

Why Do Developers Prefer MongoDB?

Tim Vaillancourt
Software Engineer, Percona



`whoami`

```
{  
  name: "tim",  
  lastname: "vaillancourt",  
  employer: "percona",  
  techs: [  
    "mongodb",  
    "mysql",  
    "cassandra",  
    "redis",  
    "rabbitmq",  
    "solr",  
    "mesos",  
    "kafka",  
    "couch*",  
    "python",  
    "golang"  
  ]  
}
```



Agenda

- Data Models
 - Key/Value
 - Relational
 - Columnar
 - Document
- Common Problems
- MongoDB Features
- Comparisons
- Misconceptions
- Percona MongoDB Software

Data Models

Key / Value Stores

- A simple database for storing associative arrays
- Data can be queried by key only
- Values are opaque “blobs”
- Use cases:
 - Caches
 - *Do something that takes a long time (JOINS, API calls, etc)*
 - *Serialise to something like JSON*
 - *Write a key/value pair to Memcache*
 - *Read the key/value pair*
 - *Deserialize from something like JSON*



| Key | Value |
|-----|------------------|
| K1 | AAA,BBB,CCC |
| K2 | AAA,BBB |
| K3 | AAA,DDD |
| K4 | AAA,2,01/01/2015 |
| K5 | 3,ZZZ,5623 |

Key / Value Stores

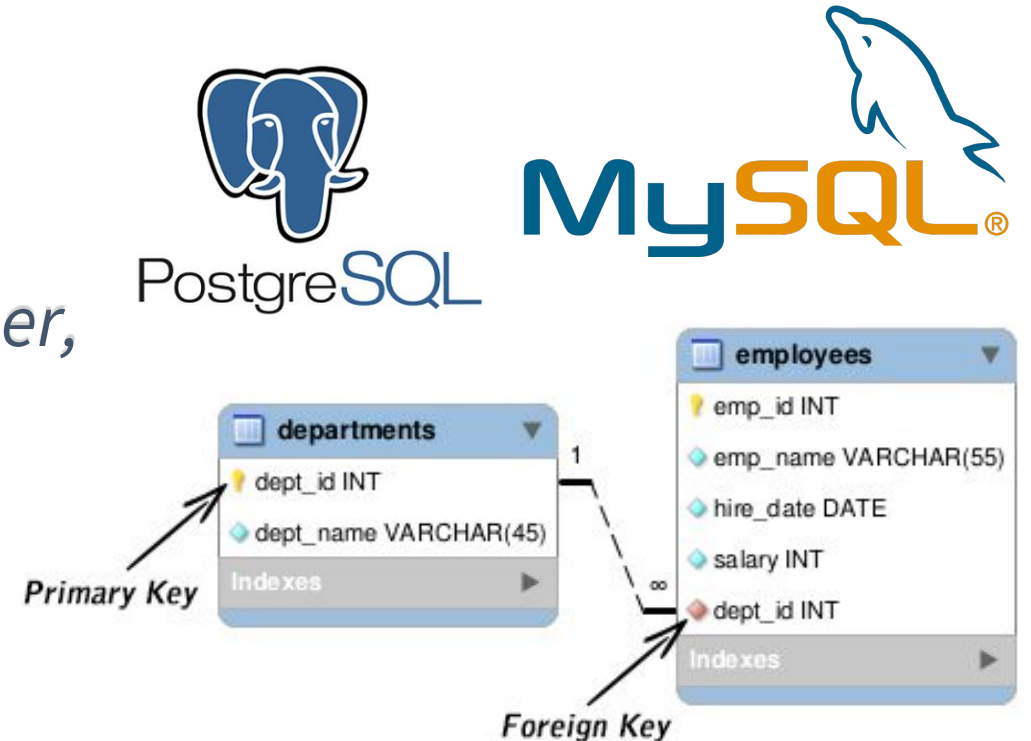
- Benefits
 - Simple
 - Fast / efficient
- Drawbacks
 - Query language is very limited
 - *Get / Set / Incr. / Decr. / Delete*
 - Values are meaningless / keys only



| Key | Value |
|-----|------------------|
| K1 | AAA,BBB,CCC |
| K2 | AAA,BBB |
| K3 | AAA,DDD |
| K4 | AAA,2,01/01/2015 |
| K5 | 3,ZZZ,5623 |

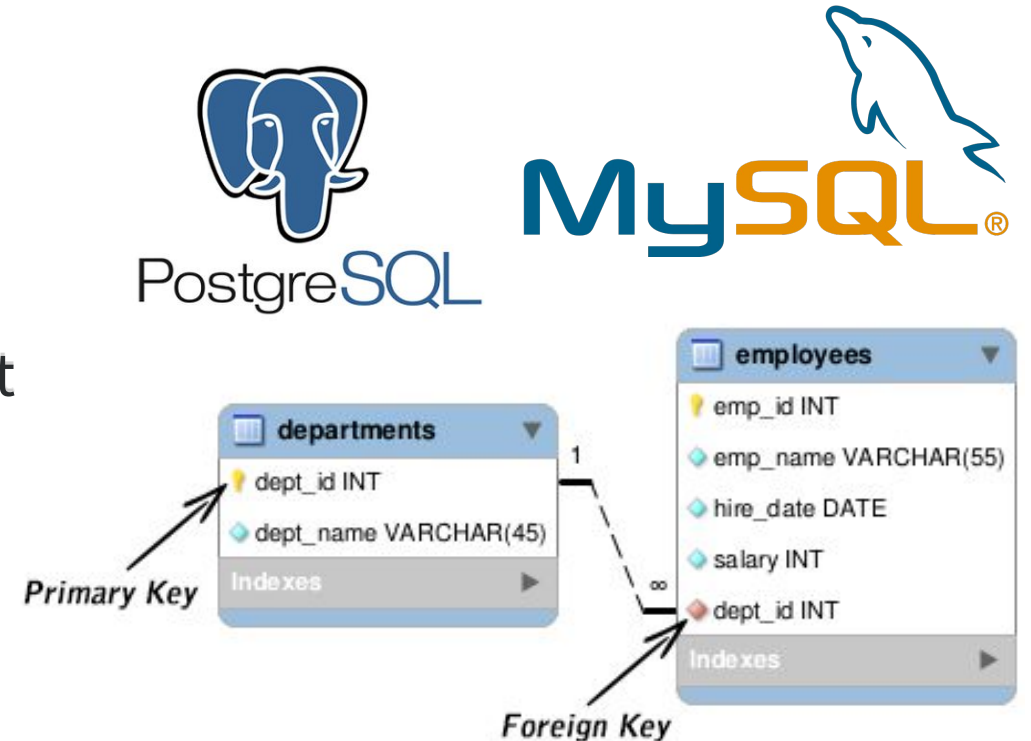
Relational Model

- Started in 1970
- Data organised into table(s) of columns and rows
- Generally each “entity type” (eg: *customer*, *product*, etc) is a table
- Schema and data types are static
- SQL query language
- Relationships are generally made using foreign keys to other tables
 - Querying these relationships require JOINS



