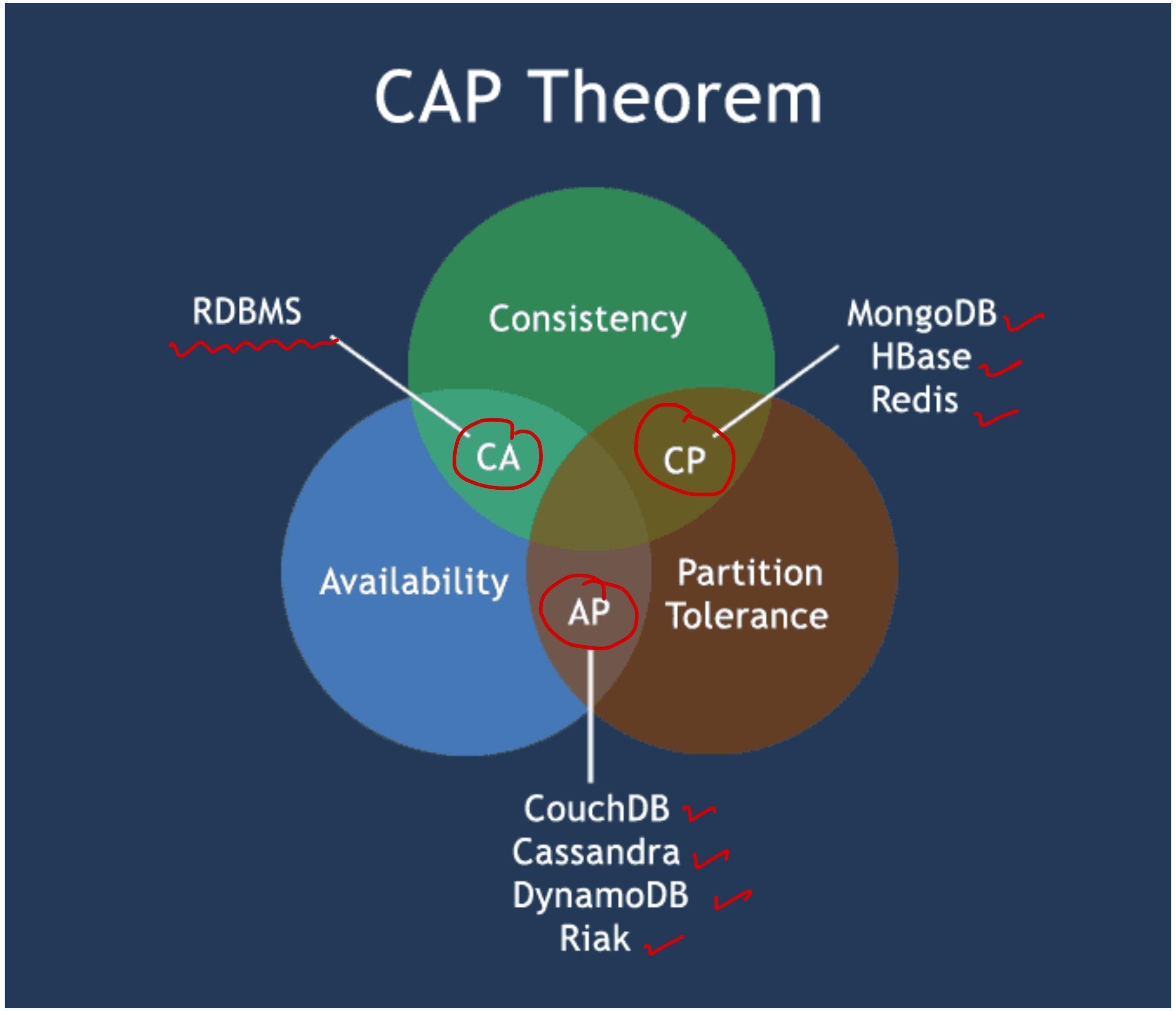
