



# Ground Up

Syed Awase

Syed Awase earned his PhD from University of Zurich in GIS, supported by EU V Framework Scholarship from SPIRIT Project ([www.geo-spirit.org](http://www.geo-spirit.org)). He currently provides consulting services through his startup [www.territorialprescience.com](http://www.territorialprescience.com) and [www.sycliq.com](http://www.sycliq.com)



SYED AWASE KHIRNI

# 0. NO SQL DATA STORES

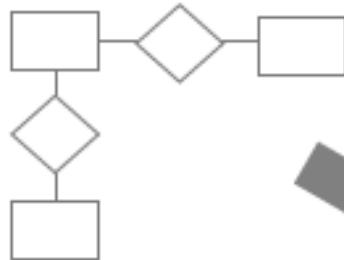


# Outline

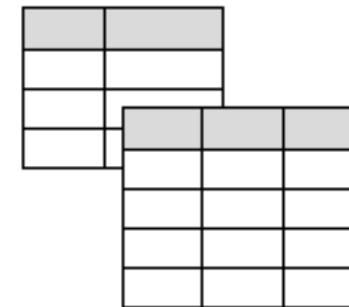
- Evolution and History of Database Management Systems
- Outline of various database types.
- Traditional Scalability
- Common Problems
- Non-traditional Databases
- Non-traditional DB characteristics
- Why NoSQL?
  - Design Goals
- NoSQL vs RDBMS
- Choosing a NoSQL Solution
- Categories of NoSQL Database

# Database Design

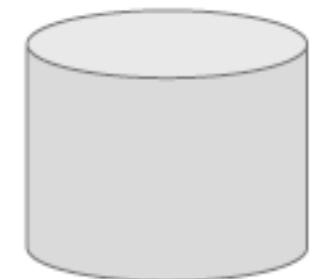
## Conceptual Design



## Implementation Design



## Physical Design



# Database Management Systems

Conceptual modelling  
Data access and representation

**Client Interface Layer**

E/R  
SQL, JDBC, ODBC

Data semantics  
Operation semantics

**Data Model Layer**

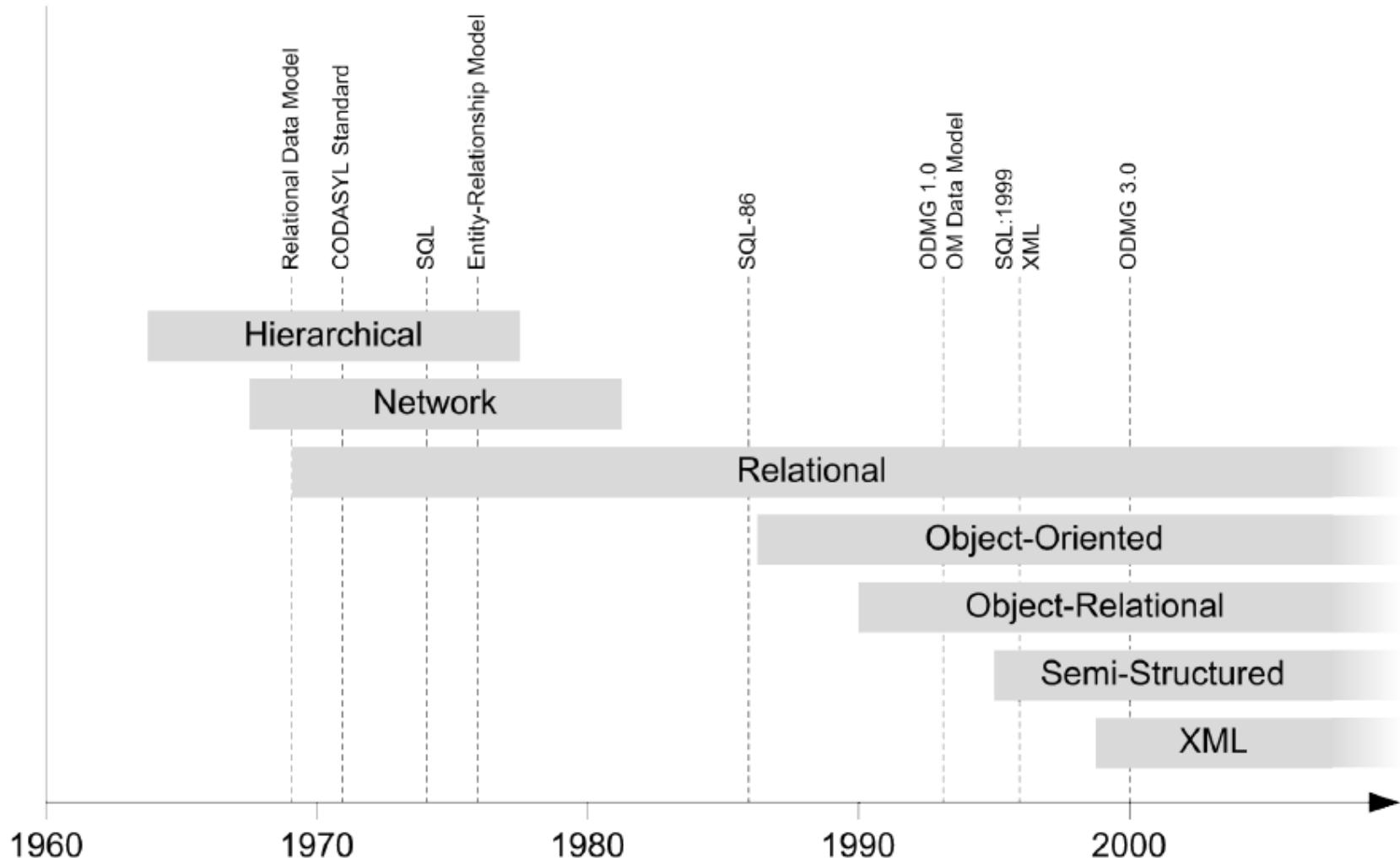
Relational Model

Persistence  
ACID  
Distribution

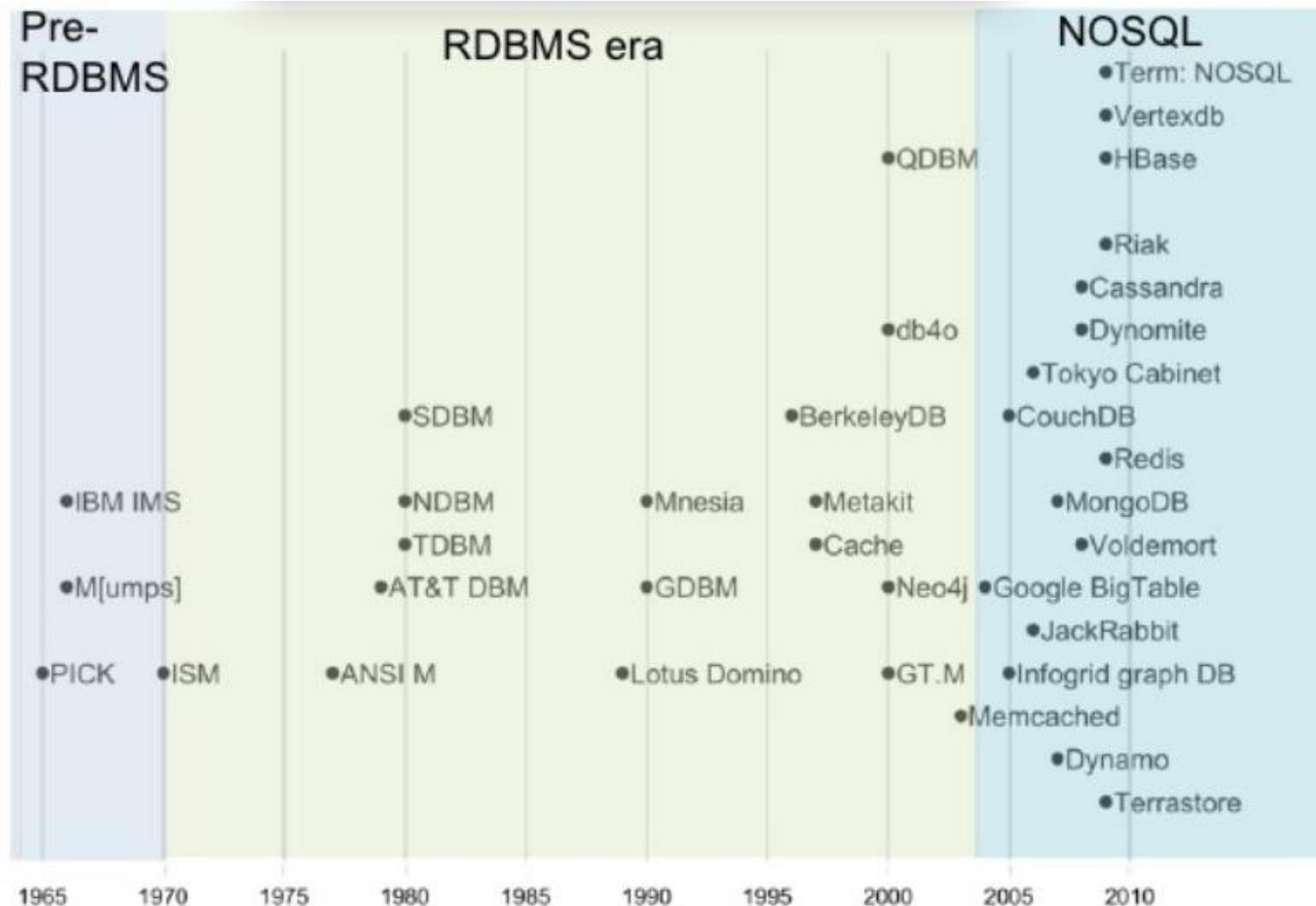
**Storage Layer**

RDBMS

# Evolution and History

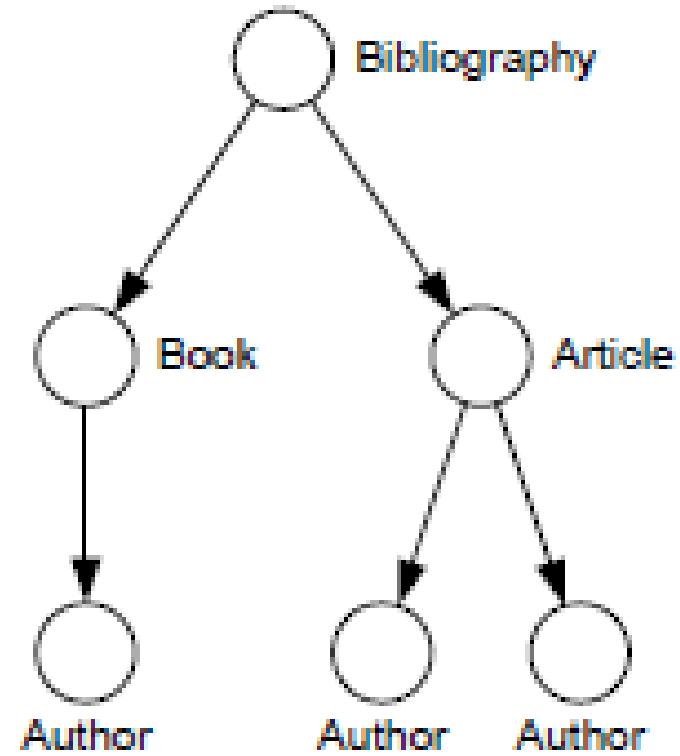


# NoSQL past and Present



# Hierarchical Databases

- Data organized in a tree
  - A parent can have many children
  - A child can have only one parent
- Records described by entity type
- 1:N (one-to-many) relationships
- Query by path navigation
- Examples
  - File system
  - LDAP
  - Windows Registry and Active Directory
  - XML documents and Xquery.

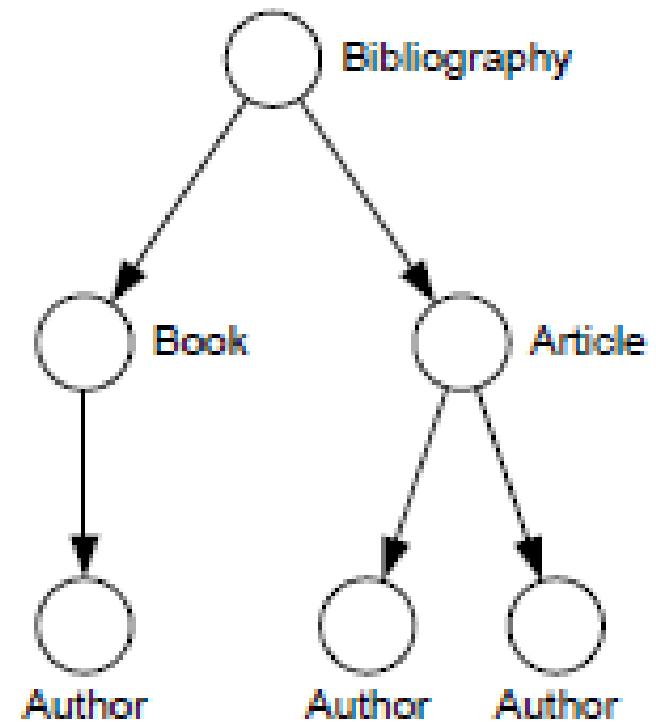


# Network Databases

- Data organized in graph(lattice)

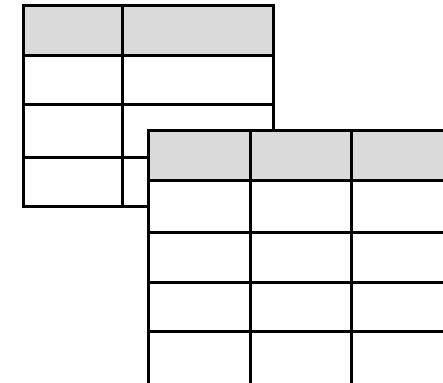
  - A parent can have many children
  - A child can have many parents

- Bachmann diagrams
- Record types define properties
- Set types defined relationships
  - Parent-child, (double) linked list..
- Query by graph navigation
- Examples
  - CODASYL



# Relational Databases

- Data organized as tuples in relations
- Link between data tuples
  - Primary and foreign keys
- Relational algebra
  - Project, select, join
- Relational normal forms
- Declarative language
  - Data definition, consistency, manipulation and querying
- Examples
  - Oracle, Microsoft SQL Server, IBM DB2
  - PostgreSQL, MySQL



# Relational Databases

- Relational model is very simple
  - Only basic concepts – references need to be simulated
  - Restricted type system – no user-defined types
- Lack of semantic modeling
  - Complex data, versioning, roles
- Little support of data and schema evolution
- Object-relational impedance mismatch.

# RDBMS Problem

- Design for ACID
  - Atomicity, Consistency, Isolation and Durability
- Hard to Scale
- Availability?
- Flexibility



# Traditional Scalability

- Scale-up
  - Memory and hardware has limitations
- Scale-out
  - Scaling reads
    - Cache is the King
      - Query cache, Memcache, OLAP
    - Prefetching
    - Replication
- Scaling Writes
  - Redundant disk arrays, RAID
  - Sharding

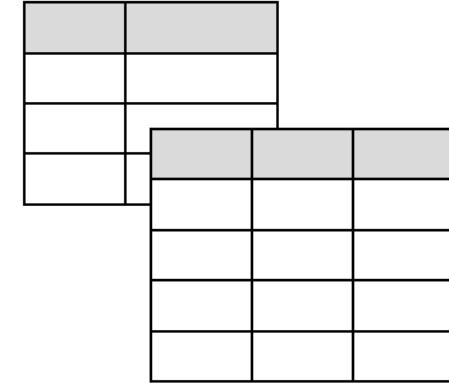
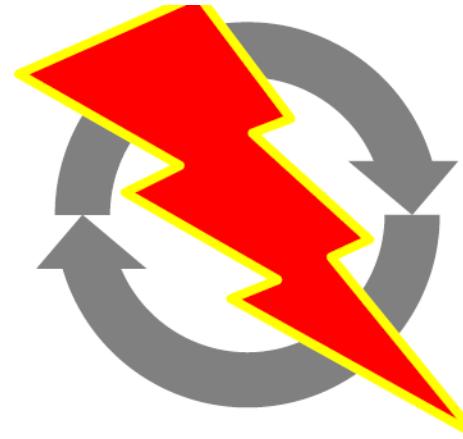
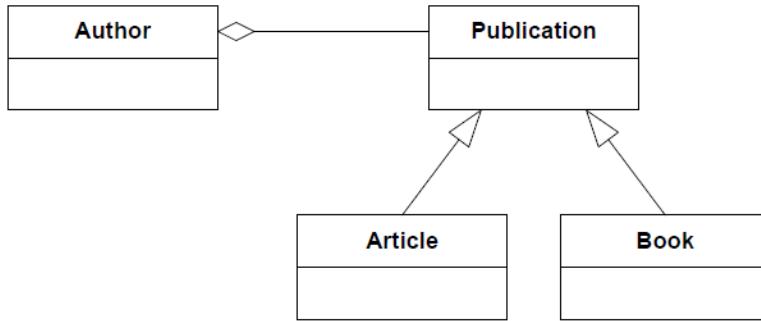
# Common Problems

- Relation model is heavy
  - Parsing
  - Locking
  - Logging
  - Buffer pool and thread
- Not every case can work within a single node SMP
- Sharding does not solve all problems
  - Cross shard or join between shards
  - Need to update across multiple shards within a transaction.
  - Shard failure
  - Online Schema changes without taking the shard offline
  - Add or replace shards in-line.

# Lack of Semantic Modeling

- All advantages of the object-oriented paradigm which enable to produce models of an application domain that capture more of its aspects, e.g. using
  - Inheritance
  - Relationships
- Inheritance and relationships can be mapped to the relational model, however, they result in a complex database schema. Such complex schemata are difficult to develop, maintain and decrease runtime performance.

# Object-Relational Impedance Mismatch



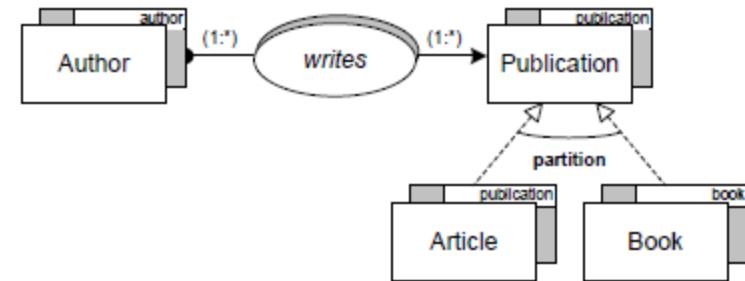
- Object-oriented application development and relational data management results in clash of two incompatible models.
- Code to map between models is considerable overhead, costly and hard to maintain.

# Database Evolution

- Object-relational mapping (ORM) layers don't really help
  - Writing manual code for ORM poses problem
  - Evolution code can usually not be generated automatically
- Automatic evolution is not possible for complex cases
  - Semantic changes
  - Certain type changes ( String to Person)
  - Ambiguity of attribute add-remove vs. rename
  - Class hierarchy changes

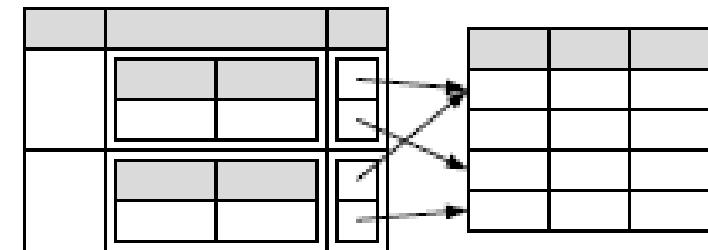
# Object-Oriented Databases

- Data represented as objects
  - Object Identity
  - Attributes and methods
  - Reference, relationships, associations
- Extensible type hierarchy
  - User-defined types, abstract data types
  - single or multiple inheritance
  - Overloading, overriding, late binding
- Declarative language for ad hoc purposes
- Binding for Object-oriented programming language



# Object-Relational Database

- Relational model extended
  - Nested relations
  - References
  - Sets
  - Row types, abstract types
  - Functions
- Declarative language extended
  - Computationally complete
- Fundamental impedance mismatch remains



# Non-traditional Databases

- XML Database
- Graph Databases
- Document Databases
- Key-value Stores
- Column Store
- Others
  - Geospatial
  - File System

# NoSQL

**Non-Relational, Web-scale database**

First used by Carlo Strozzi in 1998

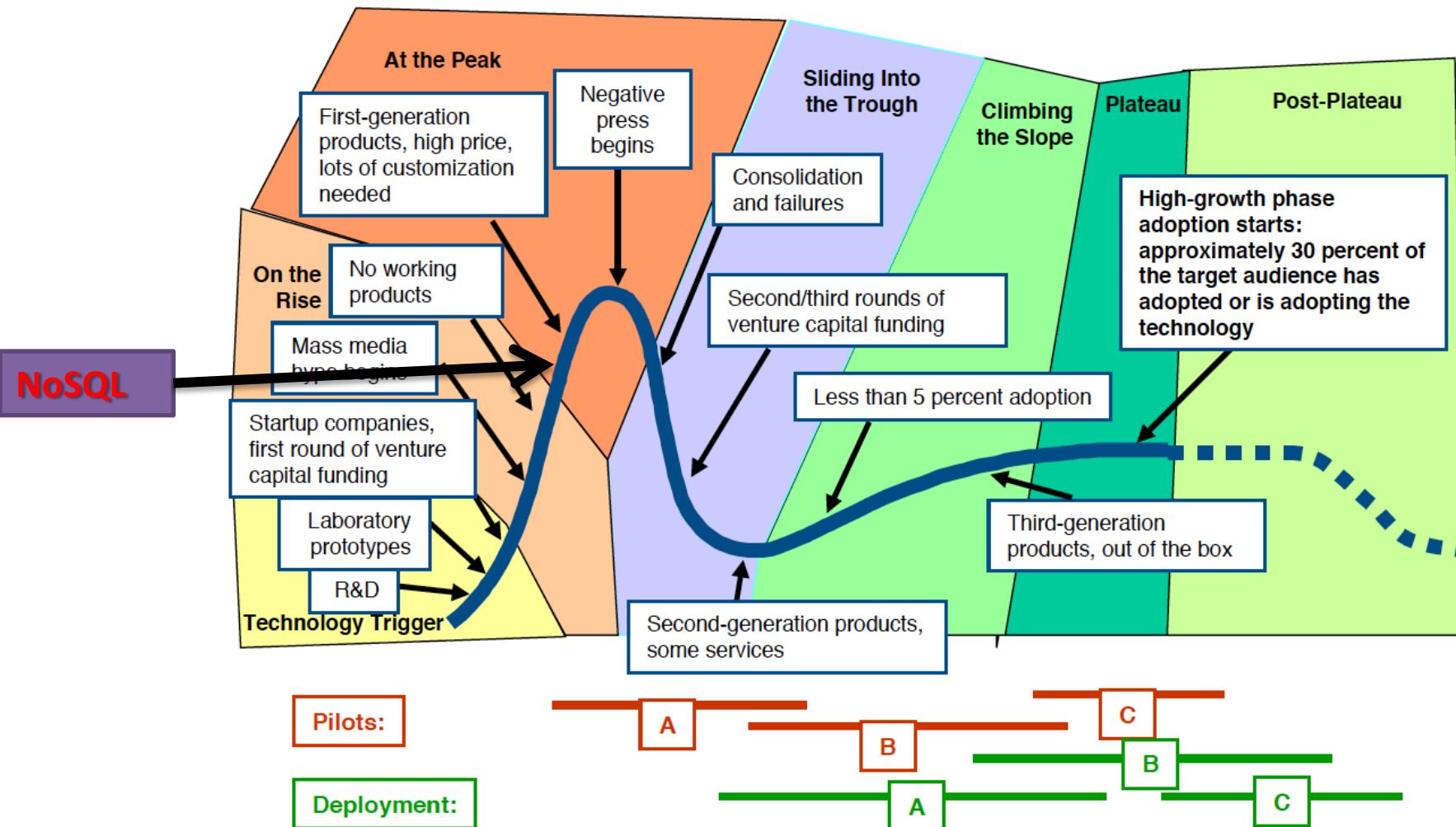
Reintroduced by Eric Evans in 2009



# Non-traditional Database Characteristics

- Non-relational in Nature
- Distributed
- Horizontal Scalability
- Schema-less/Schema-free
- Eventual Consistency

# Gartner Hype Cycle



# NO-SQL

Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontal scalable.. Schema-free, easy replication support, simple API, eventually consistent”

- **Non-Relational**
- **Distributed**
- **Open-Source**
- **Horizontally Scalable**
- **Schema-Free**
- **Replication Support**
- **Eventually Consistent**

## Classification -nosql-database.org

- Wide Column Store/Column Families
- Document Store
- Key value/Tuple Store
- Graph Database
- Multimodel Database
- Object Database
- Grid and Cloud Database Solutions
- XML Databases
- Multidimensional Databases
- Multivalue databases
- Time Series/Streaming Databases
- Scientific and Specialized DB



# Horizontal Scaling

- Scale by adding more machines into your pool of resources
- Often based on partitioning of data i.e. Each node contains only part of the data.
- Easier to scale dynamically by adding more machines into the existing pool.
- Through **partitioning**
- **Increase the capacity on the fly, using clusters to improve performance and provide high availability (HA)**
- **No single point of failure and fault tolerant**
- **E.g: Apache Cassandra, MongoDB, CouchBase**

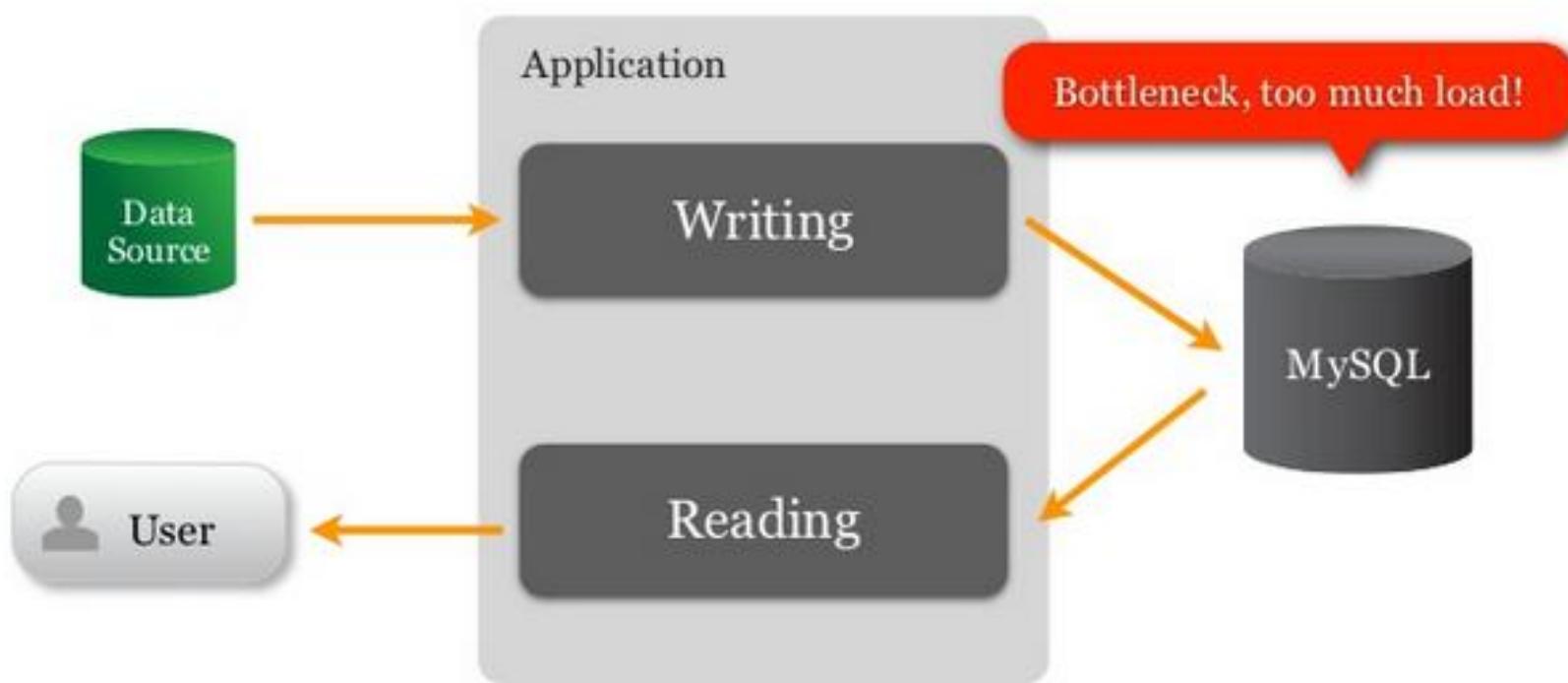
# Vertical Scaling

- Scale by adding more power (CPU, RAM) to your existing machine.
- The data resides on a single node and scaling is done through multi-core i.e. Spreading the load between the CPU and RAM resources of that machine.
- Limited to the capacity of a single machine, scaling beyond that capacity often involves downtime and comes with an upper limit
- Through **multi-core support**
- **Single point of failure**
- **Additional architecture choice : SQL-based database services that enable horizontal scaling without the complexity of manual sharding.**  
**Scale-out similar to NoSQL engines.**
- **E.g: Amazon RDS**

# Why NoSQL?

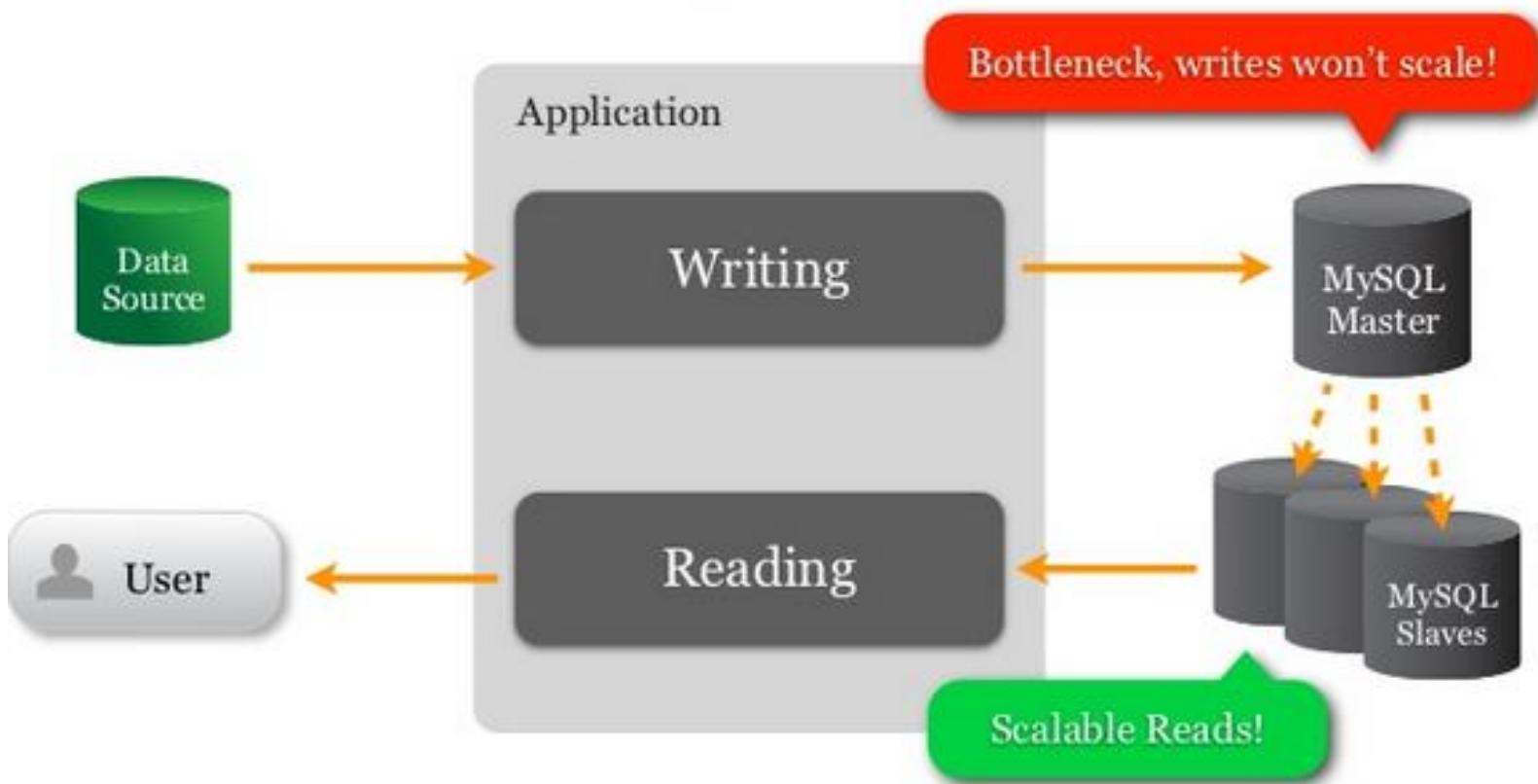
## The MySQL Problem

### 1. Default



# Why NoSQL?

## The MySQL Problem 2. Replication

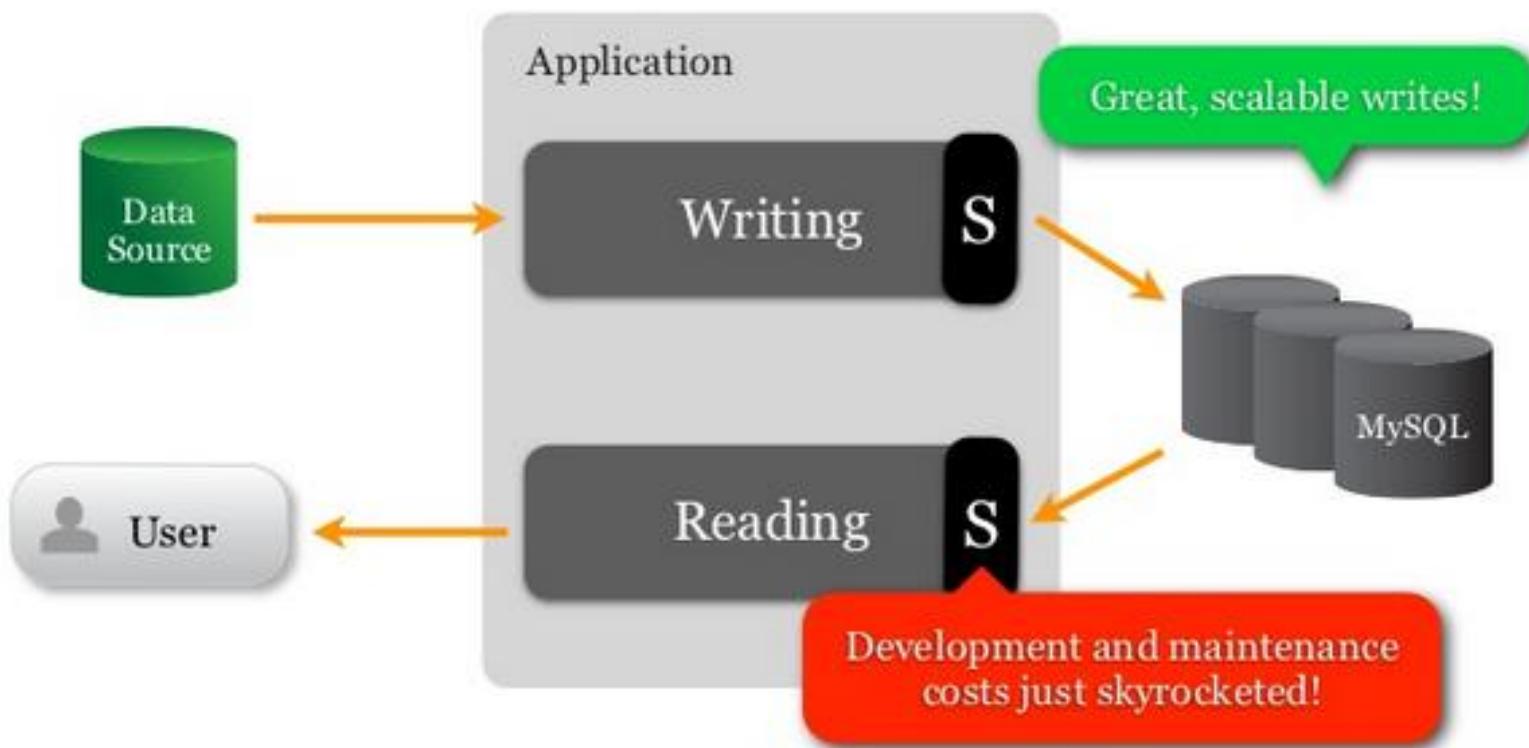


Sharding: A database shard is a horizontal partition in a database or search engine. Each individual partition is referred to as a shard or database shard.

# Why NoSQL?

## The MySQL Problem

### 3. Sharding



# Why NoSQL?

## DESIGN GOALS

- Semi Structured >>
- Big Data >>
- Real-time >>
- Ubiquity >>
- Schema-Free
- Scalable reads/writes
  - Horizontal Scalability
    - To increase the no-of machines but maintaining proportional performance.
  - Vertical Scalability
    - To add more resources to your single machine to optimize performance.
- High-performance
- High-availability
- Reliability, Fault tolerant.

# NoSQL vs RDBMS

## NoSQL

- Schema-free
- Scalable writes/reads
- Auto high-availability
- Limited queries
- Eventual Consistency – applies to most NoSQL systems
- BASE :  
Basic Availability, Soft state, Eventual consistency

## RDBMS

- Relational Schema
- Scalable reads
- Custom high-availability
- Flexible
- Consistent
- ACID : Atomic, Consistent, Isolated and Durable.

# RDBMS vs NoSQL

- Traditional distributed data management solutions focus on ACID semantics
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- Modern Web-scale data management focuses on
  - Basic Availability
  - Soft-state
  - Eventual Consistent
  - CAP (Consistency, Availability, Partition and Tolerance)

# ACID

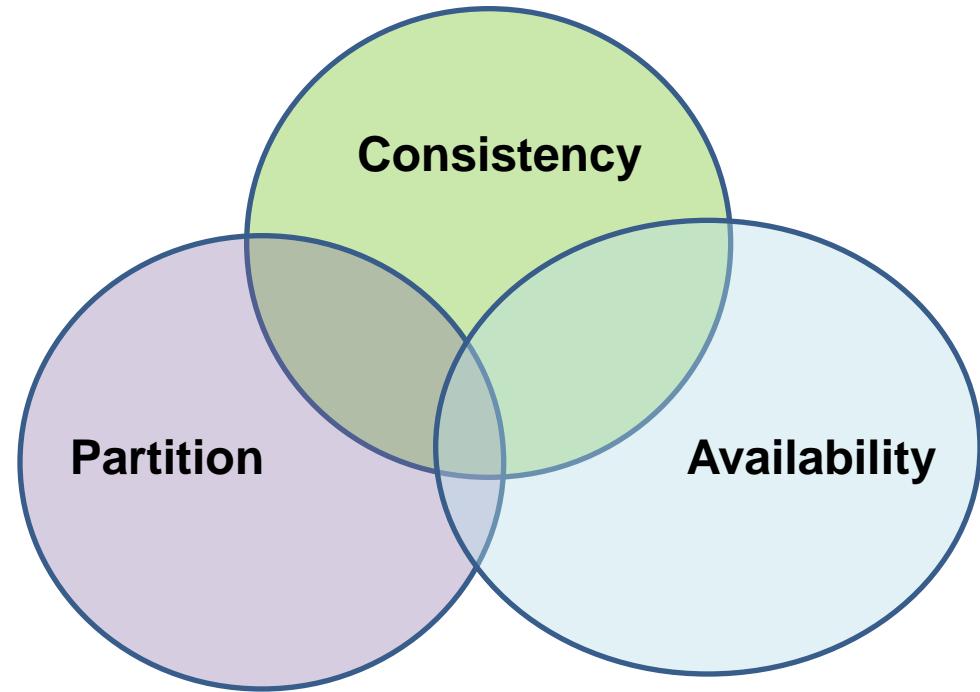
- Atomicity : All of the operations in the transaction will complete or none will
- Consistency: the database will be in a consistent state when the transaction begins and ends.
- Isolation: the transaction will behave as if it is the only operation being performed upon the database. (No interference of transaction).
- Durability : Upon completion of the transaction, the operation will not be reversed.

# BASE

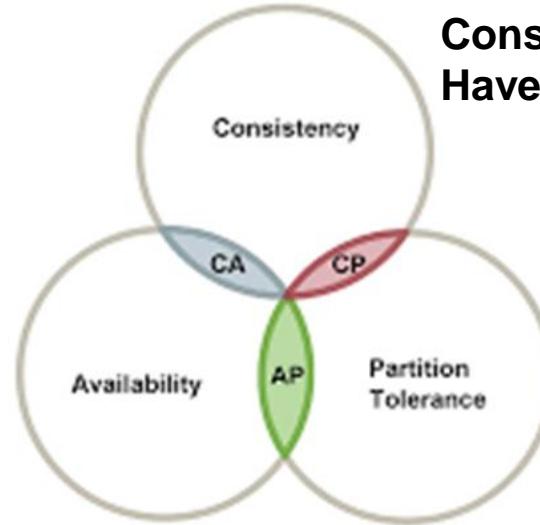
- **BASIC AVAILABILITY**
  - The database appears to work most of the time
- Soft-state
  - Stores don't have to be write-consistent, nor do different replicas have to be mutually consistent all the time
- Eventual consistency
  - Stores exhibit consistency at some later point (e.g. Lazily at read time).

# CAP Theorem/Brewer's Theorem

- It states that it is impossible for a distributed computer system to simultaneously provide all three of the following guarantees i.e.
  - Consistency
  - Availability
  - Partition-Tolerance
- **You can achieve only 2 of them, but you cannot do all 3.**



**Availability:** Each client can  
Can always read and write



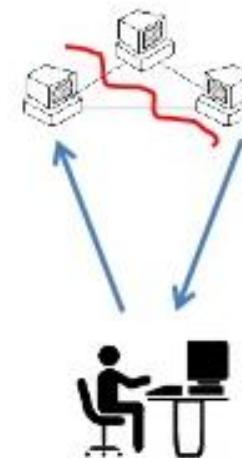
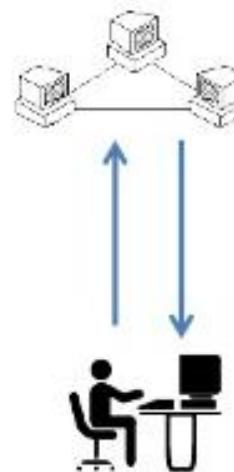
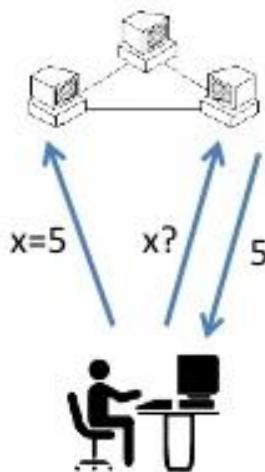
**Consistency:** All clients always  
Have the same view of the data

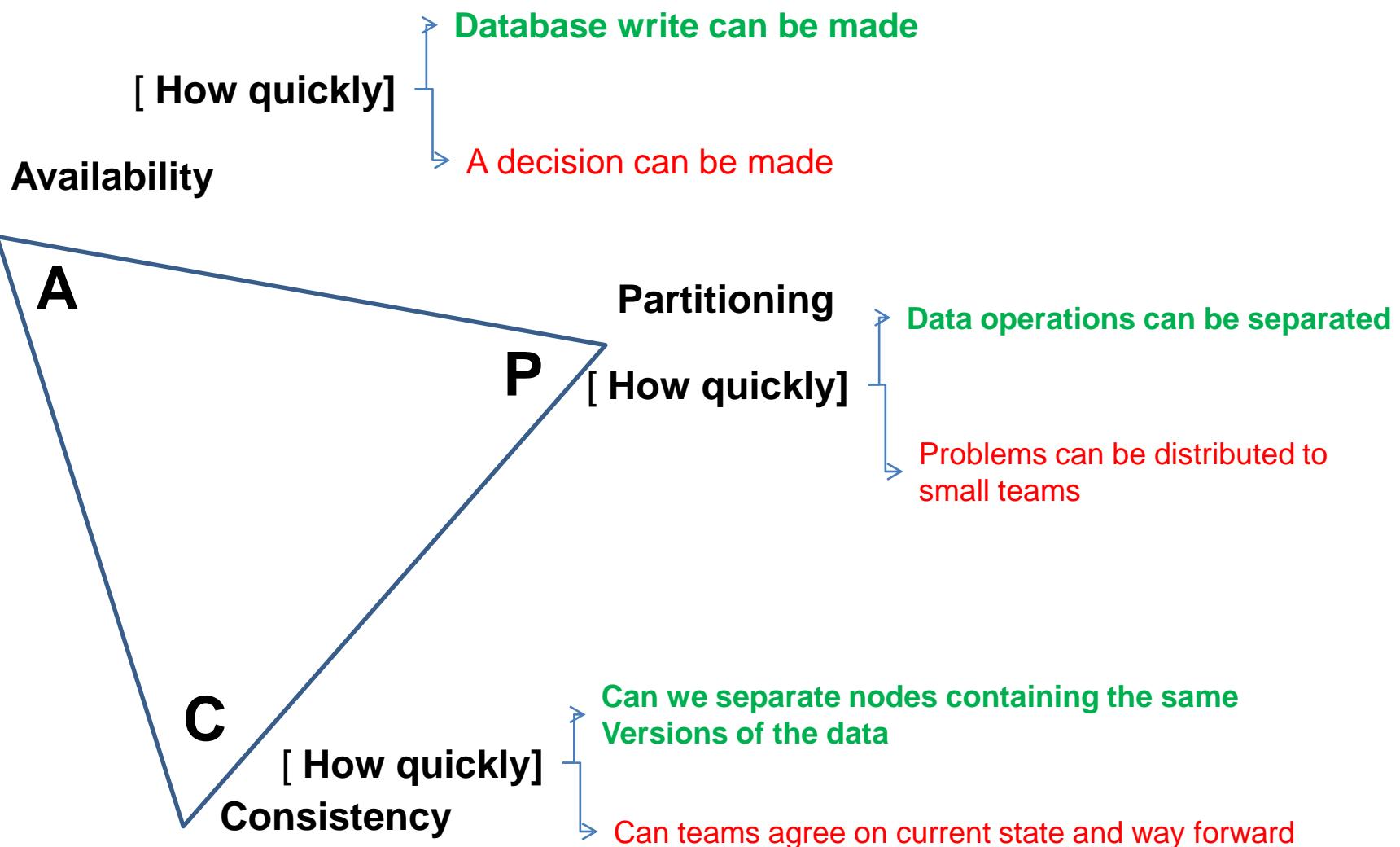
**Partition:** System works well  
despite Network partitions or  
any network failure

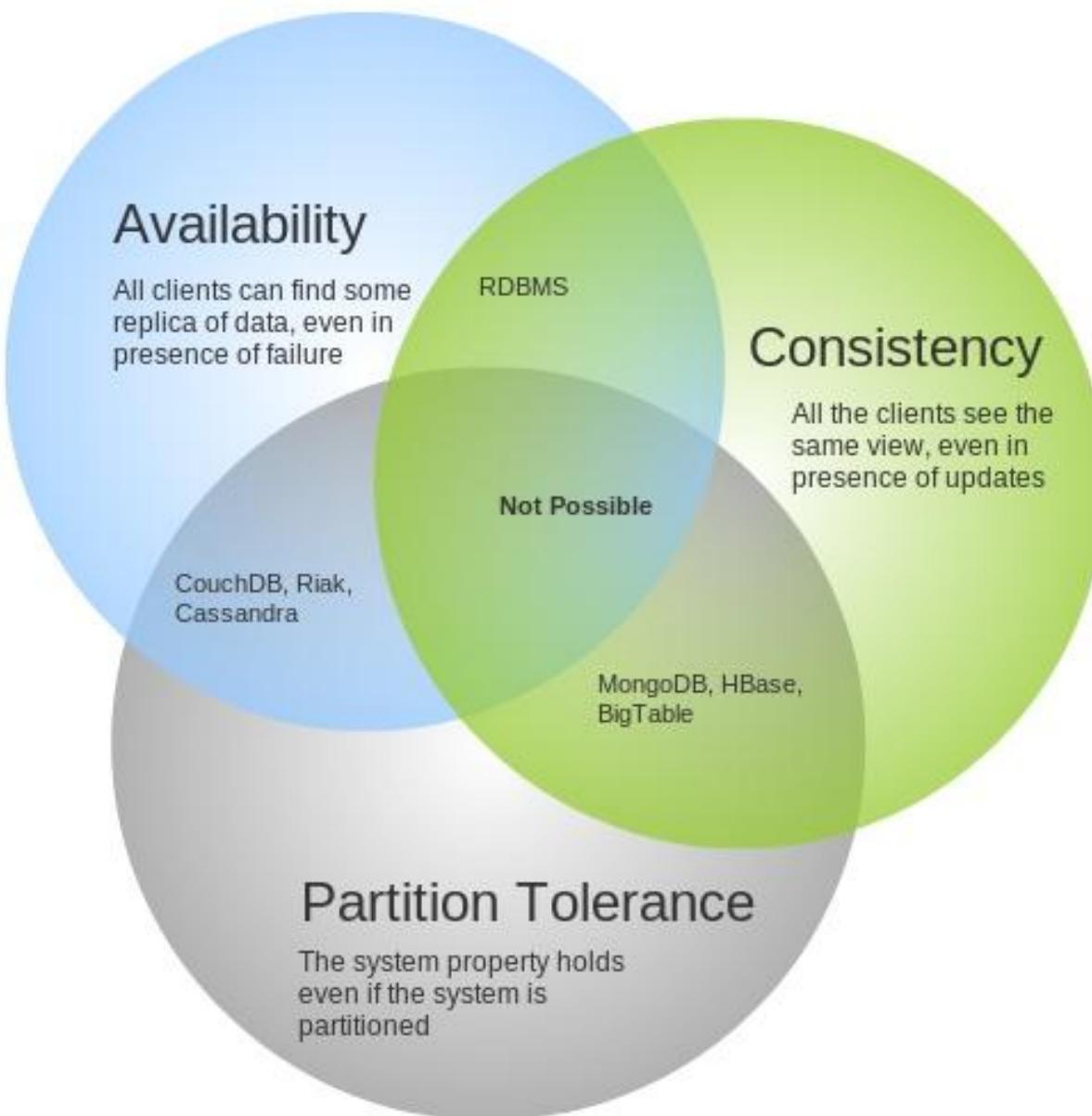
Consistency

Availability

Partition tolerance

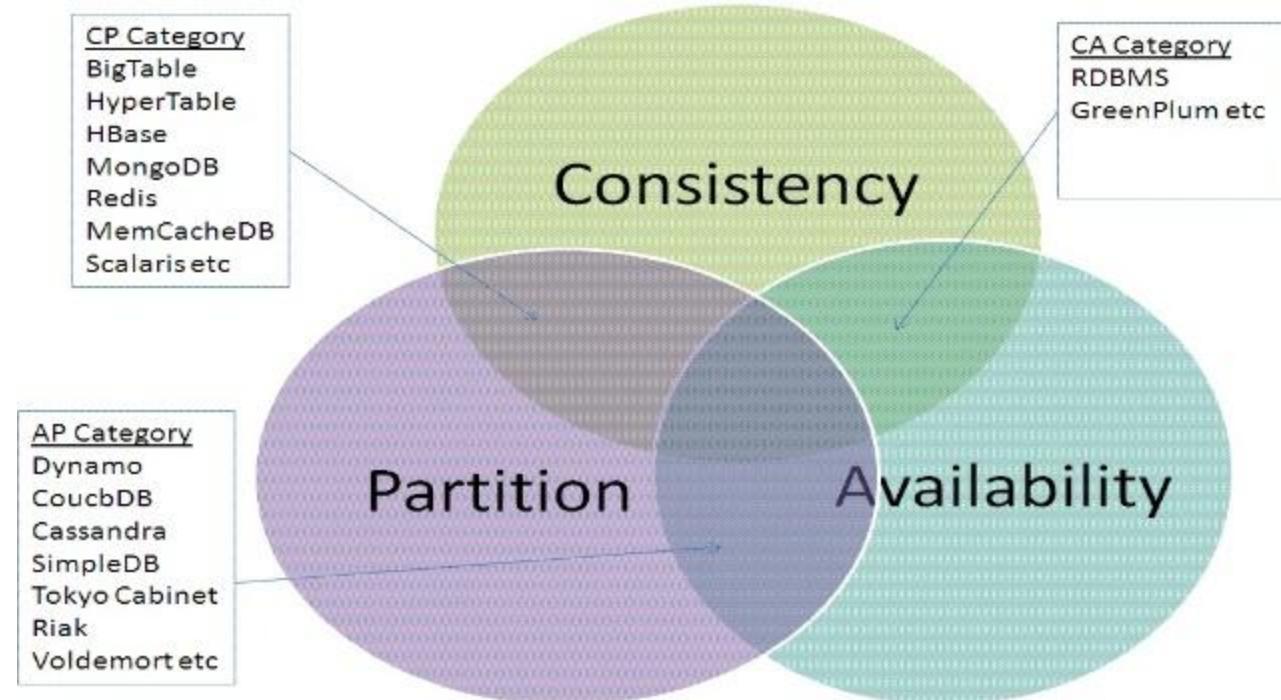






# CAP

- Consistency: No contradiction b/w data.
- Availability: Every operation must terminate in an intended response.
- Partition tolerance: Operations will complete, even If individual components are unavailable.
- NoSQL are based on CAP.