Relational Model

- Benefits
 - Powerful SQL query language
 - Data strictness (type, size, etc)
- Drawbacks
 - JOIN-heavy database models are inefficient at scale
 - *Caching JOINS became popular with release of memcached (a Key/Value store) in 2003*
 - Rigid schema



Column-oriented Model

- Stores data tables by column rather than by row
- Accesses data more-precisely rather than scanning unwanted data in rows
- Generally abstracted to the user



Row Storage

| Last Name | First Name | E-mail | Phone # | Street Address |
|-----------|------------|--------|---------|----------------|
| | | | | |
| | | | | |
| | | | | |

Columnar Storage

| Last Name | First Name | E-mail | Phone # | Street Address |
|-----------|------------|--------|---------|----------------|
| | | | | |
| | | | | |
| | | | | |

Column-oriented Model

- Benefits
 - Compression
 - Efficiency
- Drawbacks
 - Generally more restrictions



Row Storage

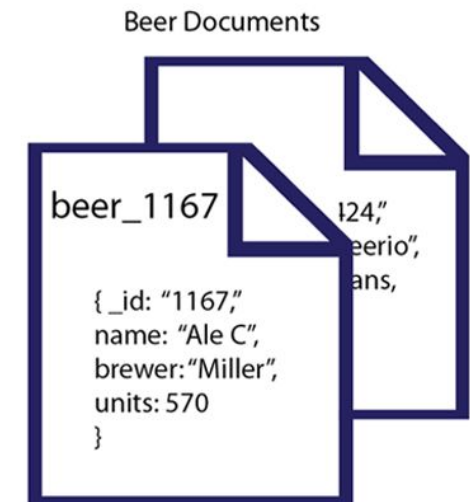
| Last Name | First Name | E-mail | Phone # | Street Address |
|-----------|------------|--------|---------|----------------|
| | | | | |
| | | | | |
| | | | | |

Columnar Storage

| Last Name | First Name | E-mail | Phone # | Street Address |
|-----------|------------|--------|---------|----------------|
| | | | | |
| | | | | |
| | | | | |

Document-oriented Model

- Allows storage of rich, semi-structured/nested documents
- Fields and data types do not need to be predefined
- Data is stored into collections of documents
- Aligns more closely with Objects in modern programming languages
- Nested objects are the new standard



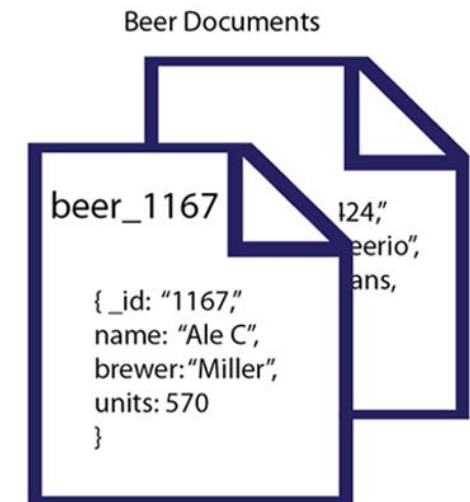
Document-oriented Model

- Benefits

- Documents == objects
- Flexible schema
- Data locality with documents
 - *ie: relationships are already "JOINED"*
 - *"Ever cached a complex structure built from SQL queries? Why not make it a single MongoDB document?"*
 - *No cross-node JOINing (at scale)*
- Rapid development

- Drawbacks

- Flexible schema :)



Documents == Objects

- Most data sources are structured JSON now
- Developers like Objects
 - Most database-driven applications use nested objects in their code:
 - *Python dictionaries*
 - *Golang structs*
 - *Ruby+Perl hashes*
 - Object-Relational Mappers
 - *A lot of SQL-driven apps are pretending their not SQL-driven apps :)*

```
{ "Situation": {
  "instance": "sit001",
  "data properties": {
    "date": "25/02/2015",
    "updateTime": "07:29 pm",
    "uncertainty": "60%"
  },
  "object properties": {
    "has_a": {
      "robberyReport": {
        "instance": "den001",
        "data properties": {
          "dataReport": "25/02/2015",
          "transcribedCall": "good night ... black cap"
        }
      }
    }
  },
  "Criminal": {
    "instance": "crim_001",
    "data properties": {
      "vehicle": "moto",
      "completeness": "70%",
      "object properties": {
        "runaway": {
          "site": {
            "instance": "local_001",
            "data properties": {
              "street": "avenue...",
              "completeness": "80%"
            }
          }
        }
      }
    }
  }
}
```


MongoDB Features

Overview

- Flexible document model
- Built-in High-availability
- Built-in Scalability
- Secondary, partial, geo and text Index support
- ACID Transaction (4.0+)
- Powerful Aggregation Framework
- Query profiling
- Role-based access control
- Change streams



Document Model

- Document schema does not need to be pre-defined
 - However, you can enable Schema Validation (*more later*)
- Documents can be up to 16MB total
- Documents are stored as BSON
 - Data is displayed as JSON in the shell, logs, etc
- Up to 100 levels of nesting
- Supports sub-documents, arrays, strings, references, date/time and numeric types
- Sub-documents and arrays are accessed with dot-notation syntax



Document Model

- Schema best practice
 - Avoid cross document/collection relationships
 - Pack as much data as possible into a single document
 - *le: move data you would get via JOINS into a single document*
 - *Specify only the required document-fields on .find() queries*
 - Use correct data types
 - *Do not store dates as strings (use the time types)*
 - *Do not store booleans as strings (use booleans)*
 - *Do not store numbers as strings (use real numbers)*
 - Indexes
 - *Add them only if you actually need them*
 - *Do not duplicate indexes*