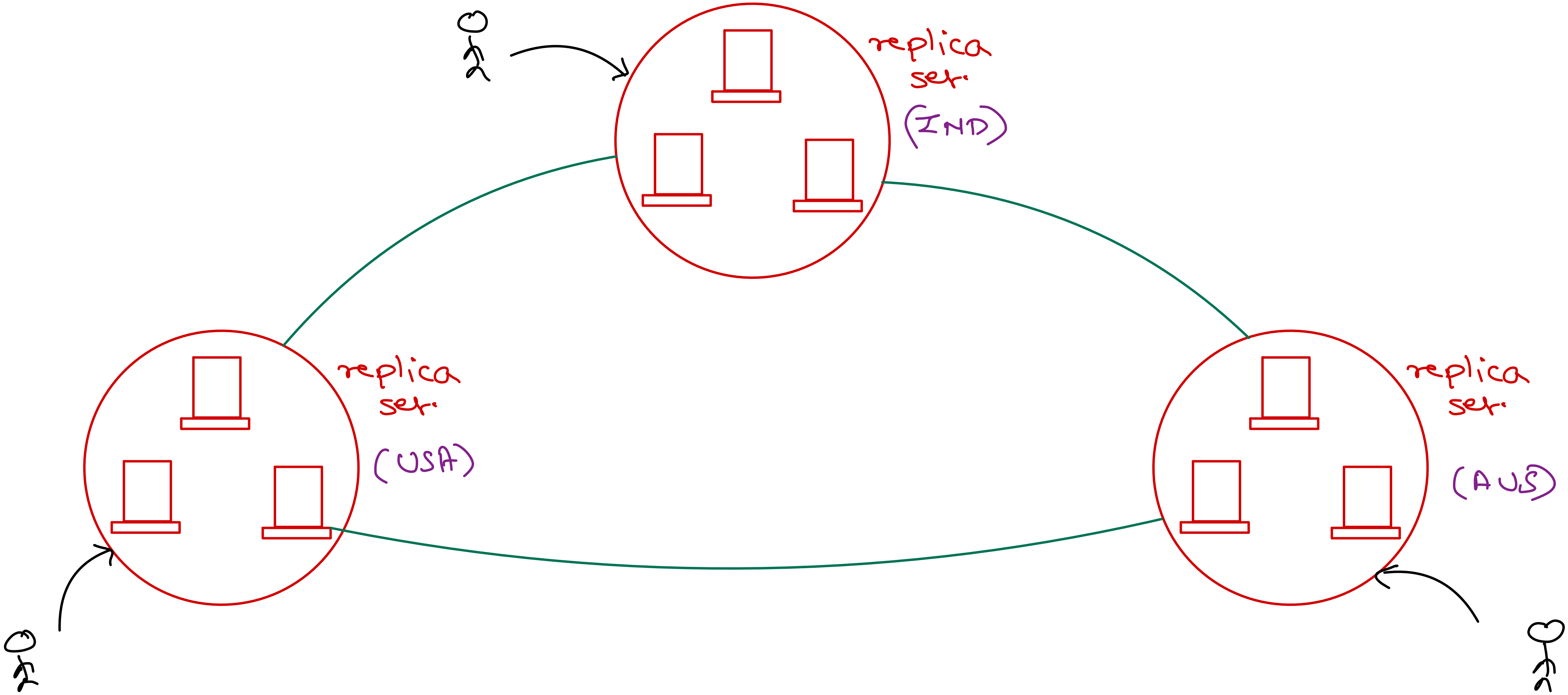
NoSQL Databases

