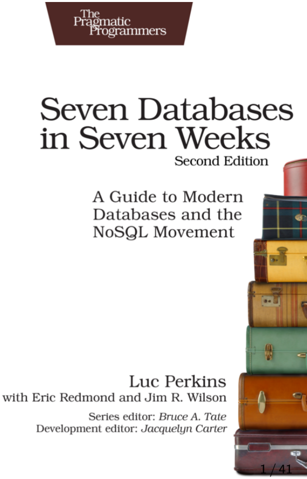
UMD DATA605 - Big Data Systems

MongoDB and CouchDB

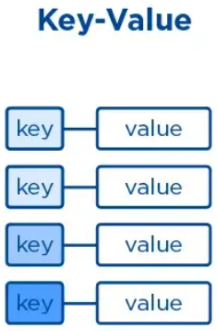
Instructor: Dr. GP Saggese - gsaggese@umd.edu

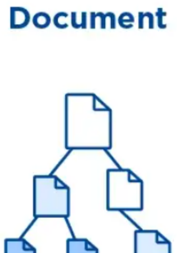
- All concepts in slides
- MongoDB tutorial
- Web
  - https://www.mongodb.com/
  - Official docs
  - pymongo
- Book
  - Seven Databases in Seven Weeks, 2e


The Pragmatic Programmers
Seven Databases in Seven Weeks
Second Edition
A Guide to Modern Databases and the NoSQL Movement
Luc Perkins
with Eric Redmond and Jim R. Wilson
Series editor: Bruce A. Tate
Development editor: Jacquelyn Carter

Key-Value Store vs Document DBs

- **Key-value stores**
  - Basically a map or a dictionary
    - E.g., HBase, Redis
  - Typically only look up values by key
    - Sometimes can do search in value field with a pattern
  - Uninterpreted value (e.g., binary blob) associated with a key
  - Typically one namespace for all key-values
- **Document DBs**
  - Collect sets of key-value pairs into *documents*
    - E.g., MongoDB, CouchDB
  - Documents represented in JSON, XML, or BSON (binary JSON)
  - Documents organized into *collections*
    - Similar to *tables* in relational DBs
    - Large collections can be partitioned and indexed





MongoDB

- Developed by MongoDB Inc.
  - Founded in 2007
  - Based on DoubleClick experience with large-scale data
  - Mongo comes from "hu-mongo-us"
- One of the most used NoSQL DBs (if not the most used)
- **Document-oriented NoSQL database**
  - Schema-less
    - No Data Definition Language (DDL), like for SQL
    - You can store maps with any keys and values
    - Application tracks the schema, mapping between documents and their meaning
  - Keys are hashes stored as strings
    - Document Identifiers **_id** created for each document (field name reserved by Mongo)
  - Values use BSON format
    - Based on JSON (B stands for Binary)
- Written in C++
- Supports APIs (drivers) in many languages
  - E.g., JavaScript, Python, Ruby, Java, Scala, C++, . . .

MongoDB: Example of Document

- **A document is a JSON data structure**
- It corresponds to a row in a relational DB
  - Without schema
  - Primary key is `_id`
  - Values nested to an arbitrary depth

```
{
   "_id" : ObjectId("4d0b6da3bb30773266f39fea"),
   "country" : {
      "$ref" : "countries",
      "$id" : ObjectId("4d0e6074deb8995216a8309e")
   },
   "famous_for" : [
      "beer",
      "food"
   ],
   "last_census" : "Sun Jan 07 2018 00:00:00 GMT -0700 (PDT)",
   "mayor" : {
      "name" : "Ted Wheeler",
      "party" : "D"
   },
   "name" : "Portland",
   "population" : 582000,
   "state" : "OR"
}
```