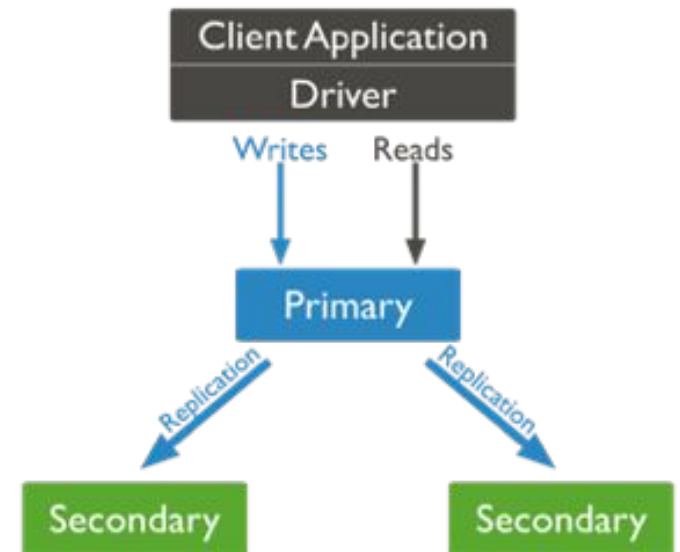
Indexes

- Compound Indexes are supported
 - Read left -> right
 - Can be partially read
- Indexes have a forward or backward direction
 - Try to cover `.sort()` with index and match direction!
- Partial
 - Updates an index based on a condition
 - Only documents matching the condition are indexed
- Text Indexes
 - Allows quick string-based searches on text
- Geo Indexes



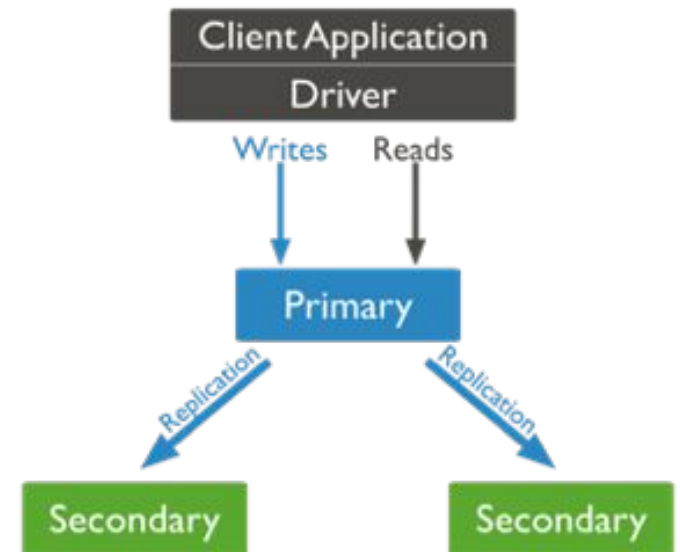
Replica Sets

- Replication
 - Changelog based, using the “Oplog”
 - Asynchronous, idempotent changes to data
- Auto-recovery/failover during failures
- More members == more read capacity
- Maximum 50 members
- Maximum 7 voting members
- Oplog
 - The “oplog.rs” capped-collection in “local” database
 - Read by secondary members for replication



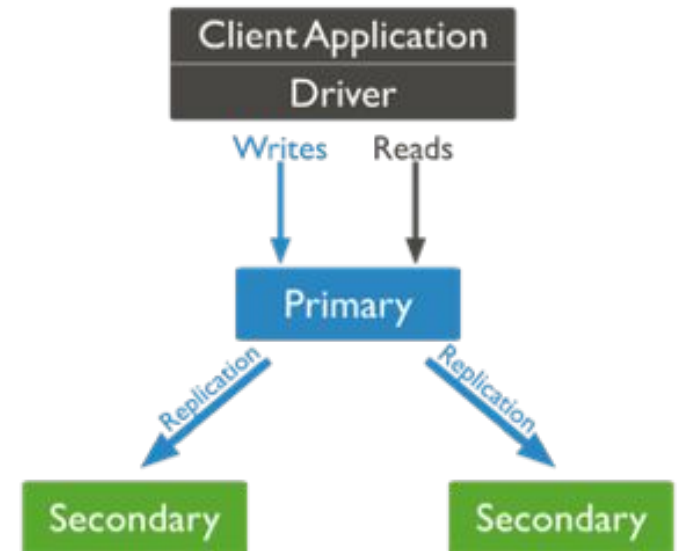
Write Concerns

- Allow fine control of data integrity of a write to a Replica Set
- Tuneable per operation or database session
- Write Concern Modes
 - “*w: <num>*” - Writes must ack to # of nodes
 - “majority” - Writes must ack on a majority of nodes
 - “*<replica set tag>*” - Writes must ack to a member with the specified replica set tags
- Durable
 - Add option “*j: true*” to journal to disk before ack!



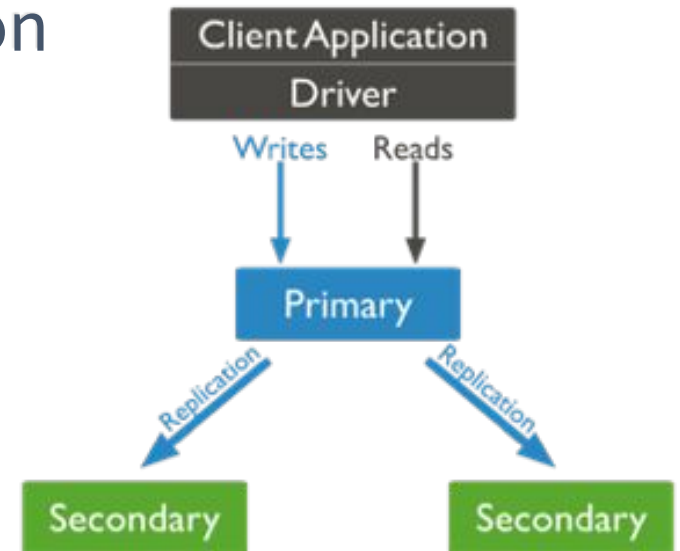
Read Concern

- Like write concerns, the consistency of reads can be tuned per session or operation
- Levels
 - **“local”** - Default, return the current node’s most-recent version of the data
 - **“majority”** - Reads return the most-version of the data that has been ack’d on a majority of nodes. Not supported on MMAPv1.
 - **“linearizable”** (3.4+) - Reads return data that reflects a “majority” read of all changes prior to the read