Trainer: Mr. Nilesh Ghule

CAP (Brewer's) Theorem

- **Consistency** - Data is consistent after operation. After an update operation, all clients see the same data.
- **Availability** - System is always on (i.e. service guarantee), no downtime.
- **Partition Tolerance** - System continues to function even the communication among the servers is unreliable.
- Brewer's Theorem
  - It is impossible for a distributed data store to simultaneously provide more than two out of the above three guarantees.




replica set.
(IND)
replica set.
(USA)
replica set.
(AUS)

Applications

- When to use NoSQL?
  - ✓ Large amount of data (TBs)
  - ✓ Many Read/Write ops
  - ✓ Economical Scaling
  - ✓ Flexible schema
- Examples:
  - ✓ Social media
  - ✓ Recordings
  - ✓ Geospatial analysis
  - ✓ Information processing

- When Not to use NoSQL?
  - ✓ Need ACID transactions
  - ✓ Fixed multiple relations
  - ✓ Need joins
  - ✓ Need high consistency
- Examples
  - ✓ Financial transactions
  - ✓ Business operations

RDBMS vs NoSQL

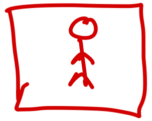
	RDBMS	NoSQL
Types	All types support SQL standard	Multiple types exists, such as document stores, key value stores, column databases, etc
History	Developed in 1970	Developed in 2000s
Examples	SQL Server, Oracle, MySQL ✓	MongoDB, HBase, Cassandra, Redis, Neo4J ✓
Data Storage Model	Data is stored in rows and columns in a table, where each column is of a specific type	The data model depends on the database type. It could be Key-value pairs, documents etc
Schemas	Fixed structure and schema	Dynamic schema. Structures can be accommodated
Scalability	Scale up approach is used (vertical)	Scale out approach is used (horizontal)
Transactions	Supports ACID ~~and~~ transactions	Supports partitioning and availability BASE
Consistency	Strong consistency	Dependent on the product [Eventual Consistency]
Support	High level of enterprise support ✓	Open source model ✓
Maturity	Have been around for a long time ✓	Some of them are mature; others are evolving ✓

NoSQL database

- NoSQL databases are non-relational. → ~~tabular~~
- There is no standardization/rules of how NoSQL database to be designed.
- All available NoSQL databases can be broadly categorized as follows:
  - ✓ Key-value databases
  - ✓ Column-oriented databases
  - ✓ Graph databases
  - ✓ Document oriented databases

Key-value database

- Based on Amazon’s Dynamo database.
- For handling huge data of any type.
- Keys are unique and values can be of any type i.e. JSON, BLOB, etc.
- Implemented as big distributed hash-table for fast searching.
- Example: redis, dynamodb, riak, ...

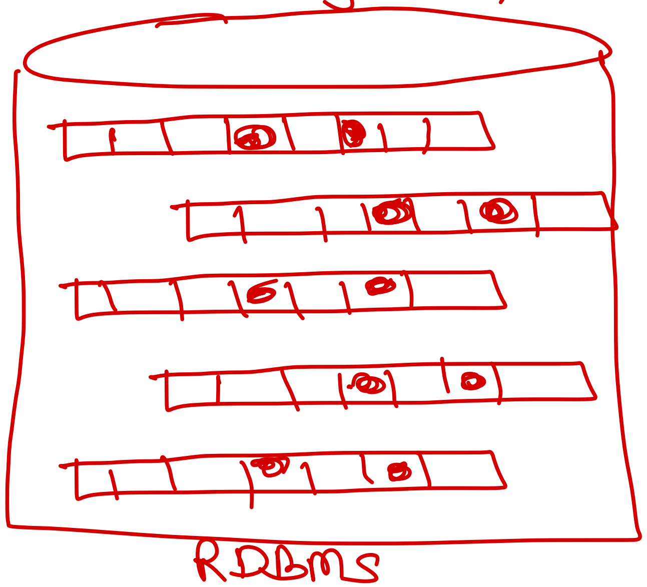
key	value
/sunbeam/dac/1/name	Abc
/sunbeam/dac/1/addr	Pune
/sunbeam/dac/1/hobbies	[Prog, Cooking, Read]
/sunbeam/dac/1/photo	 (binary).

