Introduction to NoSQL and MongoDB

Outline for today

- Introduction to NoSQL
  - Architecture
    - Sharding
    - Replica sets
  - NoSQL Assumptions and the CAP Theorem
  - Strengths and weaknesses of NoSQL
- MongoDB
  - Functionality
  - Examples

Taxonomy of NoSQL

- **Key-value**

- **Graph database**

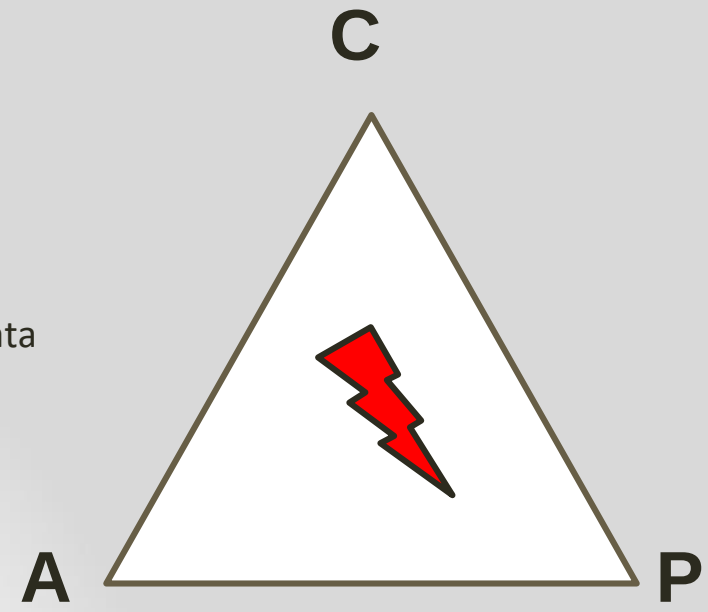
- **Document-oriented**

- **Column family**

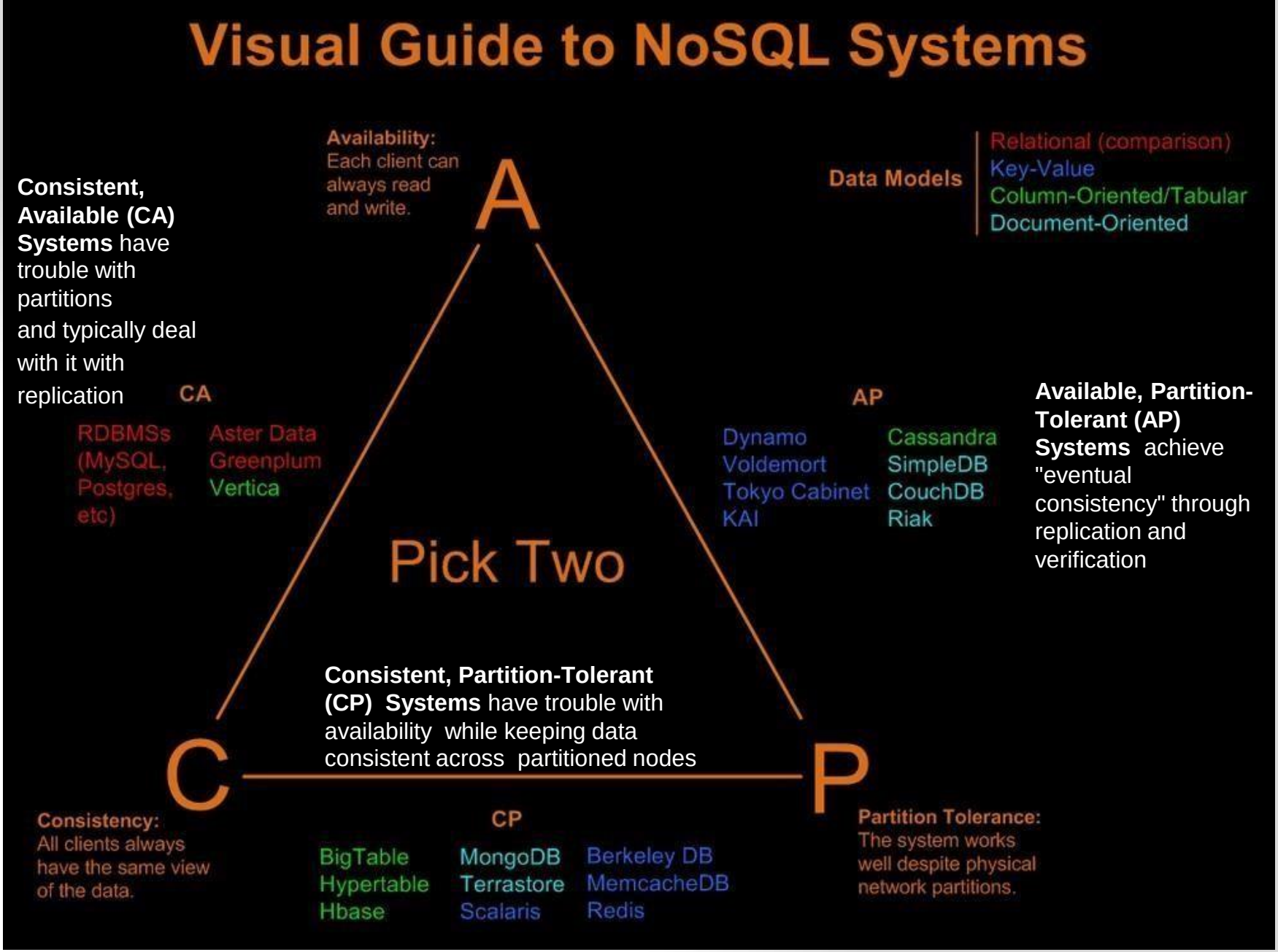
Theory of NOSQL: CAP

GIVEN:

- Many nodes
- Nodes contain ***replicas of partitions*** of the data
- **C**onsistency
  - All replicas contain the same version of data
  - Client always has the same view of the data (no matter what node)
- **A**vailability
  - System remains operational on failing nodes
  - All clients can always read and write
- **P**artition tolerance
  - multiple entry points
  - System remains operational on system split (communication malfunction)
  - System works well across physical network partitions



CAP Theorem: satisfying all three at the same time is impossible



http://blog.nahurst.com/visual-guide-to-nosql-systems

Sharding of data

- Distributes a single logical database system across a cluster of
  machines
- Uses range-based partitioning to distribute documents based on a specific shard key
- Automatically balances the data associated with each shard
- Can be turned on and off per collection (table)

How does NoSQL vary from RDBMS?

- Looser schema definition
- Applications written to deal with specific documents/ data
  - Applications aware of the schema definition as opposed to the data
- Designed to handle distributed, large databases
- Trade offs:
  - No strong support for ad hoc queries but designed for speed and growth of database
    - Query language through the API
  - Relaxation of the ACID properties

Benefits of NoSQL

Elastic Scaling

- RDBMS scale up – bigger load , bigger server
- NO-SQL scale out – distribute data across multiple hosts seamlessly

DBA Specialists

- RDMS require highly trained expert to monitor DB