Scaling: Read Preference

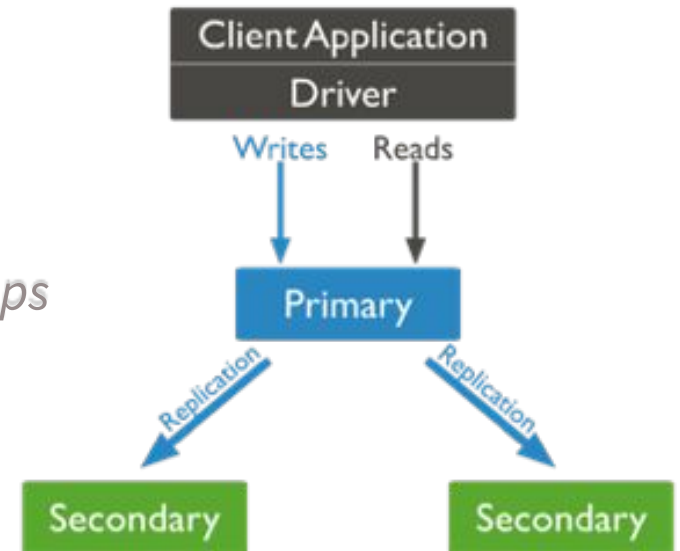
- Defines which nodes can perform a read operation
 - Can be changed per operation or session
 - Example (*probably a bad one*):
 - Read unread email messages using “secondaryPreferred”
 - Read deleted email message from “primary”
- Read Preference modes
 - primary (*default*)
 - primaryPreferred
 - secondary
 - secondaryPreferred (recommended for Read Scaling!)
 - nearest



Scaling: Read Preference

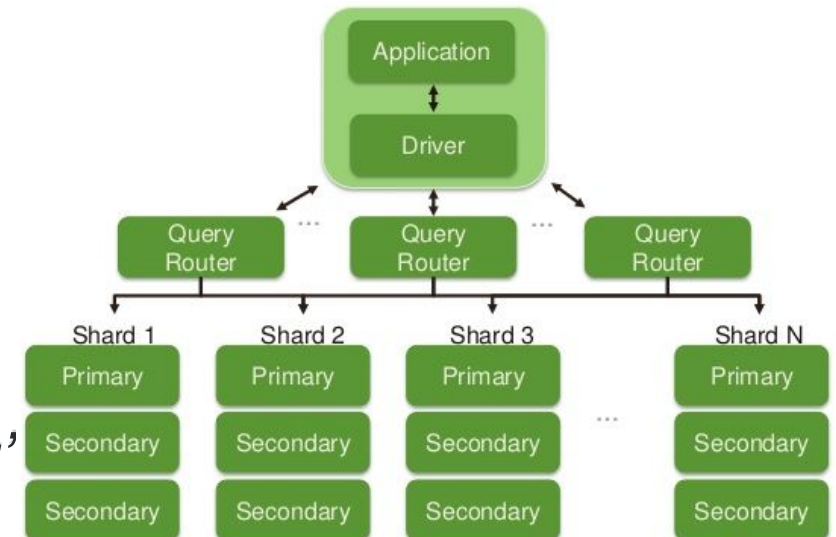
- Tags

- Select nodes based on key/value pairs (*one or more*)
- Often used for
 - *Datacenter awareness, eg: { "dc": "eu-east" }*
 - *Specific workflows, eg: Analytics, BI, Batch summaries, Backups*



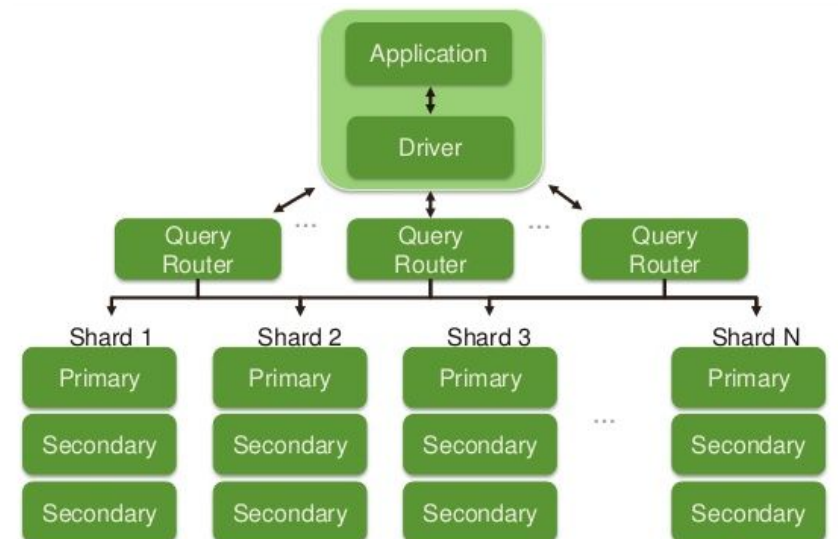
Scaling: Sharding

- A MongoDB configuration providing
 - Automatic data partitioning (*by shard key*) and balancing
 - *For now, shard key must be defined up-front!*
 - High-availability (*via replica sets*)
 - Online scaling of cluster resources
 - *Adding shards provides more write AND read capacity!*
 - Sharding based on “tags”, eg: `{ datacenter: “eu” }`
 - Cluster-wide causal consistency + clock (3.6+)



Scaling: Sharding

- Additional components
 - Config Server replica set
 - Router (“mongos”) processes



Hardware: Mainframe vs Commodity

- Databases: The Past

- Buy some really amazing, expensive hardware
- Buy some crazy expensive license
 - *Don't run a lot of servers due to above*
- Scale up:
 - *Buy even more amazing hardware for monolithic host*
 - *Hardware came on a truck*
- HA: When it rains, it pours