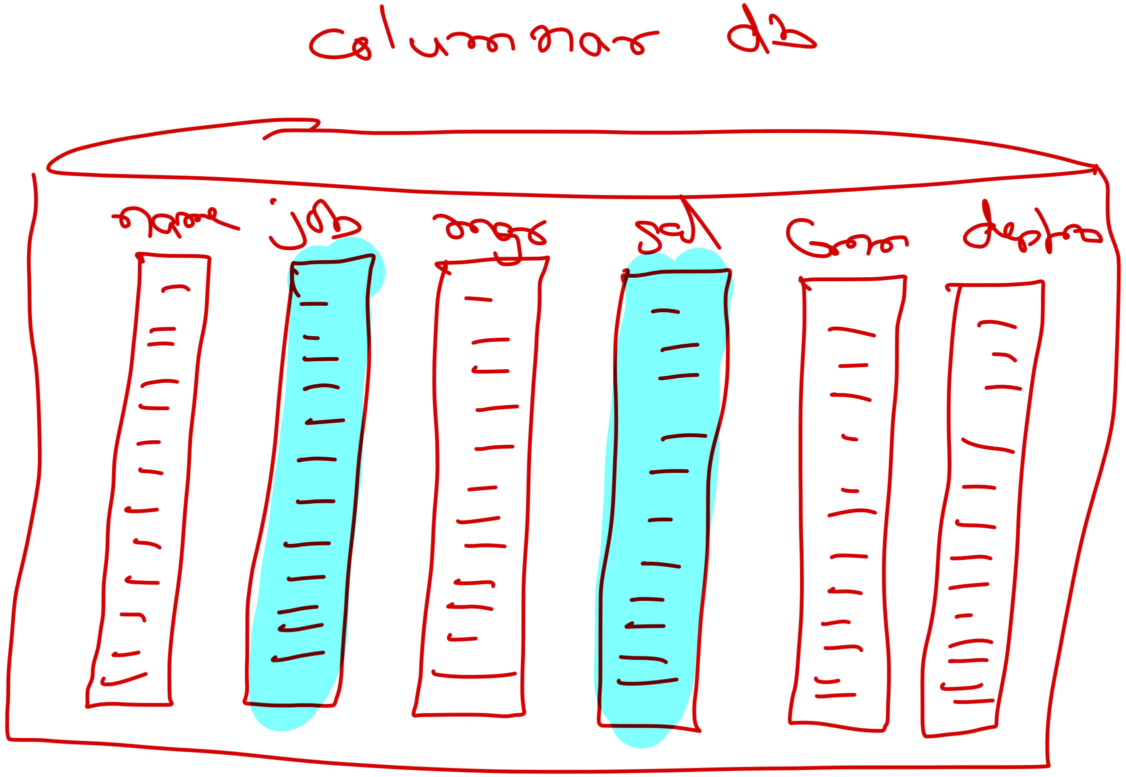
across multiple machines

Column-oriented databases

- Values of columns are stored contiguously.
- Better performance while accessing few columns and aggregations.
- Good for data-warehousing, business intelligence, CRM, ...
- Examples: hbase, cassandra, bigtable, …