# ACID vs BASE

## ACID

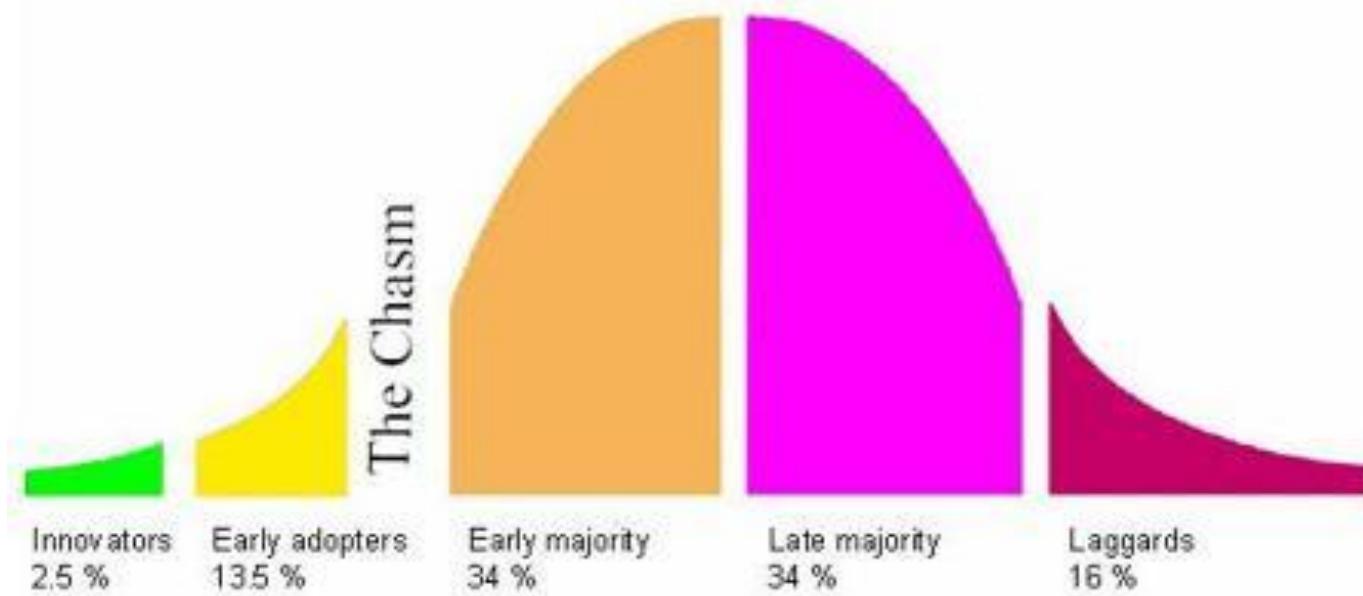
- Strong consistency
- Isolation
- Focus on “commit”
- Nested Transactions
- Availability?
- Pessimistic/Conservative
- Difficult evolution (e.g. schema)
- Provides vertical scaling
- Expensive joins and relationships
- High maintenance costs

## BASE

- Weak consistency
- Availability first
- Best effort
- Approximate answers (OK)
- Optimistic (aggressive)
- Fast and simple
- Easier evolution
- Provides horizontal scaling
- Low maintenance cost
- Free from joins and relationships

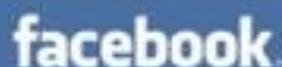
# Who's using NoSQL?

Roger's Innovation Adoption Curve



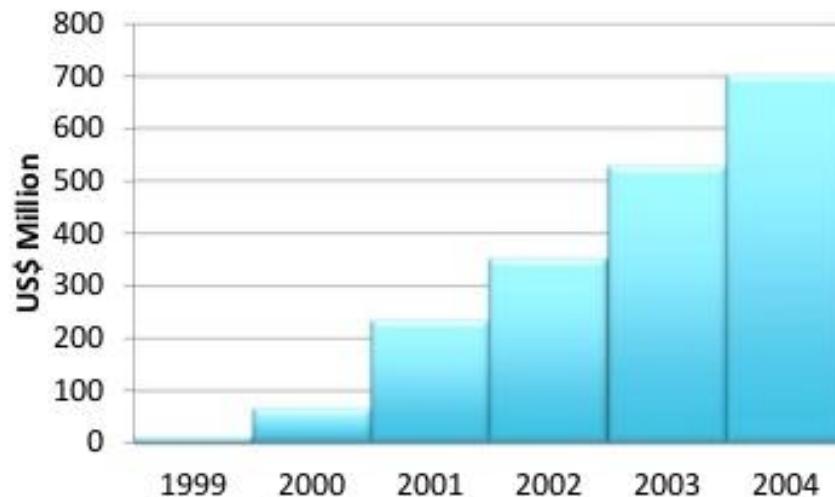
Trying to convince the mass of a new idea is *useless*.  
Convince *innovators and early adopters* first.

# Who's using NoSQL?

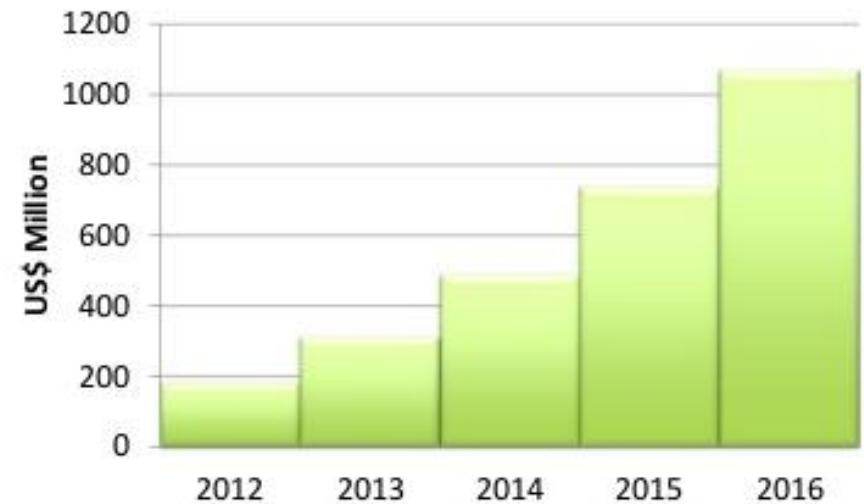


# Growth

**XML Databases Predicted Growth**



**NoSQL Databases Predicted Growth**



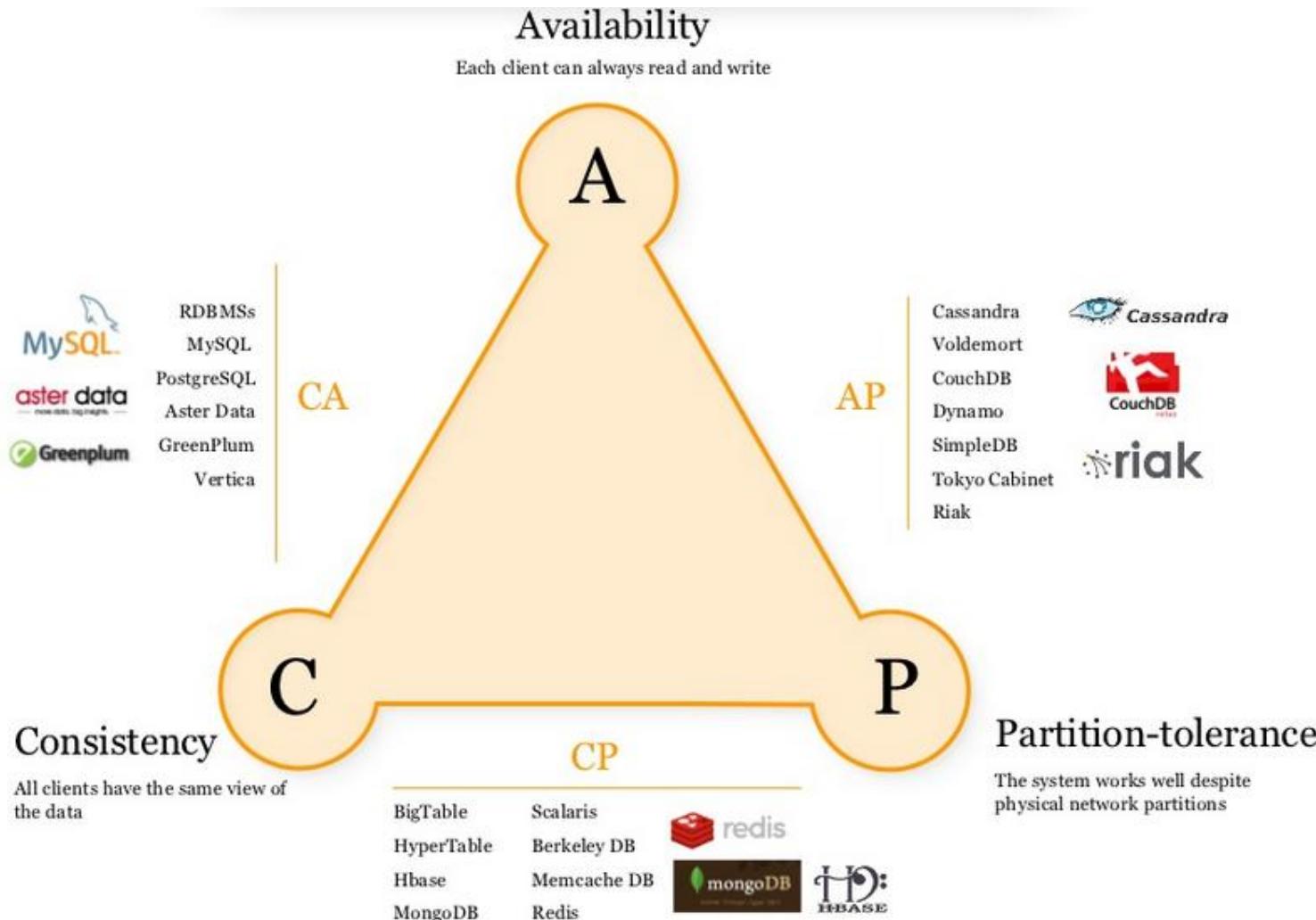
# NoSQL Use Cases

- Consumer Use Cases
- Facebook
- Twitter
- NetFlix
- Enterprise Use Cases
- Rackspace
- TrendMicro
- NewsCred

## Facebook

- Hbase – Facebook messages
- Scribe- Real-time click logs
- Hive- SQL Queries -> MapReduce jobs
- Hadoop
  - Web analytics warehouse
  - Distributed datastore
  - MySQL backups

# Choosing a NoSQL Solution



# Consistent, Available (CA)

- CA- systems have trouble with partitions and deal with it with replication
- Examples
  - MySQL (relational)
  - AsterData (relational)
  - Greenplum(relational)
  - Vertica (Column)

# Availability, Partition-Tolerant (AP)

- AP-systems have trouble with consistency, achieve “eventual consistency” through replication.
- Examples
  - Cassandra(column/tabular)
  - Dynamo (Key-value)
  - Voldemort (Key-value)
  - Tokyo Cabinet (Key-value)
  - CouchDB(document)
  - SimpleDB(document)
  - Riak(document)

# Consistent, Partition-Tolerant (CP)

- CP-systems have trouble with availability while keep data consistent across partitioned nodes.
- Examples
  - **MongoDB(document)**
  - BigTable(Column/tabular)
  - HyperTable(Column/tabular)
  - Hbase (Column/tabular)
  - Redis(key-value)
  - Scalaris(key-value)
  - MemcachedDB (Key-value)

# Categories of NoSQL databases

## 1) Key Value stores

- Don't have any schema
- Fast lookups facility
- Can be used in a forum software where user's statistic and messages are recorded. User's id will serve as a key and will retrieve a string that represents all the relevant info of the user. And a background process recalculates the information and writes to the store independently after fixed interval of time.
- Example
  - ❖ Redis : Redis is an open source, advanced key-value store. It is often referred to as a data structure server since keys (data types) can contain strings, hashes, lists, sets and sorted sets.

# Categories (cont...)

- ❖ API: Tons of languages, Written in: C,
- ❖ Concurrency: in memory and saves asynchronous disk after a defined time.

## 2) Document databases

- ❖ web application
- ❖ tolerance of incomplete data
- ❖ low query performance
- ❖ no standard query syntax
- ❖ Used when we don't have complete data about all the entities of database but we still need to create database.

Example

**CouchDB:** API: JSON, Protocol: REST, Query Method: MapReduceR of JavaScript Funcs, Replication: Master Master, Written in: Erlang.

# Categories (cont...)

- ❖ **MongoDB:** API: BSON, Protocol: lots of langs, Query Method: dynamic object-based language, Replication: Master Slave, Written in: C++.

## 3) Graph databases

- ❖ Social networking
- ❖ graph algorithms, connectedness, degree of relationships
- ❖ has to traverse the entire graph to get definitive answer.
- ❖ not easy to cluster.
- ❖ used in a situation where we want to analyze the on going trends and take decision on the basis of those trends.

# Categories (cont...)

- ❖ Example:

**Neo4J:** API: lots of langs, Protocol: Java embedded / REST, Query Method: SparQL, nativeJavaAPI, JRuby, Replication: typical MySQL style master/slave, Written in: Java, Concurrency: non-block reads, writes locks involved nodes/relationships until commit, Misc: ACID possible

## 4) XML databases

- ❖ Publishing
- ❖ mature search technologies
- ❖ schema validation
- ❖ re-writing is easier than updating

# Categories (cont...)

- ❖ Used in a situation where one wants to produce documents of articles etc from a huge amount of documents but the format of those article doesn't allow the publisher to perform search on it. Those articles are converted into xml database and wrap it in a readable-URL web service for the document production systems.

## 5) **Distributed Peer Stores**

- ❖ distributed file systems
- ❖ Fast lookups
- ❖ Good distributed storage of data
- ❖ Very low level API
- ❖ Best for voting system. In such a situation one store/user and one store/piece of content is created. The user store will hold all the votes they have ever casted and the content will store a copy of the content on which vote was casted.

# Categories (cont...)

## Example

**Cassandra:** API: many Thrift languages, Query Method: MapReduce, Written in: Java, Concurrency: eventually consistent , Misc: like "Big-Table on Amazon Dynamo alike", initiated by Facebook.

# Object Databases

- Versant
- Db40
- Objectivity
- Startcounter
- Perst
- VelocityDB
- HSS Database
- ZODB
- Magma
- NEO
- Ninja Database
- PicoLisp
- Acid-state
- ObjectDB
- Siaqodb
- Morantex
- EyeDB
- FrameD

# Key-Value Stores

- HASH
- Memcached
- DynamoDB
- Redis
- Riak
- Tokyo Cabinet
- Azure Table Storage
- Riak
- Aerospike
- FoundationDB
- LevelDB
- BerkeleyDB
- GenieDB
- BangDB
- Chordless
- Allegro-C
- nessDB
- HyperDex
- TokyoCabinet/Tyrant
- Scalien
- Voldemort
- Dynomite
- KAI
- MemcahcheDB
- Faircom C-Tree
- LSM
- KitaroDB
- HamsterDB
- STSdb
- Tarantool/Box
- MaxTable
- QuasarDb
- Pincaster
- RaptorDB
- TIBCO Active Spaces
- Sophia

# XML Database

- EMC Documentum xDB
- eXist
- Sedna
- BaseX
- Qizx
- Berkeley DB XML

# Multidimensional databases

- Globals
- Intersystems Cache
- GT.M
- SciDB
- MiniM DB
- Rasdaman

# Document Store (Semi-Structured)

- IBM Lotus
- **MongoDB**
- CouchDB
- MongoDB
- ElasticSearch
- RethinkDB
- RavenDB
- MarkLogic Server
- Clusterpoint Server
- ThruDB
- Terrastore
- JasDB
- RaptorDB
- Djondb
- EJDB
- Amisa Server
- SisoDB
- SDB
- NoSQL embedded Db

- Document Stores
  - Supports complex data model than Key Value Pairs
  - Good at handling content management, session, profile data
  - Multi index support
  - Dynamic schemas, Nested schemas
  - Auto distributed, eventual consistency
  - MVCC (CouchDB) or automic (MongoDB)
  - MongoDB, SimpleDB: widely adopted in this space
  - Use Case: Search by complex patterns & CRUD apps

# Column-Oriented Stores

- Semi-Structured
  - BigTable (inspiration Source)
  - Hbase
  - Apache Cassandra
  - Hypertable
  - Accumulo
  - Amazon Simple DB
  - Cloudata
  - Cloudera
  - HPCC
  - Stratosphere
- Hbase (Apache), Cassandra (Facebook) and HyperTable (Baidu)  
Hbase – CA  
Cassandra – AP  
Model consists of rows and columns  
Scalability: Splitting of both rows and columns  
Rows are split across nodes using primary key, range  
Columns are distributed using groups  
Horizontal and vertical partitioning can be used simultaneous  
Extension of document store  
HBase uses HDFS; Pig, Hive, Cascading can help  
Use case: Grouping of frequently used and un-used over data centers / stream of writes

# Graph Database

- InfoGrid
  - Neo4j
  - InfiniteGraph
  - DEX
  - Titan
  - InforGrid
  - HyperGraphDB
  - GraphBase
  - Trinity
  - AllegroGraph
  - BrightstarDB
  - BigData
  - Meronymy
  - WhiteDB
  - Openlink Virtuoso
  - VertexDB
  - FlockDB
- Social Graph  
Relationship between entities  
Data modeling on social networks  
Common Use Cases  
List of friends  
Recommendation system  
Following  
Followers  
Common Connections  
FAN IN/OUT

# Multimodel databases

- ArangoDB
- OrientDB
- Datanomic
- FatDB
- AlchemyDB
- CortexDB

# NoSQL (Not ONLY SQL)

- Fits very well for volatile data
- High read or write throughput.
- Automatic horizontal scalability (Consistent Hashing)
- Simple to implement, no investment for developers to design and implement relational model
- Application logic defines object model
- Support of MVCC (MultiVersion Concurrency Control) in some form
- Compaction and un-compaction happens at top tier
- In-memory or disk based or combination



# Benefits of NoSQL

- Elastic scaling
- Avoid joins
- No need for data to fit to a schema
- Quick and easy development
- Ability to cope with network failure
- Low latency and high performance
- Integrated Caching facility
- Maintaining NoSQL server is cheaper

# NoSQL - Disadvantages

- Packing and Un-packing of each key
- Lack of relation from one key to another
- Need whole value from the key; to read/write any practical information.
- No Security or authentication
- Data store is merely a storage layer, can't be used for
  - Analytics
  - Reporting
  - Aggregation
  - Ordered Values



# Some Popular NoSQL Databases



# MongoDB 3.2

- **Written in:** C++
- **Main point:** Retains some friendly properties of SQL. (Query, index)
- **License:** AGPL (Drivers: Apache)
- **Protocol:** Custom, binary (BSON)
- Master/slave replication (auto failover with replica sets)
- Sharding built-in
- Queries are javascript expressions
- Run arbitrary javascript functions server-side
- Better update-in-place than CouchDB
- Uses memory mapped files for data storage
- Performance over features
- Journaling (with --journal) is best turned on
- On 32bit systems, limited to ~2.5Gb
- An empty database takes up 192Mb
- GridFS to store big data + metadata (not actually an FS)
- Has geospatial indexing
- Data center aware

**Best used:** If you need dynamic queries. If you prefer to define indexes, not map/reduce functions. If you need good performance on a big DB. If you wanted CouchDB, but your data changes too much, filling up disks.

**For example:** For most things that you would do with MySQL or PostgreSQL, but having predefined columns really holds you back.

# Riak 1.2v

- **Written in:** Erlang & C, some JavaScript
- **Main point:** Fault tolerance
- **License:** Apache
- **Protocol:** HTTP/REST or custom binary
- Stores blobs
- Tunable trade-offs for distribution and replication
- Pre- and post-commit hooks in JavaScript or Erlang, for validation and security.
- Map/reduce in JavaScript or Erlang
- Links & link walking: use it as a graph database
- Secondary indices: but only one at once
- Large object support (Luwak)
- Comes in "open source" and "enterprise" editions
- Full-text search, indexing, querying with Riak Search
- In the process of migrating the storing backend from "Bitcask" to Google's "LevelDB"
- Masterless multi-site replication replication and SNMP monitoring are commercially licensed

**Best used:** If you want something Dynamo-like data storage, but no way you're gonna deal with the bloat and complexity. If you need very good single-site scalability, availability and fault-tolerance, but you're ready to pay for multi-site replication.

**For example:** Point-of-sales data collection. Factory control systems. Places where even seconds of downtime hurt. Could be used as a well-update-able web server.

# CouchDB 1.2v

- **Written in:** Erlang
- **Main point:** DB consistency, ease of use
- **License:** Apache
- **Protocol:** HTTP/REST
- Bi-directional (!) replication,  
continuous or ad-hoc,  
with conflict detection,  
thus, master-master replication. (!)
- MVCC - write operations do not block reads
- Previous versions of documents are available
- Crash-only (reliable) design
- Needs compacting from time to time
- Views: embedded map/reduce
- Formatting views: lists & shows
- Server-side document validation possible
- Authentication possible
- Real-time updates via '\_changes' (!)
- Attachment handling
- thus, [CouchApps](#) (standalone js apps)

**Best used:** For accumulating,  
occasionally changing data, on which  
pre-defined queries are to be run.  
Places where versioning is  
important.

**For example:** CRM, CMS systems.  
Master-master replication is an  
especially interesting feature,  
allowing easy multi-site deployments.

# Redis 2.8v

- **Written in:** C
- **Main point:** Blazing fast
- **License:** BSD
- **Protocol:** Telnet-like, binary safe
- Disk-backed in-memory database,
- Dataset size limited to computer RAM (but can span multiple machines' RAM with clustering)
- Master-slave replication, automatic failover
- Simple values or data structures by keys
- but complex operations like ZREVRANGEBYSCORE.
- INCR & co (good for rate limiting or statistics)
- Bit operations (for example to implement bloom filters)
- Has sets (also union/diff/inter)
- Has lists (also a queue; blocking pop)
- Has hashes (objects of multiple fields)
- Sorted sets (high score table, good for range queries)
- Lua scripting capabilities (!)
- Has transactions (!)
- Values can be set to expire (as in a cache)
- Pub/Sub lets one implement messaging

**Best used:** For rapidly changing data with a foreseeable database size (should fit mostly in memory).

**For example:** Stock prices. Analytics. Real-time data collection. Real-time communication. And wherever you used memcached before

# Additional Reads

- <http://blog.couchbase.com/understanding-performance-benchmark-published-cisco-and-solarflare-using-couchbase-server>
- <http://horicky.blogspot.in/2012/07/couchbase-architecture.html>
- <http://beta.json-generator.com/>
- <http://www.generatedata.com/>
- <https://www.mockaroo.com/>
- <http://jsonstudio.com/resources/>



# 1. MONGO DB INTRODUCTION

# Why choose NoSQL?

- If scalability is a concern?
- Simpler data model (no joins)
- Needs to be implemented on a very low budget and “scaled out”(Horizontal Scaling)
- Real time analysis
- Distributed storage
- Provides flexibility
- Rapid development
- Redundancy/Reliability
- Streaming/Volume

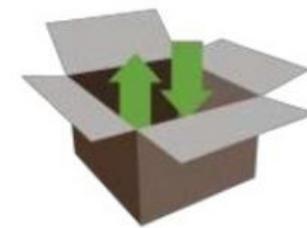
# MONGODB



Fully Featured  
High Performance  
Scalable

```
{  
    name: "John Smith",  
    pfxs: ["Dr.", "Mr."],  
    address: "10 3rd St.",  
    phones: [  
        { number: "555-1212",  
          type: "land" },  
        { number: "444-1212",  
          type: "mobile" }  
    ]  
}
```

Document  
Data Model



Open-  
Source



# MongoDB

- MongoDB (from humongous) is an open source document database written in C++.
- Document-oriented storage
  - BSON style documents with dynamic schemas offer simplicity and power
  - Full index support
  - Replication and High availability
  - Auto-sharding
  - Querying
  - Fast In-place updates

**Developed by**  
**Dwight Merriman**  
**Kevin P Ryan**  
**Eliot Horowitz**



# MongoDB

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.

- Document-oriented Database
  - Uses JSON(BSON actually)
- Schema-Free
- Performant
  - Written in C++
  - Full index support
  - No transactions (has atomic operations)
  - Memory-mapped files (delayed writes)
- Scalable
  - Replication
  - Auto-sharding
- Commercially supported (10gen)
  - Lots of documentation

**MongoDB, the document-oriented database, uses **BSON** as both **the network and on-disk representation of documents****

# MongoDB Key Features

- High performance data persistence
  - Support for embedded data models reduces I/O activity on database system.
  - Indexes support faster queries and can include keys from embedded documents and arrays.
- High Availability
  - Highly available replication facility called replica sets provide
    - Automatic failover
    - Data redundancy
  - Replica Set: a group of MongoDB server that maintain the same data set, providing redundancy and increasing data availability
  - Automatic Scaling
    - Horizontal scalability as part of its core functionality
    - Automatic sharding distributes data across a cluster of machines.
    - Replica sets can provide eventually consistent reads for low-latency high through put deployments

# MongoDB

- Document-based queries
  - Flexible document queries expressed in JSON/JavaScript
- MapReduce
  - Flexible aggregation and data processing
  - Queries run in parallel on all shards
- GridFS
  - Store files of any size easily
- Geospatial Indexing
  - Find Object based on location(i.e find closest n items to x)
- Full Index support
- Replication and High Availability
- Querying
- Auto-Sharding
- Many Production Deployments
  - Sourceforge
  - Github
  - Sugarcrm
  - Shutterfly
  - Bit.ly
  - NYTIMES
- Supported Platforms
  - OSX
  - Linux
  - Solaris
  - Windows
  - FreeBSD

# Why use MongoDB?

## Flexibility

- Data in MongoDB has a flexible schema. Collections do not enforce document structure. This flexibility gives you data-modeling choices to match your application and its performance requirements

## Scalability

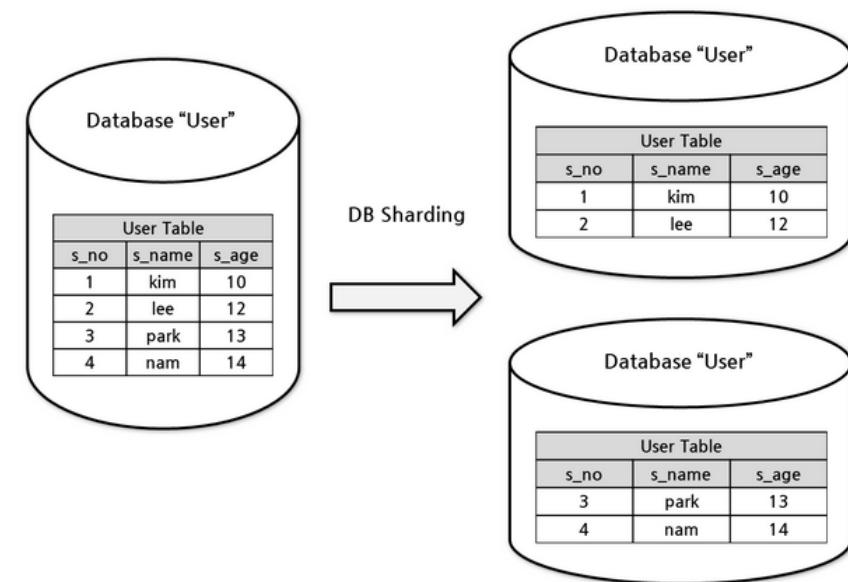
- With increase in size of data, single machine may not be sufficient to store the data nor provide an acceptable read and write throughput.
- Sharding solves the problem with horizontal scaling., it meets the demands of data growth.

## Performance

- Faster than any traditional database.

# Sharding

- A type of database partitioning that separates very large databases into smaller, faster, more easily managed parts called data shards.
- **Shard means a small part of a whole**
- **MongoDB support auto-sharding**
- **Synonymous for horizontal partitioning**



# Scalability

- the ability of a program to scale, where a software solution can handle increased loads of work
- When talking about systems scalability, we usually differentiate between
  - Scale up – the ability to grow by using stronger hardware
  - Scale out – the ability to grow by adding more hardware

Scalability can be achieved in 2 ways

**Vertical** – By adding more hardware like more RAM, processors or more nodes.

Introduce a load balancer, which will help routing the incoming calls to various servers based upon the routing algorithm used. Application will be able to handle more load as load is being shared across the servers

**Horizontal** –architecting the application in such a way that it behaves well in more parallel traffic. Program manages the memory, sessions, cache & state etc..

**MongoDB implements HORIZONTAL SCALABILITY AS PART OF ITS CORE FUNCTIONALITY**



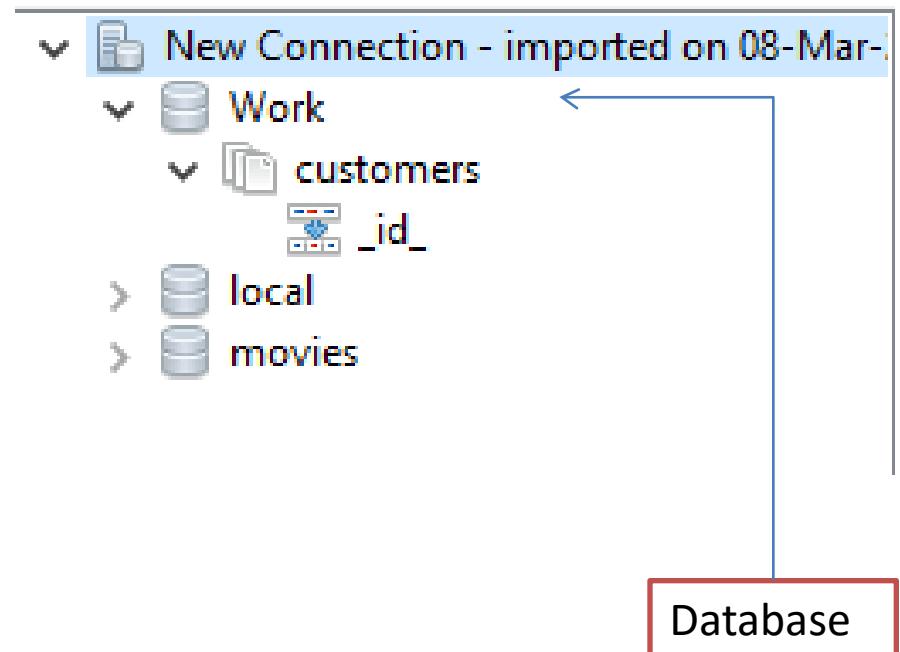
## RDBMS

## MongoDB

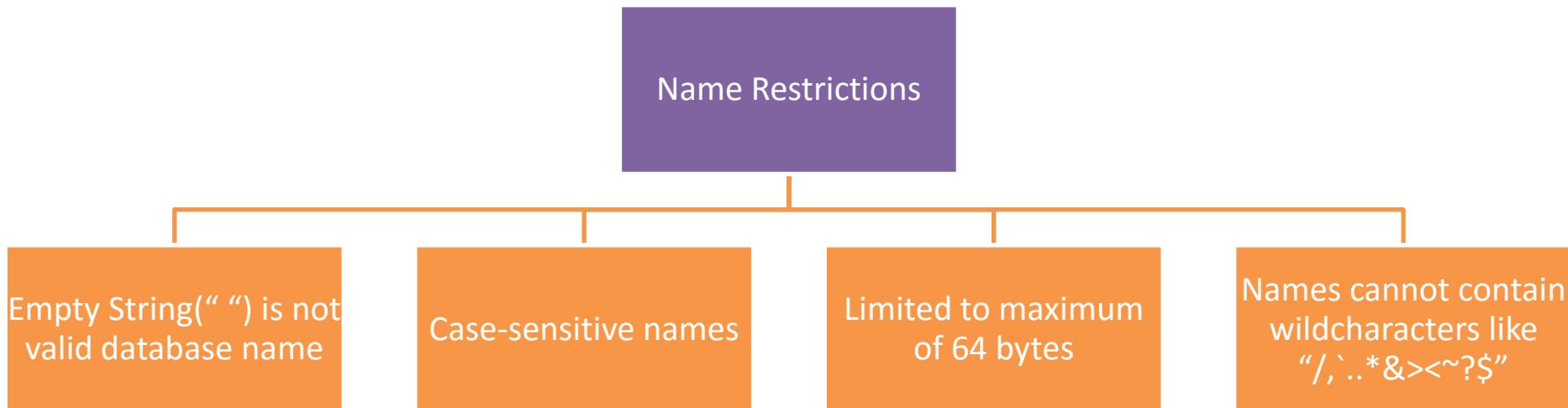
Database	→	Database
Table	→	Collection
Index	→	Index
Row	→	Document
Column	→	Field
Join	→	Embedding & Linking & \$lookup

# Database in MongoDB

- MongoDB groups collections into databases.
- A single instance of MongoDB can host several databases and each database can have zero or more collections.



# DB Name Restrictions



- **Several reserved databases names**
  - **Admin** : “root” database used for authentication. If a user is added to the admin database, the user automatically inherits permissions for all databases.
  - **local** : This database will never be replicated and can be used to store any collection that should be local to a single server
  - **config** : In a sharded setup, MongoDB uses **config** database to store information about the shards.

# MongoDB Terms

operations	MongoDB	description
C	INSERT	<b>Insert</b> <u>document</u> into the collection
R	FIND	Query any collection and <b>FIND</b> <u>documents</u>
U	UPDATE	<b>Update</b> whole <u>document or any specific ones</u>
D	REMOVE	<b>Remove</b> whole <u>document or any specific ones</u>

# Documents in MongoDB

## Document

A data structure composed of field and value pairs.

- Analogous to a row in RDBMS
- Represented as JSON(BSON)
  - Binary JSON
- Hierarchical
- Can reference other documents, arrays and arrays of documents

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



The diagram illustrates a MongoDB document represented as a JSON object. It consists of four key-value pairs: 'name: "sue"', 'age: 26', 'status: "A"', and 'groups: [ "news", "sports" ]'. Blue arrows point from the text 'field: value' to each of these four pairs, indicating the structure of a MongoDB document.

# Documents in MongoDB

## Advantages

- Documents (i.e objects) correspond to native data types in many programming languages
- Embedded documents and arrays reduce need for expensive joins.
- Dynamic schema supports fluent polymorphism

# Collections in MongoDB

- Analogous to a table in RDBMS
- Collection of documents
- Documents can be anything
- Advantageous if they are similar

# Java Script Object Notation

Douglas Crockford

- Lightweight and human readable
- JSON Notation contains these basic elements which we also use in JS:
  - **Objects** : Begin and end with curly braces {}
  - **Object Members** : Members consist of strings and values comma (,) separated
  - **Arrays**: Arrays begin and end with braces and contain different values
  - **Values**: A value can be a string, can be an object, an array or the literals
  - **Strings**: string data type surrounded by "" and Members are seperated by commas, values are seperated by commas, true, false, or null.character or common blackslash escapes.

```
{  
  "first_name": "Awase Khirni",  
  "last_name": "Syed",  
  "email_address": "sXXXX@sycliq.com",  
  "is_alive": true,  
  "age": 38,  
  "height_cm": 186,  
  "education": "Ph.D",  
  "University": "university of zurich"  
  "billing_address": {  
    "address": "Umiya Business Bay, Level 8, Tower 1, Cessna Business Park",  
    "city": "Marthahalli ORR, Bangalore",  
    "state": "Karnataka",  
    "postal_code": "560069"  
  },  
  "shipping_address": {  
    "address": "Umiya Business Bay, Level 8, Tower 1, Cessna Business Park",  
    "city": "Marthahalli ORR, Bangalore",  
    "state": "Karnataka",  
    "postal_code": "560069"  
  },  
  "phone_numbers": [  
    {  
      "type": "home",  
      "number": "9035XX3XXX"  
    },  
    {  
      "type": "office",  
      "number": "9035XX3XXX"  
    }  
  ],  
  "date_of_birth": null  
}
```



# JSON- JAVASCRIPT OBJECT NOTATION

Douglas Crockford

```
{  
  "_id" : 1,  
  "name" : { "first" : "John", "last" : "Backus" },  
  "contribs" : [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ],  
  "awards" : [  
    {  
      "award" : "W.W. McDowell Award",  
      "year" : 1967,  
      "by" : "IEEE Computer Society"  
    },  
    {  
      "award" : "Draper Prize",  
      "year" : 1993,  
      "by" : "National Academy of Engineering"  
    }  
  ]  
}
```

# JSON

<http://json.parser.online.fr/>

## Advantages

- Designed for being generated and decoded serially.
- JSON is faster
- Schema support across browsers and OS
- Fast Server Parsing
- Ideal format for data sharing

## Disadvantages

- Less good at being stored and traversed (serial read).
- Has no error handling for JSON calls.
- can be quite dangerous if used with untrusted services or untrusted browsers



Student

```
{  
    "name": {  
        "first": "Awase ",  
        "last": "Syed"  
    },  
    "email": "sak@sycliq.com",  
    "major": "Geographic Information Systems",  
    "test_scores": [  
        {  
            "test": "ACT",  
            "score": 78  
        },  
        {  
            "test": "SAT",  
            "score": 680  
        }  
    ],  
    "course_gpas": [  
        3.09,  
        2.72,  
        3.89,  
        3.45,  
        3.9,  
        4,  
        3.5,  
        3.6,  
        3.7,  
        3.8,  
        3.9,  
        3,  
        3.9,  
        3.5  
    ]  
}
```

```
{  
    "name": {  
        "first": "Sainath",  
        "middle": "Peribotula",  
        "last": "korigantla"  
    },  
    "email": "psainath@gmail.com",  
    "bio": " born in south africa, have done socioeconomic studies, pursued doctoral studies in socioeconomic at london school of economics. Have been a critic for various government policies which are not pro-people",  
    "publications": {  
    },  
    "courses": [  
        1010,  
        1012,  
        1013,  
        4012,  
        5012,  
        7012  
    ]  
}
```

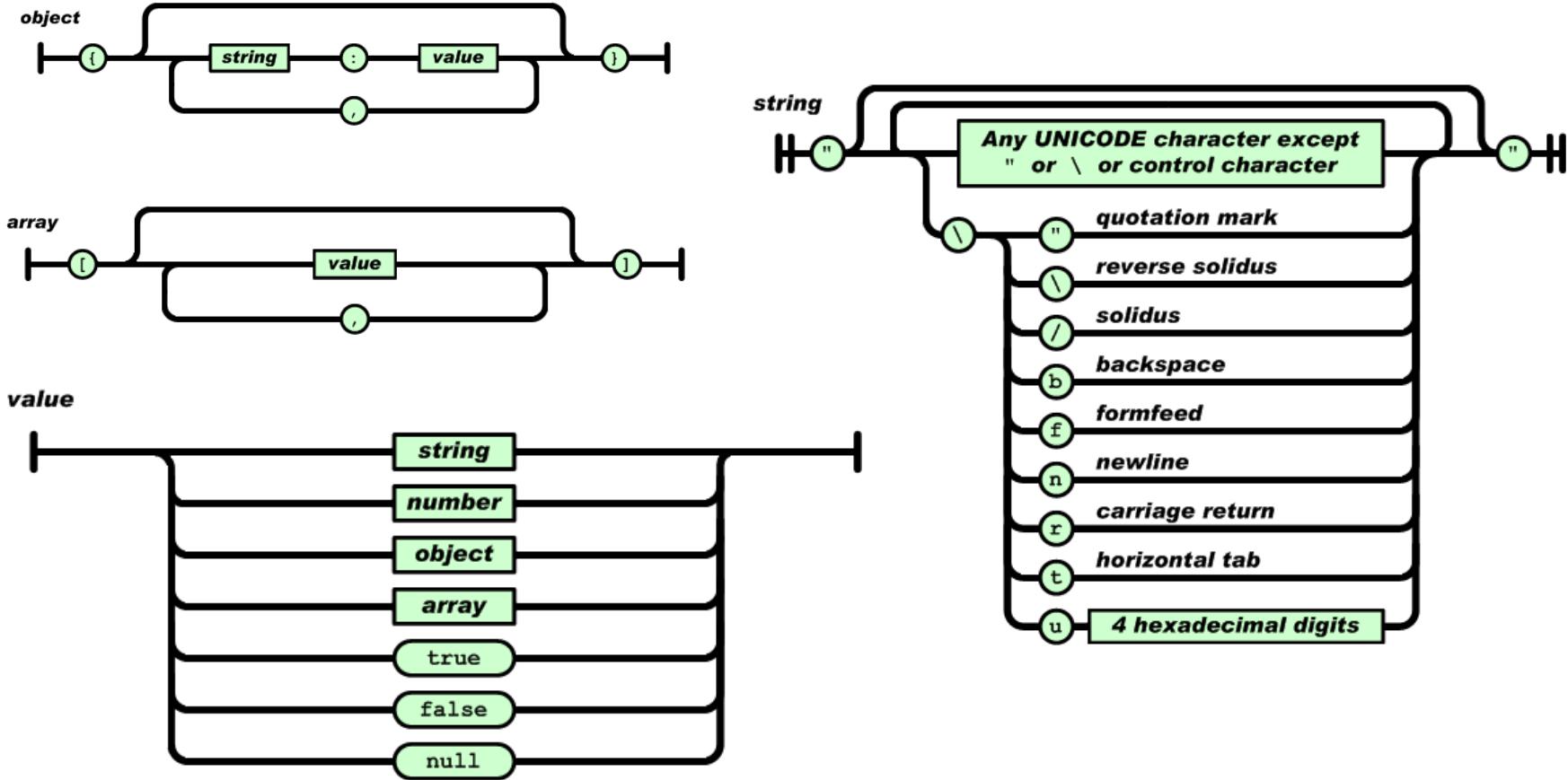
Professor



# JSON Syntax Rules

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays
- A common use of JSON is to read data from a web server and display the data in a web page
- Often used to serialize and transfer data over a network connection

# JSON DATA TYPES



# JSON DATA TYPES

Type	Description
Number	double- precision floating-point format in JavaScript
String	double-quoted Unicode with backslash escaping
Boolean	true or false
Array	an ordered sequence of values
Value	it can be a string, a number, true or false, null etc
Object	an unordered collection of key:value pairs
Whitespace	can be used between any pair of tokens
null	empty



- Lightweight, AJAX Friendly, key value pairs
- No validation to an underlying schema required.
- Traversing requires to navigate through all the elements in a sequential manner.
- Simple and loosely structured as key-value pairs

## XML

- XML is attached to an underlying schema that enforces structure to the data with hierarchy.
- Validation of XML data to conform to underlying schema
- XSLT to extract data from an XML document using arbitrarily complex XPath expression.

# BSON, a binary variant of JSON

- A single entity in BSON is called as document.
- A document comprised of zero or more key/value pairs(like associative arrays) in binary format.
- BSON supports the following basic data types:
  - Byte – 8 bits
  - Int32 – 4 bytes (32bit signed integer)
  - Int 64 – 8 bytes (64 bit signed integer)
  - Double – 8 bytes – 64 bit IEEE 754 floating point
  - Date Type and BinData Type
- BSON is faster than JSON when it comes to encoding and decoding.
- BSON supports the embedding of documents and arrays with-in other documents and arrays.
- BSON also contains extensions that allow representation of data types that are not part of the JSON spec.

# BSON Characteristics

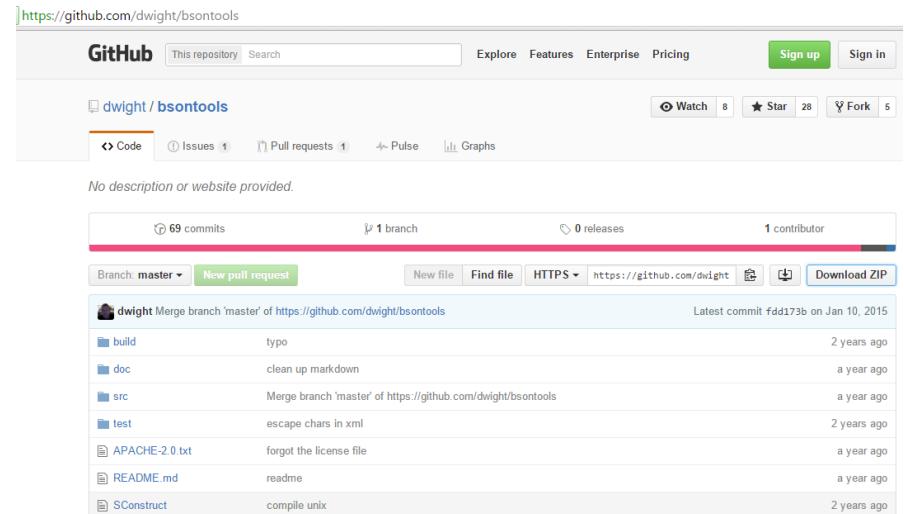
1. Lightweight: Keeping spatial overhead to a minimum is important for any data representation format, especially when used over the network
2. Traversable : BSON is designed to be traversed easily. This is a vital property in its role as the primary data representation for MongoDB.
3. Efficient : Encoding data to BSON and decoding from BSON can be performed very quickly in most languages due to the use of C data types.