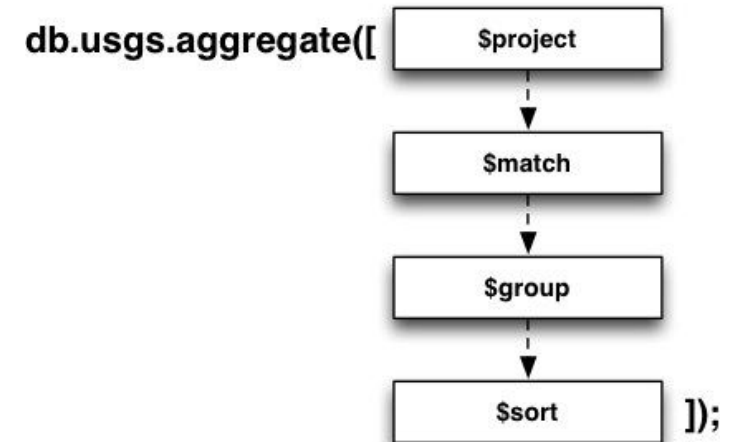
- Databases: Today

- Buy what you can afford
- Use a scalable database
- Add resources as needed



Aggregation Pipeline

- Run as a pipeline of “stages” on a MongoDB collection
 - Each stage passes it’s result to the next
 - Aggregates the entire collection by default
 - *Add a \$match stage to reduce the aggregation data*
- Runs inside the MongoDB Server
 - Much more efficient than .mapReduce() operations
- Example stages:
 - **\$match** - only aggregate documents that match
 - *Must be 1st stage to use indexes!*
 - **\$group** - group documents by certain conditions
 - *Similar to “SELECT GROUP BY”*



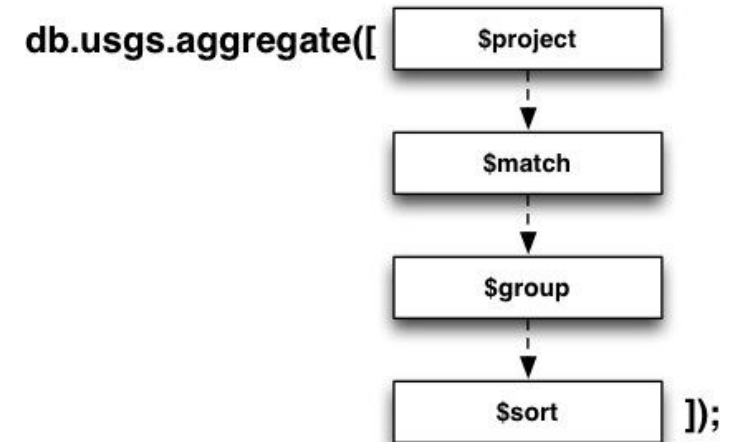
Aggregation Pipeline

- Example stages:
 - **\$count** - count the # of documents
 - **\$project** - only output specific pieces of the data
 - **\$bucket** and **\$bucketAuto** - Group documents based on specified expression and bucket boundaries
 - *Useful for Faceted Search*
 - **\$geoNear** - Returns documents based on geo-proximity
 - **\$graphLookup** - Performs a recursive search on a collection
 - **\$sample** - Returns a random sample of documents of a specified size
 - **\$unwind** - Unwinds arrays into many separate documents
 - **\$facet** - Runs many aggregation pipelines within a single stage

Aggregation Pipeline

- Just a few examples of operators that can be used each stage:

- \$and / \$or / \$not
- \$add / \$subtract / \$multiply
- \$gt / \$gte / \$lt / \$lte / \$ne
- \$min / \$max / \$avg / \$stdDevPop
- \$log / \$log10
- \$sqrt
- \$floor / \$ceil
- \$in (*inefficient*)
- \$dayOfWeek / \$dayOfMonth / \$dayOfYear
- \$concat / \$split / ...



Aggregation Pipeline

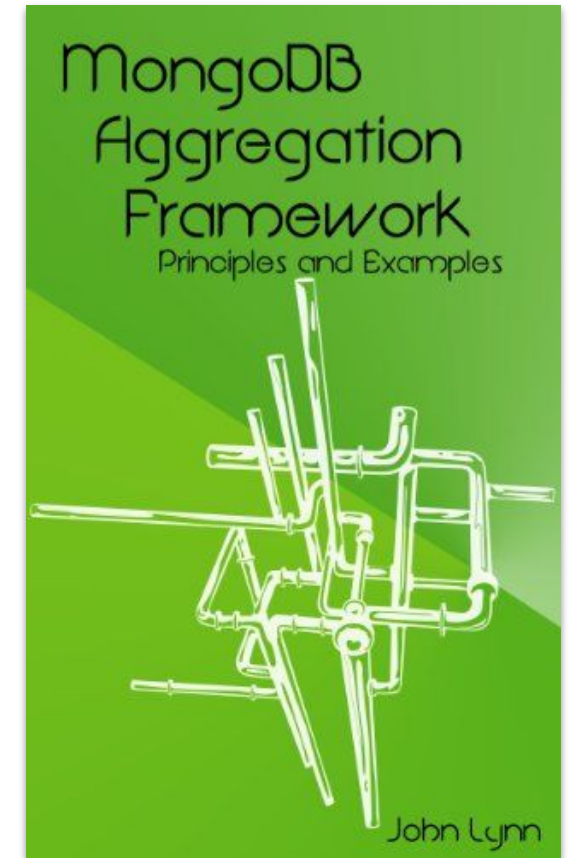
Bucket example:

```
JavaScript
1 > db.items.find()
2 { "_id" : ObjectId("58502ade9a49537a011226fb"), "name" : "scotch", "price_usd" : 90, "department" : "fc
3 { "_id" : ObjectId("58502ade9a49537a011226fc"), "name" : "wallet", "price_usd" : 95, "department" : "cl
4 { "_id" : ObjectId("58502ade9a49537a011226fd"), "name" : "watch", "price_usd" : 900, "department" : "cl
5 { "_id" : ObjectId("58502ade9a49537a011226fe"), "name" : "flashlight", "price_usd" : 9, "department" :
```

```
JavaScript
1 > db.items.aggregate([
2 ... { $bucket: {
3 ...   groupBy: "$price_usd",
4 ...   boundaries: [ 0.99, 9.99, 99.99, 999.99 ],
5 ...   output: {
6 ...     count: { $sum: 1 },
7 ...     departments: { $addToSet: "$department" }
8 ...   }
9 ... } }
10 ... ])
11 { "_id" : 0.99, "count" : 1, "departments" : [ "hardware" ] }
12 { "_id" : 9.99, "count" : 2, "departments" : [ "clothing", "food and drinks" ] }
13 { "_id" : 99.99, "count" : 1, "departments" : [ "clothing" ] }
```

Aggregation Pipeline: .aggregate()

- More on the Aggregation Pipeline:
 - <https://www.percona.com/blog/2016/12/13/mongodb-3-4-facet-aggregation-features-and-server-27395-mongod-crash/>
 - <https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/>
 - <https://docs.mongodb.com/manual/reference/operator/aggregation/>
 - <https://www.amazon.com/MongoDB-Aggregation-Framework-Principles-Examples-ebook/dp/B00DGKKGWE4>