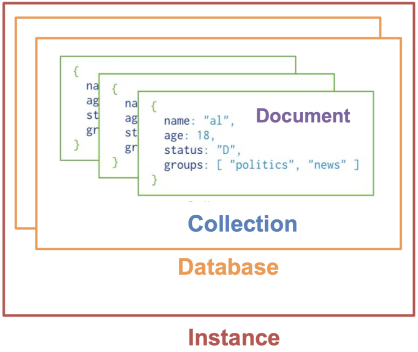
SCIENCE
ACADEMY

MongoDB: Functionalities

- **Design goals**
  - Performance
  - Availability / scalability
  - Rich data storage (not rich querying!)
- **Dynamic schema**
  - No DDL (Data Definition Language)
  - Secondary indexes
  - Query language via an API
- **Several levels of data consistency**
  - E.g., atomic writes and fully-consistent reads (at document level)
- **No joins nor transactions across multiple documents**
  - Makes distributed queries easy and fast
- **High availability through replica sets**
  - E.g., primary replication with automated failover
- **Built-in sharding**
  - Horizontal scaling via automated range-based partitioning of data
  - Reads and writes distributed over shards

MongoDB: Hierarchical Objects

- A Mongo **instance** has:
  - Zero or more "databases"
  - Mongo instance ∼ Postgres instance
- A Mongo **database** has:
  - Zero or more "collections"
    - Mongo collection ∼ Postgres tables
  - Mongo database ∼ Postgres database
- A Mongo **collection** has:
  - Zero or more "documents"
    - Mongo document ∼ Postgres rows
- A Mongo **document** has:
  - One or more "fields"
    - It has always primary key `_id`
    - Mongo field ∼ Postgres columns



```
{
   name: "al",
   age: 18,
   status: "D",
   groups: [ "politics", "news" ]
}
```

ref

Relational DBs vs MongoDB: Concepts

RDBMS Concept	MongoDB Concept	Meaning in MongoDB
database	database	Container for collections
relation / table / view	collection	Group of documents
row / instance	document	Group of fields
column / attribute	field	A name-value pair
index	index	Automatic
primary keys	_id field	Always the primary key
foreign key	reference	Pointers
table joins	embedded documents	Nested name-value pairs

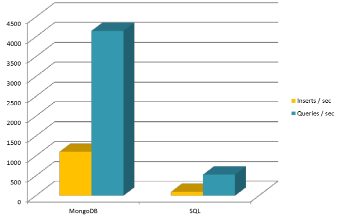
```
{
   "_id" : ObjectId("4d0b6da3bb30773266f39fea"),
   "country" : {
      "$ref" : "countries",
      "$id" : ObjectId("4d0e6074deb8995216a8309e")
   },
   "famous_for" : [
      "beer",
      "food"
   ],
   "last_census" : "Sun Jan 07 2018 00:00:00 GMT -0700 (PDT)",
   "mayor" : {
      "name" : "Ted Wheeler",
      "party" : "D"
   },
   "name" : "Portland",
   "population" : 582000,
   "state" : "OR"
}
```

Relational vs Document DB: Workflows

- Relational DBs
  - E.g., PostgreSQL
  - Know what you want to store
    - Tabular data
  - Do not know how to use it
    - Static schema allows query flexibility (e.g., joins)
  - Complexity is at insertion time
    - Decide how to represent the data (i.e., schema)
- Document DBs
  - E.g., MongoDB
  - No assumptions on what to store
    - E.g., irregular JSON data
  - Know a bit how to access data
    - You want to access the data by key
    - E.g., it's a nested key-value map
  - Complexity is at access time
    - Get the data from the server
    - Process data on the client side

Why Use MongoDB?

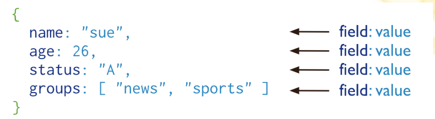
- Simple to query
  - Do the work on client side
- It's fast
  - 2-10x faster than Postgres
- Data model / functionalities suitable for most web applications
  - Semi-structured data
  - Quickly evolving systems
- Easy and fast integration of data
- Not well suited for heavy and complex transactions systems
  - E.g., banking system



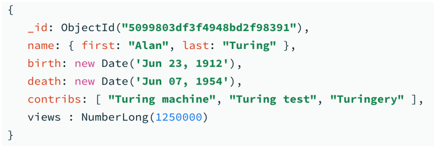
SCIENCE ACADEMY

MongoDB: Data Model

- **Documents** are composed of field and value pairs
  - **Field names** are strings
  - **Values** are any BSON type
    - Arrays of documents
    - Native data types
    - Other documents
- E.g.,
  - `_id` holds an `ObjectId`
  - name holds a document that contains the fields first and last
  - birth and death are of Date type
  - contribs holds an array of strings
  - views holds a value of the NumberLong type