# BSON- BINARY JSON

1. SCANABILITY
2. DATA TYPES

## BSON TOOLS

<https://github.com/dwight/bsontools>



The screenshot shows the GitHub repository page for dwight/bsontools. The repository has 69 commits, 1 branch, 0 releases, and 1 contributor. The latest commit was on Jan 10, 2015. The commits are listed below:

File	Message	Time Ago
build	typo	2 years ago
doc	clean up markdown	a year ago
src	Merge branch 'master' of https://github.com/dwight/bsontools	a year ago
test	escape chars in xml	2 years ago
APACHE-2.0.txt	forgot the license file	a year ago
README.md	readme	a year ago
SConstruct	compile unix	2 years ago



# BSON

Student

```
{  
  "_id": ObjectId("1231231sda435435sasdas"),  
  "name": {  
    "first": "Awase",  
    "last": "Syed"  
  },  
  "email": "sak@sycliq.com",  
  "major": "Geographic Information Systems"  
}
```

# BSON Data Types in MongoDB

Type	Describing Number
Double	1
String	2
Object	3
Array	4
Binary Data	5
Object ID	7
Boolean	8
Date	9
Null	10
Regular Expression	11

- BSON is a binary serialization format used to store documents
- Each data type has a corresponding Number that can be used with **\$type** Operator to query documents by BSON Type

```
db.corp.employee.find({first_name:{$type:2}})
```

- Query returns first name field for each document, if its type is “**string**” ! Otherwise no result set will be returned.



# BSON Data Types in MongoDB

Type	Describing Number
DBPointer	12
JavaScript	13
Symbol	14
JavaScript(with Scope)	15
32-bit Integer (int)	16
Timestamp	17
64-bit Integer (long)	18
Min Key	-1
Max key	127



# BSON USE IN MONGODB

**It is a binary form for  
representing simple data  
structures and  
associative arrays**

**MongoDB uses BSON as the  
data storage and  
network transfer format  
for “documents”.**

# Collection

- A collection is a group of documents, if a document is roughly equivalent to a row in a relational database management system then collection is roughly equivalent to tables.
- Collections have **dynamic schemas**
  - Document with single collection can have any number of different “shapes”.
- **Collection Naming Restriction**
  - Empty String(“ ”) not valid
  - \0 character in collection name is not valid
  - Must not start with “system”
  - Reserved character “\$” not to be used

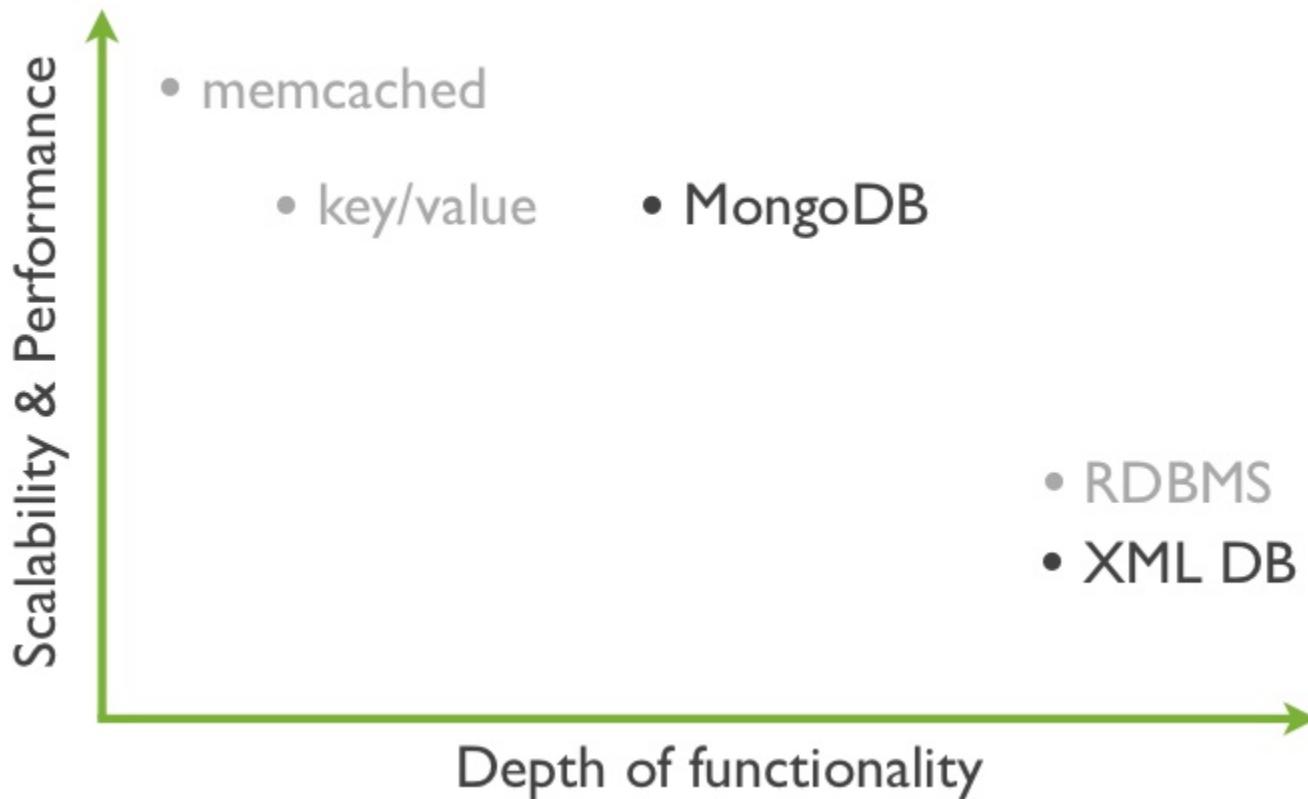
## Embedding

- Analogous to a foreign key
- Think “pre-joined relationship”
- Clearer conceptual model
- DDD Aggregate Root
- Can be
  - Sub Objects
  - Collections

## References

- Analogous to a foreign key
- Think “relationship”

# DB Landscape





# MongoDB System Components

## **Mongod.exe**

The database Server

## **Mongos.exe**

The sharding Server

## **Mongo.exe**

The interactive shell



# Window Installation

```
C:\Program Files\MongoDB\Server\3.2\bin>mongod
2016-03-08T13:19:18.264+0530 I CONTROL  [initandlisten] MongoDB starting : pid=8380 port=27017 dbpath=C:\data\db\ 64-bit host=DESKTOP-I57JOS9
2016-03-08T13:19:18.266+0530 I CONTROL  [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2016-03-08T13:19:18.266+0530 I CONTROL  [initandlisten] db version v3.2.0
2016-03-08T13:19:18.267+0530 I CONTROL  [initandlisten] git version: 45d947729a0315accb6d4f15a6b06be6d9c19fe7
2016-03-08T13:19:18.268+0530 I CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.0.1p-fips 9 Jul 2015
2016-03-08T13:19:18.269+0530 I CONTROL  [initandlisten] allocator: tcmalloc
2016-03-08T13:19:18.270+0530 I CONTROL  [initandlisten] modules: none
2016-03-08T13:19:18.270+0530 I CONTROL  [initandlisten] build environment:
2016-03-08T13:19:18.271+0530 I CONTROL  [initandlisten]     distmod: 2008plus-ssl
2016-03-08T13:19:18.272+0530 I CONTROL  [initandlisten]     distarch: x86_64
2016-03-08T13:19:18.273+0530 I CONTROL  [initandlisten]     target_arch: x86_64
2016-03-08T13:19:18.273+0530 I CONTROL  [initandlisten] options: {}
2016-03-08T13:19:18.276+0530 I STORAGE [initandlisten] exception in initAndListen: 29 Data directory C:\data\db\ not found., terminating
2016-03-08T13:19:18.276+0530 I CONTROL  [initandlisten] dbexit: rc: 100
```

## Create default data location c: -> data ->db

Starting the mongodb      **Mongod &**

```
C:\Program Files\MongoDB\Server\3.2\bin>mongod
2016-03-08T13:22:03.716+0530 I CONTROL  [initandlisten] MongoDB starting : pid=6224 port=27017 dbpath=C:\data\db\ 64-bit host=DESKTOP-I57JOS9
2016-03-08T13:22:03.717+0530 I CONTROL  [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2016-03-08T13:22:03.717+0530 I CONTROL  [initandlisten] db version v3.2.0
2016-03-08T13:22:03.717+0530 I CONTROL  [initandlisten] git version: 45d947729a0315accb6d4f15a6b06be6d9c19fe7
2016-03-08T13:22:03.717+0530 I CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.0.1p-fips 9 Jul 2015
2016-03-08T13:22:03.718+0530 I CONTROL  [initandlisten] allocator: tcmalloc
2016-03-08T13:22:03.718+0530 I CONTROL  [initandlisten] modules: none
2016-03-08T13:22:03.718+0530 I CONTROL  [initandlisten] build environment:
2016-03-08T13:22:03.718+0530 I CONTROL  [initandlisten]     distmod: 2008plus-ssl
2016-03-08T13:22:03.718+0530 I CONTROL  [initandlisten]     distarch: x86_64
2016-03-08T13:22:03.718+0530 I CONTROL  [initandlisten]     target_arch: x86_64
2016-03-08T13:22:03.719+0530 I CONTROL  [initandlisten] options: {}
2016-03-08T13:22:03.771+0530 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=1G/session_max=20000,eviction=(threads_max=4),config_base=false,statistics=(fast),log=(enabled=true
,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000),checkpoint=(wait=60,log_size=2GB),statistics_log=(wait=0),direct_io=(data,
2016-03-08T13:22:04.244+0530 I NETWORK [HostnameCanonicalizationWorker] Starting hostname canonicalization worker
2016-03-08T13:22:04.244+0530 I FTDC   [initandlisten] Initializing full-time diagnostic data capture with directory 'C:/data/db/diagnostic.data'
2016-03-08T13:22:04.535+0530 I NETWORK [initandlisten] waiting for connections on port 27017
```



# Using --dbpath

## Starting database

Start mongodb database by specifying the dbpath

By default all the data and log files are in

C:\data\db and c:\logs

➤ **Mongod –dbpath c:\data\db**

```
mongod --dbpath E:\mongodata\data --port 27017 --logpath  
E:\mongodata\mongod.log
```

## Shutdown database

- Invoke shell using
  - mongo
  - show dbs
  - use admin
  - db.shutdownServer()
  - db.shutdownServer({timeou  
tSecs: 60});



Administrator: Windows PowerShell (x86)

PS C:\Program Files\MongoDB\Server\3.2\bin> ls

Directory: C:\Program Files\MongoDB\Server\3.2\bin

Mode	LastWriteTime	Length	Name
d----	08/03/2016	13:17	data
-a---	04/12/2015	21:19	bsondump.exe
-a---	14/07/2015	20:16	lbeay32.dll
-a---	04/12/2015	21:25	mongo.exe
-a---	04/12/2015	21:29	mongod.exe
-a---	04/12/2015	21:29	19110400 mongod.pdb
-a---	04/12/2015	21:21	30604800 mongodump.exe
-a---	04/12/2015	21:20	10216448 mongoexport.exe
-a---	04/12/2015	21:20	10084352 mongofiles.exe
-a---	04/12/2015	21:20	10313216 mongoimport.exe
-a---	04/12/2015	21:21	9807360 mongooplog.exe
-a---	04/12/2015	21:29	16425984 mongoperf.exe
-a---	04/12/2015	21:20	49544704 mongorestore.exe
-a---	04/12/2015	21:28	7814656 mongos.exe
-a---	04/12/2015	21:28	84553728 mongos.pdb
-a---	04/12/2015	21:19	10032128 mongostat.exe
-a---	04/12/2015	21:21	9894400 mongotop.exe
-a---	14/07/2015	20:16	351232 ssleay32.dll

PS C:\Program Files\MongoDB\Server\3.2\bin> & .\mongod.exe --config c:\mongo\mongo.cfg

## Mongo.cfg (configuration file)

mongo.cfg C:\mongo

```
1 systemLog:
2   destination: file
3   path: "C:/mongo/logs/mongod.log"
4   logAppend: true
5   quiet: false
6 storage:
7   dbPath: "C:/mongo/data/db"
8 -
```



# Mongo Shell

```
C:\Windows\system32\cmd.exe - mongo localhost/test
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\SyedAwase>cd C:\Program Files\MongoDB\Server\3.2\bin

C:\Program Files\MongoDB\Server\3.2\bin>mongo localhost/test
MongoDB shell version: 3.2.0
connecting to: localhost/test
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
      http://docs.mongodb.org/
Questions? Try the support group
      http://groups.google.com/group/mongodb-user
> db
test
> show dbs
local 0.000GB
> show collections
> -
```



# Monitor log files using BareTail

mongod.log (5.5 KB) - BareTail

File Edit View Preferences Help

Open Highlighting Follow Tail ANSI C:\mongo\logs\mongod.log (5.5 KB)

```
2016-03-08T21:46:18.061+0530 I CONTROL [initandlisten] MongoDB starting : pid=1020 port=27017 dbpath=C:/mongo/data/db 64-bit host=DESKTOP-I57JOS9
2016-03-08T21:46:18.068+0530 I CONTROL [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2016-03-08T21:46:18.082+0530 I CONTROL [initandlisten] db version v3.2.0
2016-03-08T21:46:18.082+0530 I CONTROL [initandlisten] git version: 45d947729a0315acccb6d4f15a6b06be6d9c19fe7
2016-03-08T21:46:18.082+0530 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.0.1p-fips 9 Jul 2015
2016-03-08T21:46:18.082+0530 I CONTROL [initandlisten] allocator: tcmalloc
2016-03-08T21:46:18.083+0530 I CONTROL [initandlisten] modules: none
2016-03-08T21:46:18.083+0530 I CONTROL [initandlisten] build environment:
2016-03-08T21:46:18.083+0530 I CONTROL [initandlisten] distmod: 2008plus-ssl
2016-03-08T21:46:18.083+0530 I CONTROL [initandlisten] distarch: x86_64
2016-03-08T21:46:18.083+0530 I CONTROL [initandlisten] target_arch: x86_64
2016-03-08T21:46:18.091+0530 I CONTROL [initandlisten] options: { config: "c:/mongo/mongo.cfg", storage: { dbPath: "C:/mongo/data/db" }, systemLog: { destination: "file", logAppend: true, p
2016-03-08T21:46:18.117+0530 I - [initandlisten] Detected data files in C:/mongo/data/db created by the 'wiredTiger' storage engine, so setting the active storage engine to 'wiredTige
2016-03-08T21:46:18.119+0530 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=1G,session_max=20000,eviction=(threads_max=4),config_base=false,statistics=(fast),log=(enabl
2016-03-08T21:46:19.884+0530 I NETWORK [HostnameCanonicalizationWorker] Starting hostname canonicalization worker
2016-03-08T21:46:19.884+0530 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory 'C:/mongo/data/db/diagnostic.data'
2016-03-08T21:46:19.971+0530 I NETWORK [initandlisten] waiting for connections on port 27017
2016-03-08T21:51:11.323+0530 I NETWORK [initandlisten] connection accepted from 127.0.0.1:22438 #1 (1 connection now open)
2016-03-08T21:51:29.497+0530 I COMMAND [conn1] command admin.$cmd command: listDatabases { 1.0 } ntnoreturn:1 ntoskip:0 keyUpdates:0 writeConflicts:0 numYields:0 reslen:211 lo
2016-03-08T21:51:40.489+0530 I COMMAND [conn1] terminating, shutdown command received
2016-03-08T21:51:40.489+0530 I FTDC [conn1] Shutting down full-time diagnostic data capture
2016-03-08T21:51:40.514+0530 I CONTROL [conn1] now exiting
2016-03-08T21:51:40.527+0530 I NETWORK [conn1] shutdown: going to close listening sockets...
2016-03-08T21:51:40.527+0530 I NETWORK [conn1] closing listening socket: 616
2016-03-08T21:51:40.527+0530 I NETWORK [conn1] shutdown: going to flush diaglog...
2016-03-08T21:51:40.527+0530 I NETWORK [conn1] shutdown: going to close sockets...
2016-03-08T21:51:40.527+0530 I STORAGE [conn1] WiredTigerKVEngine shutting down
2016-03-08T21:51:41.072+0530 I STORAGE [conn1] shutdown: removing fs lock...
2016-03-08T21:51:41.073+0530 I CONTROL [conn1] dbexit: rc: 0
2016-03-08T21:52:10.967+0530 I CONTROL [main] ***** SERVER RESTARTED *****
2016-03-08T21:52:11.211+0530 I CONTROL [initandlisten] MongoDB starting : pid=7204 port=27017 dbpath=C:/mongo/data/db 64-bit host=DESKTOP-I57JOS9
2016-03-08T21:52:11.211+0530 I CONTROL [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2016-03-08T21:52:11.211+0530 I CONTROL [initandlisten] db version v3.2.0
2016-03-08T21:52:11.211+0530 I CONTROL [initandlisten] git version: 45d947729a0315acccb6d4f15a6b06be6d9c19fe7
2016-03-08T21:52:11.211+0530 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.0.1p-fips 9 Jul 2015
2016-03-08T21:52:11.212+0530 I CONTROL [initandlisten] allocator: tcmalloc
2016-03-08T21:52:11.212+0530 I CONTROL [initandlisten] modules: none
2016-03-08T21:52:11.212+0530 I CONTROL [initandlisten] build environment:
2016-03-08T21:52:11.212+0530 I CONTROL [initandlisten] distmod: 2008plus-ssl
2016-03-08T21:52:11.212+0530 I CONTROL [initandlisten] distarch: x86_64
2016-03-08T21:52:11.212+0530 I CONTROL [initandlisten] target_arch: x86_64
2016-03-08T21:52:11.212+0530 I CONTROL [initandlisten] options: { config: "c:/mongo/mongo.cfg", storage: { dbPath: "C:/mongo/data/db" }, systemLog: { destination: "file", logAppend: true, p
2016-03-08T21:52:11.213+0530 I - [initandlisten] Detected data files in C:/mongo/data/db created by the 'wiredTiger' storage engine, so setting the active storage engine to 'wiredTige
2016-03-08T21:52:11.213+0530 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=1G,session_max=20000,eviction=(threads_max=4),config_base=false,statistics=(fast),log=(enabl
2016-03-08T21:52:12.369+0530 I NETWORK [HostnameCanonicalizationWorker] Starting hostname canonicalization worker
2016-03-08T21:52:12.369+0530 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory 'C:/mongo/data/db/diagnostic.data'
2016-03-08T21:52:12.443+0530 I NETWORK [initandlisten] waiting for connections on port 27017
```

Activate



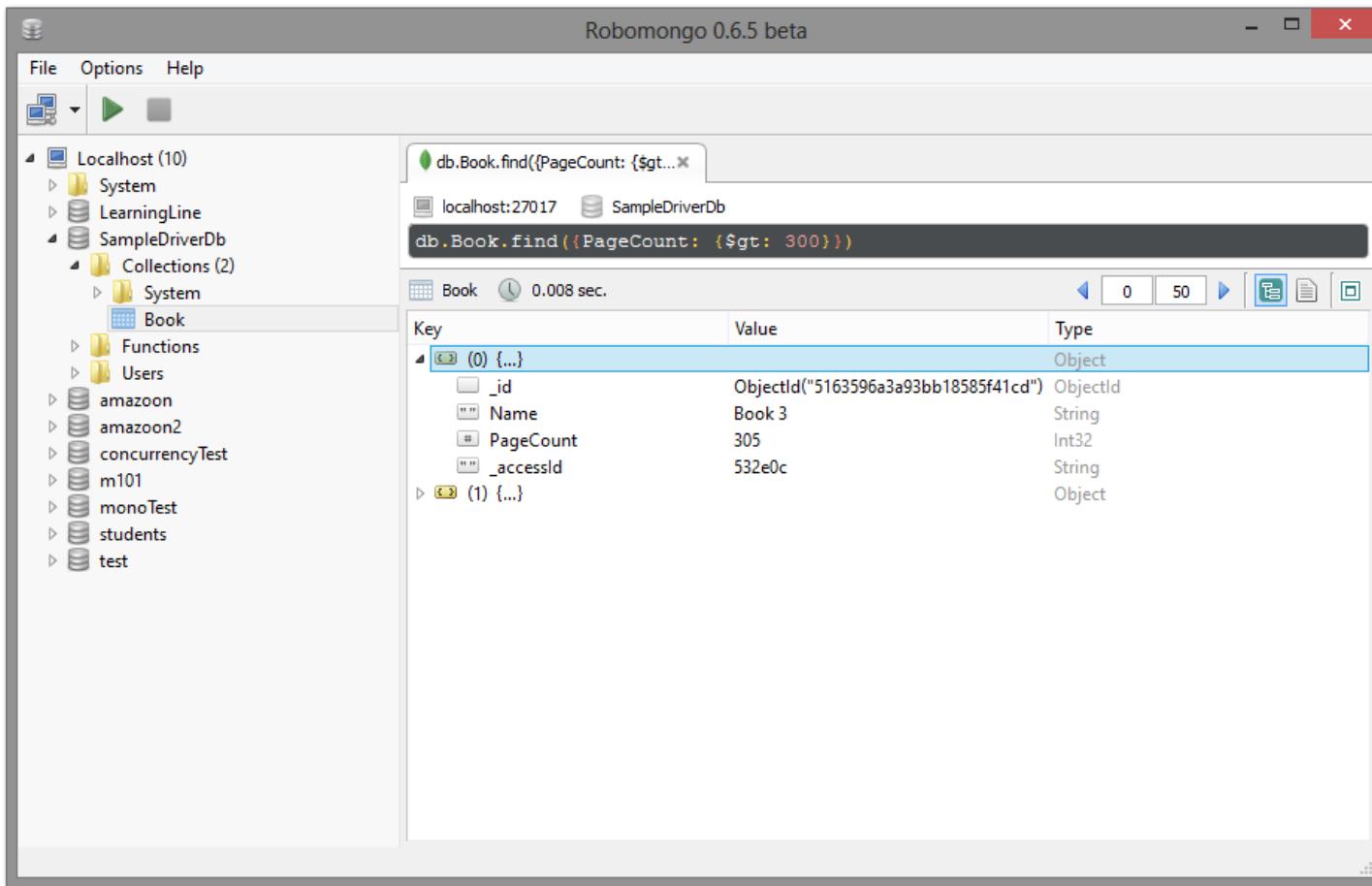
# Configuring Startup with a System(service Integration)

```
PS C:\Program Files\MongoDB\Server\3.2\bin> & .\mongod.exe --config c:\mongo\mongo.cfg
PS C:\Program Files\MongoDB\Server\3.2\bin> & .\mongod.exe --config c:\mongo\mongo.cfg --install
PS C:\Program Files\MongoDB\Server\3.2\bin>
```

-- install

- enables mongod as a service in windows.

# RoboMongo



The screenshot shows the Robomongo interface. On the left is a sidebar with a tree view of databases and collections. The 'SampleDriverDb' database is selected, and its 'Book' collection is highlighted. In the main pane, a search bar contains the query `db.Book.find({PageCount: {$gt: 300}})`. Below the search bar, the results are displayed in a table:

Key	Value	Type
0 (0) ...	Object	Object
_id	ObjectId("5163596a3a93bb18585f41cd")	ObjectId
Name	Book 3	String
PageCount	305	Int32
__accessId	532e0c	String
1 (1) ...	Object	Object



# MongoChef

MongoChef Professional - 3T Software Labs - Non-Commercial License

File Edit Database Collection Index Document GridFS View Help

Connect IntelliShell Aggregate Map-Reduce Export Import Users Roles Feedback

New Connection - imported on 08-Mar-2016 (localhost:27017)

- demo
- things
- local
  - hello
  - startup\_log
- test
  - mycollection

Welcome Recent Connections

New Connection - imported on 08-Mar-2016 (localhost:27017)

Connect...

What's new

**IntelliShell: Your Intelligent MongoDB Shell**

Use IntelliShell to access the power of the shell in the beautiful MongoChef environment. Enjoy MongoChef's intelligent auto-completion, including auto-completion of shell methods and document field names.

IntelliShell output is editable with MongoChef's in-place editors and can always be exported to a file or copied to another collection.

Select a collection, click the **IntelliShell** button in the main tool bar and check it out!

Find out more about IntelliShell on our blog: [Getting Started with MongoChef's IntelliShell](#)

Blog Posts

- How to Have a Side-By-Side View of Multiple Tabs with MongoChef  
15 Feb 2016
- How to Do MongoDB Map-Reduce Queries Easily with 3T MongoChef  
01 Feb 2016
- How to prevent your connection from dropping with hosted MongoDB instances  
06 Oct 2015
- Connecting to your MongoDB at MongoLab  
02 Oct 2015
- How to Do MongoDB Aggregation Queries Easily with 3T MongoChef  
08 Sep 2015
- Running MongoChef on CentOS  
03 Sep 2015
- Updated: MongoChef on OS X 10.11 (El Capitan)  
13 Jul 2015
- How to Query MongoDB Secondary Replica Set Members  
10 Jun 2015

[View Online](#)

Activate Windows  
Go to Settings to activate Windows.

# Example

## Relational

Customer ID	First Name	Last Name	City
0	John	Doe	New York
1	Mark	Smith	San Francisco
2	Jay	White	Dallas
3	Meagan	White	London
4	Edward	Daniels	Boston

Phone Number	Type	DNC	Customer ID
1-212-555-1212	home	T	0
1-212-555-1213	home	T	0
1-212-555-1214	cell	F	0
1-212-777-1212	home	T	1
1-212-777-1213	cell	(null)	1
1-212-888-1212	home	F	2

## MongoDB

```
{ vers: 1,  
  customer_id : 1,  
  name : {  
    "f":"Mark",  
    "l":"Smith" },  
  city : "San Francisco",  
  phones: [ {  
    number : "1-212-777-1212",  
    dnc : true,  
    type : "home"  
  },  
  {  
    number : "1-212-777-1213",  
    type : "cell"  
  } ]  
}
```



# MongoDB Tools

<http://mongodb-tools.com/>

**MongoDB Tools** *a guide to some interesting tools for working with mongodb* 

PURPOSE	LICENSE	LANGUAGE	COMPATIBILITY
 Chef-MongoDB Cookbook Actively Maintained	 Luamongo Actively Maintained	 Django MongoDB Backend Actively Maintained	
 NoSQL Viewer for BigData Actively Maintained	 mongobee Actively Maintained	 Nucleon BI Studio Actively Maintained	

Big Data	Product & Asset Catalogs	Security & Fraud	Internet of Things	Database-as-a-Service	Complex Data Management
		 Top Investment and Retail Banks	Top Global Shipping Company  Top Industrial Equipment Manufacturer	 Top Media Company	Cushman & Wakefield Top Investment and Retail Banks
Mobile Apps	Customer Data Management	Single View	Social & Collaboration	Content Management	Embedded / ISV



# MongoDB Driver

- <http://us.php.net/manual/en/mongo.installation.php>

# Installing Node.js



Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

Important **security releases**, please update now!

Download for Windows (x64)

v4.3.2 LTS

Mature and Dependable

v5.7.1 Stable

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

Or have a look at the [LTS schedule](#).

# Node.js

Node.js® is a JavaScript runtime built on [Chrome's V8 JavaScript engine](#). Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, [npm](#), is the largest ecosystem of open source libraries in the world.

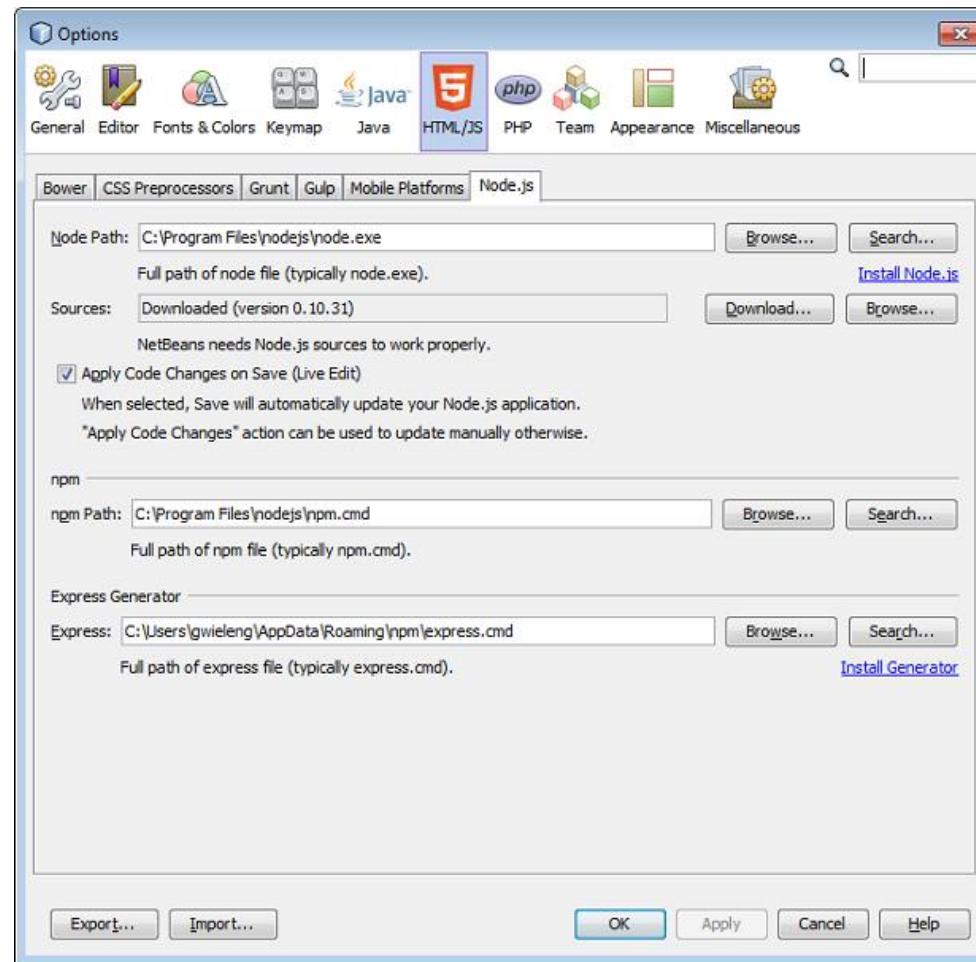
## Starting

- @ powershell console
  - node
  - Console.log("Hello world");
  - npm install -g (global)

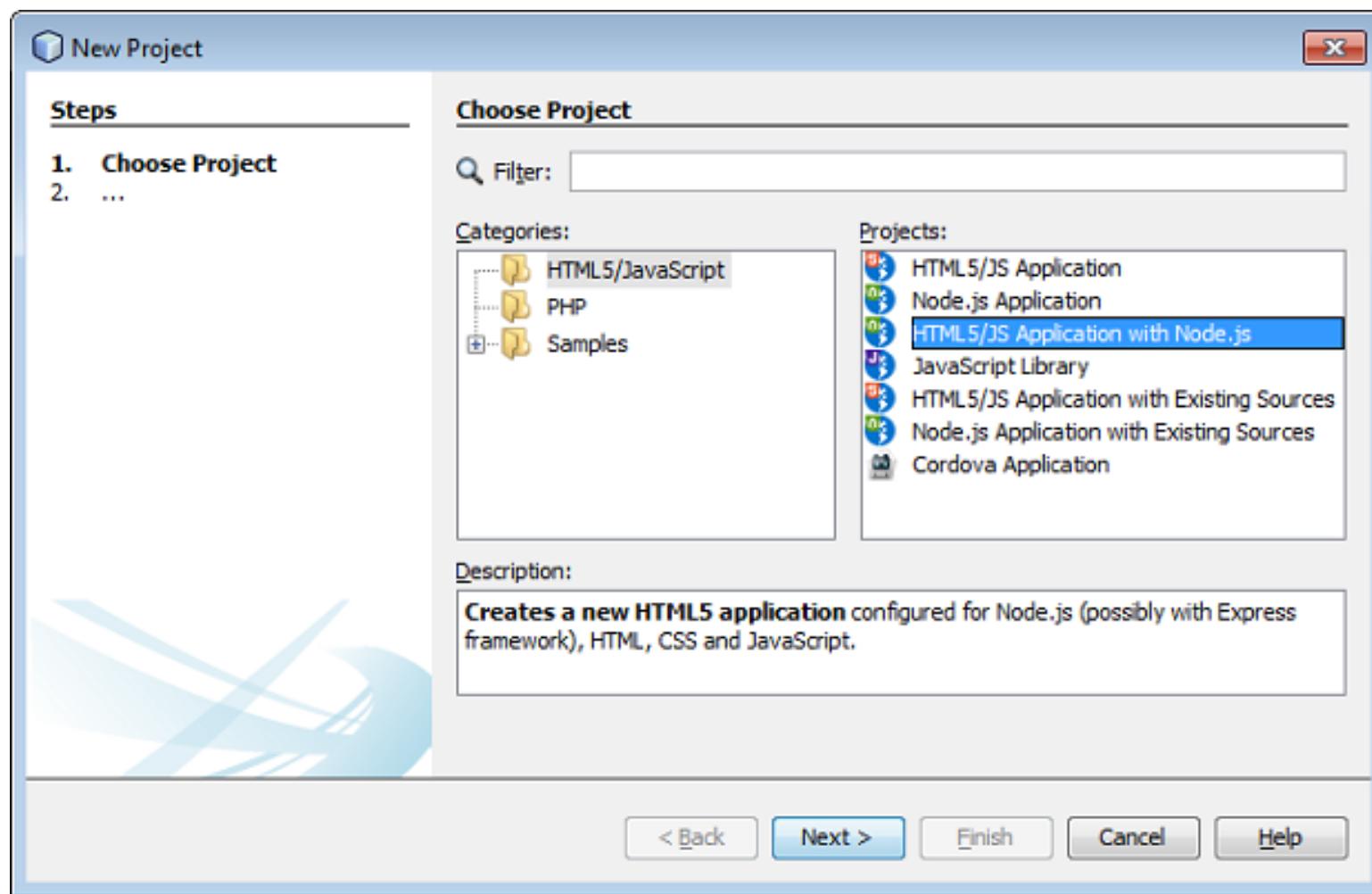
## Shutting down

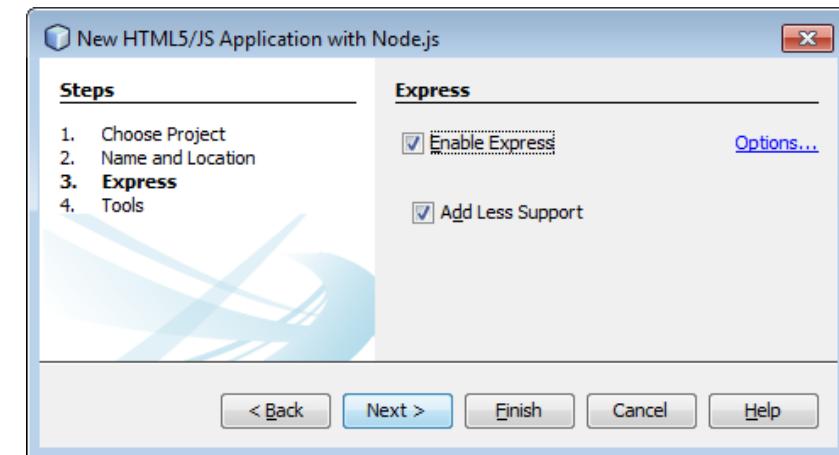
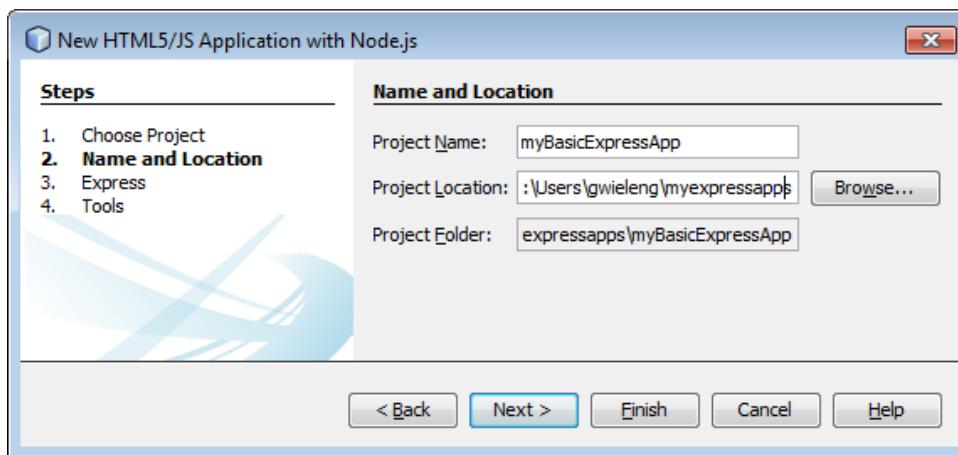
- @powershell console
  - node
  - process.exit();

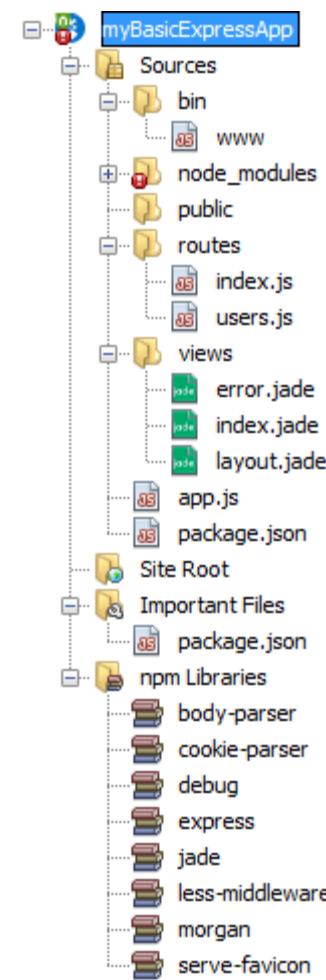
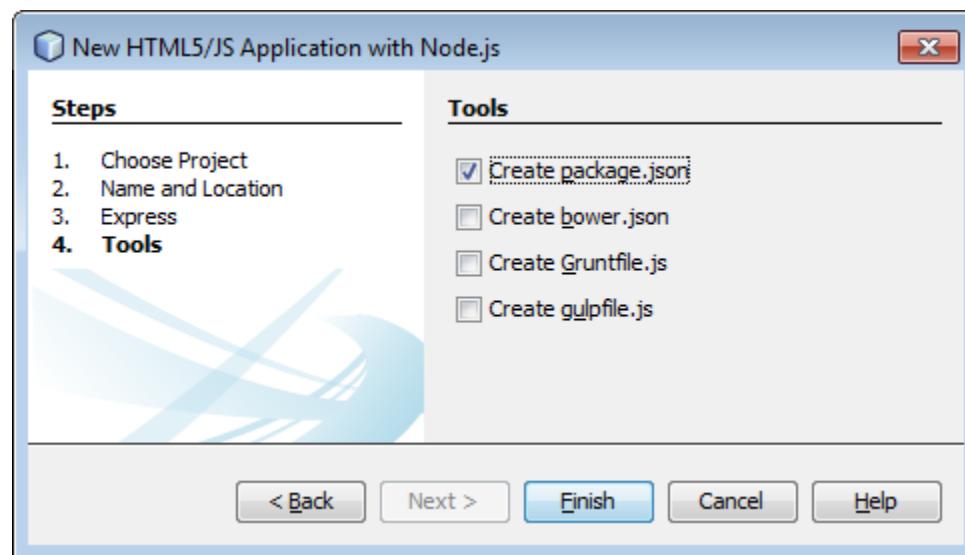
# Node.js Express app with NetBeans

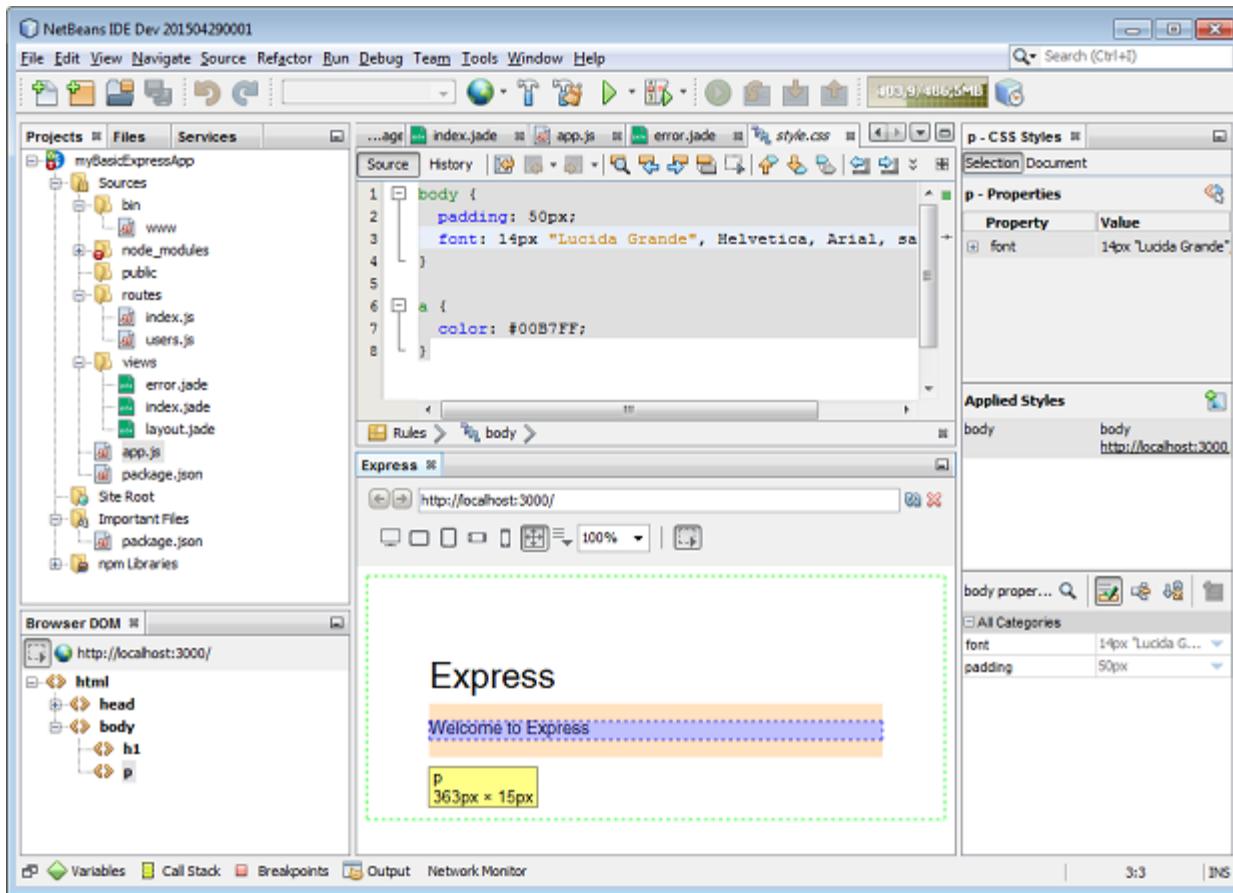


# Node.js Express app with NetBeans











# Summary



## 2. MONGO DB SCHEMA LESS



# Concepts related to a data structure

- Compare relational structure to document structure
- Demonstrate difference between table and collection
  - Tables can only hold one type of thing
  - Collections can hold a variety of things
- Demonstrate implicit schema driven by convention

# Mongod server

Database

Collection

**BSON DOCUMENT**

Key/Value pair

Key/Value pair

**BSON DOCUMENT**

Key/Value pair

Key/Value pair



# Sample Document

```
{  
  "guid": "91KA4391284",  
  "first_name": "Awase Khirni",  
  "last_name": "Syed",  
  "mobile": "+919035433124",  
  "city": "Bangalore",  
  "location": [12.712, 77.412],  
  "Profession": ["professor", "entrepreneur", "technologist"],  
  skills:[  
    { "category": "Databases",  
      "skill_list": ["SQLServer", "Oracle", "Postgresql", "MySQL", "MongoDB"]  
    },  
    { "category": "Programming",  
      "skill_list": ["java", "C#", "Python", "Ruby"]  
    },  
    { "category": "GIS",  
      "skill_list": ["ESRI", "Safe-FME", "MapServer", "GML", "Intergraph"]  
    }  
  ]  
}
```

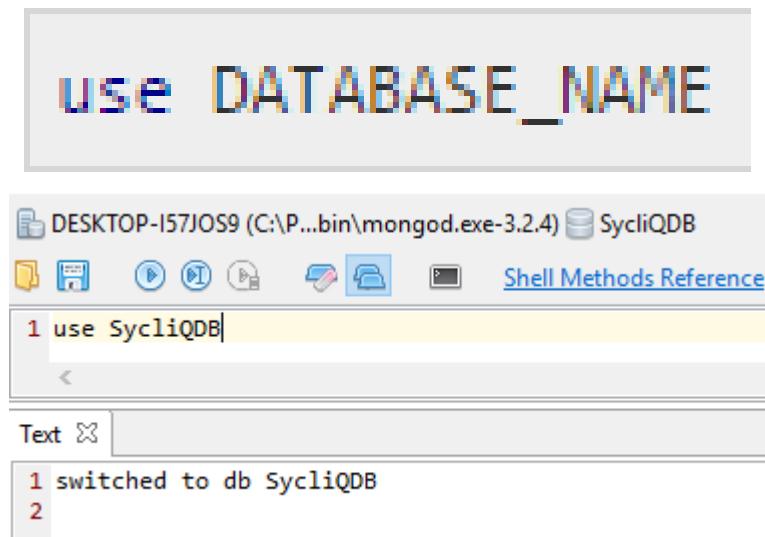
# MongoDB Document

- N-dimensional Storage
- Field can contain many values and embedded values
- Query on any field and level
- Flexible schema
- Optimal data locality requires fewer indexes and provides better performance

# Create Database

- **Use DATABASE\_NAME**  
=> to create database

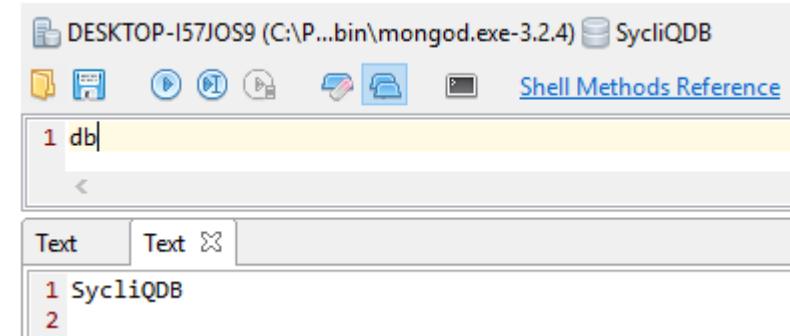
```
use DATABASE_NAME
```



The screenshot shows the MongoDB shell interface. The title bar indicates the connection is to 'DESKTOP-I57JOS9 (C:\P...bin\mongod.exe-3.2.4)' and the database is 'SycliQDB'. Below the title bar are standard shell icons. The main area has tabs for 'Text' and 'Text X'. A code input field contains the command 'use SycliQDB'. The output pane shows the response '1 switched to db SycliQDB'.

- To check your currently selected database  
=> db

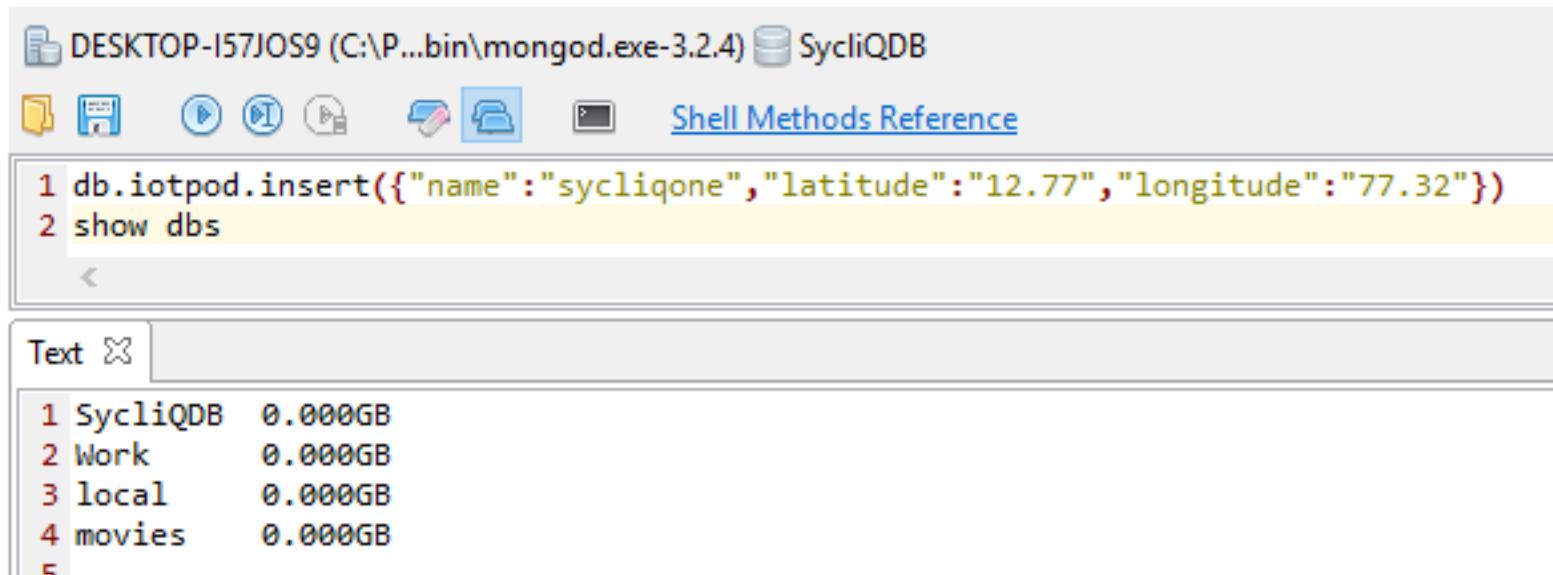
```
db
```



The screenshot shows the MongoDB shell interface. The title bar indicates the connection is to 'DESKTOP-I57JOS9 (C:\P...bin\mongod.exe-3.2.4)' and the database is 'SycliQDB'. Below the title bar are standard shell icons. The main area has tabs for 'Text' and 'Text X'. A code input field contains the command 'db'. The output pane shows the response '1 SycliQDB'.