- NoSQL require less management, automatic repair and simpler data models

Big Data

- Huge increase in data
- RDMS: capacity and constraints of data volumes at its limits
- NoSQL designed for big data

Benefits of NoSQL

Flexible data models

- Change management to schema for RDMS have to be carefully managed
- NoSQL databases more relaxed in structure of data
  - Database schema changes do not have to be managed as one complicated change unit
  - Application already written to address an amorphous schema

Economics

- RDMS rely on expensive proprietary servers to manage data
- No SQL: clusters of cheap commodity servers to manage the data and transaction volumes
- Cost per gigabyte or transaction/second for NoSQL can be lower than the cost for a RDBMS

Drawbacks of NoSQL

- Support
  - RDBMS vendors provide a high level of support to clients
    - Stellar reputation
  - NoSQL – are open source projects with startups supporting them
    - Reputation not yet established
- Maturity
  - RDMS mature product: means stable and dependable
    - Also means old no longer cutting edge nor interesting
  - NoSQL are still implementing their basic feature set

Drawbacks of NoSQL

- **Administration**
  - RDMS administrator well defined role
  - No SQL's goal: no administrator necessary however NO SQL still requires effort to maintain
- **Lack of Expertise**
  - Whole workforce of trained and seasoned RDMS developers
  - Still recruiting developers to the NoSQL camp
- **Analytics and Business Intelligence**
  - **RDMS designed to address this niche**
  - NoSQL designed to meet the needs of a Web 2.0 application - not designed for ad hoc query of the data
    - Tools are being developed to address this need

RDB ACID to NoSQL BASE

Atomicity

Consistency

Isolation

Durability

Basically

Available (CP)

Soft-state
(State of system may change over time)

Eventually consistent
(Asynchronous propagation)

Pritchett, D.: BASE:An Acid Alternative (queue.acm.org/detail.cfm?id=1394128)


mongoDB

What is MongoDB?