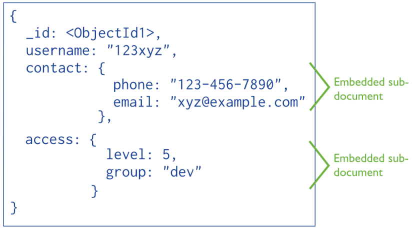
```
{
   name: "sue",
   age: 26,
   status: "A",
   groups: [ "news", "sports" ]
}
```



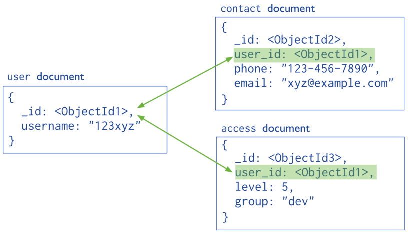
```
{
   _id: ObjectId("5099803df3f4948bd2f98391"),
   name: { first: "Alan", last: "Turing" },
   birth: new Date('Jun 23, 1912'),
   death: new Date('Jun 07, 1954'),
   contribs: [ "Turing machine", "Turing test", "Turingery" ],
   views : NumberLong(1250000)
}
```

MongoDB: Data Model

- Documents can be nested
  - Embedded sub-document
- **Denormalized data models**
  - Store multiple related pieces of information in the same record
  - Conceptually is the result of a join operation
- **Normalized data models**
  - Eliminate duplication
  - Represent many-to-many relationships



```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
           }
}
```



```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

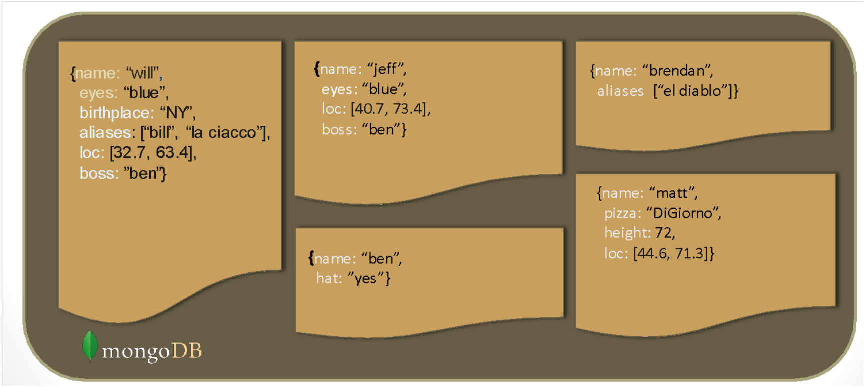
```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

Schema Free

- MongoDB does not need any pre-defined data schema
- Every **document** in a **collection** can have different fields and values
  - No need for `NULL` values / union of fields like in relational DBs
- E.g., dishomogeneous data instances

Document **Document** **Collection**



{name: “will”,
eyes: “blue”,
birthplace: “NY”,
aliases: [“bill”, “la ciacco”],
loc: [32.7, 63.4],
boss: ”ben”}

{name: “jeff”,
eyes: “blue”,
loc: [40.7, 73.4],
boss: “ben”}

{name: “ben”,
hat: ”yes”}

{name: “brendan”,
aliases [“el diablo”]}

{name: “matt”,
pizza: “DiGiorno”,
height: 72,
loc: [44.6, 71.3]}

SCIENCE ACADEMY

JSON Format

- JSON = JavaScript Object Notation
- Data is stored in field / value pairs
- A field / value pair consists of:
  - A field name (always a string)
  - Followed by a colon :
  - Followed by a typed value

```
"name": "R2-D2"
```

- **Data in documents is separated by commas ‘,‘**

```
"name": "R2-D2", race: "Droid"
```

- Curly braces {} hold documents

```
{"name": "R2-D2", race : "Droid", affiliation: "rebels"}
```