# Database

- To check the list of databases
  - **Show dbs**
  - Insert document into the database.
    - **db.iotpod.insert({})**



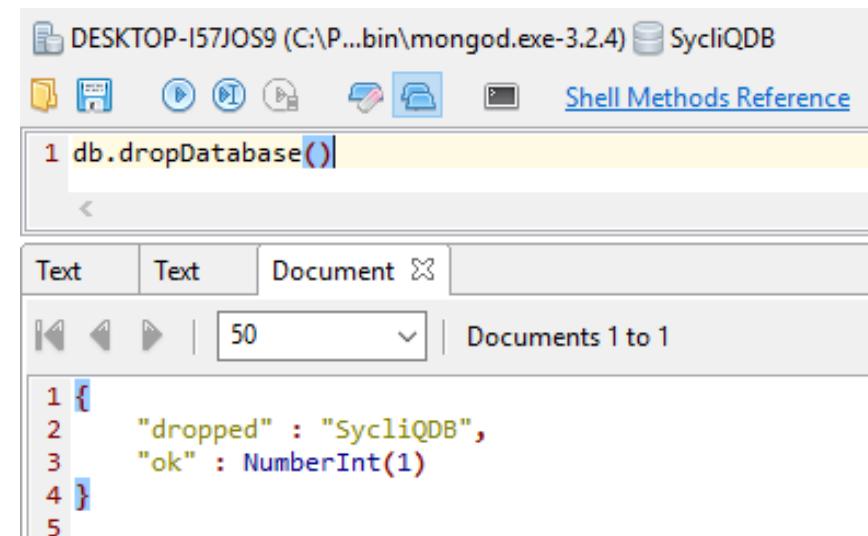
```
DESKTOP-I57JOS9 (C:\P...bin\mongod.exe-3.2.4) [SycliQDB]
File Edit View Help Shell Methods Reference

1 db.iotpod.insert({"name":"sycliqone","latitude":"12.77","longitude":"77.32"})
2 show dbs

Text
1 SycliQDB 0.000GB
2 Work 0.000GB
3 local 0.000GB
4 movies 0.000GB
5
```

# dropDatabase()

- List available databases using the command
  - **Show dbs**
- to delete new database
  - **Use SycliQDB**
  - **db.dropDatabase()**
- **Verify**
  - **Show dbs**

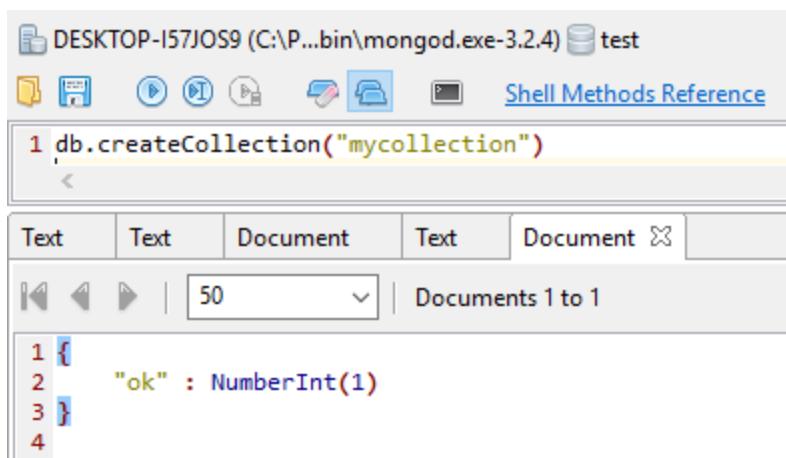


The screenshot shows the MongoDB shell interface. The title bar indicates the session is running on DESKTOP-I57JOS9 (C:\P...bin\mongod.exe-3.2.4) and connected to the database SycliQDB. The toolbar includes icons for file operations and help. A menu bar at the top right has "Shell Methods Reference". The main area contains a code editor with the command `db.dropDatabase()` highlighted in red. Below it is a results pane showing the output of the command:

```
1 {  
2   "dropped" : "SycliQDB",  
3   "ok" : NumberInt(1)  
4 }  
5
```

# Collections

- To create **collections** in a database
  - **db.createCollection(name,options)**



The screenshot shows the MongoDB shell interface. The title bar says "DESKTOP-I57JOS9 (C:\P...bin\mongod.exe-3.2.4) test". Below the title bar are several icons. The main area contains the command "1 db.createCollection("mycollection")". The status bar at the bottom shows "Text Text Document Text Document" and "Documents 1 to 1". The bottom pane displays the response to the command:

```

1 {
2   "ok" : NumberInt(1)
3 }
4

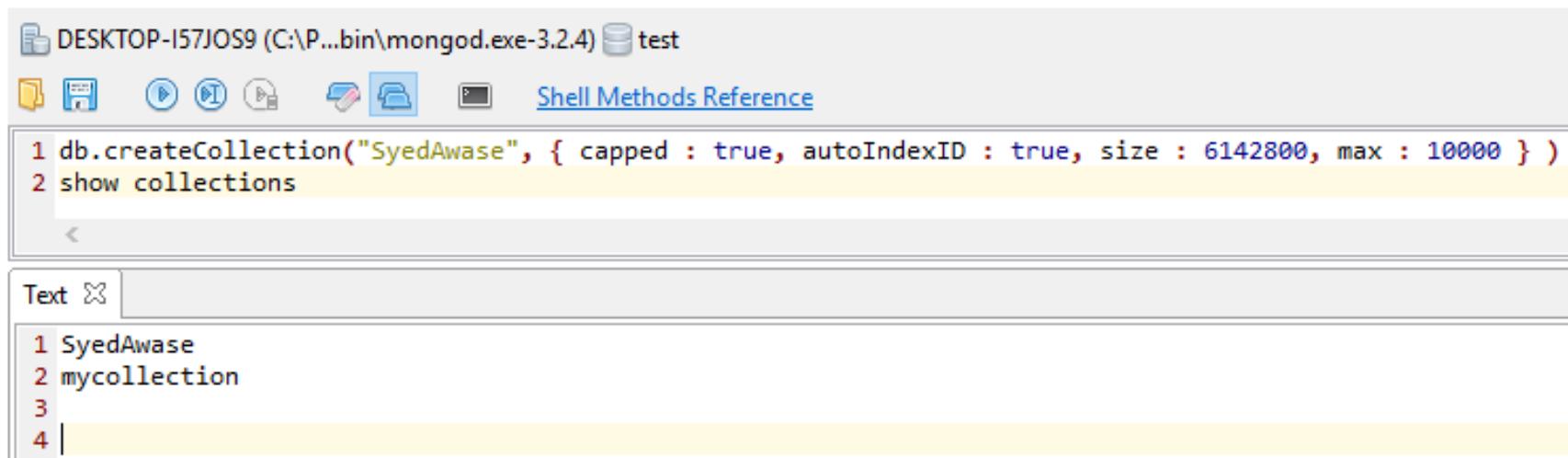
```

Parameter	Type	Description
Name	String	Name of the collection to be created
Options	Document	(Optional) Specify options about memory size and indexing

Field	Type	Description
capped	Boolean	(Optional) If true, enables a capped collection. Capped collection is a collection fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. <b>If you specify true, you need to specify size parameter also.</b>
autoIndexID	Boolean	(Optional) If true, automatically create index on _id field.s Default value is false.
size	number	(Optional) Specifies a maximum size in bytes for a capped collection. <b>If capped is true, then you need to specify this field also.</b>
max	number	(Optional) Specifies the maximum number of documents allowed in the capped collection.

# Collections

- `createCollection()` method with few important options



The screenshot shows the MongoDB shell interface. The title bar indicates the connection is to DESKTOP-I57JOS9 (C:\P...bin\mongod.exe-3.2.4) and the database is test. Below the title bar are standard shell icons. A toolbar below the title bar includes a file icon, a save icon, a back icon, a forward icon, a search icon, a help icon, and a shell methods reference link. The main area contains two code snippets. The top snippet shows the command to create a capped collection:

```
1 db.createCollection("SyedAwase", { capped : true, autoIndexID : true, size : 6142800, max : 10000 } )  
2 show collections
```

The bottom snippet shows the output of the `show collections` command, listing the collections "SyedAwase" and "mycollection".

```
Text X  
1 SyedAwase  
2 mycollection  
3  
4 |
```

# Collections

- MongoDB creates collections automatically, when we insert some document



The screenshot shows the MongoDB shell interface. The title bar indicates the connection is to DESKTOP-I57JOS9 (C:\P...bin\mongod.exe-3.2.4) and the database is test. Below the title bar is a toolbar with various icons. The main area contains two lines of code:

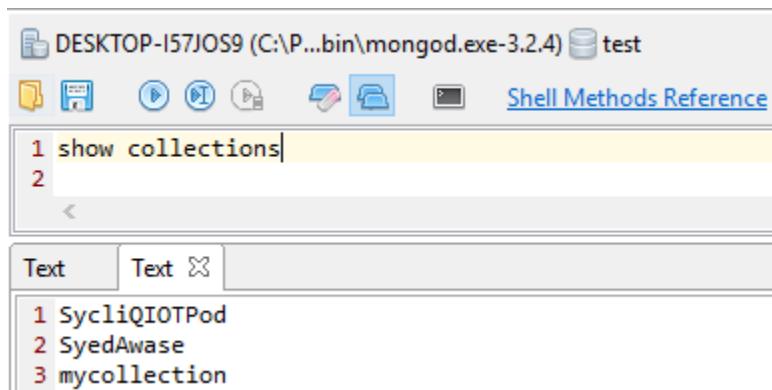
```
1 db.SycliQIOTPod.insert({"name": "SycliQOne", "Latitude": "12.77", "longitude": "77.87"})  
2 show collections
```

After running the first command, the output window below shows the results of the second command:

```
1 SycliQIOTPod  
2 SyedAwase  
3 mycollection
```

# db.collection.drop()

- Used to drop a collection from the database.

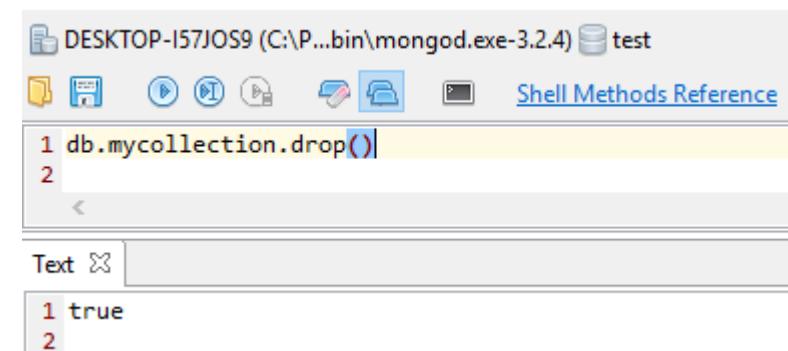


```
DESKTOP-I57JOS9 (C:\P...bin\mongod.exe-3.2.4) test
Shell Methods Reference

1 show collections
2

Text Text ×

1 SycliQIOTP Pod
2 SyedAwase
3 mycollection
```

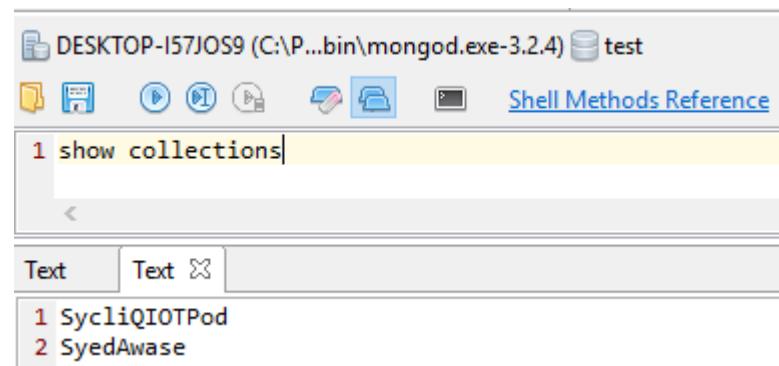


```
DESKTOP-I57JOS9 (C:\P...bin\mongod.exe-3.2.4) test
Shell Methods Reference

1 db.mycollection.drop()
2

Text ×

1 true
2
```



```
DESKTOP-I57JOS9 (C:\P...bin\mongod.exe-3.2.4) test
Shell Methods Reference

1 show collections
2

Text Text ×

1 SycliQIOTP Pod
2 SyedAwase
```

# Data Types

Type	Description
String	Most commonly used datatype to store the data UTF-8
Integer	Used to store a numerical value (32 bit or 64 bit)
Boolean	True/false value
Double	Floating point values
Min/Max Keys	Used to compare a value against the lowest and highest BSON elements
Arrays	To store arrays or lists or multiple values into one key
Timestamp	For recording when a document has been modified or added
Object	Used for embedded documents
Null	Used to store a <b>null</b> value
Symbol	Used identically to a string and generally reserved for languages that use a specific symbol type
Date	To store current date or time in UNIX time format
ObjectId	Used to store the document's ID
Binary data	Used to store binary data
Code	Used to store javascript code into document
Regular expression	Used to store regular expression

# Document

- A document is the basic unit of data for MongoDB and is roughly equivalent to a row in a relational database management system.
- A document is a set of field-value pairs.
- Stores documents as BSON Serialization format

```
{  
  "_id": ObjectId("1231231sda435435sadas"),  
  "name": {  
    "first": "Awase",  
    "last": "Syed"  
  },  
  "email": "sak@sycliq.com",  
  "major": "Geographic Information Systems"  
}
```

## Name Restrictions

- Key must not contain the \0 (null characters)
- the (.) and (\$) characters have some special properties
- MongoDB is typed-sensitive and case-sensitive
- MongoDB cannot contain duplicate keys

# Insert() Method

- To insert data into MongoDB collection
  - db.COLLECTION\_NAME.insert(document)

```
1 db.SyedAwase.insert({  
2   title: 'Syed Awase Khirni',  
3   description: 'A researcher in geographic information science domain',  
4   designation: 'researcher',  
5   url: 'www.sycliq.com',  
6   tags: ['syedawase', 'spirit', 'geo-spirit','sycliq'],  
7   likes: 100  
8 });  
  
<  
Text Text Text X  
1 WriteResult({ "nInserted" : 1 })  
2
```

```
_id" : ObjectId("570920f367a6718b39367126"),  
"title" : "Syed Awase Khirni",  
"description" : "A researcher in geographic information science domain",  
"designation" : "researcher",  
"url" : "www.sycliq.com",  
"tags" : [  
  "syedawase",  
  "spirit",  
  "geo-spirit",  
  "sycliq"  
],  
"likes" : 100.0
```



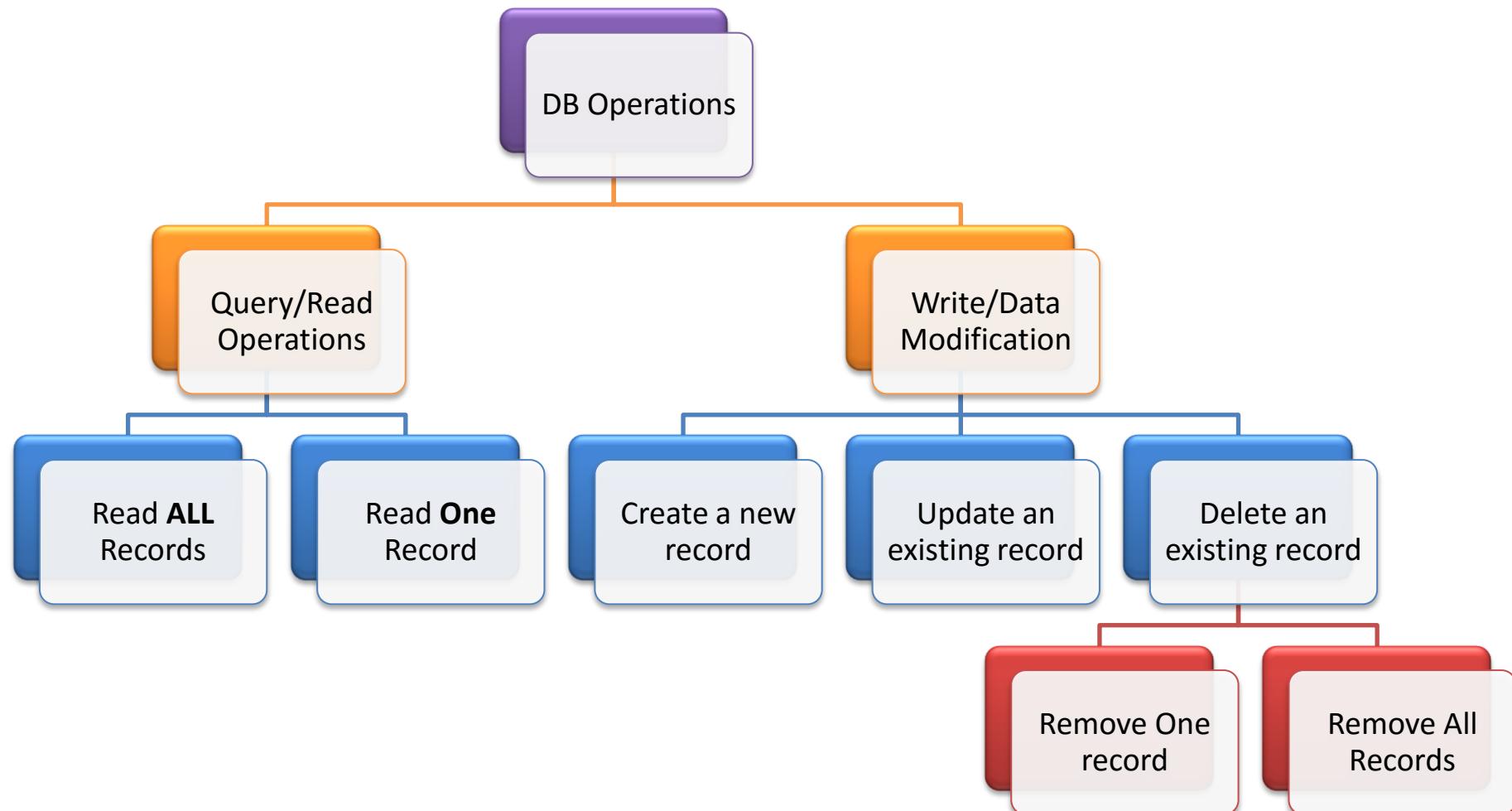
# \_id

`_id` is 12 bytes hexadecimal number unique for every document in a collection.  
12 bytes are divided as follows

```
_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer)
```

```
{  
    "_id" : ObjectId("570920f367a6718b39367126"),  
    "title" : "Syed Awase Khirni",  
    "description" : "A researcher in geographic information science domain",  
    "designation" : "researcher",  
    "url" : "www.sycliq.com",  
    "tags" : [  
        "syedawase",  
        "spirit",  
        "geo-spirit",  
        "sycliq"  
    ],  
    "likes" : 100.0  
}
```

# DB Operations



# Find()

- db.Collection\_Name.find()
- Used to query the data in the MongoDB
- To display all the documents in an unstructured way
- db.Collection\_Name.find().pretty()
- To display documents in an organized way with json pretty print

# FindOne()

- Db.Collection\_Name.findOne()
- Returns one document that satisfies the specified query criteria

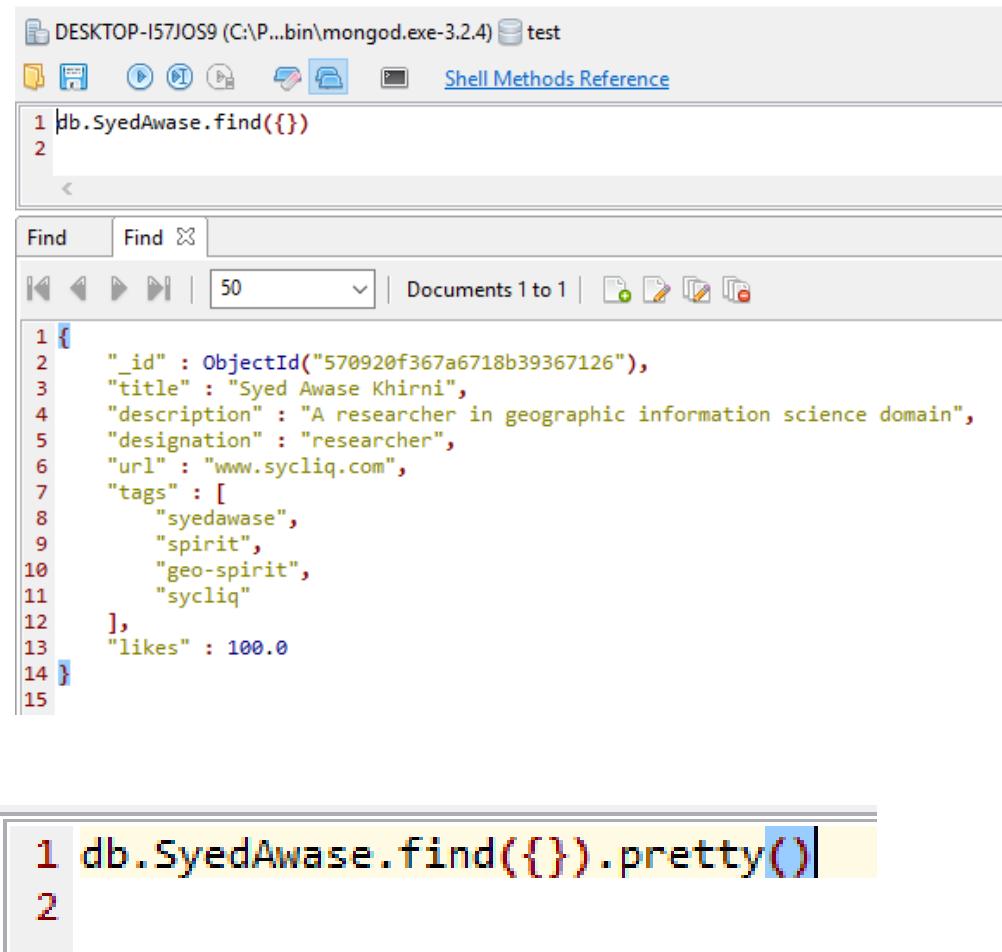
# Show

- dbs : show a list of databases
- Collections : show a list of collections

## READ OPERATIONS

# Find() method

- To query data from MongoDB collection, we use MongoDB's find() method
- To display the results in a formatted way, you can use pretty() method.

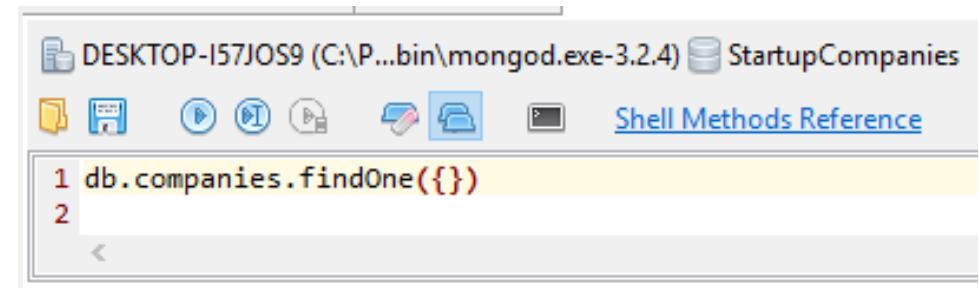


The screenshot shows the MongoDB shell interface. The title bar indicates the session is on DESKTOP-I57JOS9 (C:\P..bin\mongod.exe-3.2.4) connected to the test database. The main area displays the results of a find operation on the SyedAwase collection. The results are shown in a JSON document, with line numbers 1 through 15 on the left. The document contains fields such as \_id, title, description, designation, url, tags, and likes. Below this, another command is shown: db.SyedAwase.find({}).pretty(). This command is intended to format the output of the previous find operation for better readability.

```
1 db.SyedAwase.find({})
2
3
4
5
6
7
8
9
10
11
12
13
14
15
1 db.SyedAwase.find({}).pretty()
2
```

# findOne()

- Returns one document that satisfies the specified query criteria.
- In the event multiple documents satisfy the query, this method returns the first document according to the natural order which reflects the order of documents on the disk



The screenshot shows the MongoDB shell interface. The title bar indicates the connection is to DESKTOP-I57JOS9 (C:\P...bin\mongod.exe-3.2.4) and the database is StartupCompanies. Below the title bar, there are several icons for file operations like New, Open, Save, and Print. To the right of the icons is a link to "Shell Methods Reference". The main pane of the shell contains two lines of code:  
1 db.companies.findOne({})  
2

- In capped collections, natural order is the same as insertion order
- if no document satisfies the query, the method returns null.

Parameter	Type	Description
query	document	Optional. Specifies query selection criteria using <a href="#">query operators</a> .
projection	document	Optional. Specifies the fields to return using <a href="#">projection operators</a> . Omit this parameter to return all fields in the matching document.

# Query Operations in MongoDB

Operation	Syntax	Example
Equality	{<key>:<value>}	{"founded_year":2000}
Less Than	{<key>:{\$lt:<value>}}	{"founded_year":{\$lt:2000}}
Less Than Equals	{<key>:{\$lte:<value>}}	{"founded_year":{\$lte:2000}}
Greater Than	{<key>:{\$gt:<value>}}	{"founded_year":{\$gt:2000}}
Greater Than Equals	{<key>:{\$gte:<value>}}	{"founded_year":{\$gte:2000}}
Not Equals	{<key>:{\$ne:<value>}}	{"founded_year":{\$ne:2000}}

# Create

## **Use Database\_Name**

- Used to create the database , if it does not exist

## **db.createCollection(name)**

- Used to create a new collection

## **db.Collection\_Name.insert(document)**

- Used to insert the documents into the collection

# Update

## **db.Collection\_Name.update(Selection\_criteria, Updated\_Data)**

- To update the documents of a collection
- Update specific field
- Update an embedded field
- Update multiple documents
- Replace entire document

# Delete

## **db.Collection\_Name.remove(D eletion\_criteria)**

- Used to delete the documents from a collection based on two parameters
- Remove all documents
- Remove documents that match a condition

## **db.dropDatabase()**

- Removes the current database, deleting the associated data files

# WRITE OPERATIONS

Flexible Schema

Collections do not enforce document structure

Facilitate the mapping of documents to an entity or an object

## DATA MODELLING IN MONGODB

Challenge

Balancing the needs of the application

Challenge

Performance characteristics of the database engine

Challenge

Data Retrieval Patterns

# Data Models for MongoDB

References

Embedded  
Data



# MODEL RELATIONSHIPS BETWEEN DOCUMENTS



# References

- References store the relationships between data by including links or references from one document to another.
- Applications can resolve these references to access the related data
- References provide more flexibility than embedding.
- Client-side application must issue follow-up queries to resolve the references.
- Normalized data models describe relationships using references between documents.
- Use normalized data models
  - When embedding would result in duplication of data but would not provide sufficient read performance advantages to outweigh the implications of the duplication
  - To represent more complex many to many relationships
  - To model large hierarchical datasets.

# References

## Manual References

- Using `_id` field of one document in another document as a reference.
- Your application can run a second query to return the related data.
- Reference are simple and sufficient for most use cases

## DBRefs

- References from one document to another using the value of the first document's `_id` field, collection name and optionally, its database name.
- DBRefs allow documents located in multiple collections to be more easily linked with documents from a single collection

# Embedding

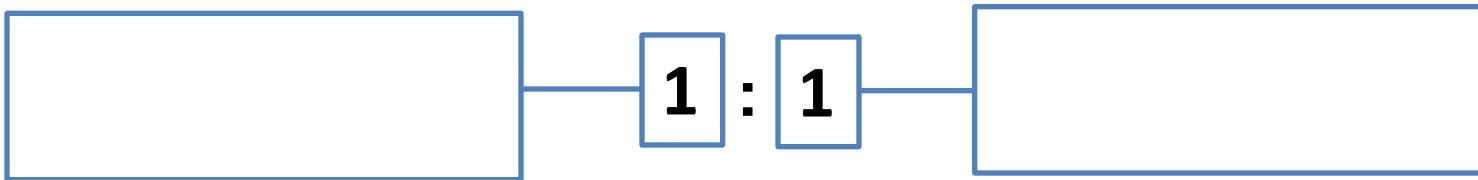
- Embedded documents capture relationships between data by storing related data in a **single document structure**.
- **MongoDB documents make it possible to embed document structures in a field or array within a document.**
  - These denormalized data models allow applications to retrieve and manipulate related data in a single database operation
  - Denormalized data model with embedded data combines all related data for a represented entity in a single document
    - Atomic write operations can insert or update the data for an entity.



# Schema Design

- Data **needed** by your application?
- Data **Read** by your application?
- Data **written** by your application?

# One-To-One

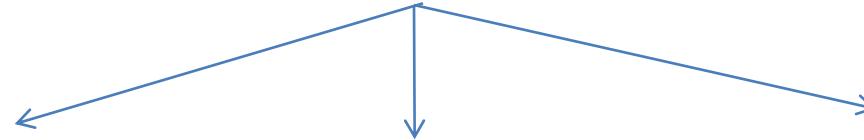


## Strategy?

Linking

Embedding

Bucket List



# 1:1 Embedding

- Strategy: Embedding
  - Embed the document within
  - Embed the vehicle object inside the user object.

```
{  
  "name": "Syed Awase Khirni",  
  "gender": "male",  
  "driving_license": true,  
  vehicle:{  
    "vehicle_type": "car",  
    "vehicle_name": "mercedes",  
    "vehicle_cc": 2000  
  }  
}
```

# 1:1 Linking

- Strategy: Linking
  - Link the vehicle and the user document using a **foreign key**

MongoDB does not enforce any foreign key constraints so the relation only exists as part of the application level schema

User Object

```
{  
  "guid": "91ka43sak212312",  
  "name": "Syed Awase Khirni",  
  "gender": "male",  
  "driving_license": true,  
}  
}
```

Vehicle Object

```
{  
  "user_guid": "91ka43sak212312",  
  "vehicle_type": "car",  
  "vehicle_name": "mercedes",  
  "vehicle_cc": 2000  
}  
}
```

# 1:1 Referencing

## Advantages

- Smaller documents
- Less likely to reach 16MB document limit
- Infrequently accessed information not accessed on every query

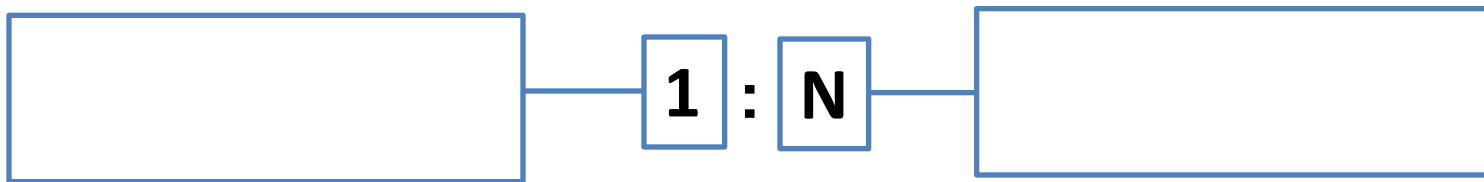
## Disadvantages

- Two queries required to retrieve information
- Cannot update related information **atomically**

# 1:1 Preferred Approach

- Strategy: Embedding is the **preferred way to model the relationship as it's a more efficient way to retrieve the document.**
- **Avoid implementing joins in application code**
- **Update related information as a single atomic operation**
- “belongs to” relationships are better embedded.
- Good for read operations/performance
- Keeps things simple
- Holistic representation of entities with their embedded attributes and relationships
- **large documents mean overhead- max document size limit is 16 MB**

# One-To-Many

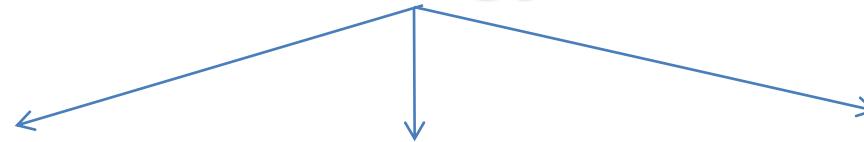


## Strategy?

Linking

Embedding

Bucket List



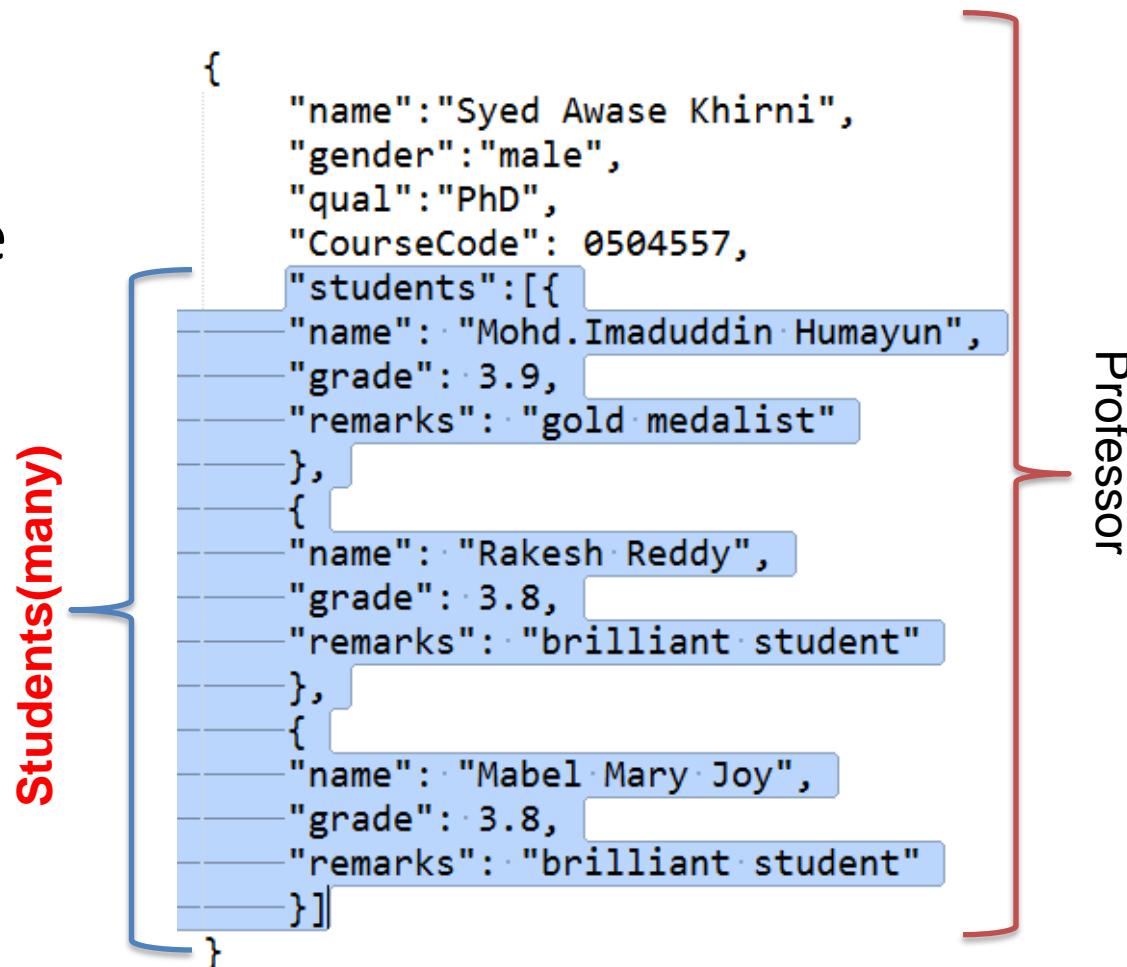
# 1:M Embedding

A Customer has many orders

```
{  
  "_id" : 123,  
  "name" : "syed awase khirni",  
  "zipcode" : "560043",  
  "orders" : [  
    {  
      "_id" : 100,  
      "product_title": "mamypoko pants",  
      "category": "diapers"  
      "price" : 579,  
      "status" : 2  
    },  
    {  
      "_id" : 100,  
      "product_title": "dell latitude e6430",  
      "category": "laptop"  
      "price" : 57900,  
      "status" : 1  
    },  
    {  
      "_id" : 100,  
      "product_title": "BenQ projector",  
      "category": "equipment"  
      "price" : 27900,  
      "status" : 3  
    }  
  ]  
}
```

# 1:M Embedding

- Strategy: Embedding
- Easy retrieval of all the students in a single read.

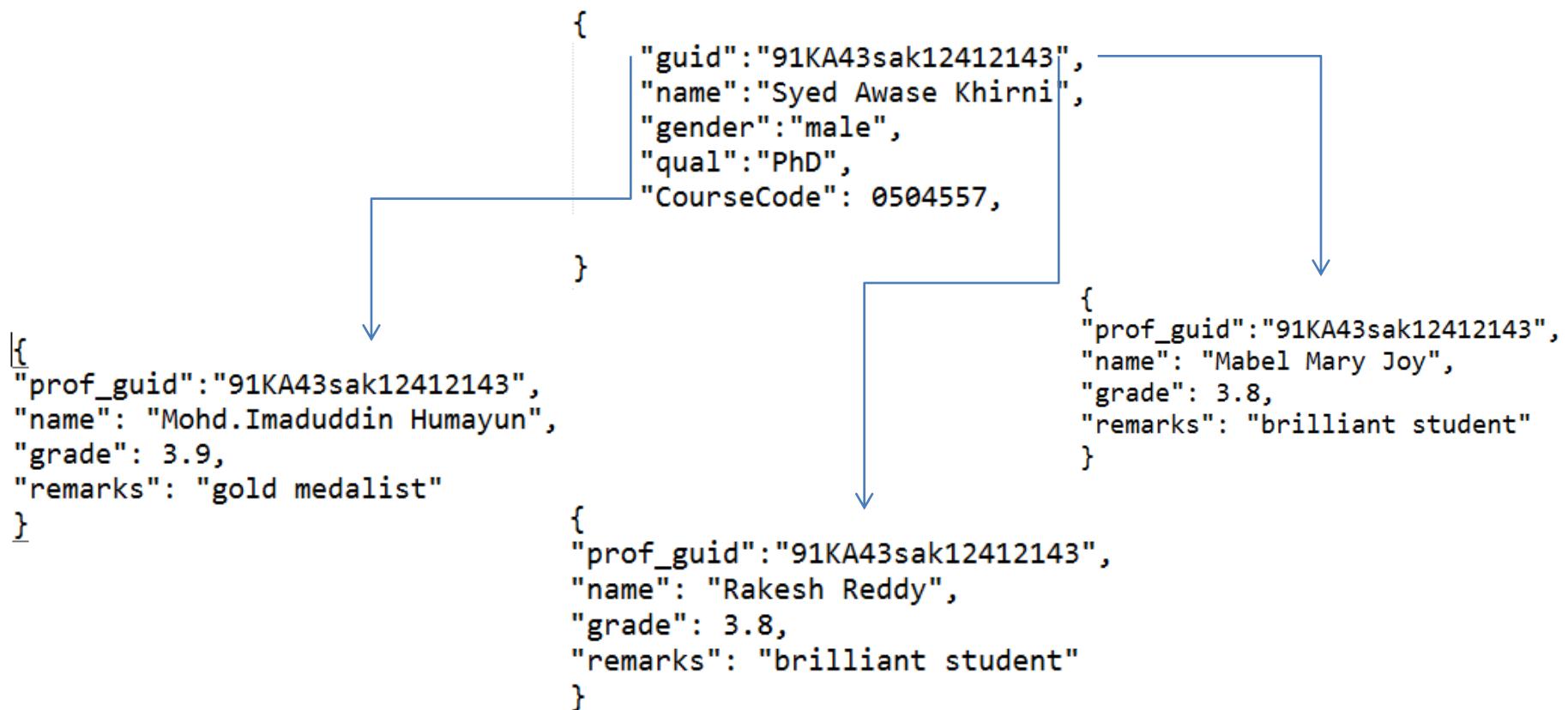


# Challenges with the Embedding Strategy

- Students array might grow larger than the maximum document size i.e 16 MB
- Write performance: it makes it hard for MongoDB to predict the correct document padding to apply when a new document is created.
  - the document has to be moved around in memory as it grows causing additional IO and impacting write performance
  - only matters for : high write traffic and might not be problem for smaller applications
- Filtering of students returned from the single Professor, will require to **retrieve all the students and filter them out.**
  - Performing pagination off the students and sorting will require fetching all the students.

# 1:M Linking

- Strategy: Linking



# 1:M Linking Approach

- Unlikely that the maximum document size of 16mb will be reached.
- Easier to filter and sort the students **BUT, we will have to fetch all the 1000 student documents causing a lot of reads from the database**

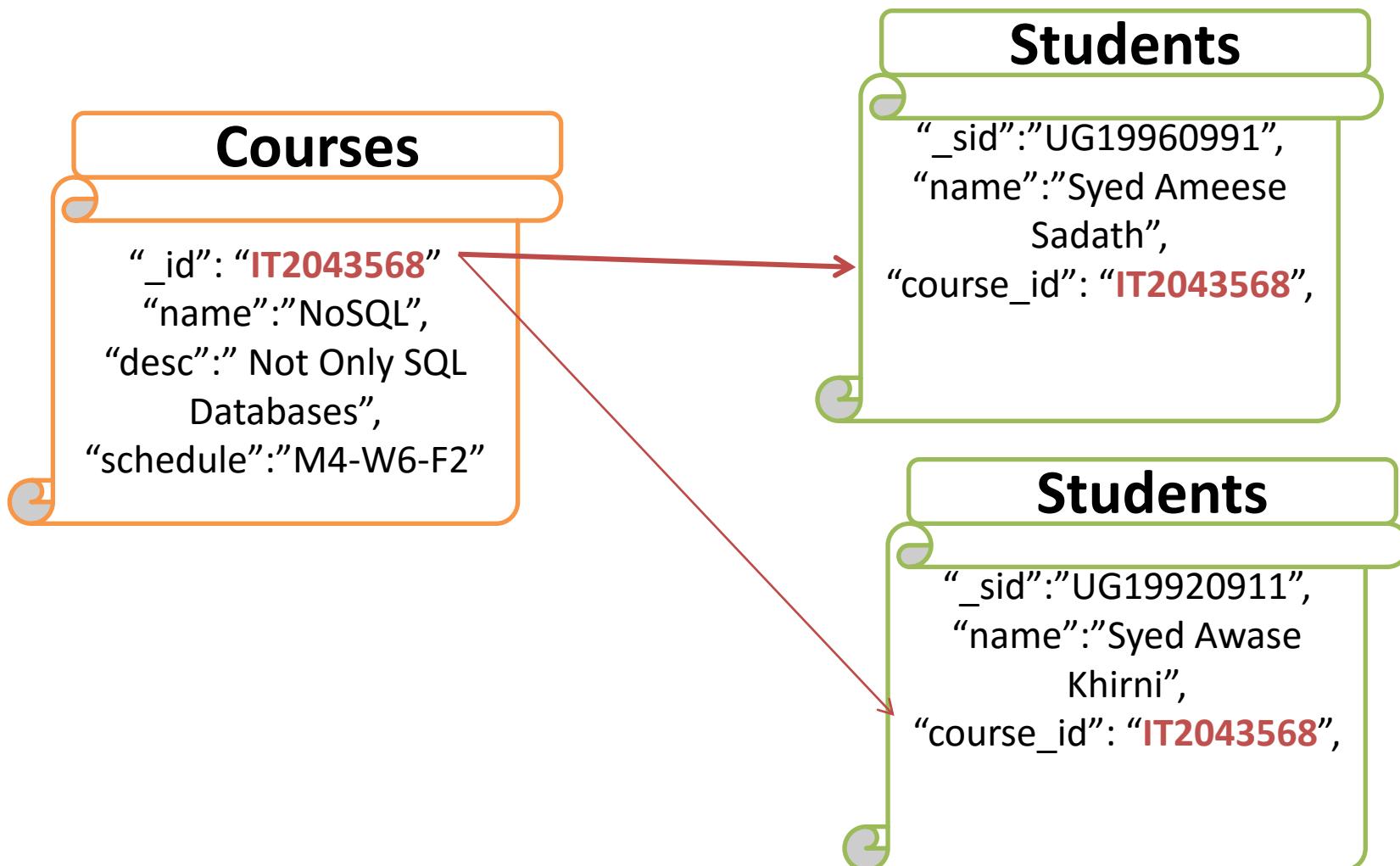
# 1:M Bucketing

```
{  
    "guid": "91KA43sak12412143",  
    "name": "Syed Awase Khirni",  
    "gender": "male",  
    "qual": "PhD",  
    "CourseCode": 0504557,  
}  
  
{  
    "prof_guid": "91KA43sak12412143",  
    "page": 2,  
    "count": 50,  
    "year": "2006"  
    "students": [ {  
        "name": "Mohd Hasan Ahmed",  
        "grade": 3.7,  
        "remarks": "brilliant student"  
    },  
    {  
        "name": "Prachi",  
        "grade": 3.8,  
        "remarks": "brilliant student"  
    }  
]  
}  
  
{  
    "prof_guid": "91KA43sak12412143",  
    "page": 1,  
    "count": 50,  
    "year": "2007"  
    "students": [ {  
        "name": "Mohd.Imaduddin Humayun",  
        "grade": 3.9,  
        "remarks": "gold medalist"  
    },  
    {  
        "name": "Rakesh Reddy",  
        "grade": 3.8,  
        "remarks": "brilliant student"  
    }  
]  
}
```

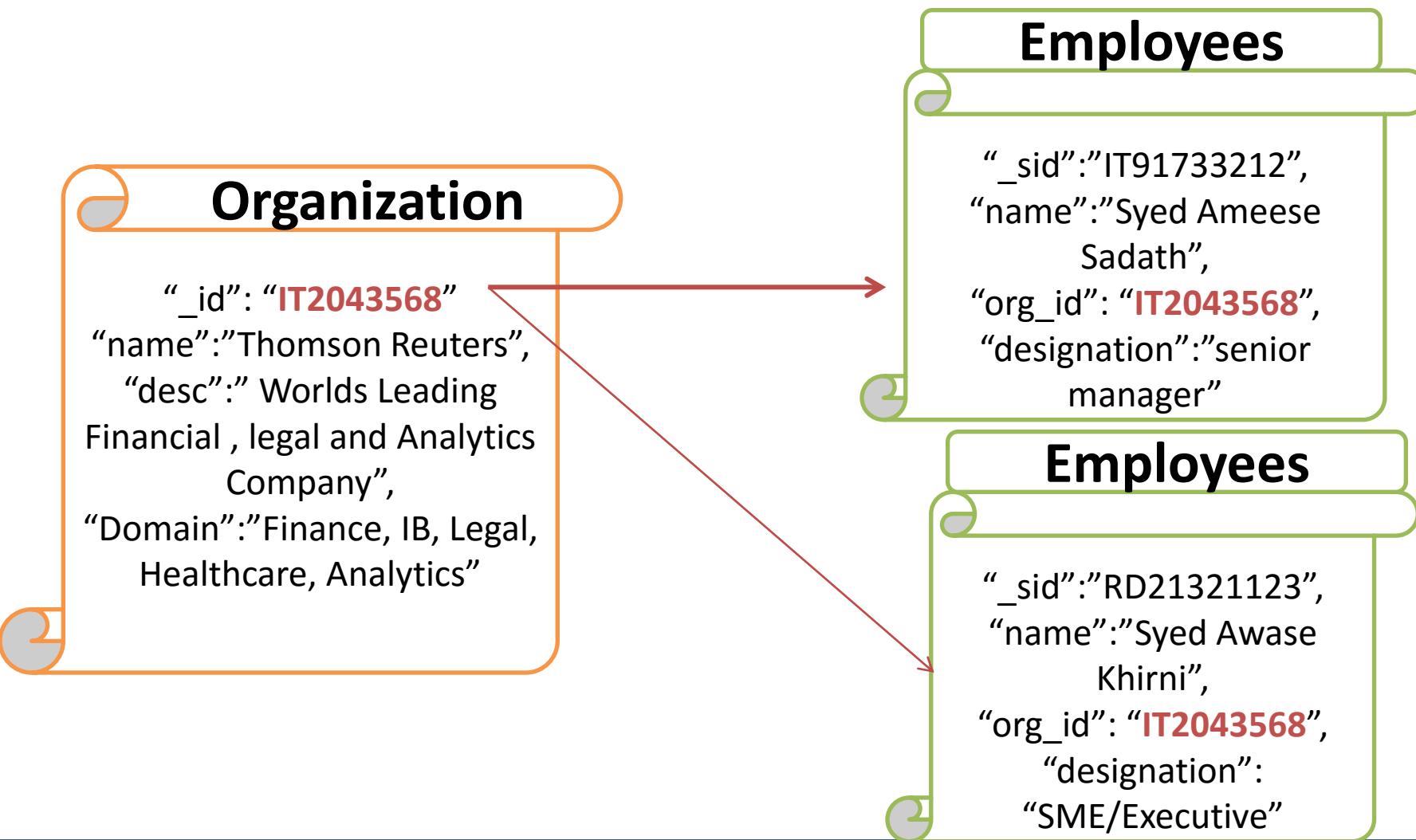
# 1:M bucketing Approach

- Possibility of splitting up your documents in discreet batches it makes sense to consider bucketing to speed up retrieval of documents.
- Bucketing data by hours, days or number of entries on a page.  
Etc..