- Developed by 10gen
  - Founded in 2007
- A document-oriented, NoSQL database
  - Hash-based, s*chema-less database*
    - No Data Definition Language
    - In practice, this means you can store hashes with any keys and values that you choose
      - Keys are a basic data type but in reality stored as strings
      - Document Identifiers (_id) will be created for each document, field name reserved by system
    - Application tracks the schema and mapping
    - Uses BSON format
      - Based on JSON – B stands for Binary
- Written in C++
- Supports APIs (drivers) in many computer languages
  - JavaScript, Python, Ruby, Perl, Java, Java Scala, C#, C++, Haskell, Erlang

Functionality of MongoDB

- Dynamic schema
  - No DDL
- Document-based database
- Secondary indexes
- Query language via an API
- Atomic writes and fully-consistent reads
  - If system configured that way
- Master-slave replication with automated failover (replica sets)
- Built-in horizontal scaling via automated range-based partitioning of data (sharding)
- No joins nor transactions

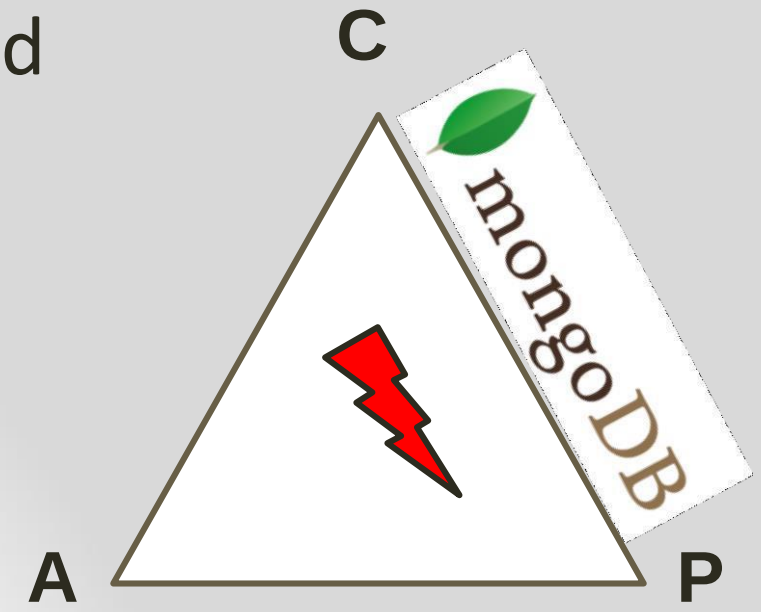
Why use MongoDB?

- Simple queries
- Functionality provided applicable to most web applications
- Easy and fast integration of data
  - No ERD diagram
- Not well suited for heavy and complex transactions systems

MongoDB: CAP approach

Focus on Consistency and Partition tolerance

- **C**onsistency
  - all replicas contain the same version of the data
- **A**vailability
  - system remains operational on failing nodes
- **P**artition tolerance
  - multiple entry points
  - system remains operational on system split



CAP Theorem:
satisfying all three at the same time is impossible

MongoDB: Hierarchical Objects

- A MongoDB instance may have zero or more ‘databases’
- A database may have zero or more ‘collections’.
- A collection may have zero or more ‘documents’.
- A document may have one or more ‘fields’.
- MongoDB ‘Indexes’ function much like their RDBMS counterparts.

RDB Concepts to NO SQL

RDBMS		MongoDB
Database	⇨	Database
Table, View	⇨	Collection
Row	⇨	Document (BSON)
Column	⇨	Field
Index	⇨	Index
Join	⇨	Embedded Document
Foreign Key	⇨	Reference
Partition	⇨	Shard

Collection is not strict about what it Stores

Schema-less

Hierarchy is evident in the design

Embedded Document ?

MongoDBProcesses and configuration

- Mongod – Database instance
- Mongos - Sharding processes
  - Analogous to a database router.
  - Processes all requests
  - Decides how many and which *mongod*s should receive the query
  - *Mongos* collates the results, and sends it back to the client.
- Mongo – an interactive shell (a client)
  - Fully functional JavaScript environment for use with a MongoDB
- You can have one *mongos* for the whole system no matter how many mongods you have.
- OR you can have one local *mongos* for every client if you wanted to minimize network latency.