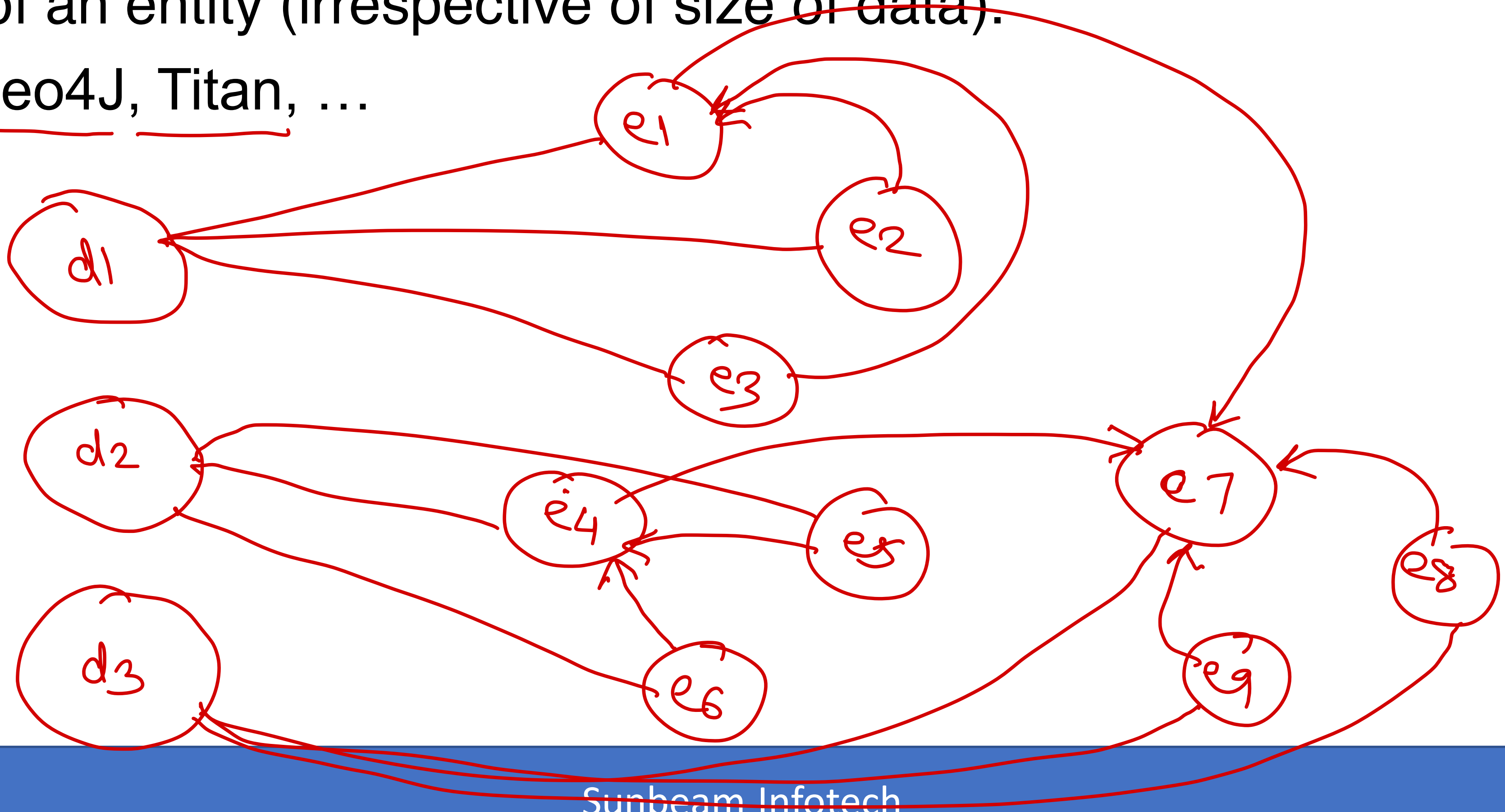
select job, sum(sal) from emp
group by job;





Graph databases

- Graph is collection of vertices and edges (lines connecting vertices).
- Vertices keep data, while edges represent relationships.
- Each node knows its adjacent nodes. Very good performance, when want to access all relations of an entity (irrespective of size of data).
- Examples: Neo4J, Titan, …



Document oriented databases

- Document contains data as key-value pair as JSON or XML.
- Document schema is flexible & are added in collection for processing.
- RDBMS tables → Collections
- RDBMS rows → Documents
- RDBMS columns → Key-value pairs in document
- Examples: MongoDb, CouchDb, …

b1 JSON → Java Script Object Notation.

```
{
   "id" : 1, → int
   "title": "Let us C", → string
   "author": "Kanetkar",
   " price" : 240.4 → double
}
```

r1 → JSON document

```
{
   id : 1,
   name : "Nilesh",
   age : 38,
   hobbies : ["Programm", "Reading", ...],
   addr : { area : "Katraj", city : "Pune", pin : 411046 },
   political : false,
   height : 5.9,
   bloodgroup : null
}
```