# Manual Reference Example



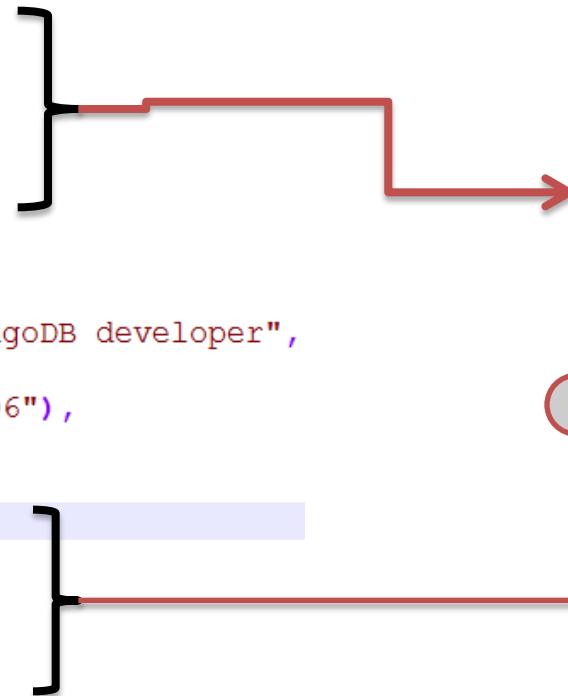
# Manual Reference Example 2



# One to Many relationship

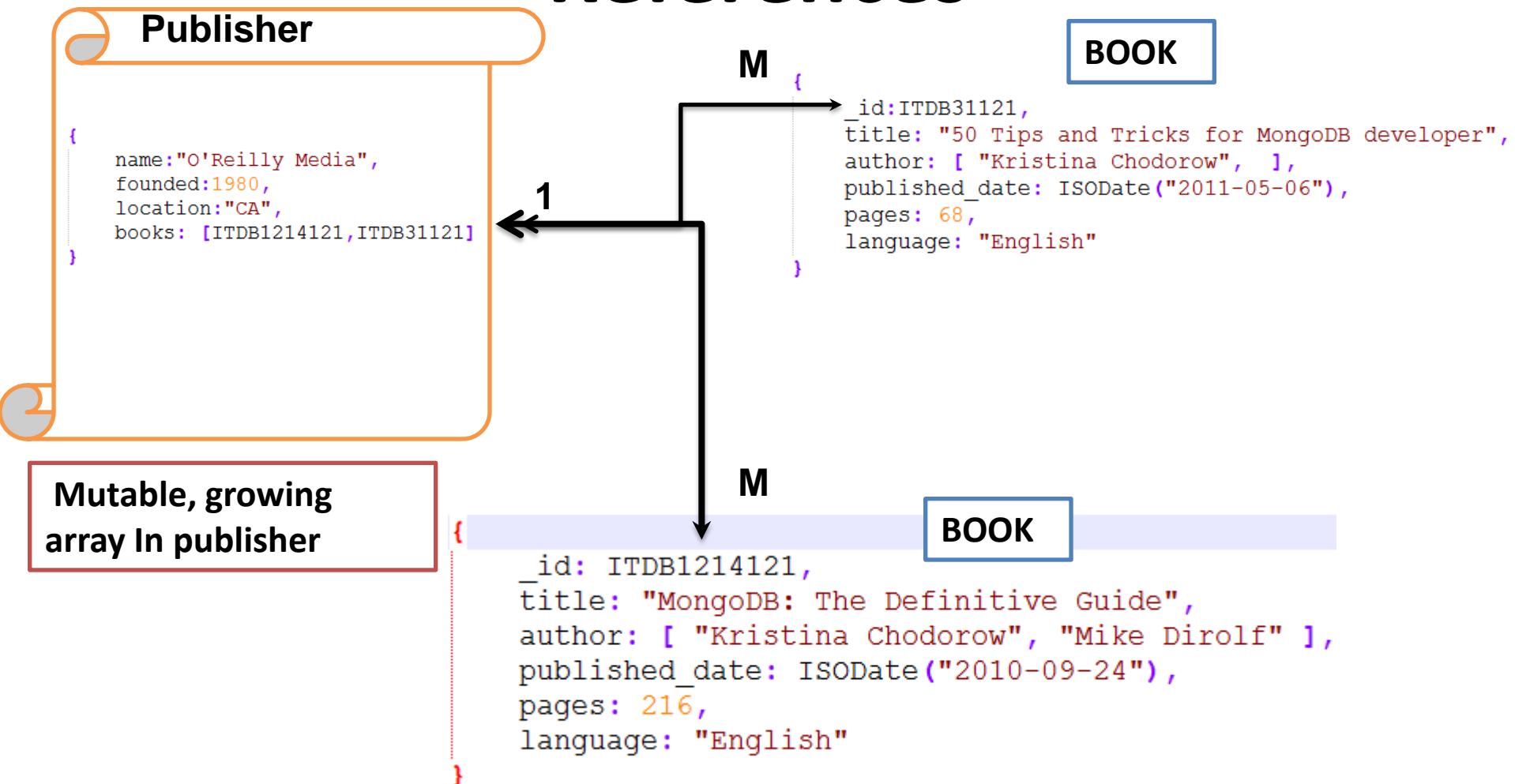
```
{  
    title: "MongoDB: The Definitive Guide",  
    author: [ "Kristina Chodorow", "Mike Dirolf" ],  
    published_date: ISODate("2010-09-24"),  
    pages: 216,  
    language: "English",  
    publisher: {  
        name: "O'Reilly Media",  
        founded: 1980,  
        location: "CA"  
    }  
}
```

```
{  
    title: "50 Tips and Tricks for MongoDB developer",  
    author: [ "Kristina Chodorow", ],  
    published_date: ISODate("2011-05-06"),  
    pages: 68,  
    language: "English",  
    publisher: {  
        name: "O'Reilly Media",  
        founded: 1980,  
        location: "CA"  
    }  
}
```

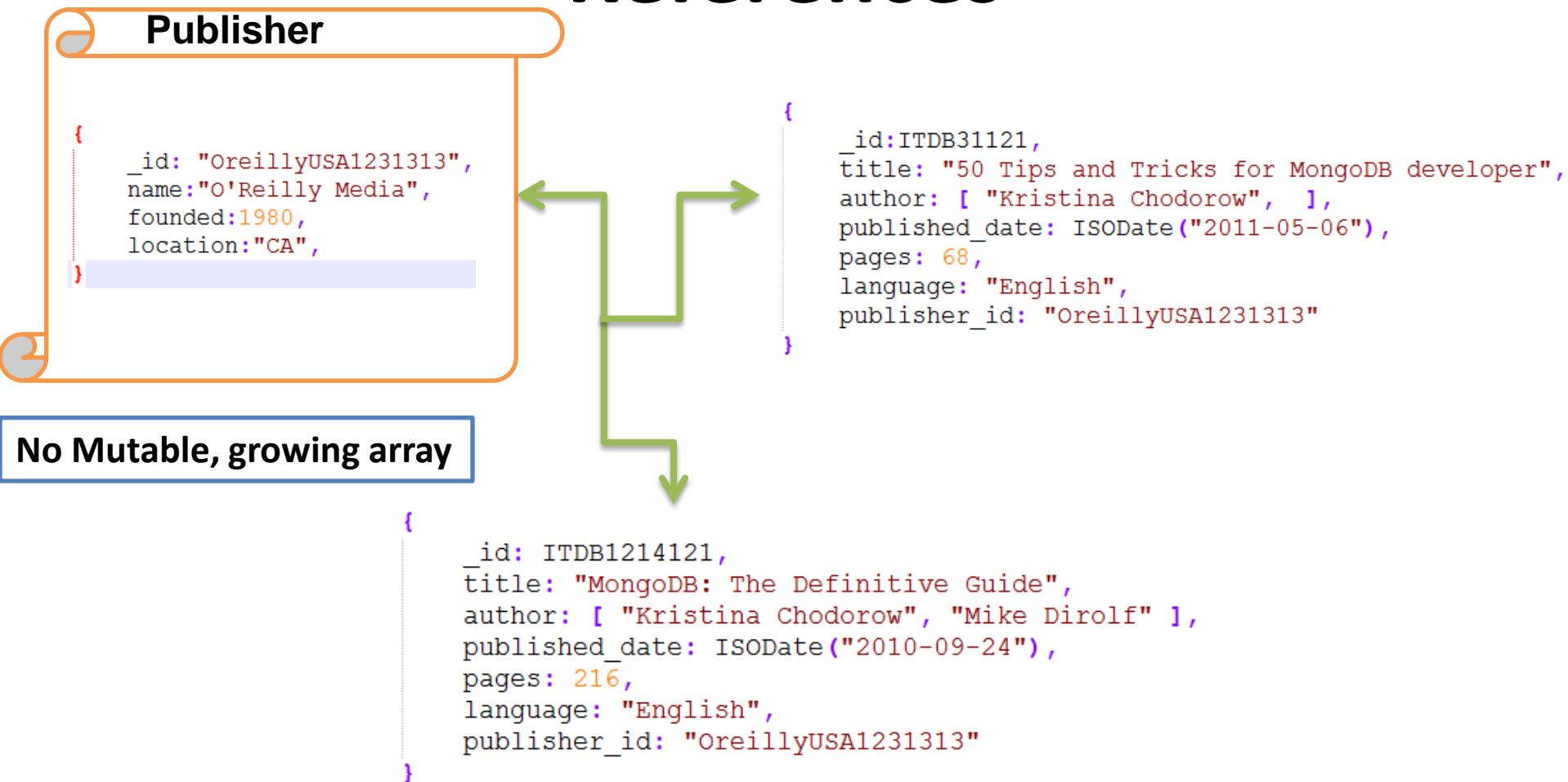


**Embedding the publisher**  
document inside  
the book  
document leads  
to **repetition of Publisher data**

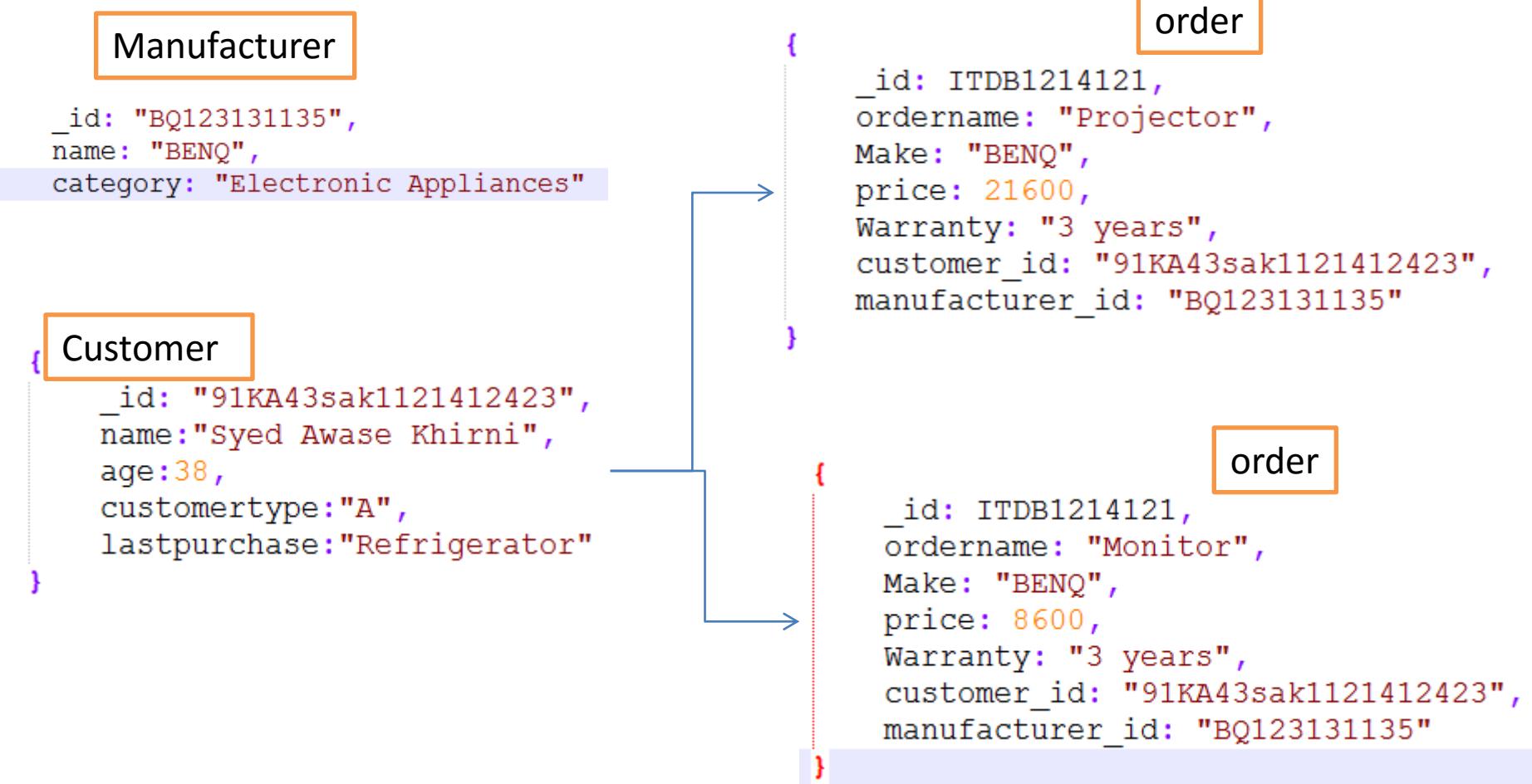
# 1-Many Relationship using References



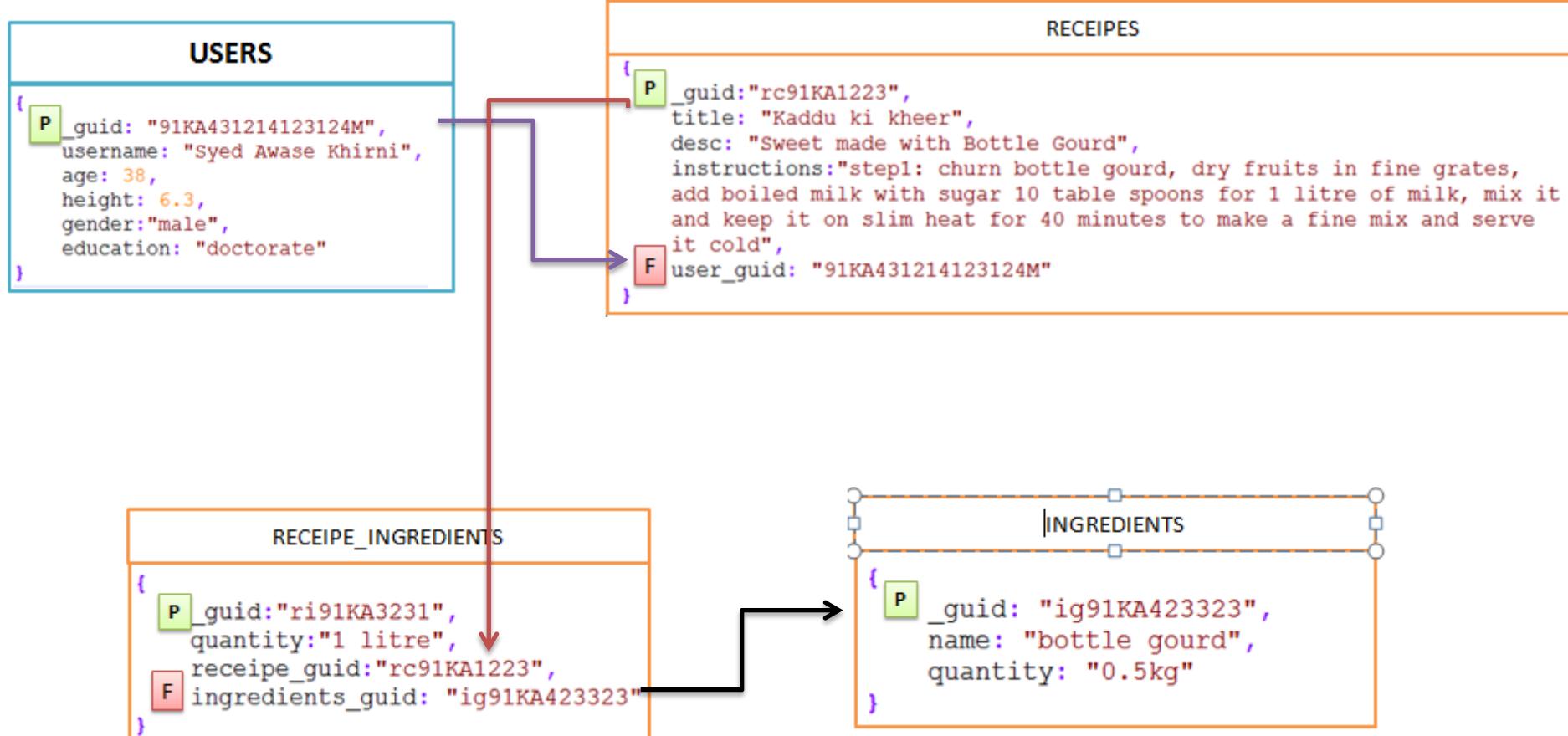
# 1-Many Relationship with References



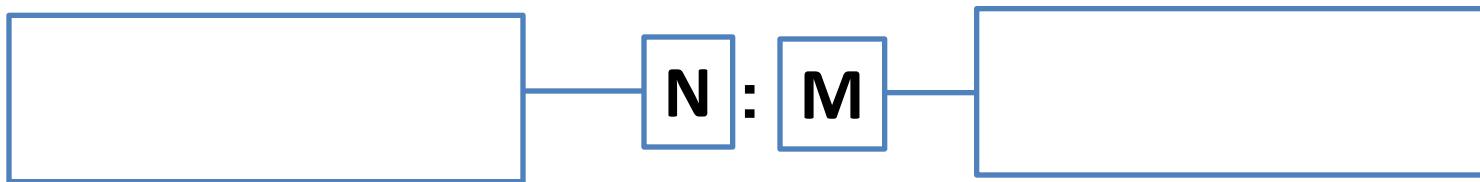
# 1-Many Relationship with References



# Example With References



# Many-To-Many

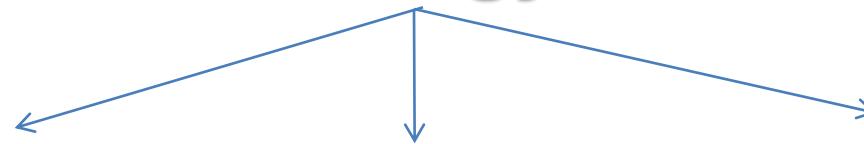


## Strategy?

Linking

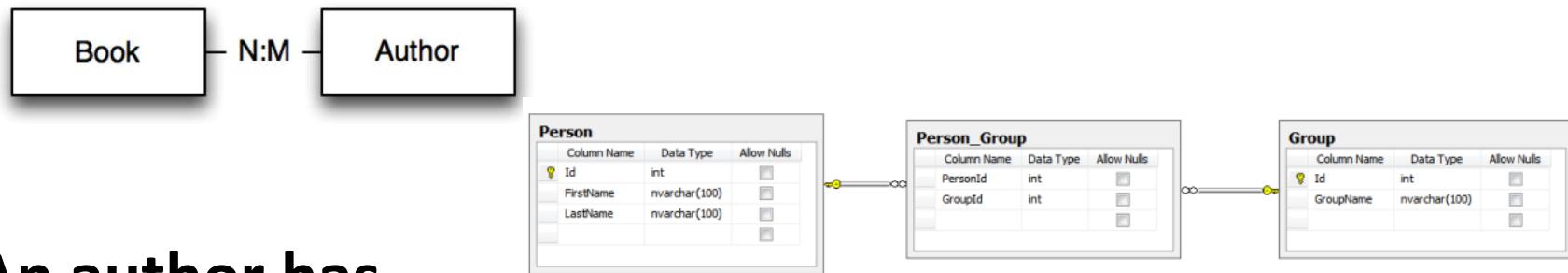
Embedding

Bucket List



# Many to Many Relationships in RDBMS

- They are modeled by using a **join table**.



- An author has authored multiple books and a book can be written by multiple authors.

# Two way embedding

```
{  
  "author_guid": "1MAR1212MGF",  
  "author_name": "Robert Sedgewick",  
  "books": [032157351X, 0321498054]  
}
```

```
{  
  "author_guid": "1TXK127126YH",  
  "author_name": "Kevin Wayne",  
  "books": [032157351X, 0321498054]  
}
```



**Embedding keys in both the entities to establish relationship : TWO WAY EMBEDDING**

```
{  
  "isbn": "032157351X",  
  "title": "Algorithms",  
  "authors": [1MAR1212MGF, 1TXK127126YH]  
}
```

```
{  
  "isbn": "0321498054",  
  "title": "Introduction to Programming in Java: An Interdisciplinary Approach",  
  "authors": [1MAR1212MGF, 1TXK127126YH]  
}
```

# Two Embedding

- Two queries are required to be performed in both directions
  - Books
  - Authors
- **If Books are massively unbalanced in size this modeling might not be feasible.**

# One Way Embedding (N:M)

- References
  - Minimizes redundancy
  - Preserves integrity
  - Reduces document growth
  - Requires multiple reads
- Embedded Documents
  - Captures point in time data
  - One document read retrieves data
  - Increases document growth

# N:M References

```
{  
  "empId": "KN1786",  
  "fName": "Awase Khirni",  
  "lName": "Syed",  
  "projects":  
    ["kdh1231412", "mdb12131321", "cdb1241"]  
}
```

```
{  
  "empId": "EN1656",  
  "fName": "Sadath",  
  "lName": "Syed",  
  "projects":  
    ["kdh1231412", "mdb12131321", "cdb1241"]  
}
```

```
{  
  "projId": "kdh1231412",  
  "projName": "SycliQ",  
  "startDate": "2008",  
  "version": "1.0"  
}
```

```
{  
  "projId": "mdb121313",  
  "projName": "TruEstate",  
  "startDate": "2007",  
  "version": "1.0"  
}
```



## Embedding

- Good for read operations
  - Aggregation
  - Complex structures
- Inserts are slower than linking
- Data integrity needs to be managed

## Linking

- Flexible
- Data integrity is built-in
- Work is done during reads
  - But not necessarily more work than any relational database.

# Db Schema Design in MongoDB

- Basic design principles stay the same.
- The focus is on how an application accesses/manipulates data
- Schema design is a bit different in MongoDB
- Evolve the schema to meet requirements as they change
- Adapt to the requirements

# DBRefs

If you have documents in a single collection that relate to documents in more than one collection, you may need to consider using DBRefs.

DBRefs are a convention for representing a document, rather than a specific reference type. They include the name of the collection, and in some cases the database name, in addition to the value from the `_id` field.

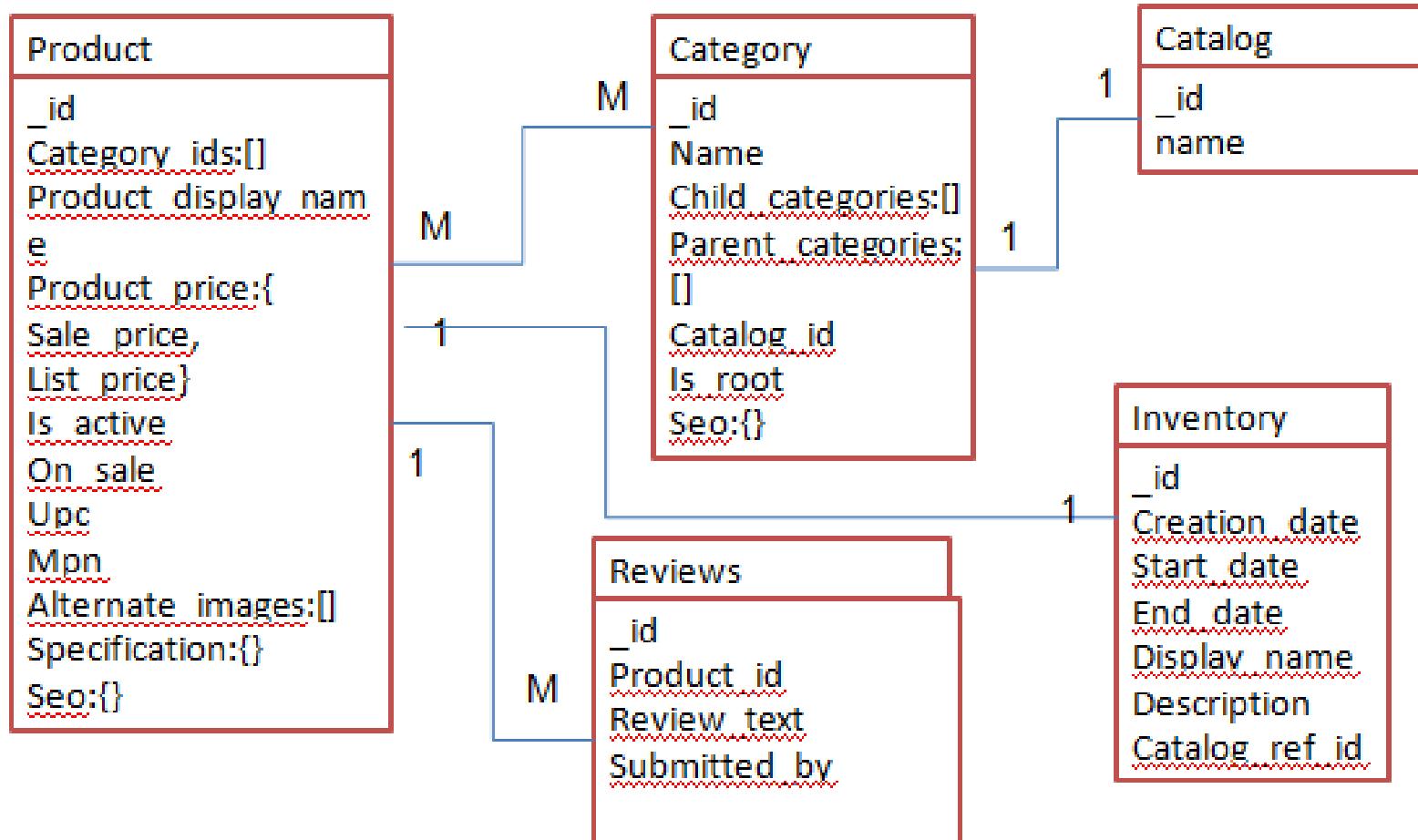
# DB Refs

- \$ref – field holds the name of the collection where the referenced document resides
- \$id – field contains the value of the \_id field in the referenced document
- \$db – Optional – contains the name of the database where the referenced document resides.
  - Only some drivers support \$db references



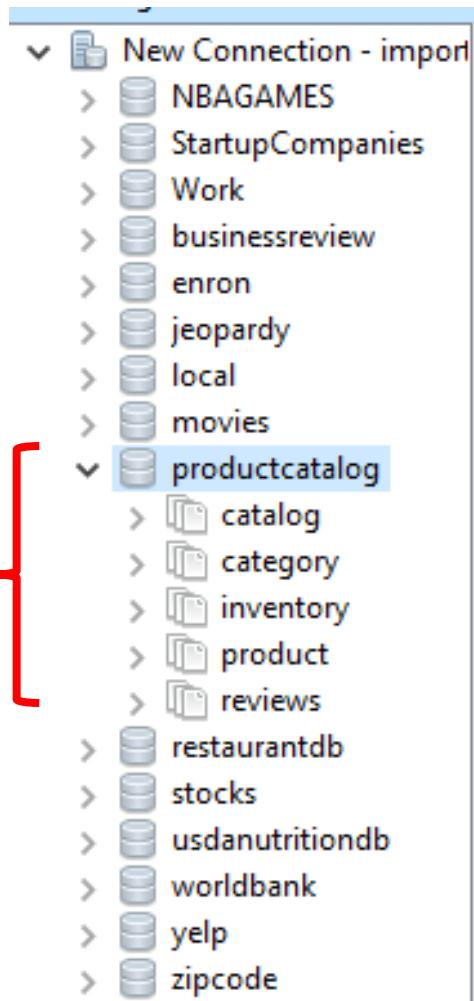
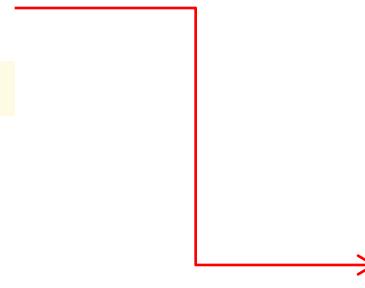
# PRODUCT CATALOG DATABASE

# Product Catalog Document Structure and Associations



# ProductCatalog

```
1 use productcatalog
2 db.createCollection("product")
3 db.createCollection("category")
4 db.createCollection("catalog")
5 db.createCollection("inventory") [highlighted]
6 db.createCollection("reviews")
```



The MongoDB Compass interface shows the database structure. The 'productcatalog' database is expanded, revealing its collections: catalog, category, inventory, product, reviews, and others like NBAGAMES, StartupCompanies, etc.

- New Connection - import
- > NBAGAMES
- > StartupCompanies
- > Work
- > businessreview
- > enron
- > jeopardy
- > local
- > movies
- > productcatalog
  - > catalog
  - > category
  - > inventory
  - > product
  - > reviews
- > restaurantdb
- > stocks
- > usdanutritiondb
- > worldbank
- > yelp
- > zipcode

Entity Name	Properties	Data Type	Description
Catalog			
	_id	int	Id, supplied by us
	name	string	name of the catalog

Entity Name	Properties	Sub Properties	Data Type	Description
category				
	_categoryid		string	category id, supplied by us
	name		string	category name
	child_categories		Array	list of child categories
	parent_categories		Array	list of parent categories
	image		string	category image
	creation_date		date	category creation date
	last_modified_date		date	category modified date
	catalog_id		string	catalog id
				tells whether the category is root category or not
	is_root		boolean	
	is_active		boolean	tells whether the category is active or not
	seo			

EntityName	Properties	SubProperties	SubProperties	Data Type	Description
product	_productid product_display_name category_ids price			string string Array double	product id, supplied by us product name the product associated categories product list price
		list_price sale_price whole_sale_price		double double double	product list price product sale price
	is_active on_sale upc mpn manufacturer product_short_description product_long_description landing_page_image thumbnail_image small_image alternate_images			boolean boolean string string string string string string string array	identifies if sale price is less than regular price universal product code manufactuer product model number manufactuer of the product brief product description detailed product description the product landing image image to use on search results page list of alternate images
		image1 image2 image3		string string string	
	shipping_info		weight dimensions	double	product weight
			width height depth	double double double	product width (inches) product height(inches) product depth(inches)
	creation_date last_modified_date avg_product_review_ratings specifications additional_product_info seo			date date double array array	Product's creation date Product's modified date average score or ratings as submitted by reviewers product's specification product additional info
		meta_title meta_keywords meta_description slug		string string string string	



Entity Name	Properties	Sub Properties	DataType	Description
Inventory	_inventoryid		objectId	Supplied by MongoDB
	creation_date		date	creation Date
	start_date		date	Inventory start date
	end_date		date	inventory end date
	display_name		string	inventory display name
	description		string	inventory description
	catalog_ref_id		int	catalogid
	avail_status		string	inventory status ex. INSTOCK, OUTOFSTOCK, PREORDER
	avaialbility_date		date	Inventory availability data
	stock_level		int	Available Quantity
Reviews	stock_thresh		int	Threshold quantity to create alerts
	product_id		string	Product id for which the inventory is created.

Entity Name	Properties	Sub Properties	Data Type	Description
Reviews	_reviewId		ObjectId	Id Supplied by MongoDB
	product_id		String	for which the review is created
	review_title		String	Title of the review
	review_text		String	Given review text
	submitted_by		String	Reviewer's id
	submitted_at		date	review created time
	verified_customer		boolean	tells whether the reviewer is verified or not
	rating		double	rating given out of 5

# Catalog Insert

```
1 db.catalog.insert({  
2   "_catalogId": "EAA43123",  
3   "name": "Electronic Appliances"  
4 })
```

```
{  
  "_id" : ObjectId("5742a3e17fd39c75ce2fe302"),  
  "_catalogId" : "EAA43123",  
  "name" : "Electronic Appliances"  
}
```

# Category Insert

```
1 db.category.insert({  
2   "_categoryId": "EA24",  
3   "categoryname": "Electronic Home Appliances",  
4   "child_categories": ["food processing", "beauty and wellness", "clothing"],  
5   "parent_categories": ["Electronics"],  
6   "image": "",  
7   "creation_date": "",  
8   "last_modified_date": "",  
9   "catalogId": "",  
10  "is_root": false,  
11  "is_active": true,  
12  "seo": {"meta_title": "", "meta_keywords": "", "meta_description": ""}  
13 })
```

```
1 {  
2   "_id" : ObjectId("5742a69b7fd39c75ce2fe303"),  
3   "_categoryId" : "EA24",  
4   "categoryname" : "Electronic Home Appliances",  
5   "child_categories" : [  
6     "food processing",  
7     "beauty and wellness",  
8     "clothing"  
9   ],  
10  "parent_categories" : [  
11    "Electronics"  
12  ],  
13  "image" : "",  
14  "creation_date" : "",  
15  "last_modified_date" : "",  
16  "catalogId" : "",  
17  "is_root" : false,  
18  "is_active" : true,  
19  "seo" : {  
20    "meta_title" : "",  
21    "meta_keywords" : "",  
22    "meta_description" : ""  
23  }  
24 }
```

# Inventory Insert

```
1 db.inventory.insert({  
2   "_inventoryId": "",  
3   "creation_date": "",  
4   "start_date": "",  
5   "end_date": "",  
6   "display_name": "",  
7   "description": "",  
8   "catalogId": "",  
9   "avail_status": "INSTOCK",  
10  "availability_date": "",  
11  "stock_level": "",  
12  "stock_thresh": "",  
13  "productId": ""  
14 })
```

Document JSON Editor

```
1 {  
2   "_id" : ObjectId("5742a8287fd39c75ce2fe304"),  
3   "_inventoryId" : "",  
4   "creation_date" : "",  
5   "start_date" : "",  
6   "end_date" : "",  
7   "display_name" : "",  
8   "description" : "",  
9   "catalogId" : "",  
10  "avail_status" : "INSTOCK",  
11  "availability_date" : "",  
12  "stock_level" : "",  
13  "stock_thresh" : "",  
14  "productId" : ""  
15 }
```



# Product insert

```
DESKTOP-I57IOS9 (C:\P...bin\mongod.exe-3.2.4) productcatalog
Shell Methods Reference

1 db.product.insert({
2 "productId": "",
3 "product_displayname": " ",
4 "categoryId": "",
5 "price": { "list_price": "", "sale_price": "", "whole_sale_price": "" },
6 "is_active": true,
7 "on_sale": false,
8 "upc": "",
9 "mpn": "",
10 "manufacturer": "",
11 "product_short_desc": "",
12 "product_long_desc": "",
13 "landing_page_image": "",
14 "thumbnail_image": "",
15 "small_image": "",
16 "alternate_images": {"image1": "", "image2": "", "image3": ""},
17 "shipping_info": {"weight": "", "dimensions": {"width": "", "height": "", "depth": ""}},
18 "creation_date": "",
19 "last_modified_date": "",
20 "avg_product_review_ratings": "",
21 "specifications": [],
22 "additional_product_info": [],
23 "seo": {"meta_title": "", "meta_keywords": "", "meta_description": "", "slug": ""}
24 })
```

```
1 _id : ObjectId("5742ac917fd39c75ce2fe306"),
2 "productId" : "",
3 "product_displayname" : " ",
4 "categoryId" : "",
5 "price" : {
6   "list_price" : "",
7   "sale_price" : "",
8   "whole_sale_price" : ""
9 },
10 "is_active" : true,
11 "on_sale" : false,
12 "upc" : "",
13 "mpn" : "",
14 "manufacturer" : "",
15 "product_short_desc" : "",
16 "product_long_desc" : "",
17 "landing_page_image" : "",
18 "thumbnail_image" : "",
19 "small_image" : "",
20 "alternate_images" : {
21   "image1" : "",
22   "image2" : "",
23   "image3" : ""
24 },
25 "shipping_info" : {
26   "weight" : "",
27   "dimensions" : {
28     "width" : "",
29     "height" : "",
30     "depth" : ""
31   }
32 },
33 "creation_date" : "",
34 "last_modified_date" : "",
35 "avg_product_review_ratings" : "",
36 "specifications" : [
37 ],
38 "additional_product_info" : [
39 ],
40 "seo" : {
41   "meta_title" : "",
42   "meta_keywords" : "",
43   "meta_description" : "",
44   "slug" : ""
45 }
```

# Review insert

```
1 db.reviews.insert({  
2   "_reviewId": "",  
3   "productId": "",  
4   "review_title": "",  
5   "review_text": "",  
6   "submitted_by": "",  
7   "submitted_at": "",  
8   "review_ipaddr": "",  
9   "verified_customer": true,  
10  "rating": 5  
11 })
```

## Document JSON Editor

```
1 {  
2   "_id" : ObjectId("5742ad947fd39c75ce2fe307"),  
3   "_reviewId" : "",  
4   "productId" : "",  
5   "review_title" : "",  
6   "review_text" : "",  
7   "submitted_by" : "",  
8   "submitted_at" : "",  
9   "review_ipaddr" : "",  
10  "verified_customer" : true,  
11  "rating" : 5.0  
12 }
```

## Update method

```
1 db.reviews.update({"_id":ObjectId("5742ad947fd39c75ce2fe307")},  
2 {$set:{rating:4}})
```



# Update

```
1 db.reviews.update({"_id":ObjectId("5742ad947fd39c75ce2fe307")},{$set:{"like":"2"})
```

_id	_reviewId	like	productId	rating	review_ipaddr	review_text	review_title	submitted_at	submitted_by
5742ad947fd3...	" "	2	" "	1.23 4.5	" "	" "	" "	" "	" "

```
db.reviews.update({"_id":ObjectId("5742ad947fd39c75ce2fe307")},{$unset:{"like":"2"})
```

_id	_reviewId	productId	rating	review_ipaddr	review_text	review_title	submitted_at	submitted_by	verified_customer
5742ad947fd3...	" "	" "	1.23 4.5	" "	" "	" "	" "	" "	true

# Remove Document and Remove Collections

	Name	Description
	Db.collection.remove()	Removes documents from a collection.
	Db.collection.remove({type:"food"})	To remove the documents that match a deletion criteria.
	Db.collection.remove({type:"food"},1)	To remove a single document from the inventory collection where the type field equals to food.
	Db.collection.drop()	Removes a collection from the database. This method also removes any indexes associated with the dropped collection

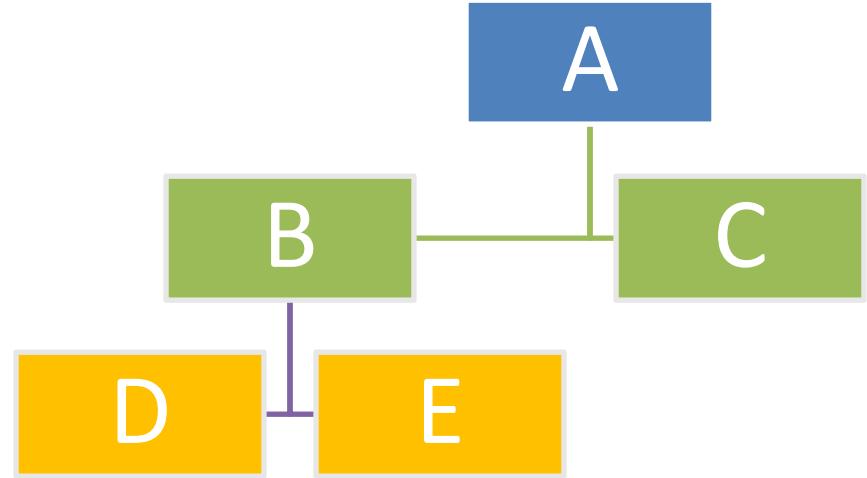
```
db.homesales.remove({"address.postcode": "SL6 7DR"})
```



# MODELING WITH TREE STRUCTURES

# Trees with Parent-Child References

- Trees
  - Single Root document
  - At most one parent
  - No cycles
- Multiple Types
  - IS-A
  - PART-OF



# Tree Reference

- Child node reference allows for **top-down navigation**
- Parent references allow for bottom up navigation
- Combination allow for bottom-up and top-down navigation
- Avoid large arrays
- Design to capture point in time data.
- Different kinds of taxonomies, site structures, demand modeling of their relationships hierarchically.



# Approaches for Modeling Hierarchical relationships in MongoDB

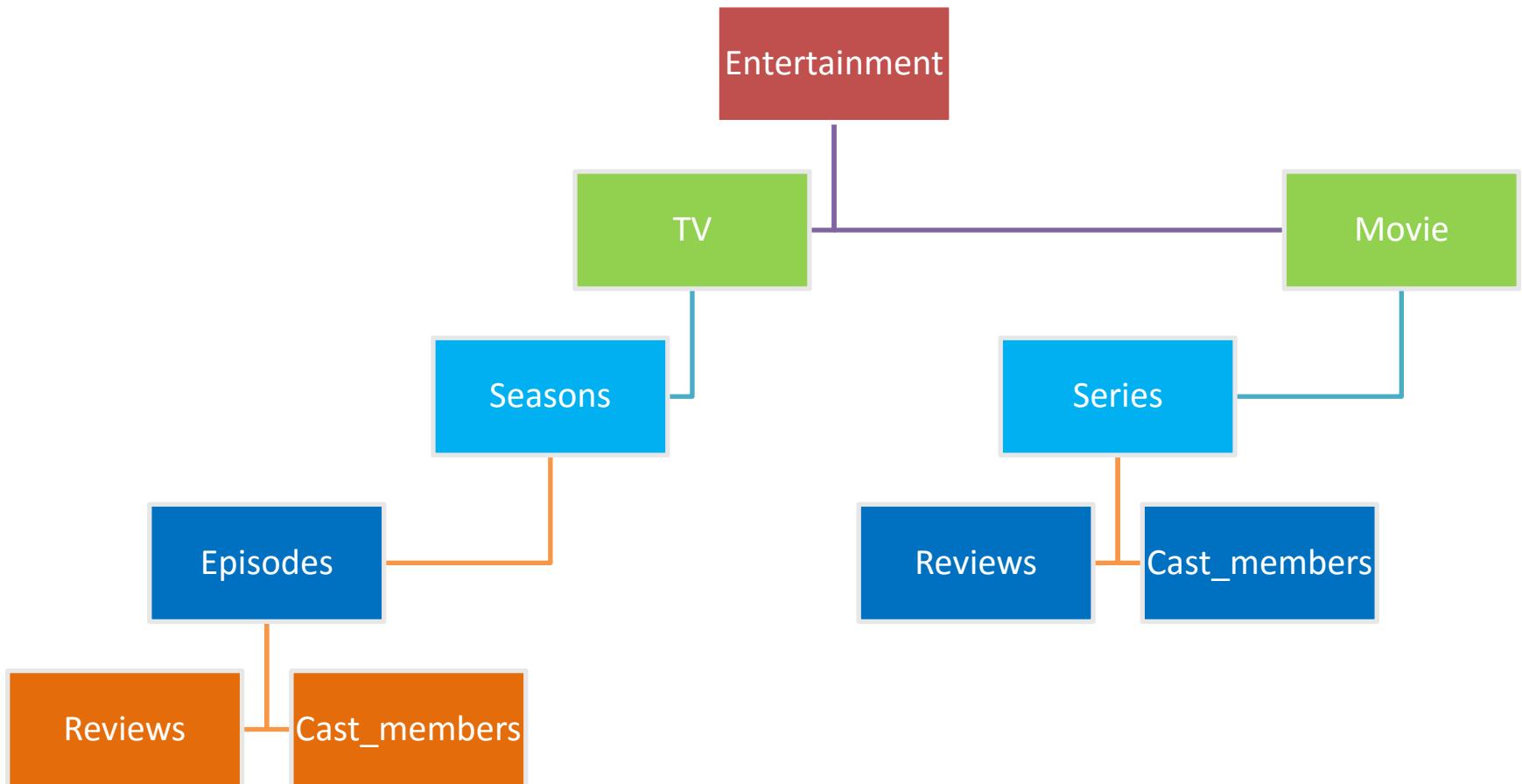
- Model Tree Structures with
  - Child references
  - Parent references
  - An Array of Ancestors
  - Materialized Paths
  - Nested Sets



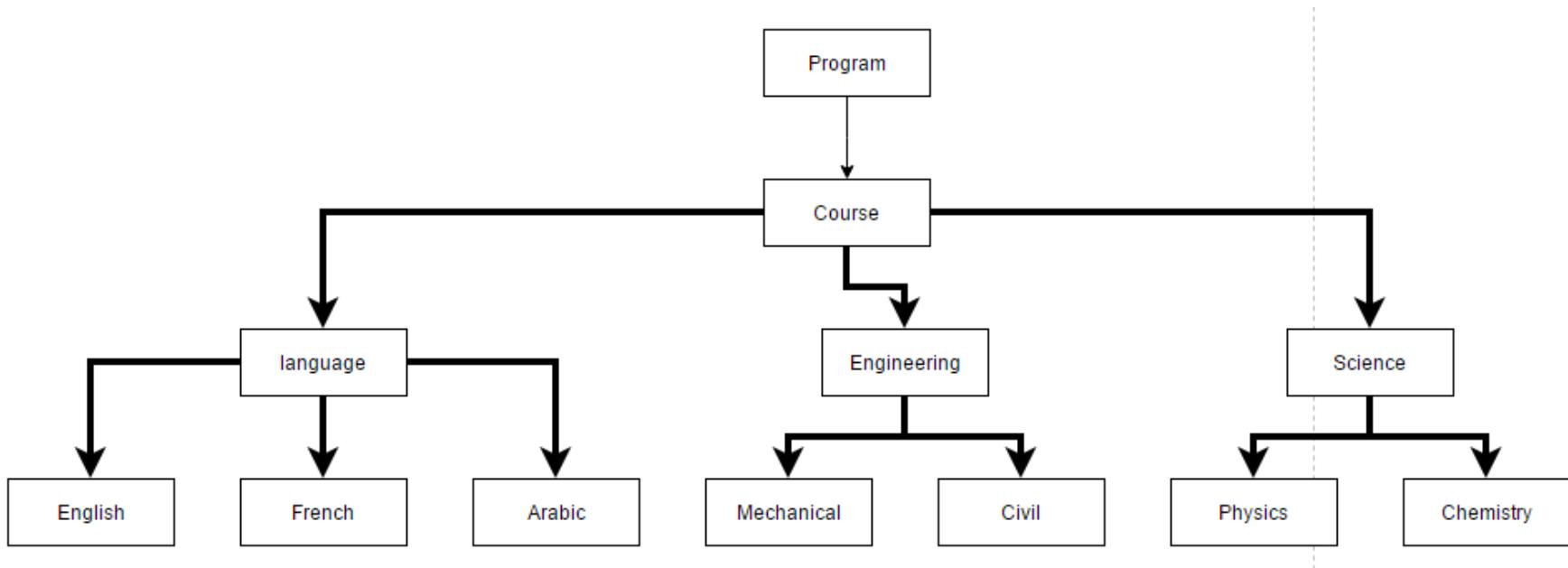
# Challenges to address when Modeling tree structures in MongoDB

- Ability to operate within the tree
  - Insert new node under specific parent
  - Update/remove existing node
  - Move node across the tree
- Ability to access the path to a node
  - Similar to XPATH navigation in XML, we should be able to navigate to the node
  - To build the breadcrumb navigation
- Ability to get all node descendants