- An array is stored in brackets []

```
[{"name": "R2-D2", race: "Droid", affiliation: "rebels"},
 {"name": "Yoda", affiliation: "rebels"}]
```

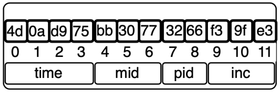
- Supports:

BSON Format

- Binary-encoded serialization of JSON-like documents
  - https://bsonspec.org
- Zero or more key/value pairs are stored as a single entity
  - Each entry consists of:
    - a field name (string)
    - a data type
    - a value
- Similar to Protocol Buffer, but more schema-less
- Large elements in a BSON document are prefixed with a length field to facilitate scanning
- MongoDB understands the internals of BSON objects, even nested ones
  - Can build indexes and match objects against query expressions for BSON keys

ObjectID

- Each JSON data contains an `_id` field of type `ObjectId`
  - Same as a SERIAL constraint incrementing a numeric primary key in PostgreSQL
- An `ObjectId` is 12 bytes, composed of:
  - a timestamp
  - client machine ID
  - client process ID
  - a 3-byte auto-incremented counter
- Each Mongo process can handle its own ID generation without colliding
  - Mongo has a distributed nature
- Details here



Indexes

- **Primary index**
    - Automatically created on the _id field
    - B+ tree indexes
- **Secondary index**
- Users can create secondary indexes to:
    - Improve query performance
    - Enforce unique values for a particular field
- Single field index and compound index (like SQL)
    - Order of the fields in a compound index matters
- Sparse property of an index
    - The index contains only entries for documents that have the indexed field
    - Ignore records that do not have the field defined
- Reject records with duplicate key value if an index is unique and sparse
- Details here

SCIENCE ACADEMY

CRUD Operations

- CRUD = Create, Read, Update, Delete
- **Create**

```
db.collection.insert(<document>)
db.collection.update(<query>, <update>, {upsert: true})
```

  - Upsert = update (if exists) or insert (if it doesn't)
- **Read**

```
db.collection.find(<query>, <projection>)
db.collection.findOne(<query>, <projection>)
```

- **Update**

```
db.collection.update(<query>, <update>, <options>)
```

- **Delete**

```
db.collection.remove(<query>, <justOne>)
```

- Details here

Create Operations

- `db.collection` specifies the collection (like an SQL table) to store the document

  ```
  db.collection.insert(<document>)
  ```

  - Without `_id` field, MongoDB generates a unique key

    ```
    db.parts.insert({type: "screwdriver", quantity: 15})
    ```

  - Use `_id` field if it has a special meaning

    ```
    db.parts.insert({\_id: 10, type: "hammer", quantity: 1})
    ```

- Update 1 or more records in a collection satisfying **query**

  ```
  db.collection.update(<query>, <update>, {upsert: true})
  ```

- Update an existing record or create a new record

  ```
  db.collection.save(<document>)
  ```

- A more modern OOP-like syntax than the COBOL / FORTRAN-inspired SQL

SCIENCE ACADEMY

Read Operations

- `find` provides functionality similar to `SQL SELECT` command

  `db.collection.find(<query>, <projection>).cursor`

  with:

  - = WHERE condition
  - = fields in result set

- `db.parts.find({parts: "hammer"}).limit(5)`
  - Return cursor to handle a result set
  - Can modify the query to impose limits, skips, and sort orders
  - Can specify to return the ‘top’ number of records from the result set

- `db.collection.findOne(<query>, <projection>)`

SCIENCE ACADEMY

More Query Examples

- Mongo has a functional programming flavor
  - E.g., composing operators, like $or

SQL	Mongo
`SELECT * FROM users WHERE age>33`	`db.users.find({age: {$gt: 33}})`
`SELECT * FROM users WHERE age!=33`	`db.users.find({age: {$ne: 33}})`
`SELECT * FROM users WHERE name LIKE "%Joe%"`	`db.users.find({name: /Joe/})`
`SELECT * FROM users WHERE a=1 and b='q'`	`db.users.find({a: 1, b: 'q'})`
`SELECT * FROM users WHERE a=1 or b=2`	`db.users.find({$or: [{a: 1}, {b: 2}]})`