Security: Authorization

- Built-in Roles
 - Database User: Read or Write data from collections
 - *“All Databases” or Single-database*
 - Database Admin: Non-RW commands (*create/drop/list/etc*)
 - Cluster Admin: Add/Drop/List shards
 - Superuser/Root: All capabilities
 - Backup and Restore
- User-Defined Roles
 - Exact Resource+Action specification
 - Very fine-grained ACLs
 - *DB + Collection specific*



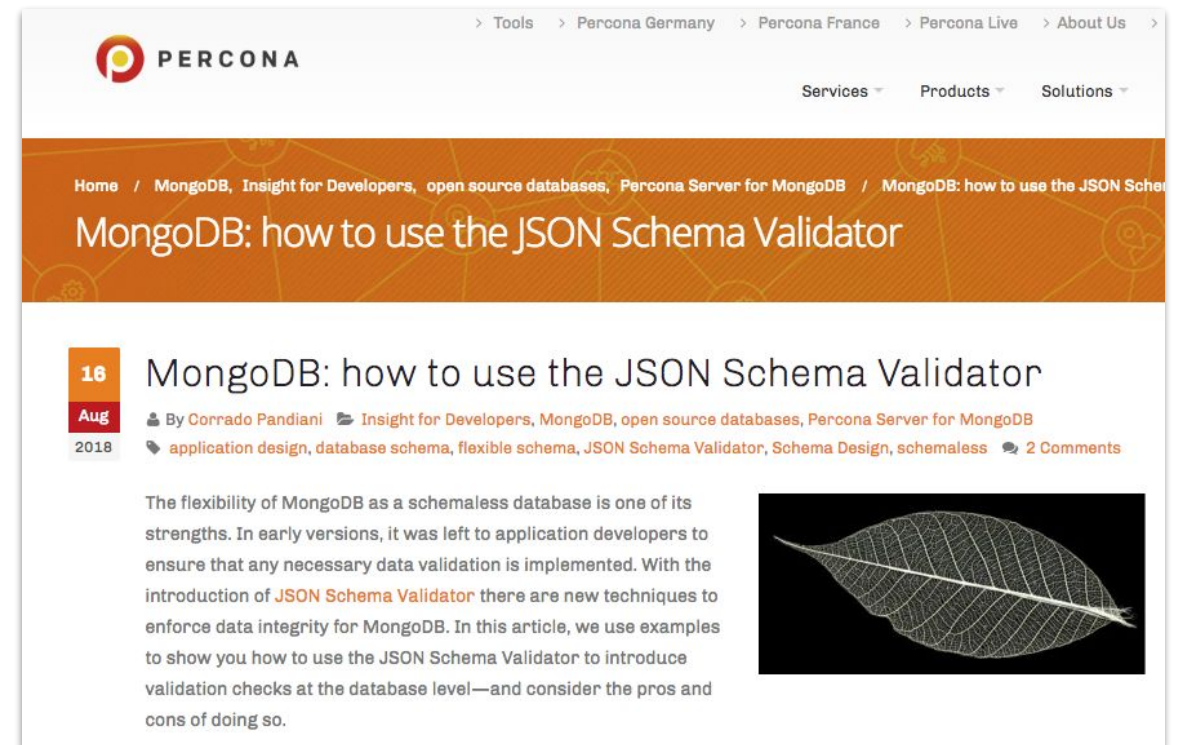
Change Streams

- Allows an app-driver to subscribe to all changes in a collection
 - Optional server-side filtering of stream
- Can be used to create a pub/sub-like workflow
- Example use cases:
 - Updating cache tiers based on inserts/updates to a collection
 - Syncing data to unrelated systems
 - Sending an email when an “email message” document changes
 - *Example of this workflow here:*
<https://www.percona.com/blog/2018/03/07/using-mongodb-3-6-change-streams/>

Schema Validation

- A feature that allows strict schema validation
 - For example:
 - *Which fields must exist in documents*
 - *What data types those field-values must have*
 - *etc*
- Refuses to write invalid documents

<https://www.percona.com/blog/2018/08/16/mongodb-how-to-use-json-schema-validator/>



The screenshot shows a web browser displaying a Percona blog article. The Percona logo is in the top left, and navigation links for 'Tools', 'Percona Germany', 'Percona France', 'Percona Live', and 'About Us' are in the top right. Below the logo, there are links for 'Services', 'Products', and 'Solutions'. The article title is 'MongoDB: how to use the JSON Schema Validator' and the date is '16 Aug 2018'. The author is 'Corrado Pandiani'. The article text discusses the flexibility of MongoDB as a schemaless database and the introduction of the JSON Schema Validator. A small image of a leaf is visible on the right side of the article.

Comparisons

Memcached / Redis

- A Key/Value store
 - Memcached: without persistence
 - Redis: with persistence to disk
- LRU caching of key/values
- Useful for
 - Caching opaque values with known keys
 - Eg: caching blobs, JOINS, API data, etc
- Drawbacks
 - Lack of query functionality
 - Lack of built-in High Availability



MySQL (PostgreSQL to some extent)

- Relational database model with SQL query language
- Support via InnoDB
- Built-in asynchronous replication
 - Semi-sync is possible
- Useful for
 - Almost anything
- Drawbacks
 - Does not have full High Availability built-in
 - *Tools like MHA themselves are rarely Highly-available*



MySQL (PostgreSQL to some extent)

- Drawbacks
 - No built-in support for scaling beyond 1 x node/set
 - *No cluster knowledge*
 - *No cluster time*
 - Lack of tunable consistency
 - Inflexible schema
 - *Schema changes usually cause a full table rewrite or significant disk I/O*
 - *Yes, there is JSON support...*
 - Depends heavily on single-node assumptions



MySQL JSON Syntax

- MongoDB was designed for semi-structured data:

```
SELECT
  *
FROM
  `e_store`.`products`
WHERE
  `category_id` = 1
AND `attributes` → '$.ports.usb' > 0
AND `attributes` → '$.ports.hdmi' > 0;
```



```
ALTER TABLE `e_store`.`products` ADD COLUMN
`port_usb` INT GENERATED ALWAYS (c->"$.ports.usb"),
ADD INDEX `port_usb` (`port_usb`);
```

```
db.products.find({
  category_id: 1,
  "ports.usb": { $gt: 0 },
  "ports.hdmi": { $gt: 0 }
})
```

```
db.products.createIndex({
  "ports.usb": 1
})
```