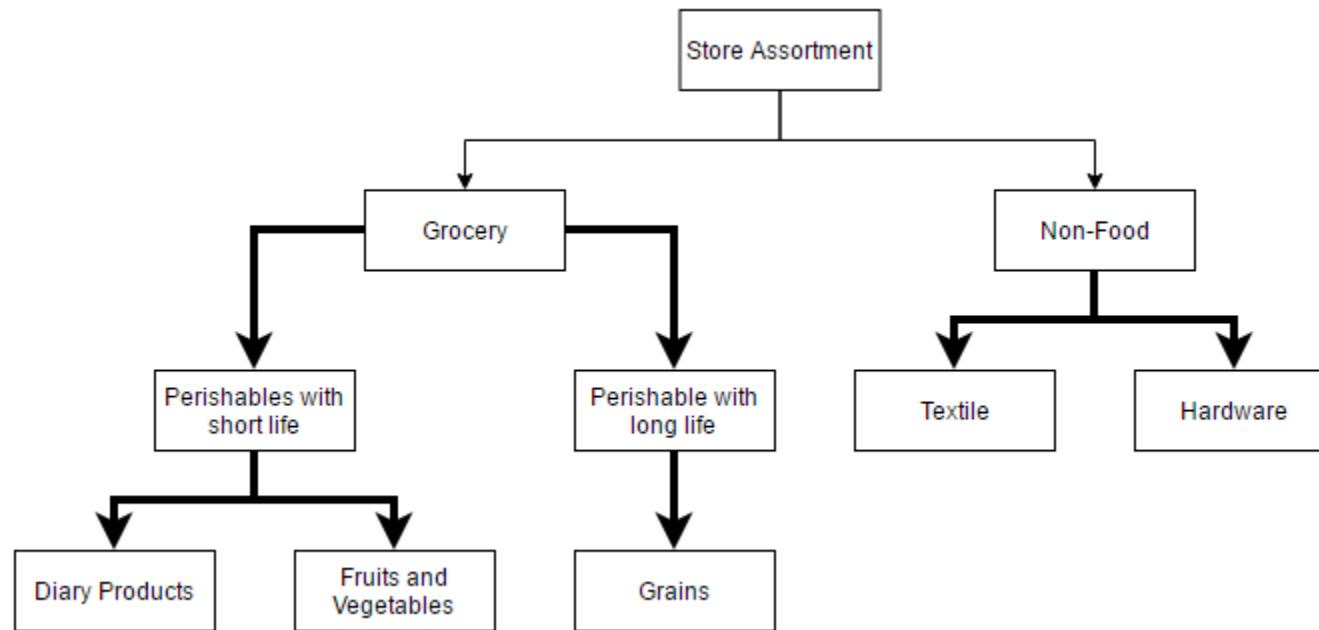
# Hierarchy of categories



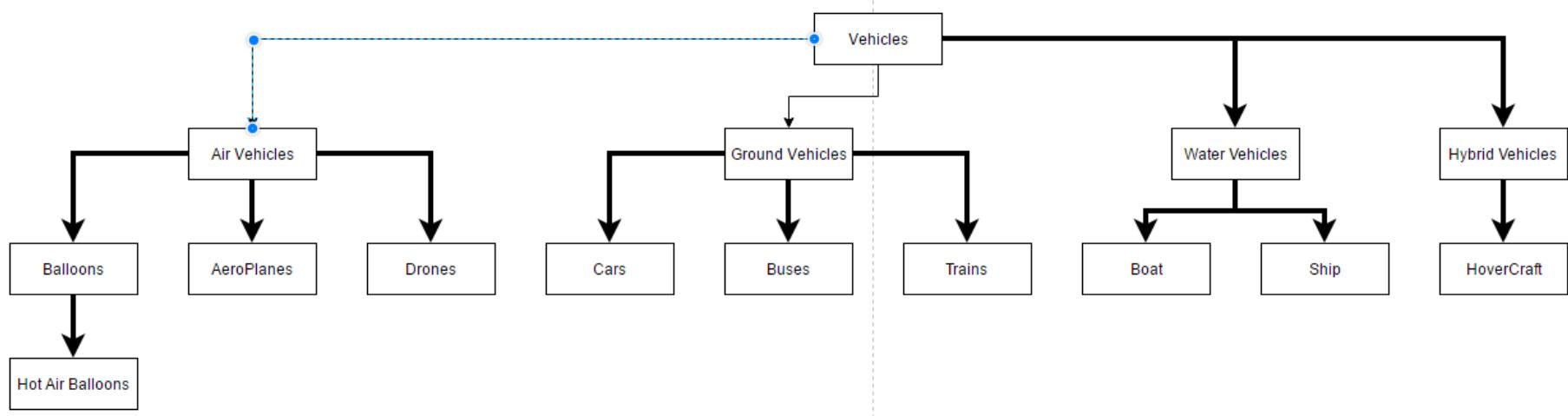
# Hierarchy of Categories



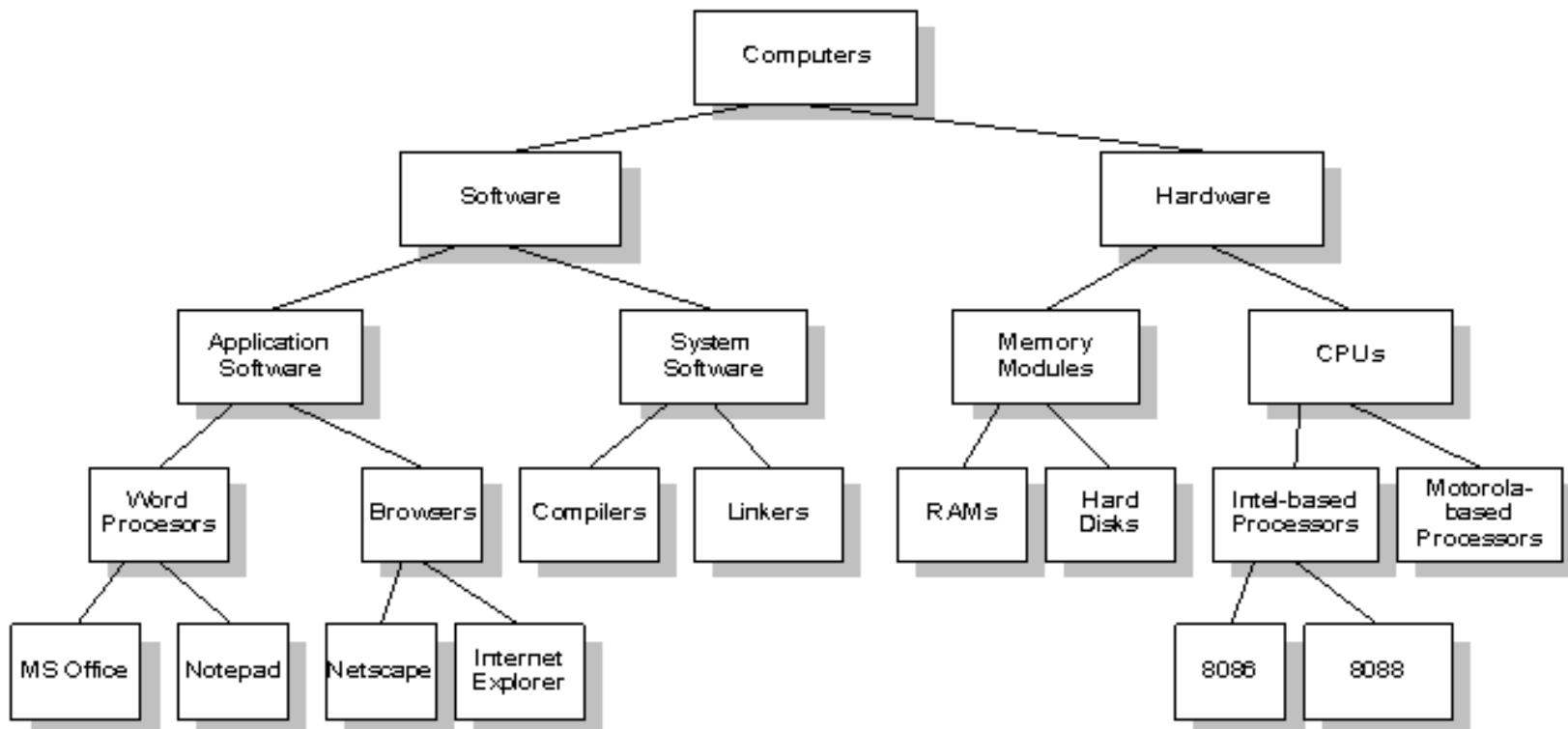
# Hierarchy of Categories



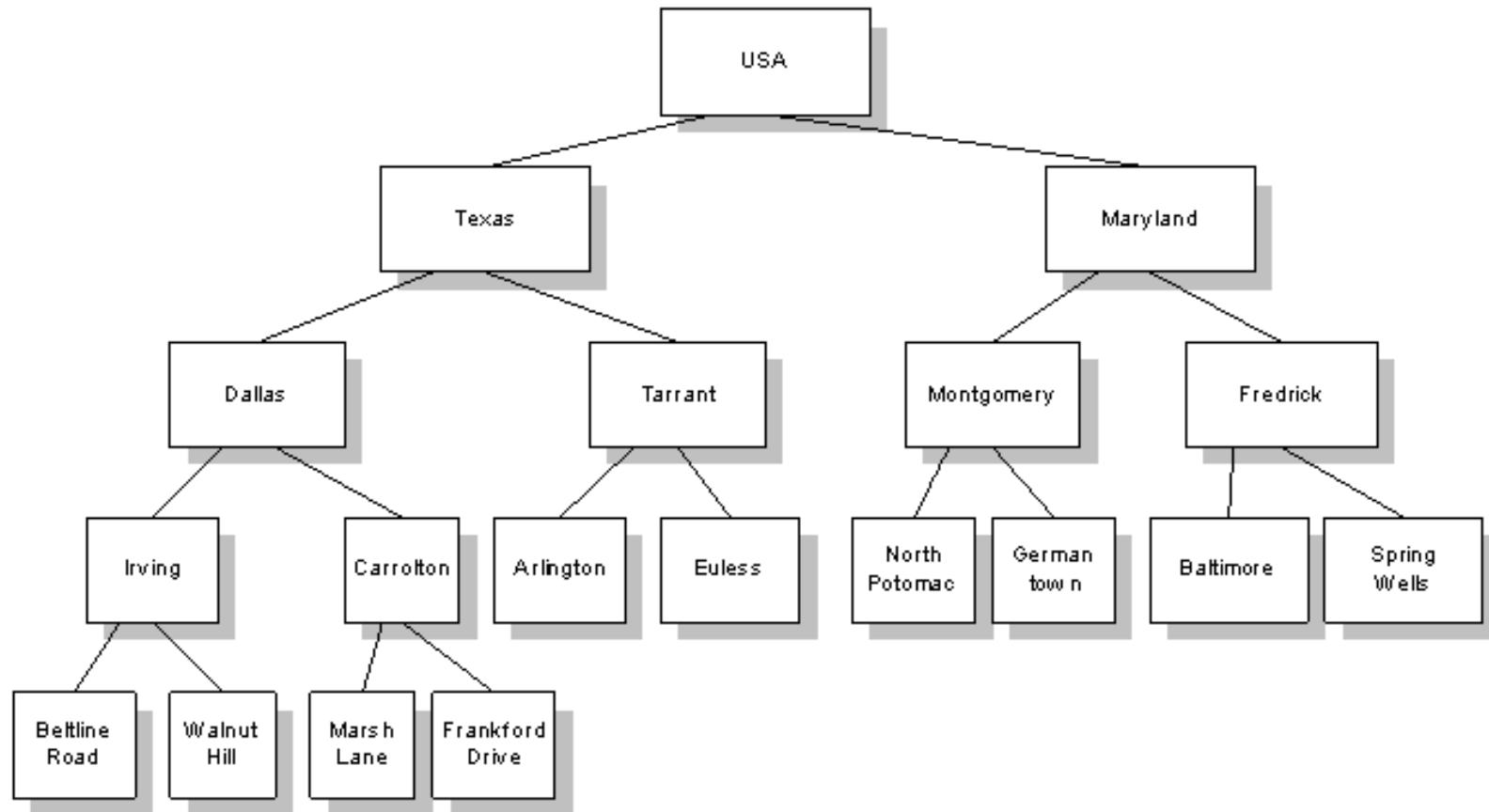
# Hierarchy of Categories



# Hierarchy of Categories



# Spatial Attribute Concept Hierarchy

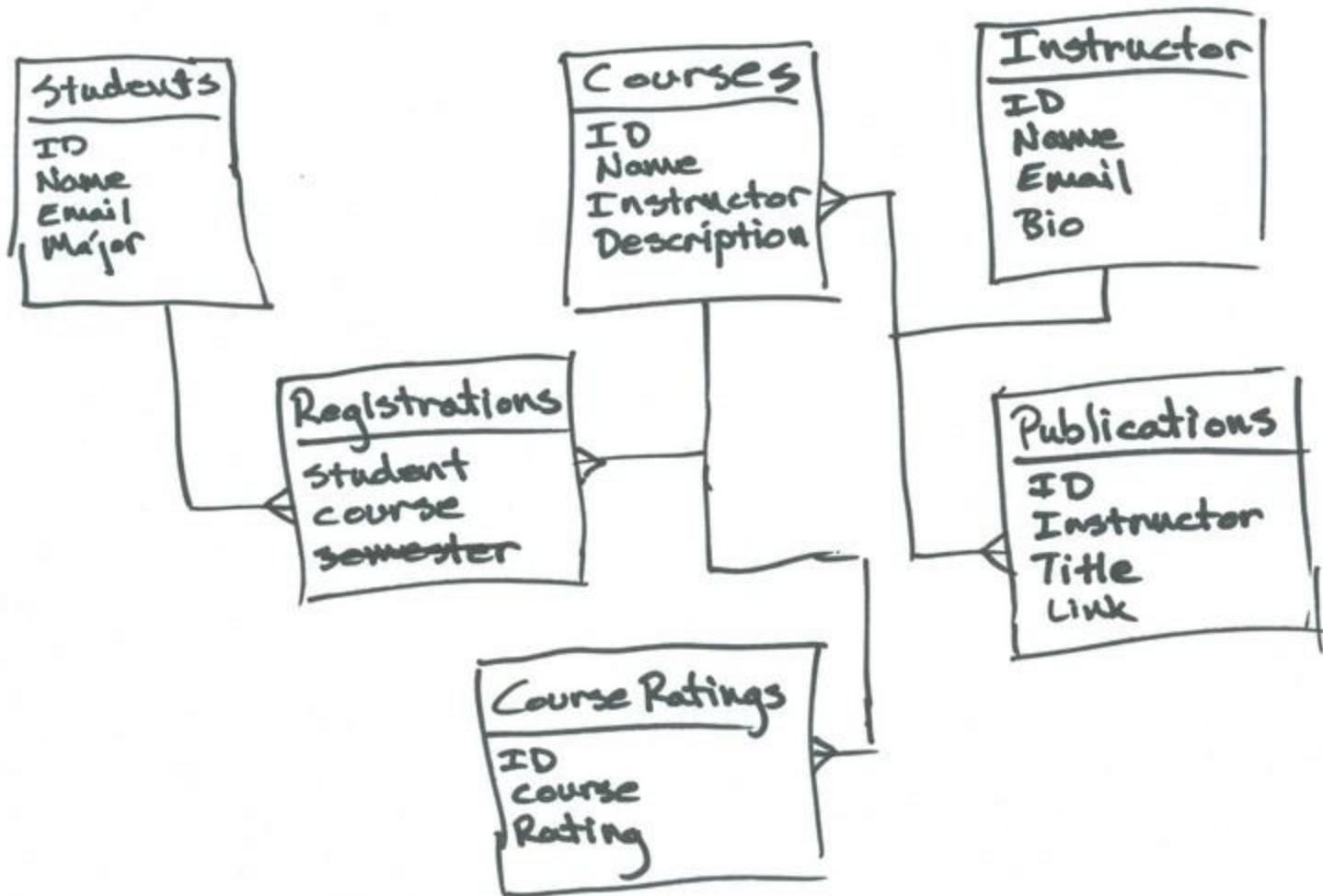


# CAPPED COLLECTIONS

- They are fixed-size collections that support high-throughput operations that insert and retrieve documents based on **insertion order**.
- **Work in a way similar to circular buffers**
  - Once a collection fills its allocated space, **it makes room for new documents by overwriting the oldest documents in the collection**
- Capped Collections guarantee preservation of the insertion order
- Queries do not need an index to return documents in insertion order
- Without this indexing overhead, capped collections can support higher insertion throughput

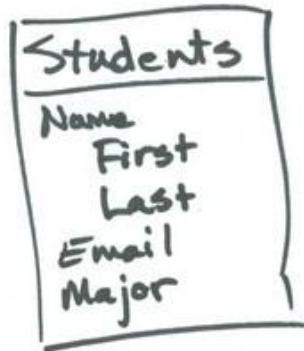
# Course Tracking System

## RDBMS



# Data Modeling in MongoDB

Course Tracking System  
MongoDB



- In the earlier example, we demonstrated how we store data into a relational structure and in a document store like mongoDB
- Demonstrated difference between table and collection
  - Tables can only hold one type of thing
  - Collections can hold a variety of things
- Demonstrated implicit schema driven by convention.



# MONGODB IMPORT -EXPORT



# Importing Dataset

- Sample dataset
  - <https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/primer-dataset.json>
  - Import data into the collection using **shell**

```
mongoimport --db test --collection restaurants --drop --file primer-dataset.json
```

```
mongoimport --host localhost --port 27017 --db test --collection restaurants  
--drop --file primer-dataset.json
```



# Import with MongoChef

The screenshot shows the MongoChef Professional interface. On the left, the main window displays a list of databases and collections. A context menu is open over the 'Collections' section, with 'Import Collections...' highlighted. The central part of the screen is the 'Import Collections (JSON)' dialog. It shows the 'Import Source Files' section with a file path selected: E:\CT\MongoDB\code\data\import-export\primer-database. The 'Import Target' section shows the target database as 'RestaurantsDatabase @ localhost:27017' and the import mode as 'Overwrite documents with same \_id'. Below this, there's a note about document naming and creation rules. At the bottom right of the dialog are 'Import' and 'Cancel' buttons. The bottom right corner of the main interface shows an 'Operations' panel with a log entry: 'Import Collections - done' and '25359 documents imported.'

MongoChef Professional - 31 Software Labs - Non-Commercial License

File Edit Database Collection Index Document GridFS View Help

Connect IntelliShell Aggregate Map-Reduce Export Import Users Roles

Get MongoChef 3.5.0 - now with support for auto-reconnect. Visit <http://3t.io/mongochef/download/>

New Connection - imported on 08-Mar-2016 (localhost:27017)

- StartupCompanies
- Work
- businessreview
- enron
- jeopardy
- local
- movies
- stocks
- worldbank
- zipcode
- RestaurantsDatabase

(localhost:27017)

- StartupCompanies
- Work
- businessreview
- enron
- jeopardy
- local
- movies
- stocks
- worldbank
- zipcode

Open IntelliShell Ctrl+L

Add Database...

Copy Collections Ctrl+C

Paste Collections Ctrl+V

Export Collections...

**Import Collections...**

Drop Database Del

Add Collection...

Add GridFS Bucket...

Manage Users

Manage Roles

Manage Stored Functions

Show DB Statistics

Show Collection Statistics

Show Current Operations

Show Server Info

Refresh Selected Item Ctrl+R

Refresh All Ctrl+Shift+R

Choose Color

Disconnect

Import Collections (JSON)

Import Source Files

Import Target

Database: RestaurantsDatabase @ localhost:27017

Import Mode: Overwrite documents with same \_id

Validate JSON before import

The documents in each source file will be imported into a collection of the same name as the file (without the path and the file extension). If no corresponding collection exists, it will be created in the target database.

Import Cancel

Operations

Import Collections - done

Target DB: RestaurantsDatabase @ New Connection -

E:\CT\MongoDB\code\data\import-export\primer-database

Importing documents

25359 documents imported.



# Exporting Collections

The screenshot shows the MongoChef Professional application interface. On the left, the main window displays a list of databases and collections under 'New Connection - imported on 08-Mar-2016 (localhost:27017)'. A context menu is open over the 'primer-dataset' collection, with 'Export Collections...' highlighted. The central part of the screen is the 'Export Collections' dialog. It shows 'Database: RestaurantsDatabase @ localhost:27017' and 'Export Target' set to 'Target Folder: E:\CT\MongoDB\code\data\exportfrommongochef' and 'Format: JSON - mongo shell / 3T MongoDB'. The 'Export Source Collections' section contains 'primer-dataset'. Below the dialog, the 'Operations' panel shows a log entry: 'Export Collections - done' with 'Source DB: RestaurantsDatabase @ New Connection -' and 'primer-dataset → E:\CT\MongoDB\code\data\exportfi' followed by '25359 document(s) exported.'



# A single restaurant Collection

```
[{  
    "_id" : ObjectId("573fcf4de658e15d5752ad2c"),  
    "address" : {  
        "building" : "1007",  
        "coord" : [  
            -73.856077,  
            40.848447  
        ],  
        "street" : "Morris Park Ave",  
        "zipcode" : "10462"  
    },  
    "borough" : "Bronx",  
    "cuisine" : "Bakery",  
    "grades" : [  
        {  
            "date" : ISODate("2014-03-03T00:00:00.000+0000"),  
            "grade" : "A",  
            "score" : NumberInt(2)  
        },  
        {  
            "date" : ISODate("2013-09-11T00:00:00.000+0000"),  
            "grade" : "A",  
            "score" : NumberInt(6)  
        },  
        {  
            "date" : ISODate("2013-01-24T00:00:00.000+0000"),  
            "grade" : "A",  
            "score" : NumberInt(10)  
        },  
        {  
            "date" : ISODate("2011-11-23T00:00:00.000+0000"),  
            "grade" : "A",  
            "score" : NumberInt(9)  
        },  
        {  
            "date" : ISODate("2011-03-10T00:00:00.000+0000"),  
            "grade" : "B",  
            "score" : NumberInt(14)  
        }  
    ],  
    "name" : "Morris Park Bake Shop",  
    "restaurant_id" : "30075445"  
}]
```



MONGODB

# PROJECTION QUERIES

# Query Data with Mongo Shell

No	Command	Output
1	Use restaurantdb	Switched to db restaurantdb
2	db['primer-dataset']	Restaurantdb.primer-dataset
Query by a Top Level Field		
3.	db.restaurants.find({'borough':'Manhattan'})	Gives a list of documents matching the query
Query by a Field in an Embedded Document		
4	db.restaurants.find({"address.zip code":"10075"})	Give a set of matching documents
Query by a field in an Array		
5.	db.restaurants.find({"grades.grade":'C'})	Querying embedded documents as its elements. To specify a condition on a field in the documents using dot notation

# Query Selectors

- For comparison of different BSON type values

	Name	Description
	\$eq	Matches values that are equal to a specified value
	\$gt	Matches values that are greater than a specified value
	\$gte	Matches values that are greater than or equal to a specified value
	\$lt	Matches values that are less than a specified value
	\$lte	Matches values that are less than or equal to a specified value
	\$ne	Matches all values that are not equal to a specified value
	\$in	Matches any of the values specified in an array
	\$nin	Matches non of the values specified in an array



# Query Selectors Example

```
// Query for documents whose grades array contains an embedded document
// with a field score greater than 30.
db.restaurants.find({"grades.score":{$gt:30}})

// Query for documents whose grades array contains an embedded document
// equal to 30
db.restaurants.find({"grades.score":{$eq:30}})

// Query for documents whose grades array contains an embedded document
// less than 30
db.restaurants.find({"grades.score":{$lt:10}})

//Query for documents whose grades array contains an embedded document
//less than or equal to 30
db. restaurants.find({"grades.score":{$lte:10}})

db.restaurants.find({"cuisine": {$in: ["Chinese", "Italian"]}}, {"borough": "Brooklyn", "address.street": "8 Avenue"})
```

# Find() and findAndModify()

- Use SalesData

```
1 db.homesales.findOne({"amount":9000})  
2 db.postcodes.find({"postcode":"SL6 0AA"})  
3 db.postcodes.find({"postcode":"SL6 0AA", "address.postcode":"SL6 7DR"})
```

```
use peoplerecords;  
db.people.find({fullname:{$type:2}});
```

```
use peoplerecords  
db.MOCK_DATA.findAndModify({  
    query:{rating:0},  
    sort:{rating:1},  
    update:{$inc:{rating:2}},  
    upsert:true,  
    new:true})
```

```
use peoplerecords;  
db.people.find({profession:{$exists:false}});
```

# findAndModify()

- It updates and returns an existing document in the people collection where the document matches the query criteria.
- Upsert: true option for update operation to either update a matching document or if no matching document exists, create a new document
- New:true option inserts a document and returns the newly inserted document in the value field.

```
db.collection.findAndModify({  
    query: <document>,  
    sort: <document>,  
    remove: <boolean>,  
    update: <document>,  
    new: <boolean>,  
    fields: <document>,  
    upsert: <boolean>  
});
```

```
use peoplerecords  
db.MOCK_DATA.findAndModify({  
    query:{rating:0},  
    sort:{rating:1},  
    update:{$inc:{rating:2}},  
    upsert:true,  
    new:true})
```



## findAndModify() => remove

```
db.MOCK_DATA.findAndModify({  
    query:{'city':'Bambas'},  
    sort:{'rating':1},  
    remove:true  
})
```

## findOneAndDelete()

- It deletes a single document based on the filter and sort criteria, returning the deleted document.
- Deletes first matching document in the collection that matches the filer.

```
use peoplerecords  
db.MOCK_DATA.findOneAndDelete({  
    'city':'Solntsevo'  
})
```

```
use peoplerecords  
db.MOCK_DATA.findOneAndDelete(  
    {'city':'Cisompet'},  
    {sort:{'rating':1}}  
)
```



## findOneAndReplace()

- Modifies and replaces a single document based on filter and sort criteria

```
use peoplerecords
db.MOCK_DATA.findOneAndReplace(
{'rating':{$lt:'2'}},
{'rating':'4','money':'$11'}
)
|db.MOCK_DATA.findOne({_id:objectId("57431d0270c61ea3e2739420")})
```

```
use peoplerecords
db.MOCK_DATA.findOneAndReplace(
{'rating':{$lt:'2'}},
{'rating':'4','money':'$11'},
{sort:{'rating':1}}
)
|db.MOCK_DATA.findOne({_id:objectId("57431d0270c61ea3e2739423")})
```

```
use peoplerecords
db.MOCK_DATA.findOneAndReplace(
{'rating':{$lt:'2'}},
{'rating':'4','money':'$11'},
{sort:{'rating':1},project:{'id':0}}
)
```



## FindOneAndUpdate()

- Updates a single document based on the filter and sort criteria

```
db.collection.findOneAndUpdate(  
<filter>,  
<update>,  
{  
  projection: <document>,  
  sort: <document>,  
  maxTimeMS: <number>,  
  upsert: <boolean>,  
  returnNewDocument: <boolean>  
}  
)
```

```
use peoplerecords  
db.people.findOneAndUpdate(  
'guid': '5a7d7753-ed8e-4510-a860-0049c2709e55',  
{$inc: {'rating':5}}  
)
```



## Db.collection.count(query,options)

- Returns the count of documents that would match a find() query.

```
use peoplerecords  
db.people.find().limit(25);
```

```
use peoplerecords  
db.people.find({ 'rating':{$gt:5}}).count()  
db.people.find({ 'country':'Indonesia'},{'rating':{$gt:10}}).count()
```

---

```
1 use peoplerecords;  
2 db.people.find({fullname:$regex:"A"});
```

```
use peoplerecords;  
db.people.find({fullname:$regex:"^A"});
```

```
use peoplerecords;  
db.people.find({fullname:$regex:"e$"});
```



## db.collection.explain()

- Returns information on the query plan for the following operations
  - Aggregate()
  - Count()
  - Distinct()
  - Find()
  - Group()
  - Remove()
  - update()

```
{  
    "queryPlanner" : {  
        "plannerVersion" : NumberInt(1),  
        "namespace" : "peoplerecords.people",  
        "indexFilterSet" : false,  
        "parsedQuery" : {  
            "country" : {  
                "$eq" : "Indonesia"  
            }  
        },  
        "winningPlan" : {  
            "stage" : "COUNT",  
            "inputStage" : {  
                "stage" : "COLLSCAN",  
                "filter" : {  
                    "country" : {  
                        "$eq" : "Indonesia"  
                    }  
                },  
                "direction" : "forward"  
            }  
        },  
        "rejectedPlans" : [  
        ]  
    },  
    "serverInfo" : {  
        "host" : "DESKTOP-I57JOS9",  
        "port" : NumberInt(27017),  
        "version" : "3.2.4",  
        "gitVersion" : "e2ee9ffcf9f5a94fad76802e28cc978718bb7a30"  
    },  
    "ok" : NumberInt(1)  
}
```

```
db.people.explain().find({'country':'Indonesia'},{'rating':{$gt:10}}).count()  
db.people.explain("executionStats").find({'country':'Indonesia'},{'rating':{$gt:10}}).count()  
db.people.explain("allPlansExecution").find({'country':'Indonesia'},{'rating':{$gt:10}}).count()
```



## Db.collection.explain()

- Returns information regarding
  - queryPlanner – which details the plan selected by the query optimizer and lists the rejected plans
  - executionStats – which details the execution of the winning plan and the rejected plans
  - serverInfo – which provides information on the MongoDB instance

## Db.collection.reIndex()

- Drop all indexes and recreate them
- Db.collection.reIndex();



## Db.Collection.dataSize()

- The size of the collection. This method provides a wrapper around the size output of the collstats()

```
db.people.dataSize()  
db.MOCK_DATA.dataSize()
```

## Db.collection.drop()

- Removes a collection from the database. The method also removes any indexes associated with the dropped collection.
- Takes no arguments and will produce an error if called with any argument.
- Returns
  - True when successfully drops a collection
  - False when the collection to drop does not exist

# Indexes

- Special data-structures that store a subset of data in an easily traversable format
- B-Tree indexes are efficient way to access content and MongoDB adopts B-Tree indexes.
- Type of indexes
  - \_id
  - Simple
  - Compound
  - Multikey
  - Full-Text
  - Geo-spatial
  - Hashed



## The `_id` index

- Automatically create and cannot be removed
- Similar to primary key
- Default value is 12-byte ObjectId
  - 4 byte time stamp
  - 3-byte machine id
  - 2-byte process id
  - 3-byte counter

## Simple index

- An index on a single key
- Similar to a book's index where you look up a word to find the pages it's referenced on.



## Compound index

- Created over two or more fields in a document
- Similar to a phone book where you can find the phone number of a person given their first and last names.

## Multikey index

- A multikey index is an index that's created on a field that contains an array.
- If using in a compound index, only a single field in a given document can be an array.
- One entry in the index for every item in the array for the given document. An array with 100 items in a document, will have 100 index entries

## Full-Text Index

- An index over a text based field, similar to how Google indexes web pages

## Geo-spatial index

- Allows you to determine distance from a given point
- Works on both planar and spherical geometries

## Hashed indexes

- Used in has based sharding, and allows for a more randomized distribution
- Cannot contain compound keys or be unique
- Can contain both a hashed and non-hashed version. The non-hashed version will allow for range based queries

## Index Properties

- **Unique**
- **Sparse**
- **TTL(TimeToLive)**
- **Partial**

# Index Properties

## Sparse

- Allows you to index only documents that contain a value for the given field.
- A sparse index will not be used if it would result in an incomplete result set, unless specifically hinted.

# Index Properties

## TTL

- Special single-field indexes that MongoDB can use to automatically remove documents from a collection after a certain amount of time.
- Data expiration is useful for certain type of information like machine generated event data, logs, and session information that only need to persist in a database for finite amount of time
- On replica sets, TTL background thread only deletes documents on the primary.

```
db.eventlog.createIndex( { "lastModifiedDate": 1 },  
{ expireAfterSeconds: 3600 } )
```

- They expire after the specified number of seconds has passed since the indexed field value, i.e the expiration threshold is the indexed field value plus the specified number of seconds.
- If the field is an array and there are multiple date values in the index, MongoDB uses lowest(i.e earliest) date value in the array to calculate the expiration threshold.
- **Db.currentOp()**
- Should contain an ISODate() value
- Runs every 60 seconds.



# Index Property

## Partial (3.2)

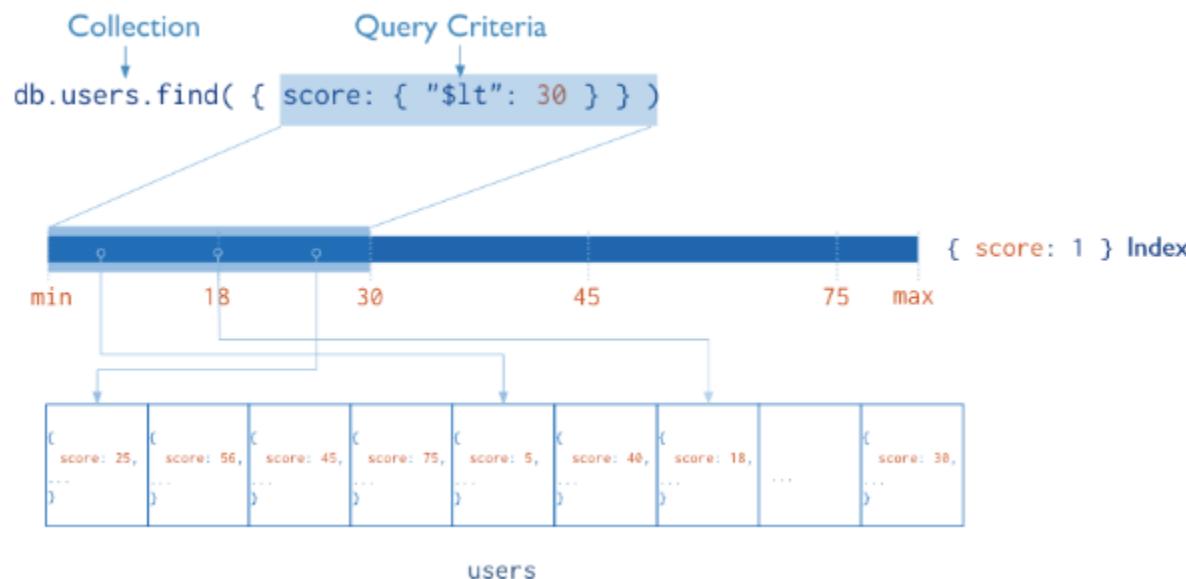
- Allows you to index a subset of your data
- The index will not be used if it would provide an incomplete result set( similar to the sparse index)

```
|db.collection.createIndex({ "movie":1, "reviews":  
| 1}, { "rating":{ "gte":4}});
```

# Indexes

## ► Definition

- Indexes are special data structures that store a small portion of the **collection's** data set in an easy to traverse form.



# Compound Index

## Creation

```
db.flights.createIndex({ "Origin":1, "Dest":1,  
"FlightDate": -1});
```

## Queries

```
db.flights.find({ "Origin": "DEN" });  
db.flights.find({ "Origin": "DEN", "Dest": "BNG"  
});  
db.flights.find({ "Origin": "DEN", "Dest": "BNG"  
}).sort({ "FlightDate": -1 });  
db.flights.find({ "Origin": "DEN", "Dest": "BNG"  
}).sort({ "FlightDate": 1 });
```

# Full-Text Index

```
db.messages.createIndex({ "body": "text" });
```

```
db.messages.find({ "$text": { "$search":  
"\\"sometext\\\""} })
```



## Covering Indexes

- Covering indexes are indexes that will answer a query without going back to the data.

```
db.flights.createIndex({"Origin":1,"Dest":1,  
"ArrDelay":1,"UniqueCarrier":1})
```

```
db.flights.find({"Origin":"DEN","Dest":"JFK"  
}, {"UniqueCarrier":1,"ArrDelay":1, "_id":0}).  
sort ({"ArrDelay": -1})
```

Nested Index

```
db.locations.createIndex ({"location.state":1});  
db.locations.find ({"location": {"state":  
"Andhra", "City": "Hyderabad"} }) ;
```

# Index Intersection

- When two or more indexes are used to satisfy a query

```
db.orders.createIndex({"qty":1});
db.orders.createIndex({"item":1});
//query
db.orders.find({"item":"CXS-12-s","qty":{$gte:15}});
```

# Index Operations in MongoDB

## ► Creation index

- db.users.ensureIndex( { score: 1 } )

## ► Show existing indexes

- db.users.getIndexes()

## ► Drop index

- db.users.dropIndex( {score: 1} )

## ► Explain—Explain

- db.users.find().explain()
- Returns a document that describes the process and indexes

## ► Hint

- db.users.find().hint({score: 1})
- Override MongoDB's default index selection

- View all indexes in a database

`db.system.indexes.find()`

- View all indexes for a given collection

`db.collection.getIndexes()`

- View the size of all indexes in a collection

`db.collection.stats()`



## Db.collection.createIndex()

- Creates indexes on collections
- Indexes can be created in ascending order on a single field
- Indexes can be created on multiple fields as a compound index.
  - A compound index cannot include a hashed index component.

## Db.collection.getIndexes()

- Returns an array that holds a list of documents that identify and describe the existing indexes on the collection.

```
db.collection.createIndex(keys, options)
```



## Db.collection.distinct()

- Finds the distinct values for a specified field across a single collection and returns the results in an array.

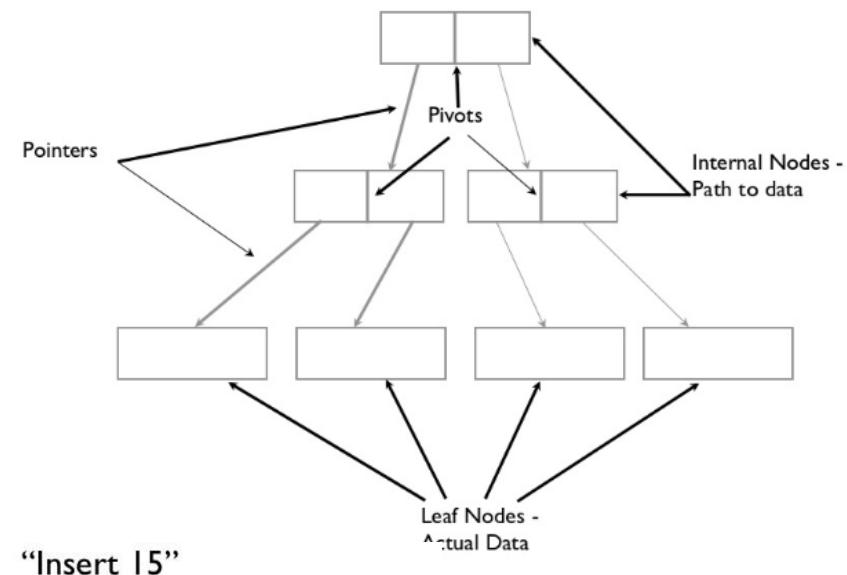
```
use peoplerecords
db.people.distinct('city');
```

Key	Value	Type
0	[ 974 elements ]	Array
1	Ninh Hòa	String
2	Bejuco	String
3	Tajan	String
4	Lusk	String
5	Asmara	String
6	Vällingby	String
7	Chiguayante	String
8	Bochum	String
9	Sergio Osmeña Sr	String
10	El Realejo	String
11	Calde	String
	Chengbei	String

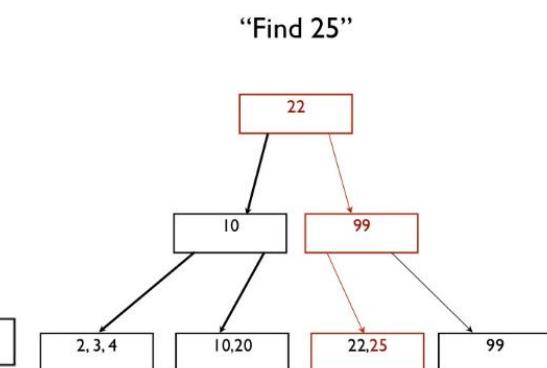
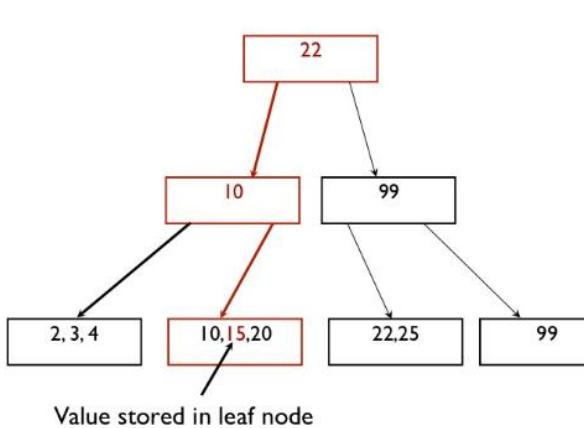
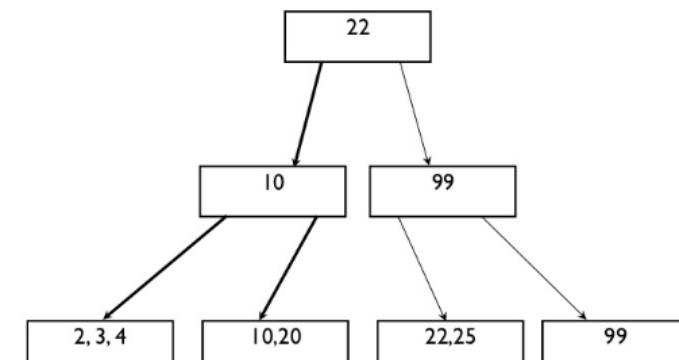
Activate  
Go to Settii

# B-Tree in MongoDB

- A tree data-structure that keeps data sorted and allows searches, sequential access, insertions, deletions in logarithmic time



"Insert 15"



\* Pivot Rule is  $\geq$

# Indexes in MongoDB

## Types

- **Single Field Indexes**
- **Compound Field Indexes**
- **Multikey Indexes**
- **Single Field Indexes**
  - `db.users.ensureIndex( { score: 1 } )`

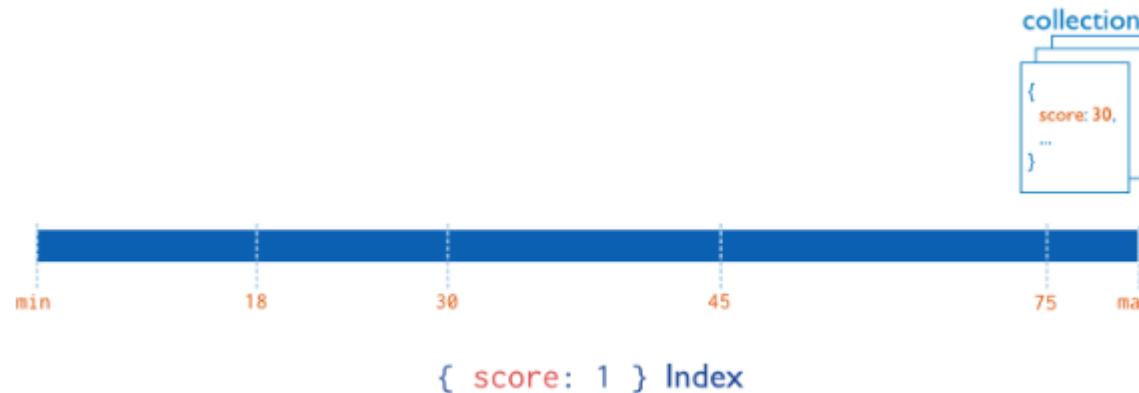


Diagram of an index on the `score` field (ascending).

# Indexes in MongoDB

## Types

- Single Field Indexes
  - Compound Field Indexes
  - Multikey Indexes
- Compound Field Indexes
- db.users.ensureIndex( { userid:1, score: -1 } )

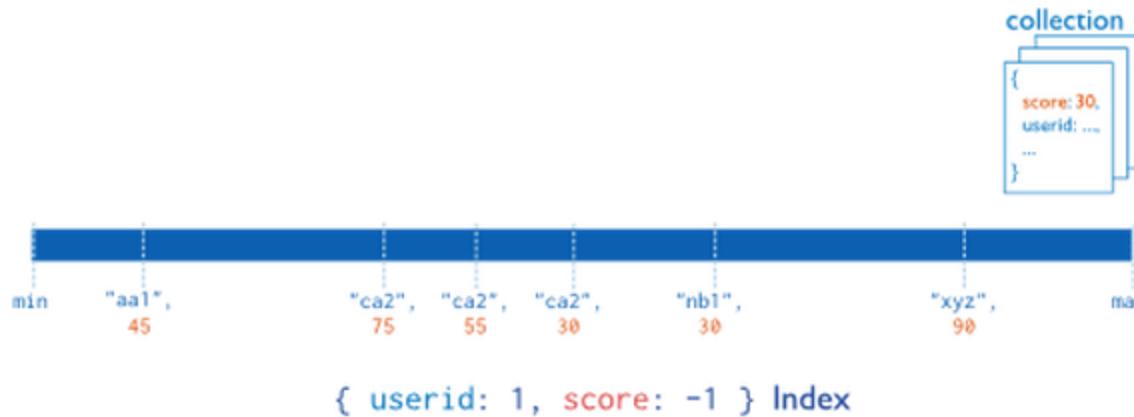


Diagram of a compound index on the `userid` field (ascending) and the `score` field (descending). The index sorts first by the `userid` field and then by the `score` field.

# Geospatial Index (2dSphere)

- Indexes on geospatial fields
  - Using GeoJSON objects
  - Geometries on spheres
- GeoJSON objects
  - Point
  - LineString
  - Polygon
  - MultiPoint
  - MultiLineString
  - MultiPolygon
  - GeometryCollection

```
//GeoJSON Object Structure for indexing
{
  name: "Bangalore",
  location:{type:"Point",
    coordinates:[12.34,77.1212]}
}

//Index on GeoJSON objects
db.articles.ensureIndex({location:"2dsphere"});

db.articles.find({
  location:{$near:{
    $geometry:{type:"point",coordinates:[12.34,
      77.1212]}},
    $maxDistance:5000
  }
})
```

# Text Search

- Use text indexes to support text search of a string content in documents of a collection
- Text indexes can include any field whose value is a string or an array of string elements
- To perform queries that access the text index, use the **\$text query operator**
- **Only one text index per collection**
- **"\$\*\*" operator to index all text fields in the collection**
- **Use weight to change the importance of fields**

## Operators

**\$text,\$search,\$language,\$meta**

```
db.articles.ensureIndex(  
  {title:"text", content:"text"});  
//index all text fields in the collection  
db.articles.ensureIndex({"$**":"text",  
 name:"MyTextIndex"});  
//weights to change the importance of fields  
db.articles.ensureIndex(  
  {"$**":"text"},  
  {weights:{"title":10,"content":5},  
 name:"MyTextIndex"}  
)
```

Use **\$text** and **\$search** operators to query and **\$meta** for scoring results

```
//search articles collection  
db.articles.find({  
  $text:{$search:"MongoDB"} });  
  
db.articles.find(  
  {$text:{$search:"MongoDB"}},  
  {score:{$meta:"textScore"},_id:0, title:1}  
)
```



## Db.collection.stats()

- Returns statistics about the collection

```
db.people.stats();
```

Key	Value	Type
✓ (1) {_id : }	{ 11 fields }	Document
i32 avgObjSize	184	Int32
t/f capped	false	Bool
i32 count	1001	Int32
> (0) indexSizes	{ 2 fields }	Object
i32 nindexes	2	Int32
"_" ns	peoplerecords.people	String
i32 ok	1	Int32
i32 size	185090	Int32
i32 storageSize	167936	Int32
i32 totalIndexSize	77824	Int32
> (0) wiredTiger	{ 13 fields }	Object



## Db.Collection.getIndexes()

- Returns an array that holds a list of documents that identify and describe the existing indexes on the collection.

## Explain

- To see if an index is used, append the **.explain()** operator to your query
- 3 levels of verbosity
  - queryPlanner – it returns the winning query plan (default)
  - executionStats – adds execution stats for the plan
  - allPlansExecution- adds stats fro the other candidate plans.

```
db.flights.find({ "Origin": "DEN" }).explain();
```

# Index-Limitations

- Collections cannot have > 64 indexes
- Index keys cannot be > 1024 bytes (1K)
- The name of an index, including the namespace, must be 128 characters
- Queries can only use 1 index
- Indexes have storage requirements, and impact the performance of writes
- In memory sort(no-index) limited to 32 mb of return data

# Logical Operators

	Name	Description
1.	\$or	Joins query clauses with a logical OR returns all documents that match the conditions of either clause
2.	\$and	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses
3.	\$not	Inverts the effect of a query expression and returns documents that do not match the query expression
4.	\$nor	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.

# Logical operators DEMO

## Logical AND

- Specify a logical conjunction (AND) for a list of query conditions by separating the conditions with **comma in** the conditions document.

```
db.restaurants.find(  
{ $and: [ { "cuisine": "Italian" }, { "address.zipcode": "10075" } ] }  
)
```

```
db.restaurants.find({ "cuisine" : "Irish", "address.zipcode": "10019" })
```

```
db.restaurants.find(  
{ $or: [ { "cuisine": "Italian" }, { "address.zipcode": "10075" } ] }  
)
```



# Logical Operators DEMO

```
db.restaurants.find(  
{ $nor: [ { "cuisine": "Italian" }, { "address.zipcode": "10075" } ] }  
)
```

# Evaluation Query Operators

	Name	Description
	\$mod	Performs a modulo operation on the value of a field and selects documents with a specified result
	\$regex	Selects documents where values match a specified regular expression
	\$text	Performs text search
	\$where	Matches documents that satisfy a javascript expression

	Name	Description
	\$geoWithin	Selects geometries within a bounding GeoJSON geometry. The 2dSphere and 2d indexes support \$geoWithin
	\$geoIntersects	Selects geometries that intersect with a GeoJSON geometry.
	\$near	Returns geospatial objects in proximity to a point. Requires a geospatial index
	\$nearSphere	Returns geospatial objects in proximity to a point on a sphere. Requires a geospatial index

# Array

	Name	Description
	\$all	Matches arrays that contain all elements specified in the query
	\$elemMatch	Selects documents if element in the array field matches all the specified \$elemMatch conditions
	\$size	Selects documents if the array field is a specified size

# Bitwise

	Name	Description
	\$bitsAllSet	Matches numeric or binary values in which a set of bit positions all have a value of 1
	\$bitsAnySet	Matches numeric or binary values in which any bit from a set of bit positions has a value of 1
	\$bitsAllClear	Matches numeric or binary values in which a set of bit positions all have a value of 0
	\$bitsAnyClear	Matches numeric or binary values in which any bit from a set of bit positions has a value of 0.