`SELECT * FROM foo`
`WHERE name='bob' and (a=1 or b=2 )`
	`db.foo.find({name: "bob",`
	`$or: [{a: 1}, {b: 2}]})`

`SELECT * FROM users`
`WHERE age>33 AND age<=40`
	`db.users.find({'age':`
	`{$gt: 33, $lte: 40}})`

SCIENCE ACADEMY

Query Operators

Command	Description
$regex	Match by any PCRE-compliant regular expression string (or just use the // delimiters as shown earlier)
$ne	Not equal to
$lt	Less than
$lte	Less than or equal to
$gt	Greater than
$gte	Greater than or equal to
$exists	Check for the existence of a field
$all	Match all elements in an array
$in	Match any elements in an array
$nin	Does not match any elements in an array
$elemMatch	Match all fields in an array of nested documents
$or	or
$nor	Not or
$size	Match array of given size
$mod	Modulus
$type	Match if field is a given datatype
$not	Negate the given operator check

Update Operations

- `db.collection.insert(<document>)`
  - Omit the _id field to have MongoDB generate a unique key
    ```
    db.parts.insert({type: "screwdriver", quantity: 15})
    db.parts.insert({\_id: 10, type: "hammer", quantity: 1})
    ```
- `db.collection.save(<document>)`
  - Updates an existing record or creates a new record
- `db.collection.update(<query>, <update>, {upsert: true})`
  - Will update 1 or more records in a collection satisfying query
- `db.collection.findAndModify(<query>, <sort>, <update>, <new>, <fields>, <upsert>)`
  - Modify existing record(s)
  - Retrieve old or new version of the record

Delete Operations

- `db.collection.remove(<query>, <justone>)`
  - Delete all records from a collection or matching a criterion
  - `<justone>` specifies to delete only 1 record matching the criterion
- Remove all records in `parts` with `type` starting with h
  - `db.parts.remove(type: /h/ )`
- Delete all documents in the `parts` collection
  - `db.parts.remove()`

SCIENCE ACADEMY

MongoDB Features

- Document-oriented NoSQL store
- Rich querying
    - Full index support (primary and secondary)
- Fast in-place updates
- Agile and scalable
    - Replication and high availability
    - Auto-sharding
    - Map-reduce functionality
- Scale horizontally over commodity hardware
    - Horizontally = add more machines
    - Commodity hardware = relatively inexpensive servers

MongoDB vs Relational DBs

- Keep the functionality that works well in RDBMSs
    - Ad-hoc queries
    - Fully featured indexes
    - Secondary indexes
- Do not offer RDBMS functionalities that don't scale up
    - Long running multi-row transactions
    - ACID consistency
    - Joins

MongoDB Tutorial

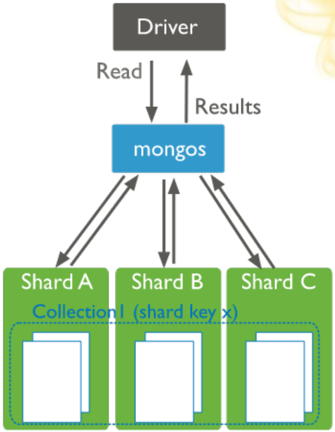
Tutorial is at GitHub The instructions are here:

```
> cd $GIT\_REPO/tutorials/tutorial\_mongodb

> vi tutorial\_mongo.md
```

MongoDB Processes and Configuration

- `mongod`: database instance (i.e., a server process)
- `mongosh`: interactive shell (i.e., a client)
  - Fully functional JavaScript environment for use with a MongoDB
- `mongos`: database router
  - Process all requests
  - Decide how many and which `mongod` instances should receive the query (sharding / partitioning)
  - Collate the results
  - Send result back to the client
- You should have:
  - One `mongos` (router) for the whole system no matter how many `mongods` you have; or
  - One local `mongos` for every client if you wanted to minimize network latency