Cassandra

- Columnar based database with SQL-like query interface
- Benefits
 - Highly write-optimised
 - Flexible schema
 - Synchronous Writes
 - Multi-master, highly available topology
- Drawbacks
 - Lack of 2ndary indexing, limited query functionality
 - Reads are sometimes slow
 - Always uses Synchronous Writes
 - Java (*ScyllaDB has made a C-based fork*)



MongoDB in Contrast

- Full High-availability
- Linear, online read and write scaling
 - Auto-balancing of cluster data
- Natural, rich document model
- Powerful Aggregation Framework / Pipeline
- Great Read and Write performance
- Great indexing support
- Cluster-wide Causal Consistency
- Operation Profiling



MongoDB in Contrast

- Built-in on-disk compression
- Simple query syntax
- Useful for:
 - Almost anything



(Mis)conceptions

Please don't let me be misunderstood

MongoDB: MongoDB can be a great choice if you need scalability and caching for real-time analytics; however, it is not built for transactional data (accounting systems, etc.). MongoDB is frequently used for mobile apps, content management, real-time analytics, and applications involving the Internet of Things. If you have a situation where you have no clear schema definition, MongoDB can be a good choice.

Which Database is Right for Your Business?

<https://blog.panoply.io/cassandra-vs-mongodb>

1. **ACID Transactions (*which Cassandra doesn't have*)**
2. **Read and Write Concerns**

Until very recently, when you shopped for a database you had to choose: Scalability or consistency? SQL databases such as MySQL guarantee strong consistency, but don't scale well horizontally. (Manual sharding for scalability is no one's idea of fun.) NoSQL databases such as MongoDB scale beautifully, but offer only eventual consistency. ("Wait long enough, and you can read the right answer"—which isn't any way to do financial transactions.)

<https://www.infoworld.com/article/3244138/sql/cockroachdb-review-a-scale-out-sql-database-built-for-survival.html>

1. **Use Read and Write Concerns for time sensitive data**

Please don't let me be misunderstood



<https://www.youtube.com/watch?v=HdnDXsqiPYo>

1. **Journaling (enabled by default)**
2. **Write Concerns**

Author: Ken Rugg

Is ACID a Priority or Not? Can MongoDB Decide?

If I'm honest, I think of MongoDB as more of a "data store" than an actual database. This may be a controversial opinion these days, but I don't think it's a hard position to support. Granted, MongoDB does a great job storing data for a particular use case and can be highly optimized for that purpose. But, as applications inevitably evolve, Mongo's single-use case formula, where applications leverage the document store to optimize one access path to the data, starts to unravel. To be a true general purpose database, you need to be able to support many different paths through the same data set to support different use cases, whether your goal is reporting, inline analytics or just supporting a new business process that wasn't envisioned when the document structure was designed.



<https://www.enterprisedb.com/blog/acid-priority-or-not-can-mongodb-decide>

1. **ACID Transactions were added in 4.0**
2. **Cluster-wide ACID coming soon**

Please don't let me be misunderstood

"Schema-less == inconsistent data"

1. Use Schema Validation

More than 26,000 vulnerable MongoDB databases whacked by ransomware

Security shortcomings return to haunt firm as it prepares for IPO



<https://www.theinquirer.net/inquirer/news/3016752/mongodb-hack-26000-databases-whacked-by-ransomware>

1. Don't leave your front door unlocked
2. Use authentication (*enabled by default 3.6+*)

Please don't let me be misunderstood

Investors are misreading MongoDB: it's a service provider, not a software seller

MongoDB are an enterprise IT company that went public October 19. Unlike recent consumer-tech IPOs, Mongo's did rather well: shares rose 30% on the first day of trading. Mongo's pitch to investors is laid out in their prospectus. The import

Quora

Search for questions, people, and topics

Quora uses cookies to improve your experience. Read more

NoSQL MongoDB MySQL Database Systems

Why is MongoDB hosting so costly in comparison to other DBs like MySQL DB?

Ad by DatadogHQ.com

MongoDB Issues New Server Side Public License for MongoDB Community Server

New License Leads the Way for Open Source in the Cloud Era

NEW YORK, NY - October 16, 2018 – Today MongoDB, Inc. (Nasdaq: MDB), the leading modern, general purpose database platform, issued a new software license, called **Server Side Public License (SSPL)**, for MongoDB Community Server. The license clearly and explicitly states the conditions of deploying MongoDB - or any other open source project licensed under the SSPL - as a service. All versions of MongoDB's Community Server released after today, including patch fixes for prior versions, will be licensed under the SSPL.

<https://www.mongodb.com/press/mongodb-issues-new-server-side-public-license-for-mongodb-community-server>

1. Percona MongoDB Offerings

Percona and MongoDB

Percona Server for MongoDB

- Free, open-source drop-in replacement for MongoDB Community Edition
 - No client-level API changes
 - No lock-in unless our additional features are used
- Enterprise-Grade features for free
 - Auditing support
 - External LDAP Authentication
 - Hot Backups (*for WiredTiger*)
 - Percona InMemory Engine
 - *Make MongoDB a sharded caching tier!*
 - New in 3.6.8: alpha release of WiredTiger Data Encryption!



PERCONA
Server for MongoDB

Percona Monitoring and Management

- Free and open-source platform for managing and monitoring MySQL[®], MariaDB[®] and MongoDB[®] performance
- Provides
 - Visualisations of Database and Operating System metrics
 - Point-in-time visibility and historical trending of performance
 - Data from the MongoDB Query Profiler
 - *Very useful for development and production troubleshooting*
 - *Avoids the need for connecting to databases*
- Docker-based deployment of server

Percona PMM: Metrics

The image displays the Percona Performance Monitoring (PMM) dashboard for a MongoDB Standalone Instance. The dashboard is divided into several sections:

- System Metrics:** A row of five cards showing current system metrics: Current Linux Load Avg (0.24), Current User CPU % (7.34%), Current System CPU % (1.29%), Current IO Wait CPU % (0.46%), and Current Memory (26.36%).
- Command Operations / sec:** A line graph showing the rate of command operations per second over time, with a peak around 25 operations per second.
- Connections:** A line graph showing the number of active connections over time, fluctuating between 10 and 12.5.
- Cursors:** A line graph showing the number of open cursors over time, with a peak around 1.25.
- Queued Operations:** A line graph showing the number of queued operations over time, with a peak around 1.0.
- Dashboard List:** A sidebar on the left containing a search bar and a list of dashboards: Home, Cross Server Graphs, Disk Performance, Disk Space, MongoDB Cluster Summary, MongoDB Replica Set, MongoDB RocksDB, MongoDB Standalone Instance, and MongoDB WiredTiger.
- SELECT sbtest:** A detailed view of a specific query. It shows the query text: `SELECT c FROM sbtest? WHERE id BETWEEN ? AND ? ORDER BY c`. Below the query, there is an 'EXPLAIN' section showing the execution plan for the query, including the 'InnoDB' storage engine and the 'CLASSIC' execution mode.

