalancer(
      new ConservativeShardBalancerPolicy(1, false, 1.3)
);

$migrateInfo = $balancer->balance($mapping,
                                  [new OrderSchema]);
```

保守式平衡規則

```php
$balancer = new ShardBalancer(
      new ConservativeShardBalancerPolicy(1, false, 1.3)
);

$migrateInfo = $balancer->balance($mapping,
                                  [new OrderSchema]);
```

一次只找出一個 Chunk 做 Migration

```php
$balancer = new ShardBalancer(
      new ConservativeShardBalancerPolicy(1, false, 1.3)
);

$migrateInfo = $balancer->balance($mapping,
                                  [new OrderSchema]);
```

找出 rows 總量超過平均 1.3 倍的 Shard

Chunk Migration

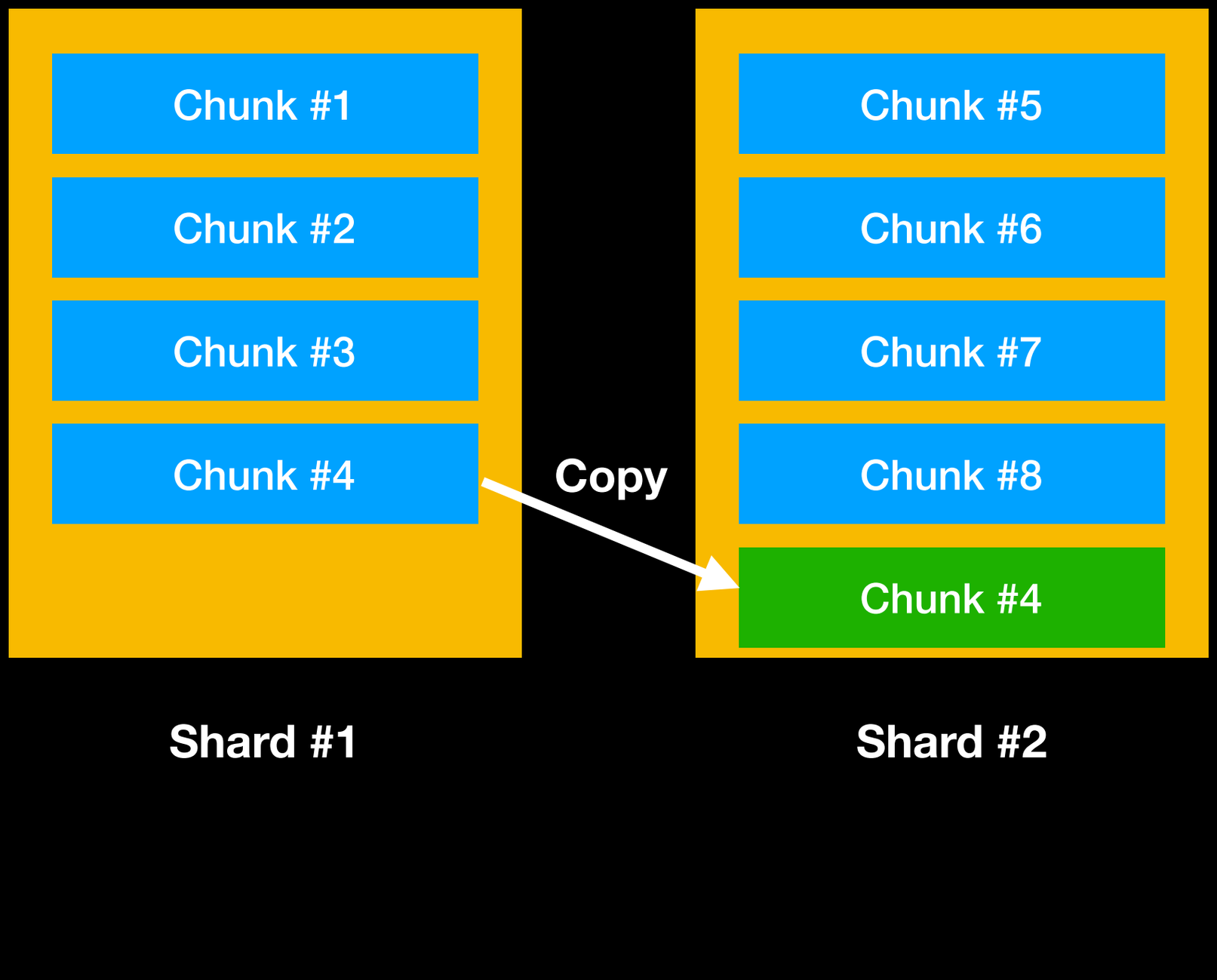
- Key(0~100) Chunk #1 -> Shard #1
- Key(101~200) Chunk #2 -> Shard #1
- Key(201~300) Chunk #3 -> Shard #2
- Key(301~400) Chunk #4 -> Shard #2
- .....

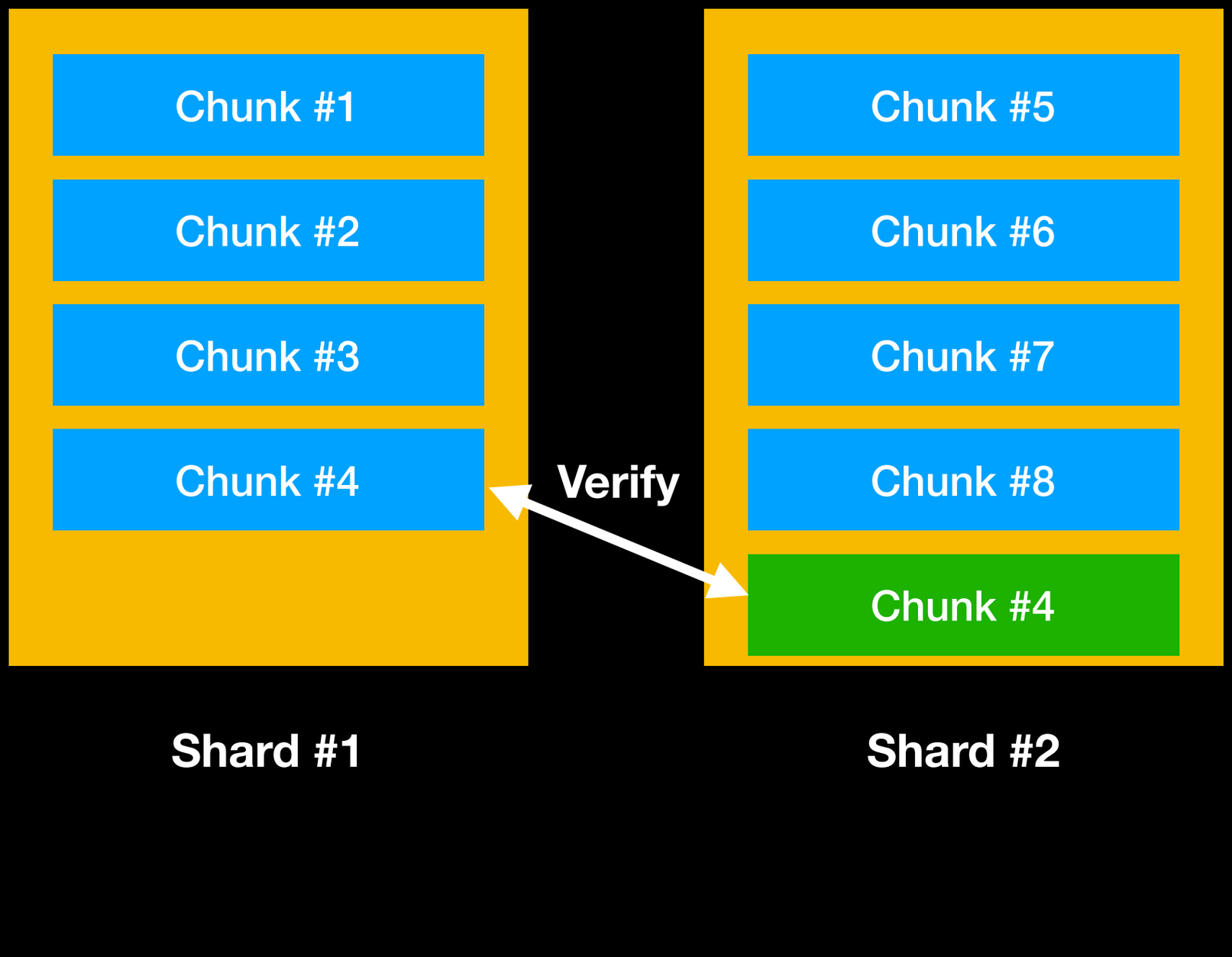
- Key(0~100) Chunk #1 -> Shard #1
- Key(101~200) Chunk #2 -> Shard #1
- Key(201~300) Chunk #3 -> Shard #2
- Key(
- .....

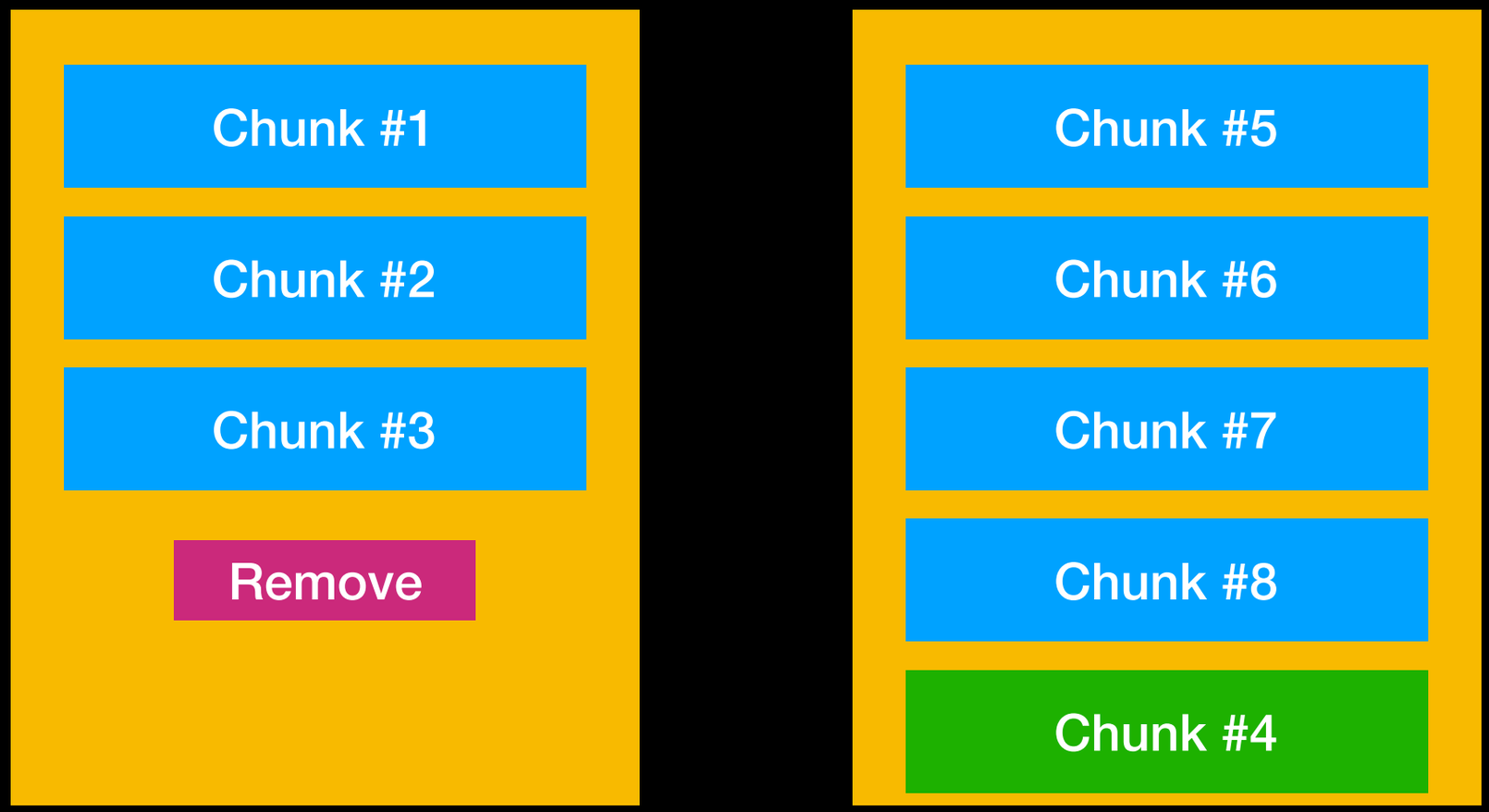
When the load of shard #2 increases, we can assign some chunks to the other shard.

- Key(0~100) Chunk #1 -> Shard #1
- Key(101~200) Chunk #2 -> Shard #1
- Key(201~300) Chunk #3 -> Shard #2
- Key(301~400) Chunk #4 -> Shard #2
- ……

When the rows in Chunk #4 grow up too much, We can split the Chunk #4 into 2+ chunks and assign them to the other shards.


Chunk #1
Chunk #2
Chunk #3
Chunk #4
Copy
Chunk #5
Chunk #6
Chunk #7
Chunk #8
Chunk #4
Shard #1
Shard #2


Chunk #1
Chunk #2
Chunk #3
Chunk #4
Verify
Chunk #5
Chunk #6
Chunk #7
Chunk #8
Chunk #4
Shard #1
Shard #2



Shard #1

Shard #2

ChunkManager API