# Projection Operators

	Name	Description
	\$	Projects the first element in an array that matches the query condition
	\$elemMatch	Projects the first element in an array that matches the specified \$elemMatch Condition
	\$meta	Projects the document's score assigned during \$text operations
	\$slice	Limits the number of elements projected from an array. Supports skip and limit slices.



Operators that enable you to modify the data in your database or add additional data.

MONGODB

# UPDATE OPERATORS

# Field Update Operators

Name	Description
\$inc	
\$mul	
\$rename	
\$setOnInsert	
\$set	
\$unset	
\$min	
\$max	
\$currentDate	

# Update Array Operators

Name	Description
\$	Acts as a placeholder to update the first element that matches the query condition in an update
\$addToSet	Adds elements to an array only if they do not already exist in the set
\$pop	Remove the first or last item of an array
\$pullAll	Removes all matching values from an array
\$pull	Removes all array elements that match a specified query
\$pushAll	Deprecated. Adds several items to an array
\$push	Adds an item to an array

# Modifiers Update Operator

Name	Description
\$each	Modifies the \$push and \$addToSet Operators to append multiple items for array updates
\$slice	Modifies the \$push operator to limit the size of the updated arrays
\$sort	Modifies the \$push operator to reorder documents stored in an array
\$position	Modifies the \$push operator to specify the position in the array to add elements.

# Bitwise Update Operator

Name	Description
\$bit	Performs bitwise AND, OR, XOR updates of integer values

# Isolation Update Operators

Name	Description
\$isolated	Modifies the behaviour of a write operation to increase the isolation of the operation



MONGODB: MAPREDUCE AND AGGREGATION FRAMEWORK

# NATIVE DATA PROCESSING TOOLS

# Two Native Data Processing Tools

## Aggregation Framework

- A powerful tool for performing analytics and statistical analysis in real-time and generating pre-aggregated reports for dashboarding.
- Executes in native code
  - Written in C++
  - JSON Parameters
- Flexible, functional and simple
  - Operation pipeline
  - Computational expressions
- Plays nice with sharding

## MapReduce

- Extremely versatile, powerful
- Overkill for simple aggregation tasks
  - Averages
  - Summation
  - Grouping
  - Reshaping
- High level of complexity
- Difficult to program and debug



Operators available to define and manipulate documents in pipeline stages

MONGODB

# AGGREGATION PIPELINE OPERATORS

# Why Aggregation?

- We are storing our data in MongoDB
- Our applications need ad-hoc queries
- We must have a way to reshape data easily
- Aggregation Framework in play to achieve this.

# Aggregation Framework

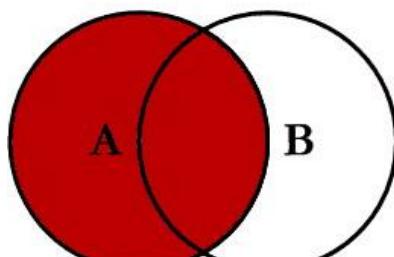
- It is a pipeline for data aggregation modeled on the concept of data processing pipelines.
- Documents enter a multi-stage pipeline that transforms the documents into aggregated results.
- Pipeline consists of stages where in each stage transforms the documents as they pass through.
- Each successive stage reduces the volume of data; removing information that isn't needed and combining other data to produce summarized results
- Running data aggregation on the [mongod](#) instance simplifies application code and limits resource requirements

# What is an Aggregation Pipeline?

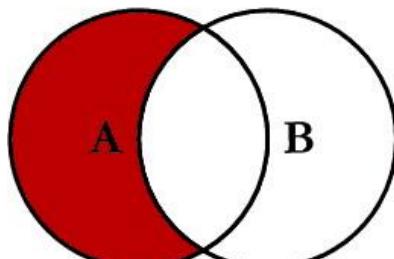
- A series of document transformations
  - Executed in stages
  - Original input is a collection
  - Output as a document, cursor or a collection
- Rich library of functions
  - Filter, compute, group and summarize data
  - Output of one stage sent to input of next
  - Operations executed in sequential order



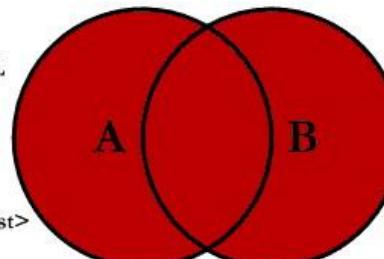
# SQL JOINS



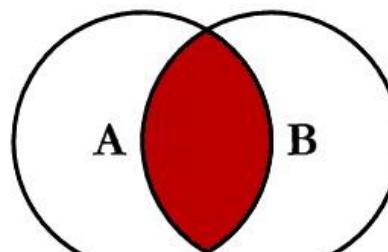
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



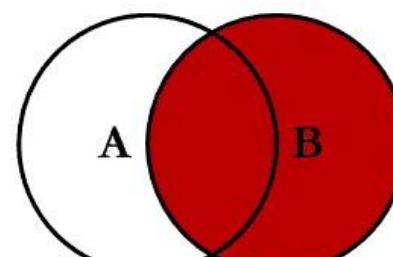
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



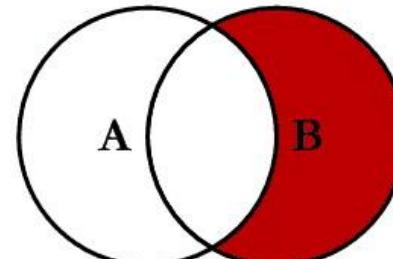
```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



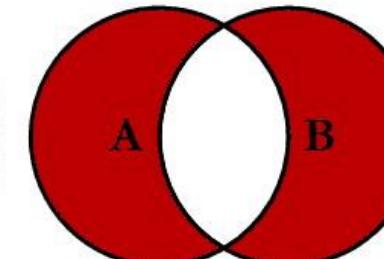
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



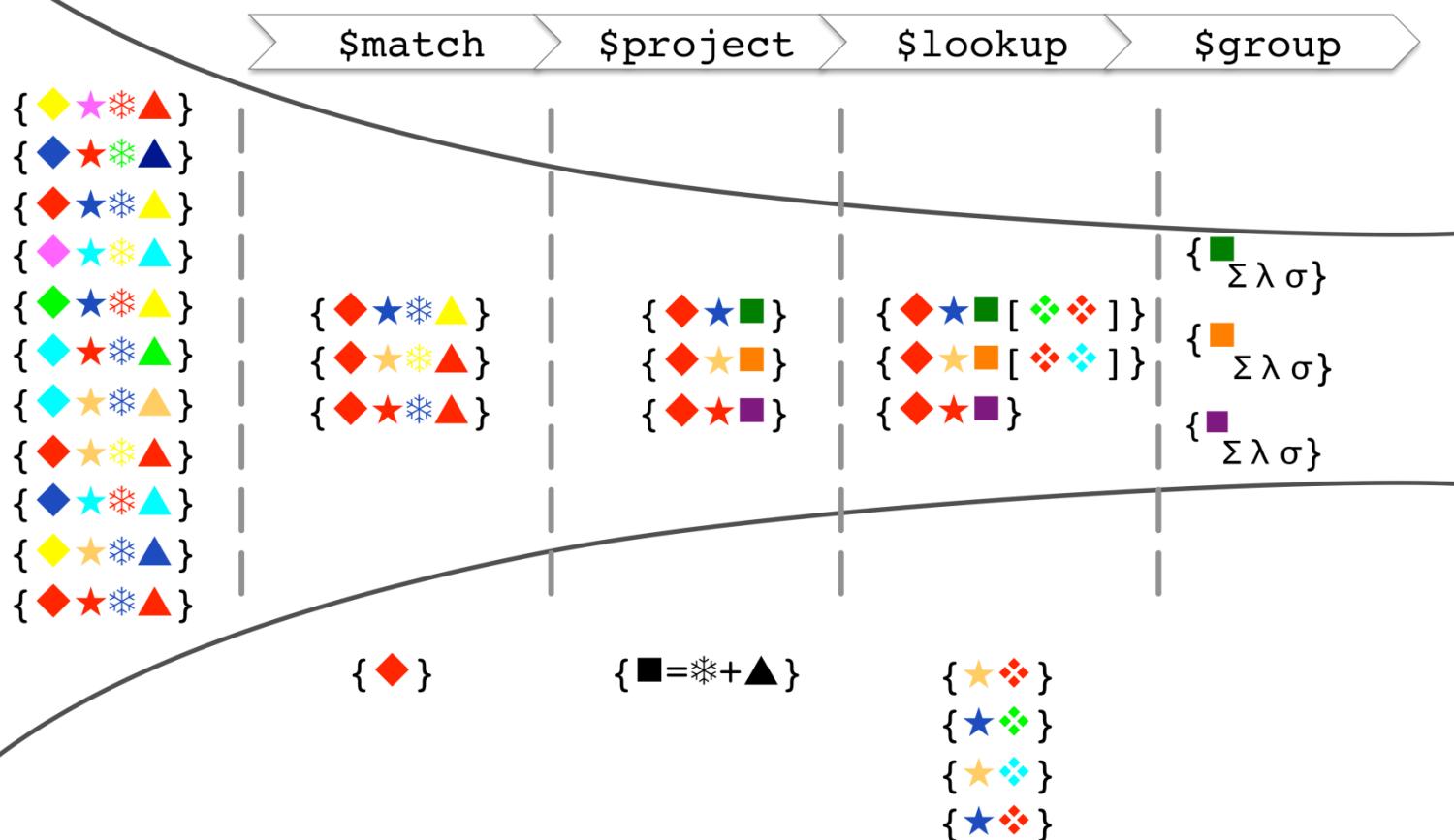
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

© C.L. Moffatt, 2008

# MongoDB Aggregation Framework Pipeline



Source: <http://www.clusterdb.com/tag/mongodb-3-2> (fig:3)

# Aggregation Pipeline:Stage Operators

Name	Description
\$project	Reshapes each document in the stream, such as by adding new fields or removing existing fields.
\$match	Filters the document stream to allow only matching documents to pass unmodified into the next pipeline stage. It uses standard MongoDB queries.
\$redact	Reshapes each document in the stream by restricting the content for each document based on information stored in the documents themselves.
\$limit	Passes the first n documents unmodified to the pipeline where n is the specified limit.
\$skip	Skips the first n documents where n is the specified skip number and passes the remaining documents unmodified to the pipeline.
\$unwind	Deconstructs an array field from the input documents to output a document for each element.

# Aggregation Pipeline:Stage Operators

Name	Description
\$group	Groups input documents by a specified identifier expression and applies the accumulator expressions. Consumes all input documents and outputs one document per each distinct group.
\$sample	Randomly selects the specified number of documents from its input
\$sort	Reorders the document stream by a specified sort key.
\$geoNear	Returns an order stream of documents based on the proximity to a geospatial point.
\$lookup	Performs a left outer join to another collection in the same database to filter in documents from the joined collection for processing
\$out	Writes the resulting documents of the aggregation pipeline to a collection. To use the \$out stage it must be the last stage in the pipeline
\$indexStats	Returns statistics regarding the use of each index for the collection

# SQL to Aggregation Mapping

	SQL Terms, Functions and Concepts	MongoDB Aggregation Operators
	WHERE	\$match
	GROUP BY	\$group
	HAVING	\$match
	SELECT	\$project
	LIMIT	\$limit
	SUM()	\$sum
	COUNT()	\$sum
	Join	\$lookup

# Aggregation Operators(Array)

Name	Description
\$slice	
\$arrayElemAt	
\$concatArrays	
\$isArray	
\$filter	
\$min	
\$max	
\$avg	
\$sum	

# Aggregation Operators

Operation	Name	Description
Standard Deviations	<code>\$stdDevSamp</code>	Based on a sample
Standard Deviations	<code>\$stdeDevPop</code>	Based on the complete population
Square Root	<code>\$sqrt</code>	
Absolute value	<code>\$abs</code>	
Rounding numbers	<code>\$trunc</code>	
	<code>\$ceil</code>	
	<code>\$floor</code>	
Logarithms	<code>\$log</code>	
	<code>\$log10</code>	
	<code>\$ln</code>	
Raise to power	<code>\$pow</code>	
Natural Exponent	<code>\$exp</code>	

# Usage

```
use peoplerecords;
db.orders.aggregate([
  {$match:{'status':'A'}},
  {$group:{'_id':'$guid', total:{$sum:'$amount'}}}
])
```

Collection

```
db.orders.aggregate([
  $match stage → { $match: { status: "A" } },
  $group stage → { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }
])
```

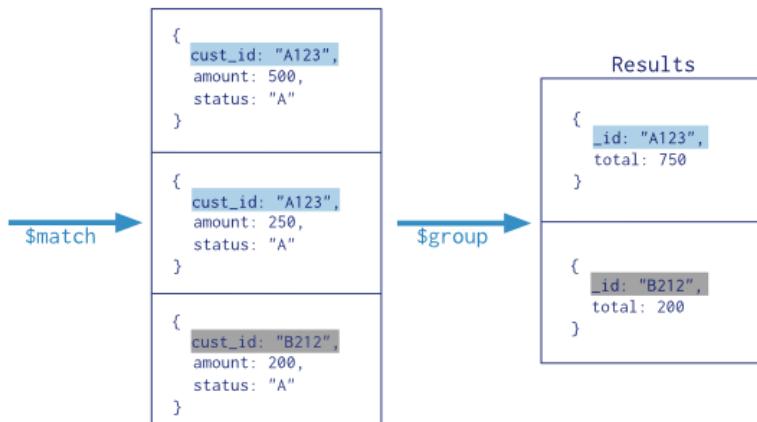
```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}

{
  cust_id: "A123",
  amount: 250,
  status: "A"
}

{
  cust_id: "B212",
  amount: 200,
  status: "A"
}

{
  cust_id: "A123",
  amount: 300,
  status: "D"
}
```

orders



Results

```
{
  "_id": "A123",
  "total": 750
}

{
  "_id": "B212",
  "total": 200
}
```

```
{
  "_id": null,
  "total": NumberInt(280557)
}
```

# Aggregation Example with ZipCode

- Output Aggregation Method

```
use zipcode;
db.zips.aggregate([
  {$group: {_id:"$state", totalPop:{$sum: "$pop"}},},
  {$match:{totalPop:{$gte:10*1000*1000}}}
])
```

```
use zipcode;
db.zips.aggregate([
  { $group: { _id: { state: "$state", city: "$city" }, pop: { $sum: "$pop" } } },
  { $group: { _id: "$_id.state", avgCityPop: { $avg: "$pop" } } }
])
```

## Equivalent SQL CODE

```
SELECT state, SUM(pop) AS totalPop
FROM zipcodes
GROUP BY state
HAVING totalPop >= (10*1000*1000)
```

- Output

_id	totalPop
"CA"	29760021
"PA"	11881643
"NY"	17990455
"TX"	16986510
"FL"	12937926
"OH"	10847115
"IL"	11430602
_id	avgCityPop
"AK"	2989.36413043...
"KY"	4676.77157360...
"LA"	10343.0710784...
"NY"	13122.1407731...
"CO"	9922.87349397...
"PA"	8679.06720233...
"VT"	2315.87654320...
"SD"	1826.78215223...
"ME"	2994.94634146...
"NJ"	15775.8938775...
"NH"	5232.32075471...
"WV"	2759.19538461...
"TN"	9656.35049504...

# Aggregate: Largest and Smallest City by State Example

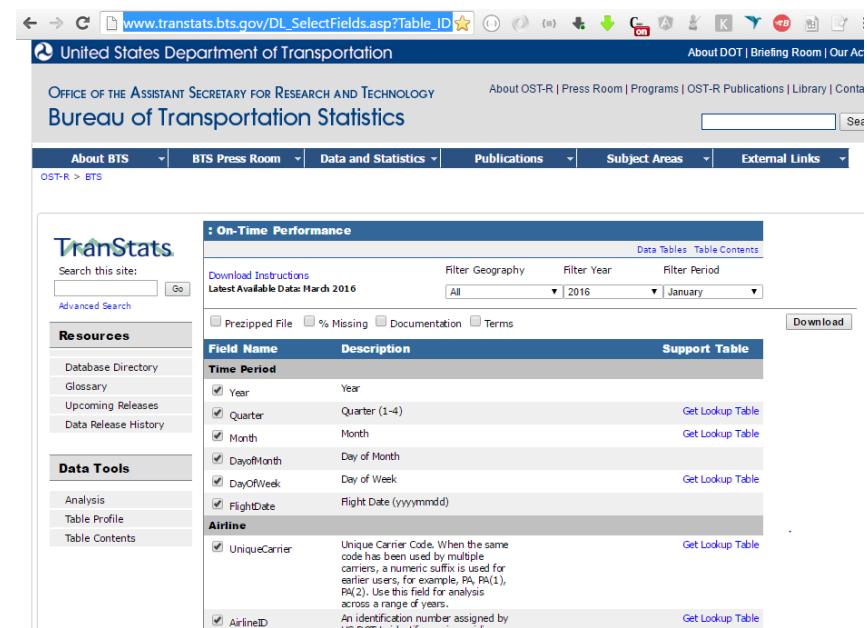
```
db.zips.aggregate([
  { $group: { _id: { state: "$state", city: "$city" },
    pop: { $sum: "$pop" } } },
  { $sort: { pop: 1 } },
  { $group: { _id: "$_id.state",
    biggestCity: { $last: "$_id.city" },
    biggestPop: { $last: "pop" },
    smallestCity: { $first: "$_id.city" },
    smallestPop: { $first: "pop" }
  },
  { $project:
    { _id: 0,
      state: "$_id",
      biggestCity: { name: "$biggestCity", pop: "$biggestPop" },
      smallestCity: { name: "$smallestCity", pop: "$smallestPop" }
    }
  }
])
```

biggestCity	smallestCity	state
○{ 2 fields }	○{ 2 fields }	"_" ID
○{ 2 fields }	○{ 2 fields }	"_" WY
○{ 2 fields }	○{ 2 fields }	"_" NM
○{ 2 fields }	○{ 2 fields }	"_" WI
○{ 2 fields }	○{ 2 fields }	"_" KS
○{ 2 fields }	○{ 2 fields }	"_" NH
○{ 2 fields }	○{ 2 fields }	"_" NJ
○{ 2 fields }	○{ 2 fields }	"_" SD
○{ 2 fields }	○{ 2 fields }	"_" ME
○{ 2 fields }	○{ 2 fields }	"_" VT
○{ 2 fields }	○{ 2 fields }	"_" NE
○{ 2 fields }	○{ 2 fields }	"_" GA
○{ 2 fields }	○{ 2 fields }	"_" LA

# Outline of Aggregation Stages

1. Grouping
2. Matching/Filtering
3. Projection
4. Sorting
5. Unwind
6. Limit,skip
7. Added in 2.6
  1. Out
  2. React

[http://www.transtats.bts.gov/DL\\_SelectFields.asp?Table\\_ID=236](http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236)



The screenshot shows a web browser displaying the United States Department of Transportation's Bureau of Transportation Statistics website. The URL in the address bar is [www.transtats.bts.gov/DL\\_SelectFields.asp?Table\\_ID=236](http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236). The page title is "On-Time Performance". The main content area displays a table with columns for "Field Name", "Description", and "Support Table". The table includes rows for Time Period (Year, Quarter, Month, DayOfMonth, DayOfWeek, FlightDate) and Airline (UniqueCarrier, AirlineID). Each row has a "Get Lookup Table" link next to it.

Field Name	Description	Support Table
Year	Year	<a href="#">Get Lookup Table</a>
Quarter	Quarter (1-4)	<a href="#">Get Lookup Table</a>
Month	Month	<a href="#">Get Lookup Table</a>
DayOfMonth	Day of Month	<a href="#">Get Lookup Table</a>
DayOfWeek	Day of Week	<a href="#">Get Lookup Table</a>
FlightDate	Flight Date (yyyyMMdd)	<a href="#">Get Lookup Table</a>
UniqueCarrier	Unique Carrier Code. When the same code has been used for multiple carriers, a suffix is used for earlier users, for example, PA, PA(1), PA(2). Use this field for analysis across a range of years.	<a href="#">Get Lookup Table</a>
AirlineID	An identification number assigned by the DOT to identify an airline.	<a href="#">Get Lookup Table</a>

Example using FlightData

# Grouping Example

```
{
  "_id" : ObjectId("5746842170c69035247ec837"),
  "guid" : "aa135af5-598a-46d4-9b94-c77b5602a8ca",
  "websitename" : "dailymotion.com",
  "hosted_at" : "'reliancedigital.com'",
  "rating" : "4",
  "ip_address" : "120.171.182.111"
}
```

```
db.hosting.aggregate( {
  $group : { _id : "$hosted_at",
    total : { $sum : 1 } },
  { $sort : {total : -1}
} );
```

```
use peoplerecords;
db.hosting.aggregate(
  { $match : {"hosted_at" : "'reliancedigital.com'"} },
  { $group : { _id : "$hosted_at", {
} } }
```

```
{
  "_id" : "'reliancedigital.com'",
  "total" : NumberInt(221)
}
```

```
SELECT hosting, SUM(hosting) AS total
FROM website
GROUP BY hosting
```

```
"_id" : "'reliancedigital.com'",
"total" : NumberInt(221)
}
```

# Inserting grouped data into a new collection

```
use peoplerecords;
var groupdata = db.hosting.aggregate( {
  $group : { _id : "$hosted_at",
    total : { $sum : 1 } },
  { $sort : {total : -1} }
} );
db.hostinggroup.insert(groupdata.toArray());
```

hostinggroup	
_id	total
"_n" 'itc-cloud.com'	1.23 541.0
"_n" 'reliancedigital....'	1.23 221.0
"_n" 'netsol.com'	1.23 117.0
"_n" 'net4domains.c...'	1.23 58.0
"_n" 'namecheap.co...'	1.23 28.0
"_n" 'cloud.google.c...'	1.23 13.0
"_n" 'hostgator.com'	1.23 11.0
"_n" 'aws.amazon.c...'	1.23 11.0
null	1.0



# Mongo Aggregation

```
db['customer-order'].aggregate([
{$group:{_id:null, count:{$sum:1}}}]));
```

```
var customerorderprice=db['customer-order'].aggregate([
{ $unwind: "$items" },
{
  $group: {
    _id: "$customerId",
    price: { $sum: "$items.price" }
  }
}]);
db.customerordergroup.insert(customerorderprice.toArray());
```

```
use peoplerecords;
db['mkretail-customerorder'].aggregate([
{$group:{_id:null, total:{$sum:"$price"}}}])
```

```
use people records
db['mkretail-customerorder'].aggregate([
{$group:{_id:"$customerId",total:{$sum:"$price"}},},
{$sort:{total:1}}]);
```

# Equivalent SQL Statement

```
SELECT COUNT(*) AS count
FROM orders
```

```
SELECT SUM(price) AS total
FROM orders
```

```
SELECT cust_id,
       SUM(price) AS total
  FROM orders
 GROUP BY cust_id
 ORDER BY total
```



# Mongo

```
use people records
db['mkretail-customerorder'].aggregate([
  { $group: {
    _id: {
      cust_id: "$customerId",
      ord_date: "$orderDate",
    },
    total: { $sum: "$price" }
  }
}
]);
```

```
use people records
db['mkretail-customerorder'].aggregate([
{
  $group: {
    _id: "$customerId",
    count: { $sum: "$price" }
  }
},
{ $match: { count: { $gt: 5000 } } }
]);
```

# Equivalent SQL Syntax

```
SELECT cust_id,
       ord_date,
       SUM(price) AS total
  FROM orders
 GROUP BY cust_id,
          ord_date
```

```
SELECT cust_id,
       count(*)
  FROM orders
 GROUP BY cust_id
 HAVING count(*) > 5000
```



## Left-Outer Join using \$lookup

```
use SalesData;
db.homesales.createIndex({amount:1});
db.homesales.aggregate([
{$match: {
  amount:{$gte:300000}}
}
]);
```

```
db.homesales.aggregate([
{$match: {
  amount: {$gte:3000000}}
},
{$lookup: {
  from: "postcodes",
  localField: "address.postcode",
  foreignField: "postcode",
  as: "postcode_docs"}
}
]);
```



## Left Outer Join and Projection

```
db.homesales.aggregate([
  {$match: {
    amount: {$gte: 3000000}}
  },
  {$lookup: {
    from: "postcodes",
    localField: "address.postcode",
    foreignField: "postcode",
    as: "postcode_docs"}
  },
  {$project: {
    _id: 0,
    saleDate: "$date",
    price: "$amount",
    address: 1,
    location: "$postcode_docs.location"}},
  {$sort:
    {
      price: -1
    }
  }
]);|
```

## Sales Analysis

```
db.homesales.aggregate([
  {
    $group:
    {
      _id: {$year: "$date"},
      higestPrice: {$max: "$amount"},
      lowestPrice: {$min: "$amount"},
      averagePrice: {$avg: "$amount"},
      priceStdDev: {$stdDevPop: "$amount"}
    }
  },
  {
    $sort: {_id: 1}
  }
]);|
```

# Sales Analysis + Projections

```
db.homesales.aggregate([
  {$group:
    { _id: { $year: "$date" },
      higestPrice: { $max: "$amount" },
      lowestPrice: { $min: "$amount" },
      averagePrice: { $avg: "$amount" },
      priceStdDev: { $stdDevPop: "$amount" }
    }
  },
  {
    $sort: { _id: 1 }
  },
  { $project:
    { _id: 1,
      higestPrice: 1,
      lowestPrice: 1,
      averagePrice: { $trunc: "$averagePrice" },
      priceStdDev: { $trunc: "$priceStdDev" }
    }
  }
])
```

# Output

_id	higestPrice	lowestPrice	averagePrice	priceStdDev
1995	1000000	12000	114168.0	81604.0
1996	975000	9000	118862.0	79871.0
1997	995860	10500	134122.0	89210.0
1998	1600000	26500	157237.0	115997.0
1999	5425000	20000	182284.0	179601.0
2000	1400000	17500	215181.0	149351.0
2001	3025000	47500	235523.0	178761.0
2002	2340000	17000	253861.0	167209.0
2003	1750000	40000	272040.0	168551.0
2004	2200000	10500	307522.0	199722.0
2005	1734804	31250	299161.0	164604.0
2006	2425000	87500	317644.0	187727.0
2007	3740000	81500	366076.0	264209.0
2008	3600000	51000	354766.0	253228.0
2009	1995000	10000	341006.0	213824.0
2010	4000000	69000	365928.0	254609.0
2011	2600000	41250	374499.0	276434.0
2012	3000000	77000	381991.0	267877.0
2013	5148500	56250	388687.0	286683.0
2014	3950000	66950	440084.0	269638.0
2015	1688000	125000	451940.0	228548.0

## \$out

```
db.homesales.aggregate([
  {$group:
    { _id: { $year: "$date" },
      highestPrice: { $max: "$amount" },
      lowestPrice: { $min: "$amount" },
      averagePrice: { $avg: "$amount" },
      priceStdDev: { $stdDevPop: "$amount" }
    }
  },
  { $sort: { _id: 1 }
  },
  { $project:
    {
      _id: 0,
      year: "$_id",
      highestPrice: 1,
      lowestPrice: 1,
      averagePrice: { $trunc: "$averagePrice" },
      priceStdDev: { $trunc: "$priceStdDev" }
    }
  },
  { $out: "annualHomePrices" }
]);
```

## Analysis from \$out

```
db.annualHomePrices.aggregate([
  {$project:
    { Year: "$year",
      hightToLowPriceGap: {
        $subtract: [ "$highestPrice", "$lowestPrice" ]
      },
      _id: 0
    }
  }
]);
```



# Location Analysis

```
db.homesales.aggregate([
  { $sort: {amount: -1}}
],
{
  $group:
  {
    _id: {$year: "$date"},  

    priciestPostCode: {$first: "$address.postcode"}
  }
},
{
  $lookup:
  { from: "postcodes",
    localField: "priciestPostCode",
    foreignField: "postcode",
    as: "locationData"
  }
},
{$sort: {_id: -1}}
],
{$project:
  { _id: 0,
    Year: "$_id",
    PostCode: "$priciestPostCode",
    Location: "$locationData.location"
  }
}
])|
```

## Hottest location

```
db.homesales.aggregate([
  {$sort: {amount: -1}}
],
{
  $group:
  {
    _id: {$year: "$date"},  

    priciestPostCode: {$first: "$address.postcode"}
  }
},
{
  $lookup:
  { from: "postcodes",
    localField: "priciestPostCode",
    foreignField: "postcode",
    as: "locationData"
  }
},
{$sort: {_id: -1}}
],
{$project:
  { _id: 0,
    Year: "$_id",
    PostCode: "$priciestPostCode",
    Location: "$locationData.location"
  }
},
{$out: "hottestLocations"
})|
```

# Using \$geoNear

## CreateIndex

- db.postcodes.createIndex({location: "2dsphere"})

```
db.postcodes.aggregate([
  {
    $geoNear:
    {
      near:
      {
        "type": "Point",
        "coordinates": [
          51.5156725,
          -0.727387
        ],
        distanceField: "distance",
        num: 5,
        spherical: true
      }
    },
    {
      $group: {
        _id: 2006,
        "neighbours": {
          $addToSet: "$postcode"
        }
      }
    }
]);

```

## \$geoNear

- Order/filter Documents by location
  - Requires a geospatial index
  - Output includes physical distance
  - Must be first aggregation stage



# \$match

- Filter documents
  - Uses existing query syntax
  - Can facilitate shard exclusion
  - No \$where (server side javascript)



MONGODB

# MAP-REDUCE

# MapReduce

A data process paradigm for condensing large volumes of data into useful aggregated results. For map-reduce operations, MongoDB provides the **mapReduce** database command

**mapReduce** command takes 2 primary inputs, the mapper function and the reducer function.

A **mapper** is to read a collection of data and building a map with only the required fields to process and group them into one array based on the key, which is fed into the **Reducer**, to process the values.



## RAW DATA

```
//raw json data  
[  
  {  
    customerId:syedawase,  
    totalPrice:780  
  },  
  {  
    customerId:syedrayyan,  
    totalPrice:999  
  },  
  {  
    customerId:syedameese,  
    totalPrice:876  
  },  
  {  
    customerId:syedazeez,  
    totalPrice:324  
  },  
  {  
    customerId:syedawase,  
    totalPrice:1279  
  },  
  {  
    customerId:syedawase,  
    totalPrice:78  
  },  
  {  
    customerId:syedrayyan,  
    totalPrice:342  
  },  
  {  
    customerId:syedameese,  
    totalPrice:1542  
  }]
```

## MAPPER

key	value
syedawase	[780, 1279, 78]
syedrayyan	[999, 342]
syedameese	[876, 1542]
syedazeez	[324]

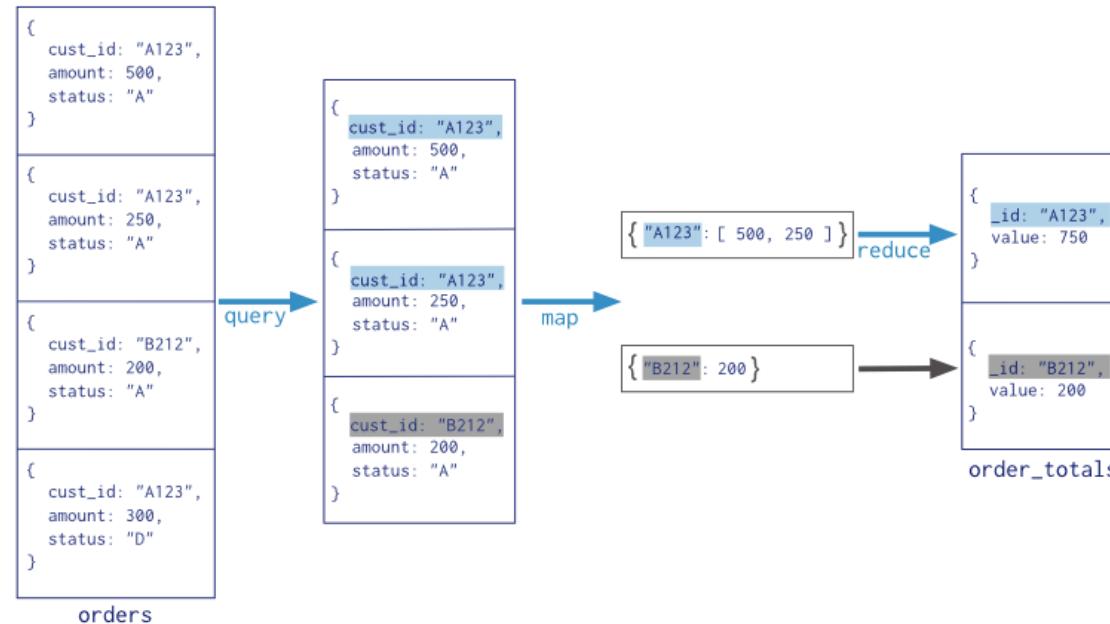
## REDUCER

key	value
syedawase	[2137]
syedrayyan	[1341]
syedameese	[2418]
syedazeez	[324]

# Map-Reduce

Collection  
`db.orders.mapReduce(`

- `map` → `function() { emit( this.cust_id, this.amount ); },`
- `reduce` → `function(key, values) { return Array.sum( values ) },`
- `query` → `query: { status: "A" },`
- `output` → `out: "order_totals"`

`)`


# Simple Vote Example

```
{  
    "_id" : ObjectId("574af4e370c64d7e46fafdae"),  
    "id" : NumberInt(1),  
    "fullname" : "Jose Wright",  
    "votes" : NumberInt(1),  
    "comment" : "Pellentesque at nulla. Suspendisse potenti.  
    "status" : "E"  
}
```

```
var map = function(){  
    emit(this.status, {votes:this.votes});  
};
```

```
var reduce= function(key,values){  
    var sum =0;  
    values.forEach(function(doc){  
        sum+=doc.votes;  
    });  
    return{votes:sum};  
};
```

```
var op = db.fictionalvote.mapReduce(map, reduce, {out:"fictionalresults"});
```



MONGODB

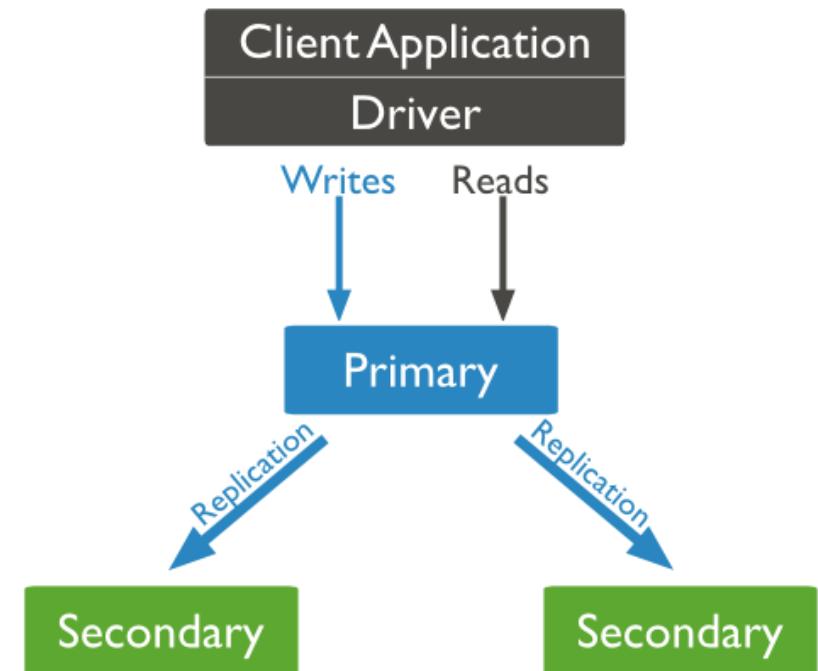
# REPLICATION AND SHARDING

# Replication

- The process of synchronizing data across multiple servers.
- Replication provides redundancy and increases data availability with multiple copies of data on different database servers.
- Protects a database from the loss of a single server
- Allows to recover from hardware failure and service interruptions.

# Why Replication?

- To keep your data safe
- High availability of data
- Disaster recovery
- No downtime for maintenance (like backups, index rebuilds, compaction)
- Read scaling (extra copies to read from)
- Replica set is transparent to the application



# Replication Working?

- A replica set is a group of **mongod** instances that host the same data set.
- One node is primary node that receives all write operations.
- All other instances, secondaries, apply operations from the primary so that they have the same dataset.
- A **replica set can have only one primary node**
- **All data replicates from primary to secondary node**
- **At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected.**
- **After the recovery of failed node, it again join the replica set and works a secondary node**

# Replica set feature

- A cluster of N nodes
- Anyone node can be primary
- All write operations goes to primary
- Automatic failover
- Automatic Recovery
- Consensus election of primary

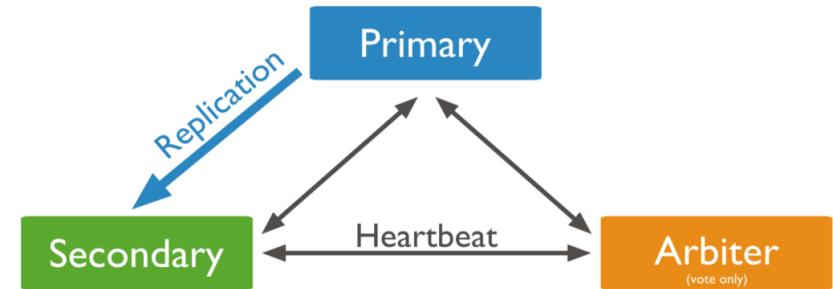


Figure 3: Diagram of a replica set that consists of a primary, a secondary, and an arbiter.

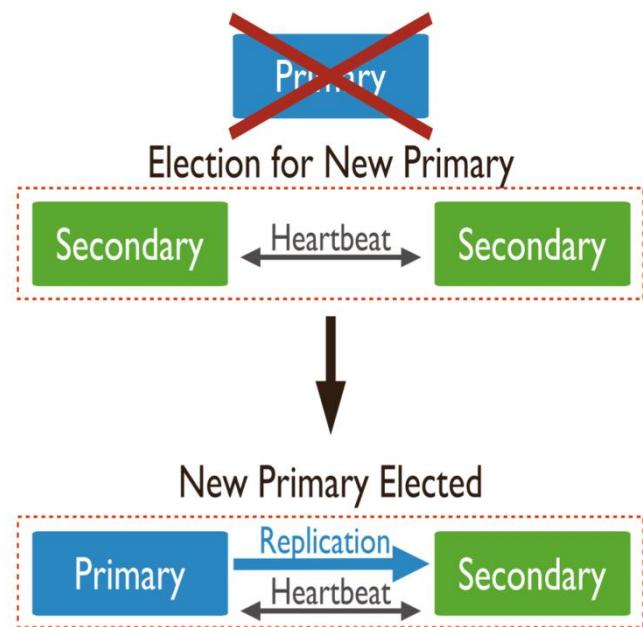


Figure 21: Diagram of an election of a new primary. In a three member replica set with two secondaries, the primary becomes unreachable. The loss of a primary triggers an election where one of the secondaries becomes the new primary

# Setting up a replica set

rs.add()	Adds a member to a replica set. To run this method, you must connect to the primary of the replica set
rs.addArb()	Adds a new arbiter to an existing replica set
rs.conf()	Returns a document that contains the current replica set configuration
rs.freeze()	Makes the current replica set member ineligible to become primary for the period specified.
rs.help()	Returns a basic help text for all of the replication related shell functions
rs.initiate()	Initiates a replica set. Optionally, the method can take an argument in the form of a document that holds the configuration of a replica set

rs.printReplicationInfo()	Prints a report of the status of the replica set from the perspective of the primary
rs.printSlaveReplicationInfo()	Prints a report of the status of the replica set from the perspective of the secondaries
rs.reconfig()	Re-configures a replica set by applying a new replica set configuration object
rs.remove()	Remove a member from a replica set
rs.slaveOk()	Sets the slaveOK property for the current connection. Deprecated.
Rs.status()	Returns a document with information about the state of the replica set
Rs.stepDown()	Causes the current primary to become secondary which forces an election
Rs.syncFrom()	Sets the member that this replica set member will sync from , overriding the default sync target selection logic

# Provisioning Your Cloud Instance

- Provision your cloud instance.
- Add necessary endpoints
- Install MongoDB

NAME	PROTOCOL	PUBLIC PORT	PRIVATE PORT
SSH	TCP	22	22
HTTP	TCP	80	80
HTTPS	TCP	443	443
PowerShell	TCP	5986	5986
Remote Desktop	TCP	AUTO	3389
SMTPS	TCP	587	587
FTP	TCP	21	21
mongoport1	TCP	27018	27018
mongoport2	TCP	27019	27019
mongoport3	TCP	27020	27020
mongoport4	TCP	27021	27021
mongoport5	TCP	27022	27022

```
apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
echo "deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen" | tee -a /etc/apt/
sources.list.d/10gen.list
apt-get -y update
apt-get -y install mongodb-10gen
```



# /etc/hosts

```
127.0.0.1 localhost
104.45.153.17 mongoprimer.cloudapp.net
40.114.9.123 mongol-second.cloudapp.net
23.101.128.34 mongolthree.cloudapp.net
```

- Mongodb.conf settings
- Port= 27018
- replSet=rs0
- Fork=true
- Set Master for Master and domain name
- Set Slave for Slave and source domain name to master domain name



New Connection

Enter a name for this connection:  
MONGOLSECONDARY

Server Authentication SSL SSH Tunnel Advanced

Import URI

Enter your connection URL:  
mongodb://40.114.9.123:27018

OK Cancel

From URI... Use this option to import connection details from a URI

To URI... Use this option to export complete connection details to a URI

Test Connection Save Cancel

New Connection

Enter a name for this connection:  
MONGOLSECONDARY

Server Authentication SSL SSH Tunnel Advanced

Authentication Mode: Standard (MONGODB-CR or SCRAM-SHA-1)

User name: awase

Password:

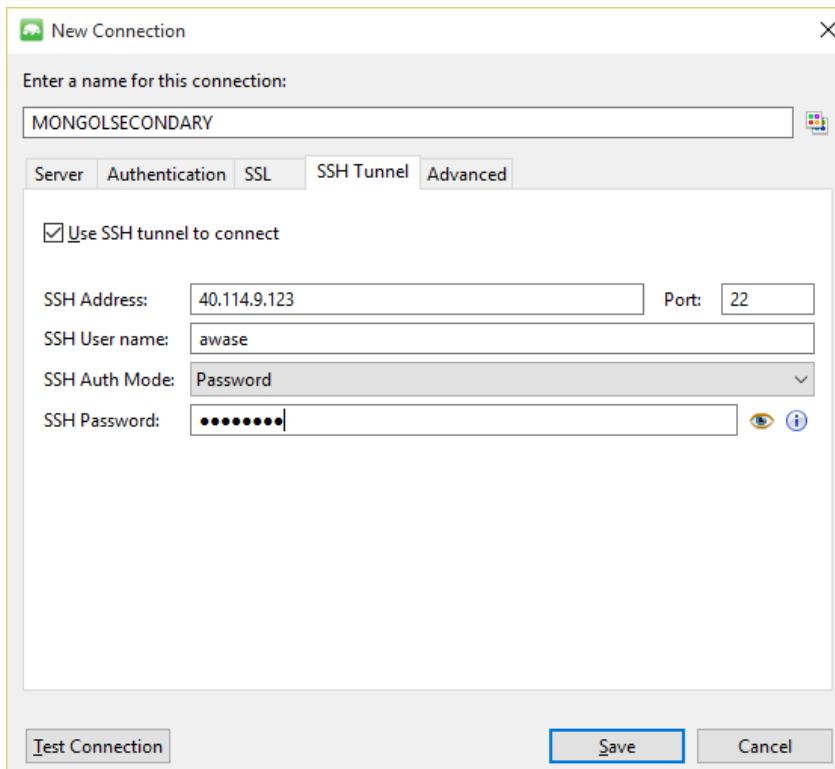
Authentication DB: admin  
The database where the user is defined

I would like to authenticate using Kerberos (GSSAPI) or LDAP (PLAIN)

Manually list visible databases

Test Connection Save Cancel

# Mongodb.conf



- Port= 27018
- replSet=rs0
- Fork=true
- Run the command
  - Mongod –config /etc/mongodbconfig/mongodb.conf



# Starting the Replication Set and Add Members

- Mongo
- rs.initiate()
- rs.conf()
- rs.add("mongol-second.cloudapp.net")

```
> rs.conf()
{
    "_id" : "master",
    "version" : 1,
    "members" : [
        {
            "_id" : 0,
            "host" : "masterone:27017"
        }
    ]
}

master:PRIMARY> rs.add("sak-slave.cloudapp.net");
{ "ok" : 1 } _
```



```
master:PRIMARY> rs.conf()
{
    "_id" : "master",
    "version" : 3,
    "members" : [
        {
            "_id" : 0,
            "host" : "masterone:27017"
        },
        {
            "_id" : 1,
            "host" : "sak-slave.cloudapp.net:27017"
        },
        {
            "_id" : 2,
            "host" : "sak-slavetwo.cloudapp.net:27017"
        }
    ]
}

master:PRIMARY> rs.initiate();
{
    "info" : "try querying local.system.replset to see current configuration",
    "ok" : 0,
    "errmsg" : "already initialized"
}
...
```



# Single Server

- Primary

```
#where to log  
logpath=/var/log/mongodb/primary/mongodb.log
```

- rs.reconfig(cfg,{force:true});

```
root@SyedAwaseMI:/# mongod --config /etc/mongodb-config/mongod-primary.conf  
about to fork child process, waiting until server is ready for connections.  
forked process: 12568  
all output going to: /var/log/mongodb/primary/mongodb.log
```

```
root@SyedAwaseMI:/# mongod --config /etc/mongodb-config/mongod-secondary.conf  
about to fork child process, waiting until server is ready for connections.  
forked process: 12616  
all output going to: /var/log/mongodb/secondary/mongodb.log
```