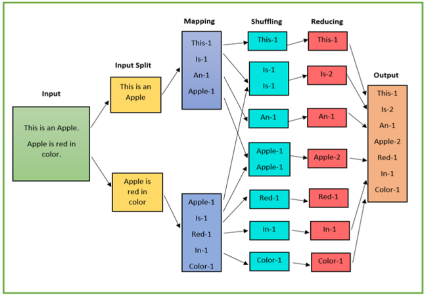


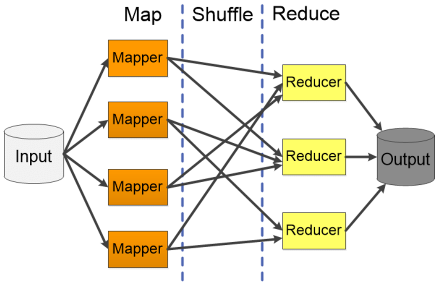
SCIENCE ACADEMY

MapReduce Functionality

- Perform map-reduce computation given a collection of (keys, value) pairs
- Must provide at least a map function, reduction function, and the name of the result set

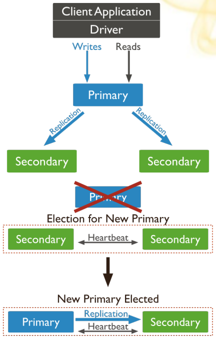
```
db.collection.mapReduce(
  <map\_function>,
  <reduce\_function>,
  {
    out: <collection>,
    query: <document>,
    sort: <document>,
    limit: <number>,
    finalize: <function>,
    scope: <document>,
    jsMode: <boolean>,
    verbose: <boolean>
})
```





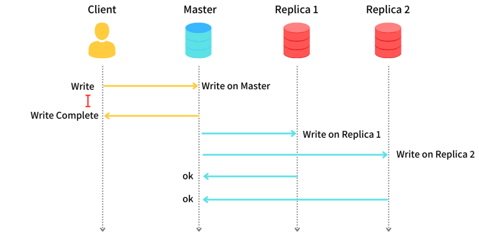
Data Replication

- **Data replication** ensure:
  - Redundancy
  - Backup
  - Automatic failover
- Replication occurs through groups of servers known as **replica sets**
  - **Primary set**: set of servers that client asks direct updates to
  - **Secondary set**: set of servers used for duplication of data
  - Different properties can be associated with a secondary set,
    - E.g., secondary-only, hidden delayed, arbiters, non-voting
- If the primary fails the secondary sets "vote" to elect the new primary set

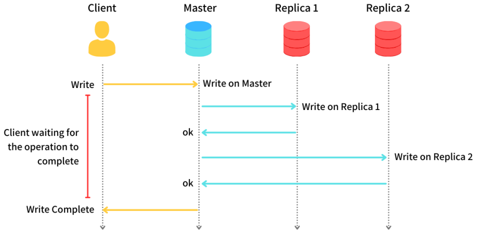


Sync vs Async Replication

- **Synchronous replication**: updates are propagated to other replicas as part of a single transaction
- Implementations
  - 2-Phase Commit (2PC)
  - Paxos
  - Both solutions are complex / expensive
- **Asynchronous replication**
  - The primary node propagates updates to replicas
  - The transaction is completed before replicas are updated (even if there are failures)
  - Commits are quick at cost of consistency



Asynchronous Replication



Synchronous Replication

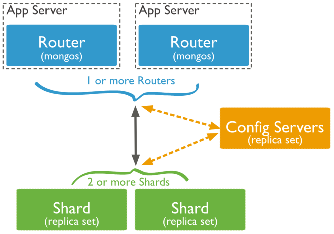
Data Consistency

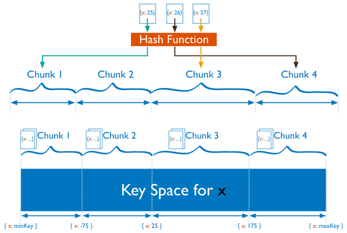
- **Client decides how to enforce consistency for reads**
- Reads to a primary have **strict consistency**
  - Reads reflect the latest changes to the data
  - All writes and *consistent* reads go to the primary
- Reads to a secondary have **eventual consistency**
  - Updates propagate gradually
  - Client may read a previous state of the database
  - All *eventually consistent* reads are distributed among the secondaries