Percona PMM: Query Analytics

PERCONA Query Analytics | _mongorsinmem_ | Query Profile | System Summary | Settings

Sep 25, 2017 12:16 AM | Sep 25, 2017 1:16 AM | DEMO

Top 10 of 113 Queries by % Grand Total Time (%GTT)

Search by Query Abstract, Fingerprint or ID

| # | Query Abstract | ID | Load | Count | Latency | | | | | | | | | | | | | | | | | | | | |
|---------------|--------------------|--|-----------------|------------|---|---------|----------|-----|-----------------|-------------|----------------|-----------------------|--|------------|-------------------|------------------------|--------------|---------------|-----------------|------------------------|----------|--------------|-----------------|--|----------|
| | TOTAL | | 36.34 (100.00%) | 102.97 QPS | 370.68 k (100.00%) 352.95 ms avg | | | | | | | | | | | | | | | | | | | | |
| 1 | FIND sbtest5_id,c | e2... | 0.88 (2.42%) | 0.27 QPS | 974.00 (0.26%) 3.25 sec avg | | | | | | | | | | | | | | | | | | | | |
| 2 | FIND sbtest16_id,c | b3... | 0.85 (2.34%) | 0.26 QPS | FIND sbtest5_id,c e282abedb6d02733a0c7701a198f1bc9 Metrics <table border="1"> <thead> <tr> <th>Metrics</th> <th>Rate/Sec</th> <th>Sum</th> <th>Per Query Stats</th> </tr> </thead> <tbody> <tr> <td>Query Count</td> <td>0.27 (per sec)</td> <td>979.00 0.26% of total</td> <td></td> </tr> <tr> <td>Query Time</td> <td>0.88 load (2.43%)</td> <td>0:53:04 2.43% of total</td> <td>3.22 sec avg</td> </tr> <tr> <td>Docs Returned</td> <td>27.19 (per sec)</td> <td>97.90 k 3.24% of total</td> <td>0.00 avg</td> </tr> <tr> <td>Docs Scanned</td> <td>27.19 (per sec)</td> <td>97.90 k 3.21% of total 1.00 per row sent</td> <td>0.00 avg</td> </tr> </tbody> </table> QUERY Fingerprint FIND sbtest5_id,c Example | Metrics | Rate/Sec | Sum | Per Query Stats | Query Count | 0.27 (per sec) | 979.00 0.26% of total | | Query Time | 0.88 load (2.43%) | 0:53:04 2.43% of total | 3.22 sec avg | Docs Returned | 27.19 (per sec) | 97.90 k 3.24% of total | 0.00 avg | Docs Scanned | 27.19 (per sec) | 97.90 k 3.21% of total 1.00 per row sent | 0.00 avg |
| Metrics | Rate/Sec | Sum | Per Query Stats | | | | | | | | | | | | | | | | | | | | | | |
| Query Count | 0.27 (per sec) | 979.00 0.26% of total | | | | | | | | | | | | | | | | | | | | | | | |
| Query Time | 0.88 load (2.43%) | 0:53:04 2.43% of total | 3.22 sec avg | | | | | | | | | | | | | | | | | | | | | | |
| Docs Returned | 27.19 (per sec) | 97.90 k 3.24% of total | 0.00 avg | | | | | | | | | | | | | | | | | | | | | | |
| Docs Scanned | 27.19 (per sec) | 97.90 k 3.21% of total 1.00 per row sent | 0.00 avg | | | | | | | | | | | | | | | | | | | | | | |
| 3 | FIND sbtest4_id,c | 35... | 0.85 (2.34%) | 0.26 QPS | | | | | | | | | | | | | | | | | | | | | |
| 4 | FIND sbtest15_id,c | 6a... | 0.85 (2.33%) | 0.26 QPS | | | | | | | | | | | | | | | | | | | | | |
| 5 | FIND sbtest11_id,c | e1c... | 0.84 (2.32%) | 0.26 QPS | | | | | | | | | | | | | | | | | | | | | |
| 6 | FIND sbtest7_id,c | 56... | 0.84 (2.31%) | 0.26 QPS | | | | | | | | | | | | | | | | | | | | | |
| 7 | FIND sbtest14_id,c | 4e... | 0.84 (2.30%) | 0.25 QPS | | | | | | | | | | | | | | | | | | | | | |
| 8 | FIND sbtest1_id,c | 9b... | 0.83 (2.30%) | 0.25 QPS | | | | | | | | | | | | | | | | | | | | | |
| 9 | FIND sbtest2_id,c | 69... | 0.83 (2.27%) | 0.25 QPS | | | | | | | | | | | | | | | | | | | | | |
| 10 | FIND sbtest6_id,c | be... | 0.82 (2.26%) | 0.25 QPS | | | | | | | | | | | | | | | | | | | | | |

60.00 at 2017-09-25 00:36:55

Load next

EXPLAIN sbtest EXPLAIN

JSON

```
{
  "queryPlanner": {
    "plannerVersion": 1,
    "namespace": "sbtest.sbtest5",
    "indexFilterSet": false,
    "parsedQuery": {
      "$and": [
        {

```

Coming soon from us...

- Percona Server for MongoDB 4.0
- Container orchestration for MongoDB...
- A new consistent backup solution for MongoDB...



PERCONA
LIVE EUROPE
FRANKFURT

Tickets are selling fast!

Percona Live Europe
Connect. Accelerate. Innovate.

Join the open source community in Frankfurt, Germany, to learn about core topics in MySQL, MongoDB, MariaDB PostgreSQL and other open source databases.

Reserve Your Seat

Frankfurt 5-7 November 2018

[Buy Your Tickets](#) >

Questions?



DATABASE PERFORMANCE
MATTERS