MongoDb Databases

Trainer: Mr. Nilesh Ghule

Agenda

- Introduction
- Installation
- JSON vs BSON
- Basic CRUD operations

Mongo Db

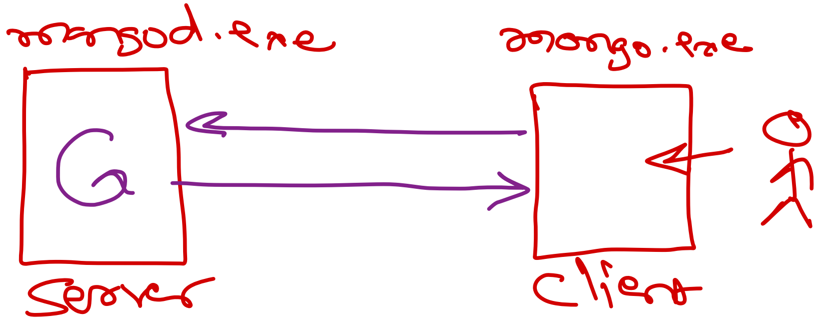
- Developed by 10gen in 2007
- Publicly available in 2009
- Open-source database which is controlled by 10gen
- Document oriented database → stores JSON documents
- Stores data in binary JSON. (BSON)
- Design Philosophy
  - MongoDB wasn't designed in a lab and is instead built from the experiences of building large scale, high availability, and robust systems.

JSON

- Java Script Object Notation
- Hierarchical way of organizing data
- Mongo stores JSON data into Binary form.

Mongo Server and Client

- MongoDb server (mongod) is developed in C, C++ and JS.
- MongoDb data is accessed via multiple client tools
  - mongo : client shell (JS). obj. method (args) ;
  - mongofiles : stores larger files in GridFS.
  - mongoimport / mongoexport : tools for data import / export.
  - mongodump / mongorestore : tools for backup / restore.
- MongoDb data can be accessed in application through client drivers available for all major programming languages e.g. Java, Python, Ruby, PHP, Perl, …
- Mongo shell is follows JS syntax and allow to execute JS scripts.



MongoDb: Data Types

data	bson	values
null	10	
boolean	8	true, false
number	1 / 16 / 18	123, 456.78, NumberInt(“24”), NumberLong(“28”)
string	2	“....”
date	9	new Date(), ISODate(“yyyy-mm-ddThh:mm:ss”)
array	4	[…, …, …, …]
object	3	{ … }

Mongo - INSERT

- show databases;
- use database;
- db.contacts.insert({name: "nilesh", mobile: "9527331338"});
- db.contacts.insertMany([

```
        {name: "nilesh", mobile: "9527331338"},
        {name: "nitin", mobile: "9881208115"}
]);
```

- Maximum document size is 16 MB.

field/key

- For each object unique id is generated by client (if _id not provided).
  - 12 byte unique id :: [counter(3) I pid(2) I machine(3) I timestamp(4)]

client process id

client

Mongo – QUERY

- db.contacts.find(); → returns cursor on which following ops allowed:
  - hasNext(), next(), skip(n), limit(n), count(), toArray(), forEach(fn), pretty()
- Shell restrict to fetch 20 records at once. Press "it" for more records.
- db.contacts.find({ name: "nilesh" });
- db.contacts.find({ name: "nilesh" }, { _id:0, name:1 });
- Relational operators: $eq, $ne, $gt, $lt, $gte, $lte, $in, $nin
- Logical operators: $and, $or, $nor, $not
- Element operators: $exists, $type
- Evaluation operators: $regex, $where, $mod
- Array operators: $size, $elemMatch, $all, $slice

Mongo – DELETE

- db.contacts.remove(criteria);
- db.contacts.deleteOne(criteria);
- db.contacts.deleteMany(criteria);
- db.contacts.deleteMany({}); → delete all docs, but not collection
- db.contacts.drop(); → delete all docs & collection as well : efficient

Mongo – UPDATE

- db.contacts.update(criteria, newObj);
- Update operators: $set, $inc, $dec, $push, $each, $slice, $pull
- In place updates are faster (e.g. $inc, $dec, …) than setting new object. If new object size mismatch with older object, data files are fragmented.
- Update operators: $addToSet
- example: db.contacts.update({ name: "peter" },
- { $push : { mobile: { $each : ["111", "222"], $slice : -3 } } });
- db.contacts.update({ name: "t" }, { $set : { "phone" : "123" } }, true);
  - If doc with given criteria is absent, new one is created before update.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>