Choices made for Design of MongoDB

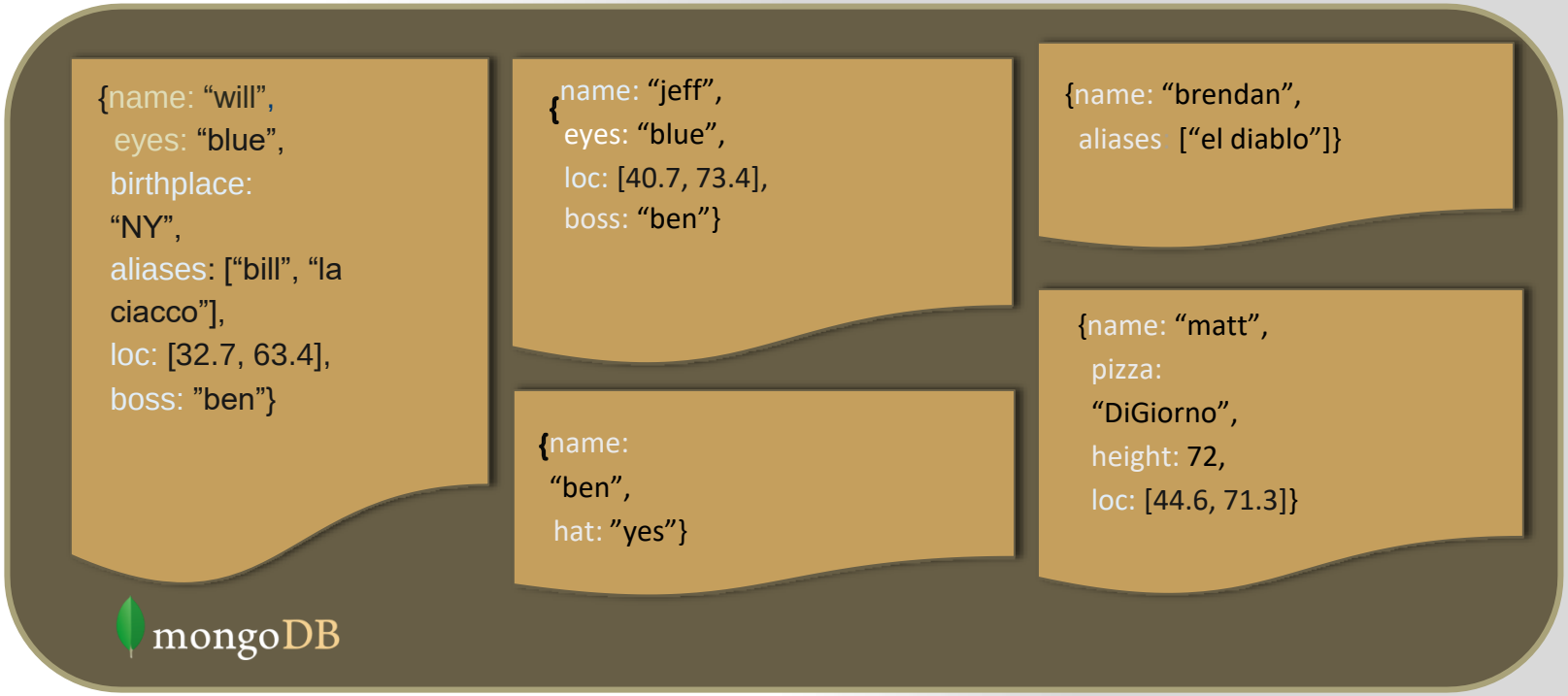
- Scale horizontally over commodity hardware
  - Lots of relatively inexpensive servers
- Keep the functionality that works well in RDBMSs
  - Ad hoc queries
  - Fully featured indexes
  - Secondary indexes

BSON format

- Binary-encoded serialization of JSON-like documents
- Zero or more key/value pairs are stored as a single entity
- Each entry consists of a field name, a data type, and a value
- Large elements in a BSON document are prefixed with a length field to facilitate scanning

Schema Free

- MongoDB does not need any pre-defined data schema
- Every document in a collection could have different data
  - Addresses NULL data fields



JSON format

- Data is in name / value pairs
- A name/value pair consists of a field name followed by a colon, followed by a value:
  - Example: “name”: “R2-D2”
- Data is separated by commas
  - Example: “name”: “R2-D2”, race : “Droid”
- Curly braces hold objects
  - Example: {“name”: “R2-D2”, race : “Droid”, affiliation: “rebels”}
- An array is stored in brackets []
  - Example