```php
namespace Maghead\Sharding\Manager;

class ChunkManager
{
    public function __construct(ShardMapping $mapping) { }

    public function distribute(array $shardIds, $numberOfChunks = 32) { }

    public function verify(Chunk $chunk, Shard $dstShard, array $schemas) { }

    public function remove(Chunk $chunk, array $schemas) { }

    public function clone(Chunk $chunk, Shard $dstShard, array $schemas) { }

    public function migrate(Chunk $chunk, Shard $dstShard, array $schemas) { }

    public function move(Chunk $chunk, Shard $dstShard, array $schemas) { }

    public function split(Chunk $chunk, $n = 2) { }
}
```

```php
$result = $chunkManager->migrate($chunk2, $shards['node3'], $schemas);
if (!$result->isSuccessful()) {
    // ...
}
```

Jumbo Chunk

Chunks contain a large number of rows

Migrating Jumbo Chunks requires more time

Migration should start from the **small chunks**

And in the **same zone**

Start Sharding in

Maghead

Define Shard Mapping or Keyspace

```
maghead shard mapping add
    --hash
     -s node1
     -s node2
     -s node3
    --key store_id
    M_store_id
```

```yaml
---
sharding:
  mappings:
    M_store_id:
      key: "store_id"
      shards:
        - node1
        - node2
        - node3
      hash: true
      chunks:
        - { from: 0, index: 536870912, shard: node1 }
        - { from: 536870912, index: 1073741824, shard: node1 }
        - { from: 1073741824, index: 1610612736, shard: node1 }
        - { from: 1610612736, index: 2147483648, shard: node2 }
        - { from: 2147483648, index: 2684354560, shard: node2 }
        - { from: 2684354560, index: 3221225472, shard: node2 }
        - { from: 3221225472, index: 3758096384, shard: node3 }
        - { from: 3758096384, index: 4294967296, shard: node3 }
```

自動分派的 Chunk 定義