SCIENCE ACADEMY

MongoDB: Sharding

- **Shard** = subset of data
  - A collection is split in pieces based on the shard key
  - Data distributed based on shard key or intervals [a, b)
- **Sharding** = method for distributing data across different machines
- **Horizontal scaling** can be achieved through sharding
  - Divide data and workload over multiple servers
  - Complexity in infrastructure and maintenance
- **mongos** acts as a query router interfacing clients and sharded cluster
  - Each shard can be deployed as a replica set
  - Config servers store metadata and configuration settings for cluster





SCIENCE ACADEMY

RDMBS Internals

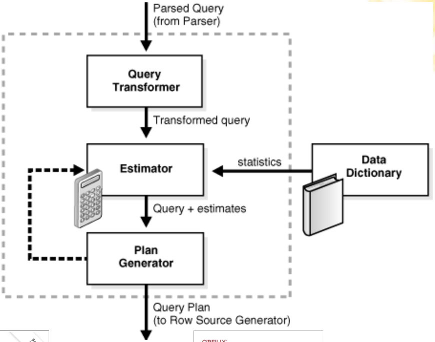
Storage hierarchy - How are tables mapped to files? - How are tuples mapped to disk blocks?

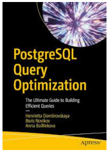
Buffer Manager - Bring pages from disk to memory - Manage the limited memory

Query Processing Engine - Given a user query, decide how to “execute” it - Specify sequence of pages to be brought in memory - Operate upon the tuples to produce results

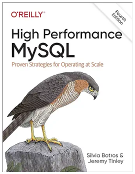
Query Optimizer

- **RDBMSs: query optimizer is static**
  - Assign a cost to each query plan
  - Estimate some cost params (e.g., time to access data)
  - Search for the best query
  - At least traditional RDBMs

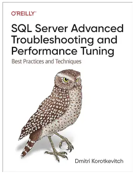




PostgreSQL Query Optimization
The Ultimate Guide to Building Efficient Queries
Henrietta Dombrovskaya
Boris Novikov
Anna Bailliekova
Apress



O'REILLY
Fourth Edition
High Performance MySQL
Proven Strategies for Operating at Scale
Silvia Botros & Jeremy Tinley

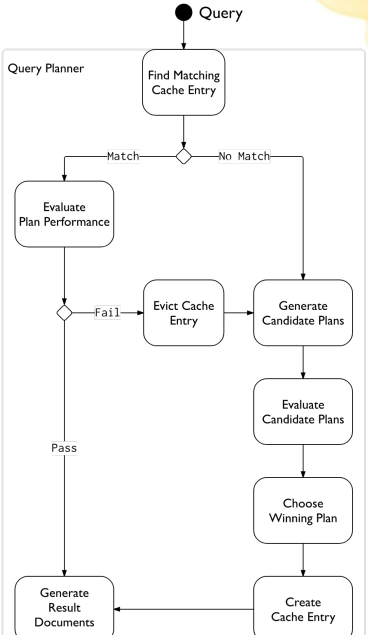


O'REILLY
SQL Server Advanced Troubleshooting and Performance Tuning
Best Practices and Techniques
Dmitri Korotkevitch

SCIENCE ACADEMY

Query Optimizer

- **MongoDB: query optimizer is dynamic**
  - Try different query plans and learn which ones perform well
  - The space of query plans is not so large, because there are no joins
  - When testing new plans
    - Execute multiple query plans in parallel
    - As soon as one plan finishes, terminate the other plans
  - Cache the result
  - If a plan that was working well starts performing poorly try again different plans
    - E.g, data in the DB has changed, parameter values to a query are different



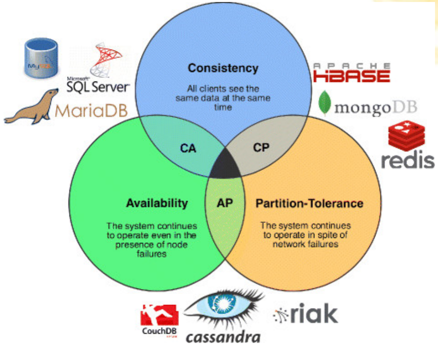
SCIENCE ACADEMY

MongoDB: Strengths

- Provide a flexible and modern query language
- High-performance
  - Implemented in C++
- Very rapid development, open source
  - Support for many platforms
  - Many language drivers
- Built to address a distributed database system
  - Sharding
  - Replica sets of data
- Tunable consistency
- Useful for working with a huge quantity of data not requiring a relational model
  - The relationships between the elements does not matter
  - What matters is the ability to store and retrieve great quantities of data