```
root@SyedAwaseMI:/# mongod --config /etc/mongodb-config/mongod-secondary2.conf  
about to fork child process, waiting until server is ready for connections.  
forked process: 12663  
all output going to: /var/log/mongodb/secondary2/mongodb.log
```

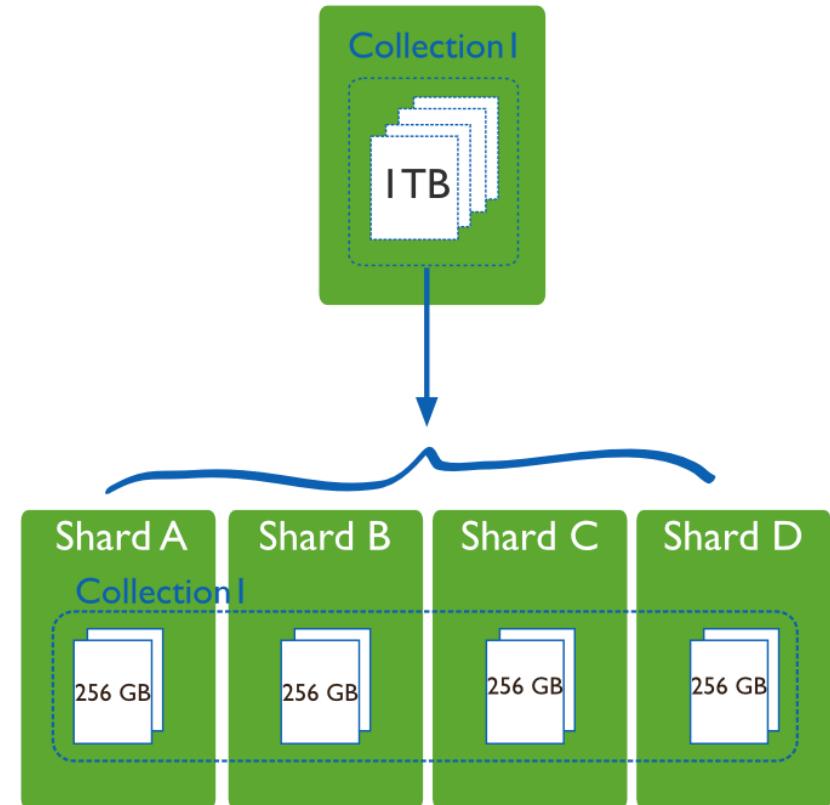
```
root@SyedAwaseMI:/# mongod --config /etc/mongodb-config/mongod-secondary3.conf  
about to fork child process, waiting until server is ready for connections.  
forked process: 12711  
all output going to: /var/log/mongodb/secondary3/mongodb.log
```

# Sharding

- If you had to increase reads by 10x, how would we achieve it?
- What does it take to increase the reads from 10x to 100x.
- Traditionally
  - We add more power
    - CPUs
    - Memory
    - Disks
  - Replication
- Expensive software and hardware
- Problems emerge at scale
  - Performance becomes less predictable

# Sharding

- Horizontal partitioning
- Database federation  
(which is a form of partitioning)
  - A shared nothing architecture
  - Dividing data across multiple containers
  - Scale containers independently
  - Data is distributed across multiple containers



# DATA DISTRIBUTION

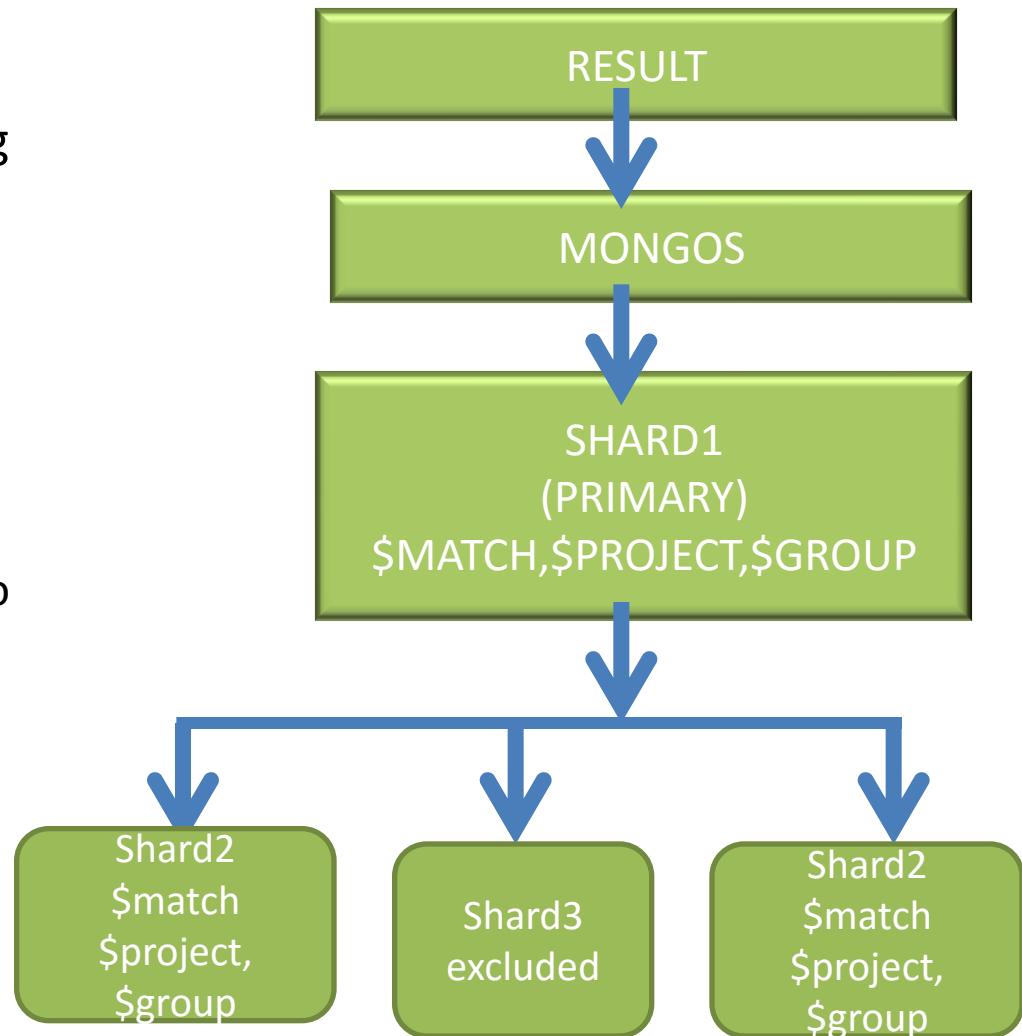
- Uniform
  - Group
  - Random
  - Distinct
- 
- Why to shard/Properties of Shard?
    - They are isolated
    - No single point of failure
    - Shards can be made redundant

# Sharding

- The process of storing data records across multiple machines. As the size of the data increases, a single machine may not be sufficient to store data nor provide and acceptable read and write throughput.
- Sharding solves the problem with horizontal scaling.
- With sharding, you add more machines to support data growth and the demands of read and write operations.

# Implementing Sharding Topology

- Shards are implemented by using clusters which are nothing but a group of MongoDB instances.
- Components of a Shard include
  - A Shard – a mongoDB instance which holds the subset of the data. In production environments, all shards need to be part of replica sets.
  - Config server – a mongoDB instance which holds metadata about the cluster.
  - A Router – responsible to redirect the commands sent by clients to the right servers.



# Sharding

- Workload split between shards
  - Shards execute pipeline up to a point
  - Primary shard merges cursors and continues processing
  - Use explain to analyze pipeline split
  - Early \$match may excuse shards
  - Potential CPU and memory implications for primary shardhost
- Sharding Mechanisms
  - Range Sharding
  - List Sharding
  - Hash Sharding

# Sharding Mechanisms

- **List Sharding**
  - Creates buckets based on intrinsic data properties
  - Grouping data based on age or gender or ethnicity or state or region
  - Similar Feature utilization
  - Similar growth patterns
  - Can be grouped by functionality, feature grouping
- **Problems**
  - Requires domain knowledge to create lists
  - Lists may see uneven growth
  - Taxonomies may become complex
- **Range Based Sharing**
  - Easy to get started with
  - Application can predict where a row goes
  - Ranges can be merged easily
    - Fractional data re-write required
- **Problems**
  - No guarantee of even distribution of data
  - No guarantee of even activity pattern
  - Difficult to scale under load, potential for write hotspots
  - Early ranges may become sparse/unused.

# Sharding Mechanism

- Hash Based Sharding
  - Apply a hash function to a key
  - Use the output to find the appropriate container, or region based data classification based on a hash key
  - Incredibly easy to get started
  - Scales well
    - No write hot spots
    - Even distributions of data across shards
- Problems with Hash Based Sharding
  - Reporting can be very difficult
    - Separate reporting systems
  - Redistributing data can be difficult
    - Data may need to be re-written
    - Consistent hashing solves this problem

# Summary of Sharding mechanisms

- Range based sharding
  - Can be difficult to scale
  - Easy to get started
- List based sharding
  - Really easy to get started
  - Requires domain knowledge
- Hash based sharding
  - Can be complex to get right
  - Handles scale well
- When to shard?
  - High transaction volume
  - Mixed workload
  - Contention on common data

# mongos

- mongos for “MongoDB Shard,” is a routing service for MongoDB shard configurations that processes queries from the application layer, and determines the location of this data in the sharded cluster, in order to complete these operations.

Sharding increases write capacity, provides the ability to support larger working sets, and raises the limits of total data size beyond the physical resources of a single node." as per the MongoDB Manual.



# Cluster Setup

- How many shards = 4
- Replication factor =3
- $3 \times 4 = 12$  mongod => shard servers
- For Demo on a Single Machine
- Best Practice
  - Run mongos on the standard mongodb – top port 27017
  - Do not run shard server mongod's nor config server on that port

# Cluster Setup: Shard settings

- For each shard
  - Initiate the replica set
  - **Add** the shard using
    - `Sh.addShard()`

Name	Description
<a href="#"><u>sh.adminCommand()</u></a>	Runs a <a href="#">database command</a> against the admin database, like <a href="#"><u>db.runCommand()</u></a> , but can confirm that it is issued against a <a href="#"><u>mongos</u></a> .
<a href="#"><u>sh.getBalancerLockDetails()</u></a>	Reports on the active balancer lock, if it exists.
<a href="#"><u>sh.checkFullName()</u></a>	Tests a namespace to determine if its well formed.
<a href="#"><u>sh.checkMongos()</u></a>	Tests to see if the <a href="#"><u>mongo</u></a> shell is connected to a <a href="#"><u>mongos</u></a> instance.
<a href="#"><u>sh.lastMigration()</u></a>	Reports on the last <a href="#"><u>chunk</u></a> migration.
<a href="#"><u>sh.addShard()</u></a>	Adds a <a href="#"><u>shard</u></a> to a sharded cluster.
<a href="#"><u>sh.addShardTag()</u></a>	Associates a shard with a tag, to support <a href="#"><u>tag aware sharding</u></a> .

<a href="#"><u>sh.addTagRange()</u></a>	Associates range of shard keys with a shard tag, to support <a href="#"><u>tag aware sharding</u></a> .
<a href="#"><u>sh.removeTagRange()</u></a>	Removes an association between a range shard keys and a shard tag. Use to manage <a href="#"><u>tag aware sharding</u></a> .
<a href="#"><u>sh.disableBalancing()</u></a>	Disable balancing on a single collection in a sharded database. Does not affect balancing of other collections in a sharded cluster.
<a href="#"><u>sh.enableBalancing()</u></a>	Activates the sharded collection balancer process if previously disabled using <a href="#"><u>sh.disableBalancing()</u></a> .

Name	Description
<a href="#"><u>sh.enableSharding()</u></a>	Enables sharding on a specific database.
<a href="#"><u>sh.getBalancerHost()</u></a>	Returns the name of a <a href="#"><u>mongos</u></a> that's responsible for the balancer process.
<a href="#"><u>sh.getBalancerState()</u></a>	Returns a boolean to report if the <a href="#"><u>balancer</u></a> is currently enabled.
<a href="#"><u>sh.help()</u></a>	Returns help text for the sh methods.
<a href="#"><u>sh.isBalancerRunning()</u></a>	Returns a boolean to report if the balancer process is currently migrating chunks.
<a href="#"><u>sh.moveChunk()</u></a>	Migrates a <a href="#"><u>chunk</u></a> in a <a href="#"><u>sharded cluster</u></a> .
<a href="#"><u>sh.removeShardTag()</u></a>	Removes the association between a shard and a shard tag.

Name	Description
<a href="#"><u>sh.setBalancerState()</u></a>	Enables or disables the <a href="#"><u>balancer</u></a> which migrates <a href="#"><u>chunks</u></a> between <a href="#"><u>shards</u></a> .
<a href="#"><u>sh.shardCollection()</u></a>	Enables sharding for a collection.
<a href="#"><u>sh.splitAt()</u></a>	Divides an existing <a href="#"><u>chunk</u></a> into two chunks using a specific value of the <a href="#"><u>shard key</u></a> as the dividing point.
<a href="#"><u>sh.splitFind()</u></a>	Divides an existing <a href="#"><u>chunk</u></a> that contains a document matching a query into two approximately equal chunks.
<a href="#"><u>sh.startBalancer()</u></a>	Enables the <a href="#"><u>balancer</u></a> and waits for balancing to start.

Name	Description
<a href="#"><u>sh.status()</u></a>	Reports on the status of a <a href="#"><u>sharded cluster</u></a> , as <a href="#"><u>db.printShardingStatus()</u></a> .
<a href="#"><u>sh.stopBalancer()</u></a>	Disables the <a href="#"><u>balancer</u></a> and waits for any in progress balancing rounds to complete.
<a href="#"><u>sh.waitForBalancer()</u></a>	Internal. Waits for the balancer state to change.
<a href="#"><u>sh.waitForBalancerOff()</u></a>	Internal. Waits until the balancer stops running.
<a href="#"><u>sh.waitForDLock()</u></a>	Internal. Waits for a specified distributed <a href="#"><u>sharded cluster</u></a> lock.
<a href="#"><u>sh.waitForPingChange()</u></a>	Internal. Waits for a change in ping state from one of the <a href="#"><u>mongos</u></a> in the sharded cluster.

# Best Practices

- Only shard big collections
- Always have replication factor as an **odd number with an arbitor assigned**
- **Pick shard key carefully**
- **Be aware of monotonically increasing shard key values on inserts**
- **Adding shards is fairly easy but isn't instantaneous**
- Consider presplitting on a bulk load.
- **Always connect to mongos except for some dba work.**
- **Put mongos on default port 27017.**
- **Use logical config server names**
  - If you change config servers, read documentation to set it right
- **For large clusters keep configuration servers on separate instance and mongos on separate instance.**



MONGODB

## **BACKUP AND RESTORE WITH MONGODB TOOLS**



# MongoRestore

- The **mongorestore** program writes data from a binary database dump created by **mongodump** to a **mongoDB instance**
- **it also accepts data to restore via the standard input**
- **it can also write data to either mongod or mongos instances**

```
mongorestore --collection people --db accounts dump/accounts/people.bson
```

```
Restore with Access Control
mongorestore --host mongodb1.example.net --port 37017 --username user --password pass /opt/backup/mongodump-2011-10-24
```

```
PS C:\Windows\system32> cd E:\CT\MongoDB\code\data\import-export\usda.nutrition\dump\usda
PS E:\CT\MongoDB\code\data\import-export\usda.nutrition\dump\usda>mongorestore --collection nutrition --db usdanutritiondb nutrition.bson
2016-05-22T08:59:19.619+0530      checking for collection data in nutrition.json
2016-05-22T08:59:19.623+0530      reading metadata for usdanutritiondb.nutrition from nutrition.metadata.json
2016-05-22T08:59:19.954+0530      restoring usdanutritiondb.nutrition from nutrition.json
2016-05-22T08:59:22.399+0530      restoring indexes for collection usdanutritiondb.nutrition from metadata
2016-05-22T08:59:22.572+0530      finished restoring usdanutritiondb.nutrition (8194 documents)
2016-05-22T08:59:22.573+0530      done
PS E:\CT\MongoDB\code\data\import-export\usda.nutrition\dump\usda>
```



# MongoDB Restore

```
mongorestore --collection people --db accounts dump/accounts/people.bson

Restore with Access Control
mongorestore --host mongodb1.example.net --port 37017 --username user --password pass /opt/backup/mongodump-2011-10-24

Restore a collection from a standard input
zcat /opt/backup/mongodump-2014-12-03/accounts.people.bson.gz | mongorestore --collection people --db accounts -

restore a database from an archive file
mongorestore --archive=test.20150715.archive --db test

Restore a Database from Standard Input
mongodump --archive --db test --port 27017 | mongorestore --archive --port 27018

Restore from Compressed Data
mongorestore --gzip --db test
mongorestore --gzip --archive=test.20150715.gz --db test
```



MONGODB

# YELP DATASET

## business

```
{  
    'type': 'business',  
    'business_id': (encrypted business id),  
    'name': (business name),  
    'neighborhoods': [(hood names)],  
    'full_address': (localized address),  
    'city': (city),  
    'state': (state),  
    'latitude': latitude,  
    'longitude': longitude,  
    'stars': (star rating, rounded to half-stars),  
    'review_count': review count,  
    'categories': [(localized category names)]  
    'open': True / False (corresponds to closed, not business hours),  
    'hours': {  
        (day_of_week): {  
            'open': (HH:MM),  
            'close': (HH:MM)  
        },  
        ...  
    },  
    'attributes': {  
        (attribute_name): (attribute_value),  
        ...  
    },  
}
```

## review

```
{  
  'type': 'review',  
  'business_id': (encrypted business id),  
  'user_id': (encrypted user id),  
  'stars': (star rating, rounded to half-stars),  
  'text': (review text),  
  'date': (date, formatted like '2012-03-14'),  
  'votes': {(vote type): (count)},  
}
```

## user

```
{  
  'type': 'user',  
  'user_id': (encrypted user id),  
  'name': (first name),  
  'review_count': (review count),  
  'average_stars': (floating point average, like 4.31),  
  'votes': {(vote type): (count)},  
  'friends': [(friend user_ids)],  
  'elite': [(years_elite)],  
  'yelping_since': (date, formatted like '2012-03'),  
  'compliments': {  
    (compliment_type): (num_compliments_of_this_type),  
    ...  
  },  
  'fans': (num_fans),  
}
```



### check-in

```
{  
    'type': 'checkin',  
    'business_id': (encrypted business id),  
    'checkin_info': {  
        '0-0': (number of checkins from 00:00 to 01:00 on all Sundays),  
        '1-0': (number of checkins from 01:00 to 02:00 on all Sundays),  
        ...  
        '14-4': (number of checkins from 14:00 to 15:00 on all Thursdays),  
        ...  
        '23-6': (number of checkins from 23:00 to 00:00 on all Saturdays)  
    }, # if there was no checkin for a hour-day block it will not be in the dict  
}
```

### tip

```
{  
    'type': 'tip',  
    'text': (tip text),  
    'business_id': (encrypted business id),  
    'user_id': (encrypted user id),  
    'date': (date, formatted like '2012-03-14'),  
    'likes': (count),  
}
```



## photos (from the photos auxiliary file)

This file is formatted as a JSON list of objects.

```
[  
  {  
    "photo_id": (encrypted photo id),  
    "business_id" : (encrypted business id),  
    "caption" : (the photo caption, if any),  
    "label" : (the category the photo belongs to, if any)  
  },  
  {...}  
]
```



```
...     # Import businesses, users, reviews and checkins
mongoimport --db yelp --collection businesses yelp_phoenix_academic_dataset/yelp_academic_dataset_business.json
mongoimport --db yelp --collection users yelp_phoenix_academic_dataset/yelp_academic_dataset_user.json
mongoimport --db yelp --collection reviews yelp_phoenix_academic_dataset/yelp_academic_dataset_review.json
mongoimport --db yelp --collection checkins yelp_phoenix_academic_dataset/yelp_academic_dataset_checkin.json
```



# SCALING MONGODB

# When to consider Scaling?

- When a specific resource becomes a bottle neck on a machine or a replica set
  - RAM
  - DISK IO
  - STORAGE
  - CONCURRENCY
- Sharding
  - User defines shard key
  - Shard key defines range of data
  - Data is partitioned into shards according to shard key
  - Increase capacity as you go
  - Commodity and cloud architectures
  - Improved operational simplicity and cost visibility



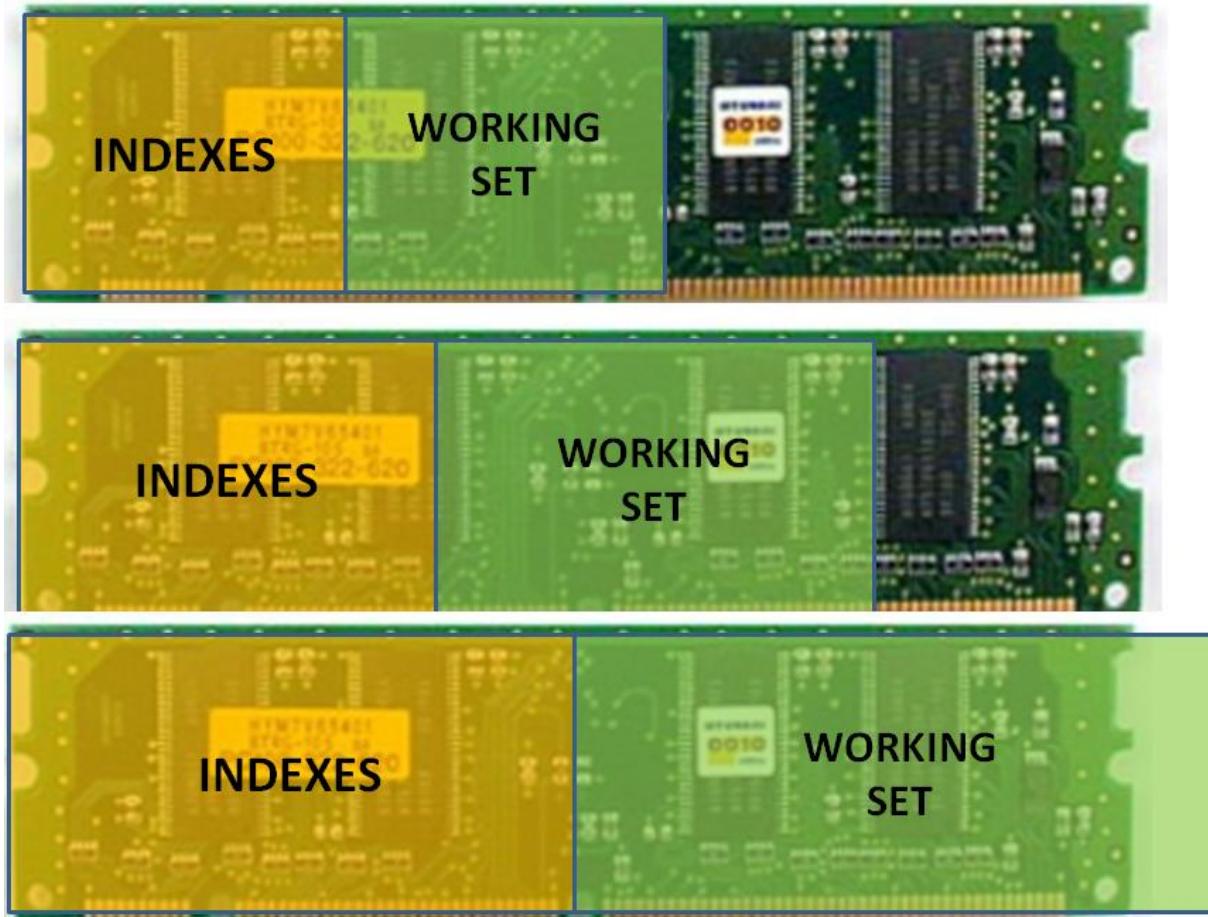
# 1.Optimize your Queries

- Optimize your queries by evaluating the queries using **explain**, which in turns gives you **queryplan/explain plan**
- Ensure that you have indexed your collections.
- Optimize compound Indexes

```
query= db.customerorder.find();
query.explain();
db.customerorder.ensureIndex({customerId:1});
```

<https://emptysqua.re/blog/optimizing-mongodb-compound-indexes/>

# Working Set Exceeds Physical Memory



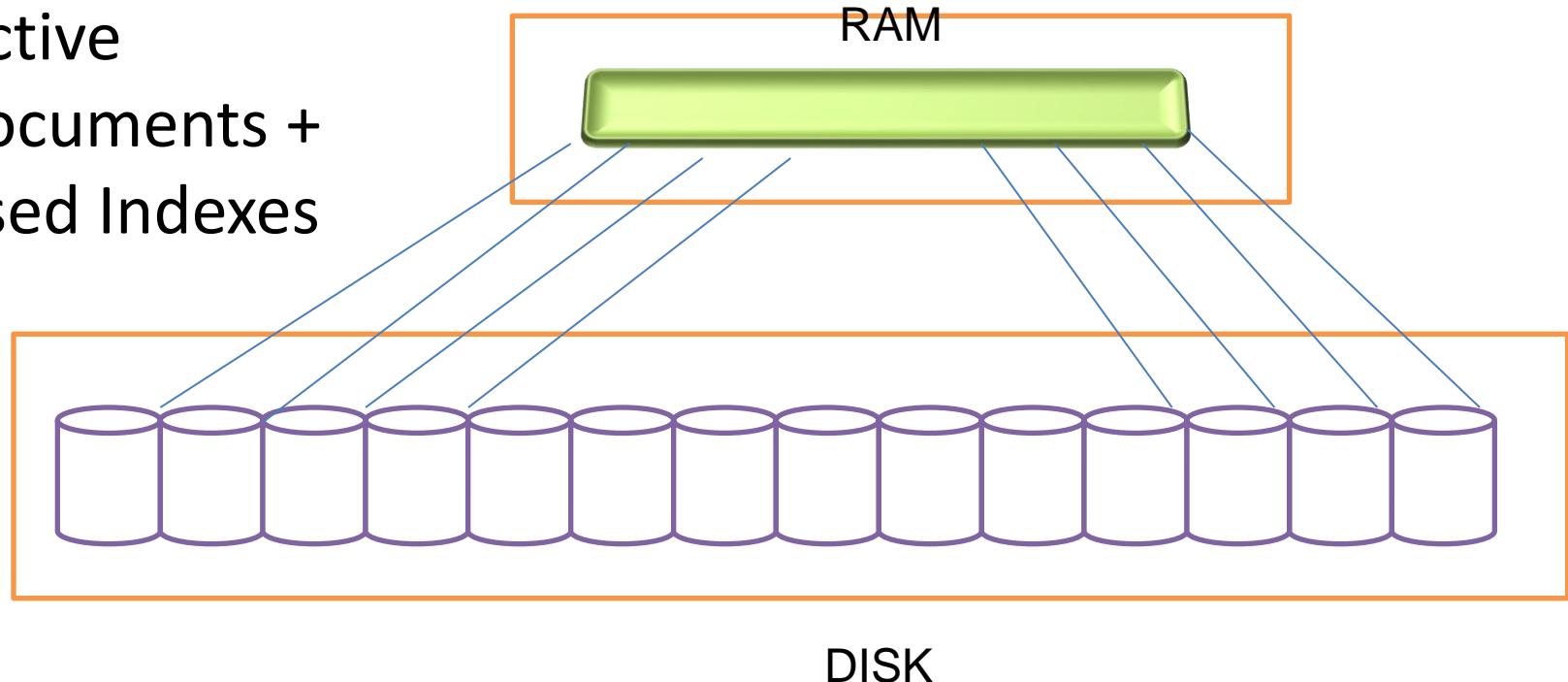
# 2. Know your working set size?

Total Database Size ~ 10TB

Point in Time Average

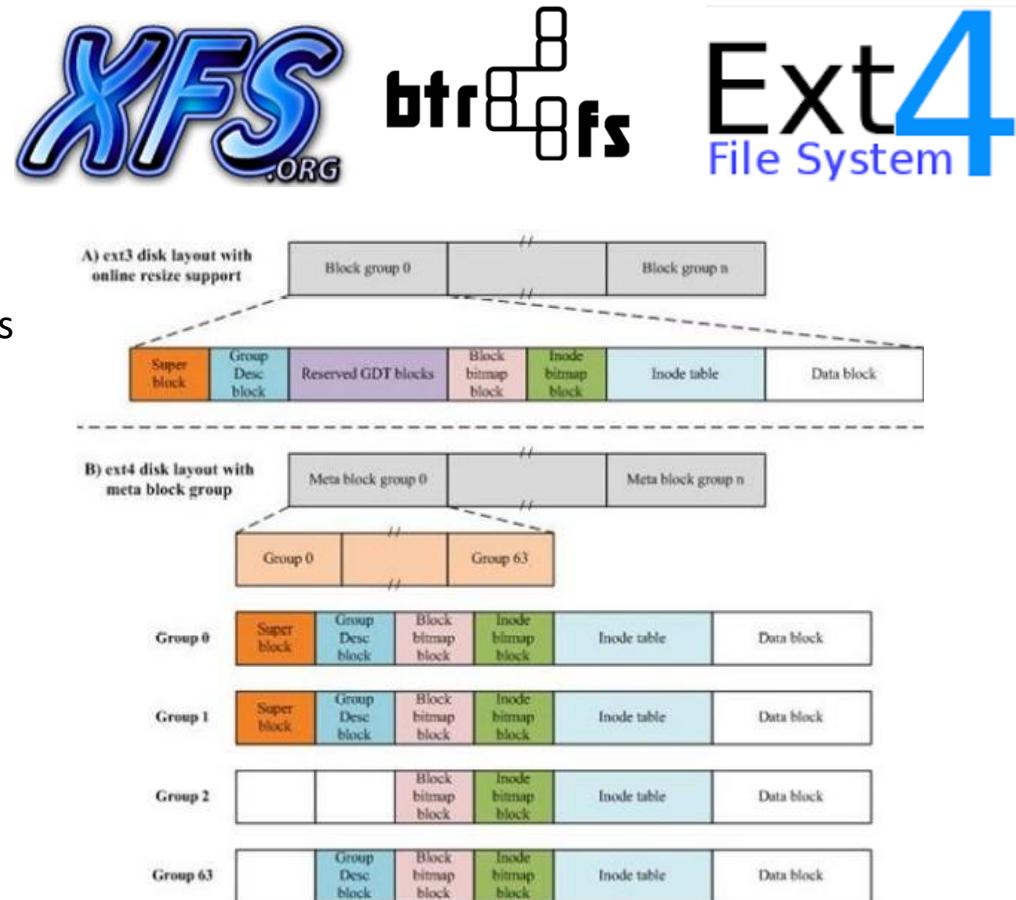
**Active Documents + Used Indexes (This is Actual) = 500GB**

- Active documents + used Indexes



# 3. Tune Your File System

- Ext4
  - Works better on cheap hardware
  - Has a well-developed fsck
  - Ext4 has an in-place upgrade from ext3
- XFS
  - Supports extremely large file systems (up to 8 exbibytes)
  - Great for storing large data files (think media content)
  - Fully mature and stable file system
  - Ideally suited for unRAID array devices
- BTRFS
  - Supports copy on write, snap-shot and compression
  - Allows mixing/matching different drives for use in an unRAID cache pool
  - Ideally suited for use in cache pool



# XFS vs. EXT4 vs. Btrfs

Apache Benchmark v2.2.17

Static Web Page Serving



PostMark v1.51

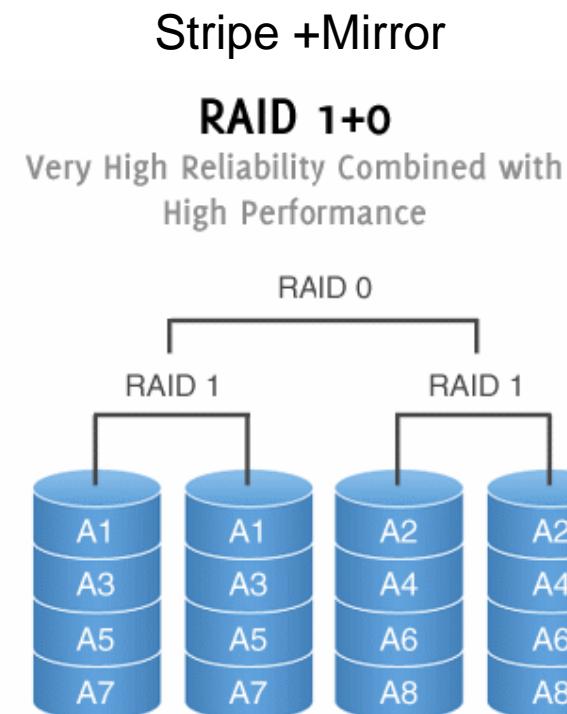
Disk Transaction Performance



[http://www.phoronix.com/scan.php?page=article&item=linux\\_2639\\_fs&num=1](http://www.phoronix.com/scan.php?page=article&item=linux_2639_fs&num=1)

# 4. Choose the right Disks

- MongoDB uses your disk for random access, so it is the disk's seek time that is the bottle neck, rather than the disk's throughput.
- Most disks are capable of 100 or so seeks per second.
- Seek performance using
  - RAID 10
  - EC2
  - SSD



# 5.Shard

- A single replica set can usually meet performance requirement, if we have optimized indexes and filesystem configured correctly.
- If the working set fits in RAM and right disks are used.
- A sharded cluster is more complex.
- Design your application for sharded cluster early, than to adapt an existing application.



MONGODB

# **PERFORMANCE TUNING, INDEXES, EXPLAIN PLAN, HINTS AND PROFILER**



Indexes are the  
single biggest  
**tunable**  
**performance factor**  
in MongoDB

- review index efficiency **early**
- **avoid duplicates**

# Covered or Index only Queries

- Returns data from the index
  - Rather than the database file
  - Performance optimization
  - Works with compound indexes
    - Invoke with a projection
  - Use **projections to reduce data sent back to the client**

# Background Index Builds

- Index creation is a blocking operation that can take a longtime
- Background creation yields to other operations
- Build more than one index in background concurrently
- Restart secondaries in standalone to build index

```
db.articles.ensureIndex(   
  |   {"author":1,"date": "-1"},  
  |   {background: true} |  
 )
```

# Explain plan

- Use to evaluate operations and indexes
  - Which indexes have been used .. If any
  - How many documents/objects have been scanned
  - View via the console or via code

```
mongos> db.customerorder.find().explain()  
{  
    "cursor" : "BasicCursor",  
    "isMultiKey" : false,  
    "n" : 0,  
    "nscannedObjects" : 0,  
    "nscanned" : 0,  
    "nscannedObjectsAllPlans" : 0,  
    "nscannedAllPlans" : 0,  
    "scanAndOrder" : false,  
    "indexOnly" : false,  
    "nYields" : 0,  
    "nChunkSkips" : 0,  
    "millis" : 0,  
    "indexBounds" : {  
    },  
    "server" : "ElevenGEN:27000",  
    "millis" : 0  
}
```

# Database Profiler

- Enable to see slow queries
  - (or all queries)
  - Default 100ms
- Works only on mongod, not on mongos
- **The profile collection is a capped collection and fixed in size**

```
db.setProfilingLevel(n, slowms=100ms);
where n=0 profiler off
      n=1 record operations longer than
      slowms
      n=2 record all queries
      ...
      db.system.profile.find()
      db.getProfilingStatus();

//Enable database profiler on the console, 0=off 1=slow 2=all
> db.setProfilingLevel(1, 100)
{ "was" : 0, "slowms" : 100, "ok" : 1 }

//View profile with
> show profile

//or
>db.system.profile.find().pretty()
```

## Hint

```
// tell the database what index to use
db.recipes.find({
  calories: {$lt:100}
}).hint({_id:1});
// tell the database to NOT use an index
db.recipes.find({
  calories: {$lt:100}
}).hint({$natural:1});
```

## Reads vs Writes Indexes

- Generally, more indexes means **faster reads**
- Generally, more indexes means **slower writes**
- Faster to build an index post import than pre-import

# currentOp()

## CurrentOp()

- Returns a document that contains information on in-progress operations for the database instance.
- db.currentOp(true)

```
db.currentOp({  
  "waitingForLock" : true,  
  $or: [  
    { "op" : { "$in" : [ "insert",  
      "update", "remove" ] } },  
    { "query.findandmodify": { $exists:  
      true } }  
  ]  
})
```

## Active Operations with no Yields

```
db.currentOp({  
  "active" : true,  
  "numYields" : 0,  
  "waitingForLock" : false  
})
```

## Active Operations on a specific database

```
db.currentOp({  
  "active" : true,  
  "secs_running" : { "$gt" : 3 },  
  "ns" : /^db1\./  
})
```

# currentOp()

## Active indexing Operations

- Returns information on index creation operations

```
db.currentOp({  
    $or: [  
        { op: "query", "query.createIndexes": {  
            $exists: true } },  
        { op: "insert", ns:  
            /\system\.indexes\b/ }  
    ]  
})
```

## killOp()

- Terminates an operation as specified by the operation ID.
- To find the operations and their corresponding IDs, we use **db.currentOp()**
- Terminate running operations with extreme caution.

```
Db.killOp(<opid>);
```



# Mongostat and mongotop

```
syedawase@ElevenGEN:~$ mongotop --port 27000
connected to: 127.0.0.1:27000
```

	ns	total	read	write	2
016-05-29T07:31:28					
	ElevenDB.customerorder	0ms	0ms	0ms	
	ElevenDB.system.indexes	0ms	0ms	0ms	
	ElevenDB.system.namespaces	0ms	0ms	0ms	
	ElevenDB.system.users	0ms	0ms	0ms	
	local.oplog.rs	0ms	0ms	0ms	
	local.replset.minvalid	0ms	0ms	0ms	
	local.slaves	0ms	0ms	0ms	

```
syedawase@ElevenGEN:~$ mongostat --port 27017
connected to: 127.0.0.1:27017
```

insert	query	update	delete	getmore	command	vsize	res	faults	netIn	netOut	conn	repl	time
0	0	0	0	0	1	221m	11m	0	62b	713b	1	RTR	07:29:06
0	0	0	0	0	1	221m	11m	0	62b	713b	1	RTR	07:29:07
0	0	0	0	0	1	221m	11m	0	62b	713b	1	RTR	07:29:08



AGGREGATION DEMO WITH NBA DATASET

# NBA GAMES DATA-SET



# NBA GAMES DATASET RESTORE

```
PS E:\CT\MongoDB\code\data\import-export\nba2\dump\nba> mongorestore --collection games --db NBAGAMES games.bson
2016-05-22T12:12:39.428+0530      checking for collection data in games.bson
2016-05-22T12:12:40.629+0530      reading metadata for NBAGAMES.games from games.metadata.json
2016-05-22T12:12:41.797+0530      restoring NBAGAMES.games from games.bson
2016-05-22T12:12:43.978+0530      [#####.....] NBAGAMES.games 64.0 MB/167.5 MB (38.2%)
2016-05-22T12:12:46.627+0530      [#####.....] NBAGAMES.games 81.0 MB/167.5 MB (48.4%)
2016-05-22T12:12:49.628+0530      [#####.....] NBAGAMES.games 167.5 MB/167.5 MB (100.0%)
2016-05-22T12:12:50.529+0530      [#####.....] NBAGAMES.games 167.5 MB/167.5 MB (100.0%)
2016-05-22T12:12:50.530+0530      restoring indexes for collection NBAGAMES.games from metadata
2016-05-22T12:12:50.763+0530      finished restoring NBAGAMES.games (31686 documents)
2016-05-22T12:12:50.763+0530      done
PS E:\CT\MongoDB\code\data\import-export\nba2\dump\nba>
```

# NBA DATA DESCRIPTION

```
"players" : [
  {
    "ast" : NumberInt(9),
    "blk" : NumberInt(2),
    "drb" : NumberInt(8),
    "fg" : NumberInt(8),
    "fg3" : NumberInt(0),
    "fg3_pct" : "",
    "fg3a" : NumberInt(0),
    "fg_pct" : ".533",
    "fga" : NumberInt(15),
    "ft" : NumberInt(3),
    "ft_pct" : ".750",
    "fta" : NumberInt(4),
    "mp" : "41:00",
    "orb" : NumberInt(6),
    "pf" : NumberInt(3),
    "player" : "Jeff Ruland",
    "pts" : NumberInt(19),
    "stl" : NumberInt(1),
    "tov" : NumberInt(5),
    "trb" : NumberInt(14)
  },
]
```

Ast	assists
Blk	blocks
Drb	Defensive rebounds
Fg	Field goals
fg3pct	3 point Field goal percentage
fg3a	3 point Field goal assists
Fg_pct	Field goal percentage
fga	Field goal attempts
ft	Free throws
Ft_pct	Free throw %
fta	Free throw attempt
Mp	
Orb	Offensive rebounds
Pf	Personal fouls
player	Player name

# NBA DATA DESCRIPTION

```
"players" : [
  {
    "ast" : NumberInt(9),
    "blk" : NumberInt(2),
    "drb" : NumberInt(8),
    "fg" : NumberInt(8),
    "fg3" : NumberInt(0),
    "fg3_pct" : "",
    "fg3a" : NumberInt(0),
    "fg_pct" : ".533",
    "fga" : NumberInt(15),
    "ft" : NumberInt(3),
    "ft_pct" : ".750",
    "fta" : NumberInt(4),
    "mp" : "41:00",
    "orb" : NumberInt(6),
    "pf" : NumberInt(3),
    "player" : "Jeff Ruland",
    "pts" : NumberInt(19),
    "stl" : NumberInt(1),
    "tov" : NumberInt(5),
    "trb" : NumberInt(14)
  },
]
```

Pts	points
Stl	steals
Tov	Turn overs
Trb	Total rebounds

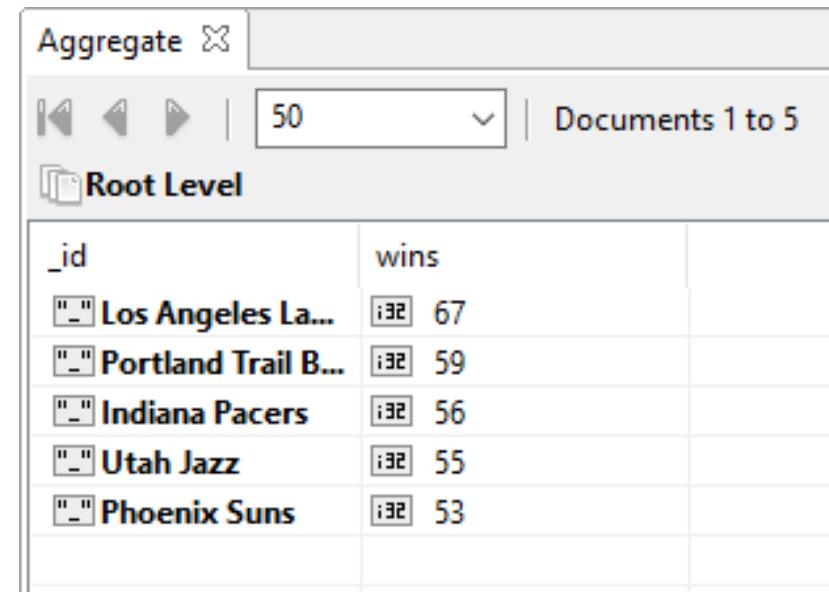
# Sanity Check

- To compute which 5 teams had the most wins in the 1999-2000 season
- Steps
  1. Use the **\$match** stage to limit ourselves to the games that took place between Aug 1, 1999 and Aug 1, 2000, two dates that are sufficiently far removed from any NBA games to safely bound the season.
  2. Use the **\$unwind** stage to generate one document for each team in the game
  3. Use **\$match** again to limit ourselves to teams that won
  4. Use the **\$group** stage to count how many times a given team appears in the output of step 3
  5. Use the **\$sort** stage to sort by number of wins, in descending order
  6. Use the **\$limit** stage to limit ourselves to the 5 winning teams

# NBA GAMES AGGREGATE FUNCTION (1999-2000)

```
db.games.aggregate([
  {
    $match : {
      date : {
        $gt : ISODate("1999-08-01T00:00:00Z"),
        $lt : ISODate("2000-08-01T00:00:00Z")
      }
    }
  },
  {
    $unwind : '$teams'
  },
  {
    $match : {
      'teams.won' : 1
    }
  },
  {
    $group : {
      _id : '$teams.name',
      wins : { $sum : 1 }
    }
  },
  {
    $sort : { wins : -1 }
  },
  {
    $limit : 5
  }
]);
```

Which teams had the most wins in the 1999-2000 season?



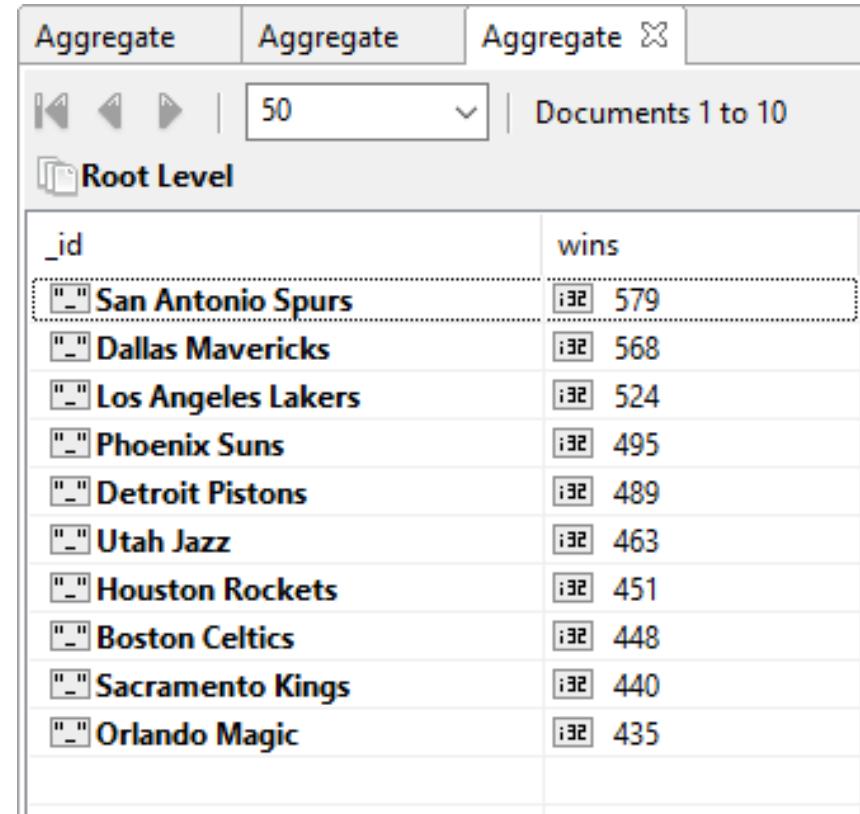
The screenshot shows the MongoDB Aggregation interface with the following details:

- Aggregate window: 50 documents from 1 to 5.
- Root Level results:

_id	wins
"_ Los Angeles La...	67
"_ Portland Trail B...	59
"_ Indiana Pacers	56
"_ Utah Jazz	55
"_ Phoenix Suns	53

# Which team won the most games between 2000-2001 season and the 2009-2010 season?

```
db.games.aggregate([
  {
    $match : {
      date : {
        $gt : ISODate("2000-08-01T00:00:00Z"),
        $lt : ISODate("2010-08-01T00:00:00Z")
      }
    }
  },
  {
    $unwind : '$teams'
  },
  {
    $match : {
      'teams.won' : 1
    }
  },
  {
    $group : {
      _id : '$teams.name',
      wins : { $sum : 1 }
    }
  },
  {
    $sort : { wins : -1 }
  },
  {
    $limit : 5
  }
]);
```



The screenshot shows the MongoDB Aggregation Pipeline interface. At the top, there are three tabs: "Aggregate", "Aggregate", and "Aggregate X". Below the tabs, there are navigation buttons (back, forward, first, last, etc.) and a document count field set to 50. To the right of the count is a dropdown menu and the text "Documents 1 to 10". Underneath these controls is a section titled "Root Level". The main area displays a table with two columns: "\_id" and "wins". The data rows are as follows:

_id	wins
"_\" San Antonio Spurs	i32 579
"_\" Dallas Mavericks	i32 568
"_\" Los Angeles Lakers	i32 524
"_\" Phoenix Suns	i32 495
"_\" Detroit Pistons	i32 489
"_\" Utah Jazz	i32 463
"_\" Houston Rockets	i32 451
"_\" Boston Celtics	i32 448
"_\" Sacramento Kings	i32 440
"_\" Orlando Magic	i32 435

# Correlating Stats with Wins

- Compute how often a team wins when they record more defensive rebounds than their opponent across the entire dataset.
- Steps
  1. Use **\$unwind** to get a document containing the box score for each team in the game.
  2. Use **\$project** with **\$cond** to transform each document so the team's defensive rebounding total is negative if the team lost, as defined by the won flag.
  3. Use **\$group** and **\$sum** to add up the rebounding totals for each game.
  4. Use **\$project** and **\$gte** to create a document which has a **winningTeamHigher** flag that is true if the winning team had more defensive rebounds than the losing team.
  5. Use **\$group** and **\$sum** to compute for how many games **winningTeamHigher** was TRUE.

```

db.games.aggregate([
  {
    $unwind : '$box'
  },
  {
    $project : {
      _id : '_id',
      stat : {
        $cond : [
          { $gt : ['$box.won', 0] },
          '$box.team.drb',
          { $multiply : ['$box.team.drb', -1] }
        ]
      }
    }
  },
  {
    $group : {
      _id : '_id',
      stat : { $sum : '$stat' }
    }
  },
  {
    $project : {
      _id : '_id',
      WinningTeamHigher : { $gte : ['$stat', 0] }
    }
  },
  {
    $group : {
      _id : '$WinningTeamHigher',
      count : { $sum : 1 }
    }
  }
]);

```

Root Level

_id	count
false	7693
true	23993

# Defensive rebounds and total rebounds vs Win Percentage

```
db.games.aggregate([
  {
    $unwind : '$box'
  },
  {
    $group : {
      _id : '$box.team.drb',
      winPercentage : { $avg : '$box.won' }
    }
  },
  {
    $sort : { _id : 1 }
  }
]);
```

Root Level	
_id	winPercentage
i32 10	1.23 0.0
i32 11	1.23 0.0
i32 12	1.23 0.0
i32 13	1.23 0.0
i32 14	1.23 0.12121212121...
i32 15	1.23 0.09433962264...
i32 16	1.23 0.06557377049...
i32 17	1.23 0.10526315789...
i32 18	1.23 0.12463768115...
i32 19	1.23 0.15141955835...
i32 20	1.23 0.18826405867...
i32 21	1.23 0.19805982215...
i32 22	1.23 0.24737816162...
i32 23	1.23 0.25130662020...
i32 24	1.23 0.29125874125...

```
db.games.aggregate([
  {
    $unwind : '$box'
  },
  {
    $group : {
      _id : '$box.team.trb',
      winPercentage : { $avg : '$box.won' }
    }
  },
  {
    $sort : { _id : 1 }
  }
]);
```

Root Level	
_id	winPercentage
i32 18	0.0
i32 19	0.0
i32 20	0.0
i32 21	0.14285714285...
i32 22	0.1875
i32 23	0.17142857142...
i32 24	0.18032786885...
i32 25	0.13684210526...
i32 26	0.16058394160...
i32 27	0.17085427135...
i32 28	0.23809523809...
i32 29	0.23045267489...
i32 30	0.23343373493...
i32 31	0.26535087719...
i32 32	0.30118577075...

# Datasets

- <http://www.ehdp.com/vitalnet/datasets.htm>
- <http://www.nber.org/patents/>
- <http://aws.amazon.com/publicdatasets>
- <http://lemurproject.org/clueweb09/>
- <http://www.census.gov/genealogy/names/dist.all.last>
- <http://archive.ics.uci.edu/ml/>
- <http://crawdad.org/>
- <http://data.austintexas.gov>
- <http://data.cityofchicago.org>
- <http://data.govloop.com>
- <http://data.gov.uk/>
- <https://kaggle.com/datasets>
- <http://usgovxml.com/>
- <http://aws.amazon.com/datasets>
- <http://databib.org/>
- <http://databib.org>
- <http://datacite.org>
- <http://figshare.com>
- <http://linkeddata.org>
- <http://reddit.com/r/datasets>
- <http://thewebminer.com/>
- <http://thedatahub.org alias http://ckan.net>
- <http://quandl.com>
- [Social Network Analysis Interactive Dataset Library \(Social Network Datasets\)](Social Network Analysis Interactive Dataset Library (Social Network Datasets))
- <Datasets for Data Mining>
- <http://enigma.io>
- <http://www.ufindthem.com/>
- <http://NetworkRepository.com - The First Interactive Network Data Repository>
- <http://MLvis.com>







## \$match

- Filter documents
  - Uses existing query syntax
  - Can facilitate shard exclusion
  - No \$where (server side javascript)

## \$geoNear

- Order/filter Documents by location
  - Requires a geospatial index
  - Output includes physical distance
  - Must be first aggregation stage