```php
class StoreSchema extends DeclareSchema
{
    public function schema()
    {
        $this->globalTable("M_store_id");
```

Store table 需要在 Shards 中被 Join 所以被定義為 Global Table

```php
namespace StoreApp\Model;

use Maghead\Schema\DeclareSchema;

class OrderSchema extends DeclareSchema
{
    public function schema()
    {
        $this->column('uuid', 'uuid-pk');

        {....cut....}

        $this->shardBy("M_store_id");
    }
}
```

Order table 需要依照定義的 keyspace 做分片

```php
namespace StoreApp\Model;

use Maghead\Schema\DeclareSchema;

class OrderSchema extends DeclareSchema
{
    public function schema()
    {
        $this->column('uuid', 'uuid-pk');

        {....cut.

        $this->shardBy("M_store_id");
    }
}
```

快速定義 UUID primary key

maghead schema build

Use the model as usual

```php
$ret = Order::create([
    'store_id' => $storeId,
    'amount' => 600,
]);
```

Automatically dispatch record to the shard by the store_id

```php
$ret->shard; // Shard object
```

The dispatched shard is returned in the result object.

Selecting Shards

```php
$store = Store::masterRepo()->findByCode('TW002');
$shard = Order::shards()->dispatch($store->id);
// Maghead\Sharding\Shard
```

Deletion works as well

```php
$order = Order::findByPrimaryKey($key);
$ret = $order->delete();
```

ShardCollection

```php
$shards = Order::shards();
$shards->locateBy(function($repo, $shard) {
    return $repo->findByCode('X01');
});
```

```php
$shards = Order::shards(); // ShardCollection
$shards->map(function($repo, $shard) {
    // do something
});
```

QueryWorker

- Maghead\Sharding\QueryMapper\GearmanQueryMapper
- Maghead\Sharding\QueryMapper\PthreadQueryMapper

```php
$query = new SelectQuery;
$query->select(['SUM(amount)' => 'amount']);
$query->from('orders');
$query->where()->in('store_id', [1,2]);
```

```php
$client = new GearmanClient;
$client->addServer();
$mapper = new GearmanQueryMapper($client);
$res = $mapper->map($shards, $query);
```

Reducer PHP7 Extension

Data Group By implemented in PHP Runtime

```php
$result = group_by($rows, $fields, [
    'total_amount' => [
        'selector' => 'amount',
        'aggregator' => REDUCER_AGGR_SUM,
    ],
    'cnt' => REDUCER_AGGR_COUNT,
]);
print_r($result);
```

```php
$rows = [
    [ 'category' => 'Food', 'type' => 'pasta', 'amount' => 1 ],
    [ 'category' => 'Food', 'type' => 'pasta', 'amount' => 1 ],
    [ 'category' => 'Food', 'type' => 'juice', 'amount' => 1 ],
    [ 'category' => 'Food', 'type' => 'juice', 'amount' => 1 ],
    [ 'category' => 'Book', 'type' => 'programming', 'amount' => 5 ],
    [ 'category' => 'Book', 'type' => 'programming', 'amount' => 2 ],
    [ 'category' => 'Book', 'type' => 'cooking', 'amount' => 6 ],
    [ 'category' => 'Book', 'type' => 'cooking', 'amount' => 2 ],
];
$result = group_by($rows, ['category','type'], [
    'total_amount' => [
        'selector' => 'amount',
        'aggregator' => REDUCER_AGGR_SUM,
    ],
    'cnt' => REDUCER_AGGR_COUNT,
]);
print_r($result);
```

See more on GitHub

Thank You

Join Us!

https://github.com/maghead/maghead

Follow me on twitter: @c9s