MongoDB: Limitations

- No referential integrity
  - Aka foreign key constraint
- Lack of transactions and joins
- High degree of denormalization
  - Need to update data in many places instead of one
- Lack of predefined schema is a double- edged sword
  - You must have a data model in your application
  - Objects within a collection can be completely inconsistent in their fields
- CAP Theorem: targets consistency and partition tolerance, giving up on availability

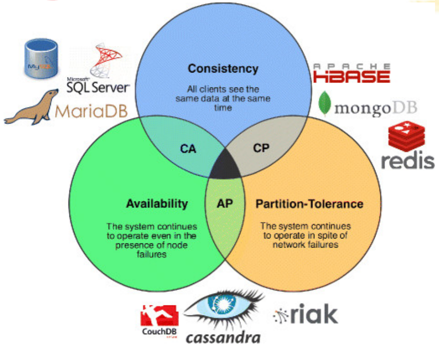


Consistency
All clients see the same data at the same time
CA
CP
Availability
The system continues to operate even in the presence of node failures
AP
Partition-Tolerance
The system continues to operate in spite of network failures
SQL Server
MariaDB
APACHE HBASE
mongoDB
redis
CouchDB
cassandra
riak

- ***Couchbase***

Couchbase

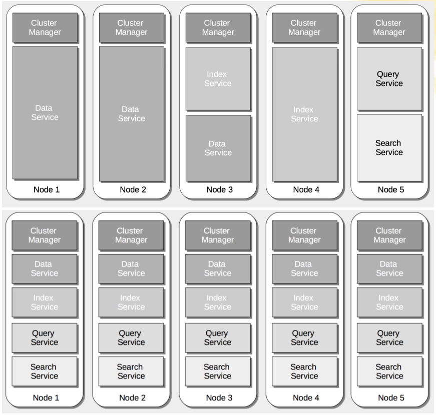
- NoSQL document-oriented DB (like MongoDB)
- Couchbase = merge of CouchDB and membase
  - *CouchDB*
    - Open source document store
    - HTTP RESTful API to add, update, delete documents
    - Support all 4 ACID properties
  - *membase*
    - Distributed key-value store (like Redis)
    - Designed to scale both up and down
    - Highly available and partition tolerant
  - Uses HTTP protocol to query and interact with objects in the DB
    - No query language
  - Objects stored in *buckets*
    - Collection of JSON docs, with no special relation to



Consistency
All clients see the same data at the same time
Availability
The system continues to operate even in the presence of node failures
Partition-Tolerance
The system continues to operate in spite of network failures
CA
CP
AP
MySQL
Microsoft SQL Server
MariaDB
APACHE HBASE
mongoDB
redis
CouchDB
cassandra
riak

Architecture

- Every Couchbase node consists of **different services**:
    - Data service
    - Index service
    - Query service
    - Cluster manager component
- Services can run on separate nodes of the cluster, if needed
- **Data replication**
    - Across nodes of a cluster
    - Across data centers
- **Data service**
    - Writes data *asynchronously* to disk after acknowledging to the client
    - Optionally *synchronous:* ensure data is written to more than one server before acknowledging a write



SCIENCE ACADEMY

Queries

- **Can create multiple views over documents**
  - Views are optimized / indexed by Couchbase for fast queries
  - Re-indexed when underlying documents changes
  - Can do full-text searches using the indexes
- **Perform well when:**
  - There are infrequent changes to the structure of documents
  - Know in advance what kinds of queries you want to execute
- **Query**
  - Uses a custom query language called N1QL (“nickel”)
  - Extends SQL to JSON documents
  - Queries over multiple documents using (server-side) joins
- **Map-reduce support**
  - (Map) First define a view with the columns of the document your are interested in
  - (Reduce) Optionally define aggregate functions over the data