[{“name”: “R2-D2”, race : “Droid”, affiliation: “rebels”},{“name”: “Yoda”, affiliation: “rebels”}]

MongoDB Features

- Document-Oriented storage
- Full Index Support
- Replication & High Availability
- Auto-Sharding
- Querying
- Fast In-Place Updates
- Map/Reduce functionality

Agile

Scalable

CRUD operations

- Create
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
  - db.collection.update(<query>, <update>, { upsert: true })
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- Update
  - db.collection.update(<query>, <update>, <options>)
- Delete
  - db.collection.remove(<query>, <justOne>)

Collection specifies the collection or the ‘table’ to store the document

Create Operations

Db.collection specifies the collection or the ‘table’ to store the document

- db.collection_name.insert(<document>)
  - Omit the _id field to have MongoDB generate a unique key
  - Example db.**parts**.insert({{type: “screwdriver”, quantity: 15 })
  - db.**parts**.insert({_id: 10, type: “hammer ”, quantity: 1 })
- db.collection_name.update(<query>, <update>, { upsert: true })
  - Will update 1 or more records in a collection satisfying query
- db.collection_name.save(<document>)
  - Updates an existing record or creates a new record

Read Operations

- db.collection.find(<query>, <projection>).cursor modified
  - Provides functionality similar to the SELECT command
    - <query> where condition , <projection> fields in result set
  - Example: varPartsCursor = db.parts.find({parts: “hammer”}).limit(5)
  - Has cursors to handle a result set
  - Can modify the query to impose limits, skips, and sort orders.
  - Can specify to return the ‘top’ number of records from the result set
- db.collection.findOne(<query>, <projection>)

Query Operators

Name	Description
$eq	Matches value that are equal to a specified value
$gt, $gte	Matches values that are greater than (or equal to a specified value
$lt, $lte	Matches values less than or (equal to) a specified value
$ne	Matches values that are not equal to a specified value
$in	Matches any of the values specified in an array
$nin	Matches none of the values specified in an array
$or	Joins query clauses with a logical OR returns all
$and	Join query clauses with a logical AND
$not	Inverts the effect of a query expression
$nor	Join query clauses with a logical NOR
$exists	Matches documents that have a specified field

https://docs.mongodb.org/manual/reference/operator/query/

Update Operations

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

Delete Operations

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

CRUD examples

```
> db.user.insert
  ({  first:
  "John",
     last :
     "Doe",
     age: 39
```

```
> db.user.find ()
{ "_id" :
        ObjectId("51"),
        "first" : "John",
        "last" : "Doe",
        "age" : 39
}
```

```
> db.user.update(
        {"_id" :
        ObjectId("51")},
        {
            $set: {
                age: 40,
                salary: 7000}
)       }
```

```
> db.user.remove
  ({  "first": /^J/
})
```

Indexes: High performance read

- Typically used for frequently used queries
- Necessary when the total size of the documents exceeds the amount of available RAM.
- Defined on the collection level
  - Can be defined on 1 or more fields
    - Composite index (SQL) □ Compound index (MongoDB)
- Only 1 index can be used by the query optimizer when retrieving data
- **Index covers a query** - match the *query conditions* **and** return the results using only the index;
  - Use index to provide the results

Replication of data

- Ensures redundancy, backup, and automatic failover
  - Recovery manager in the RDMS
- Replication occurs through groups of servers known as replica sets
  - Primary set – set of servers that client tasks direct updates to
  - Secondary set – set of servers used for duplication of data
  - At the most can have 12 replica sets
    - Many different properties can be associated with a secondary set i.e. secondary-only, hidden delayed, arbiters, non-voting
  - If the primary set fails the secondary sets ‘vote’ to elect the new primary set

Consistency of data

- All read operations issued to the primary of a replica set are consistent with the last write operation
  - Reads to a primary have **strict consistency**
    - Reads reflect the latest changes to the data
  - Reads to a secondary have **eventual consistency**
    - Updates propagate gradually
  - If clients permit reads from secondary sets – then client may read a previous state of the database
  - Failure occurs before the secondary nodes are updated
    - System identifies when a rollback needs to occur
    - Users are responsible for manually applying rollback changes

Summary

- NoSQL built to address a distributed database system
  - Sharding
  - Replica sets of data

  consistency, availability and partition tolerant
- CAP Theorem:

- MongoDB
  - Document oriented data, schema-less database, supports secondary indexes, provides a query language, consistent reads on primary sets
  - Lacks transactions, joins