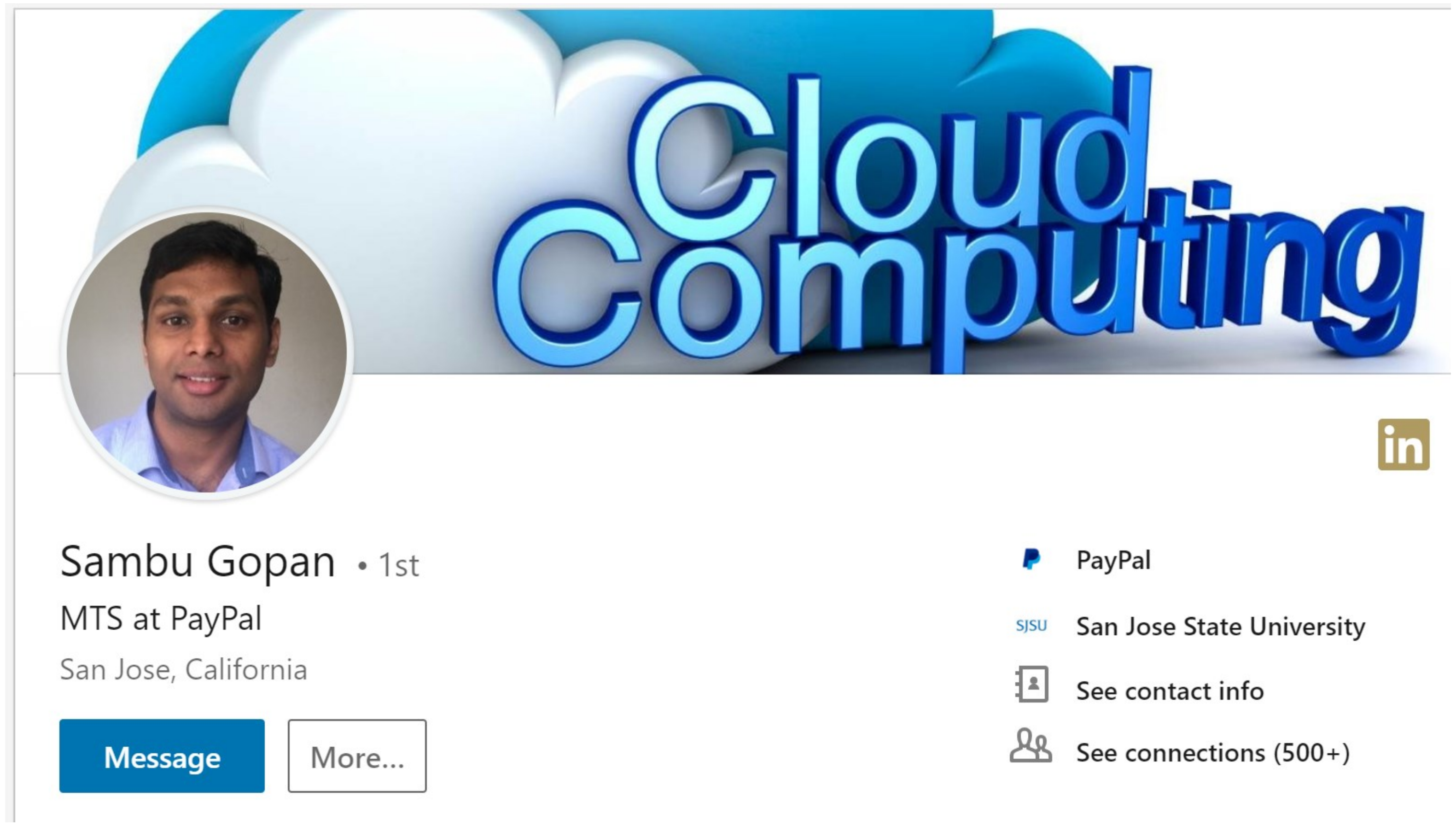
Guest Lecture 10/23



Sambu Gopan • 1st

MTS at PayPal

San Jose, California

PayPal

San Jose State University

See contact info

See connections (500+)

Introduction to MongoDB (NoSQL DB)

Why NoSql ?

- Relational databases are not designed to scale
- schema, joins

C and Latency Tradeoff

- Amazon claims that just an extra one tenth of a second on their response times will cost them 1% in sales.
- Google said they noticed that just a half a second increase in latency caused traffic to drop by a fifth.

4 Key Words on NoSQL

- Scale
- Speed
- Cloud
- New Data

What is NoSQL?

- non-relational
- simple API
- schema-free
- open-source
- horizontally scalable (sharding)
- replication support
- eventually consistent /BASE

Different types of NoSQL Databases

- NoSQL database are classified according to their data storage models:
  - Column (Cassandra)
  - Document (MongoDB)
  - Key – value Pair(Dynamo – Amazon)
  - Graph

MongoDB

- Name derived from Hu(**MONGO)**us word
- Document Oriented Database
- Built for High – Performance and scalability
- Document based queries for **Easy Readability**
- Replication and failover for **High Availability**
- Auto Sharding for **Easy Scalability**

Comparison between RDBMS and NoSQL DB

- Example: Class
- Location
- Presenter
  - Presenting at a location
- People
  - Potential attendees in context of a class
- Class
  - Presenter in location with people as actual attendees

Relational Database: Example

- Class schema in a relational database
- Presentation { id, name, location}
- People {id, name}
- Address {id, city, state, zip}

Schema for this class in a relational database model

Presentation

id	name	*location*
1	Chris	SJSU

Address

id	city	state
SJSU	San Jose	CA

People

id	name
10	Simon
11	Chris

Class

id	*person*	*presentation*
20	10	1
20	11	1

Relational database: Example

```
CREATE TABLE Presentation (
        id Integer primary key, name String, location string,
        FOREIGN KEY (location) REFERENCES Address(id));
CREATE TABLE Address (
        id String primary key, city String, state String);
CREATE TABLE People (
        id Integer primary key, name String);
CREATE TABLE Class (
        id Integer, person Integer, presentation Integer,
        PRIMARY KEY (id, person, presentation),
        FOREIGN KEY (person) REFERENCES People(id),
        FOREIGN KEY (presentation) REFERENCES Presentation(id));
```

Relational database: Example

```
select Presentation.name, Presentation.location,
        Address.city, Address.state, People.name
from Presentation, Address, People, Class
where Class.person = People.id
            and Class.presentation = Presentation.id
            and Presentation.location = Address.id;
```

```
| name | location | city | state | name |
+--------+--------+------------+-------+---------+
| Chris | SJSU | San Jose | CA   | Simon|
| Chris | SJSU | San Jose | CA   | Chris  |
```

Relational Database: Recap

1. Schema design

    Primary key (underlined) and foreign key (cursive) constraints

2. Table creation

    DDL

3. Data insertion for each table

    DML

4. Query: join

    DML

5. Data structure creation within application system

    JDBC resultset to e.g. Java objects

NoSQL Database: Use Case Example

```
use course /* database will be created if not present */
db.presentation.insert(
{"id": 1,
  "name": “Simon",
  "location": {"id": "SJSU",
                    "city": "San Jose",
                    "state": "CA"
                  },
"people": [{"id": 10, "name": "Simon"},
                {"id": 11, "name": "Chris"}
              ]
})
```

NoSQL Database: Use Case Example

- db.presentation.find()
- db.presentation.find({"id": 1})

NoSQL Database: Recap

1. ~~Schema design~~

    Primary key (underlined) and foreign key (cursive) constraints

2. Table creation

    DDL

3. Data insertion for each table

    DML

4. Query: ~~join~~

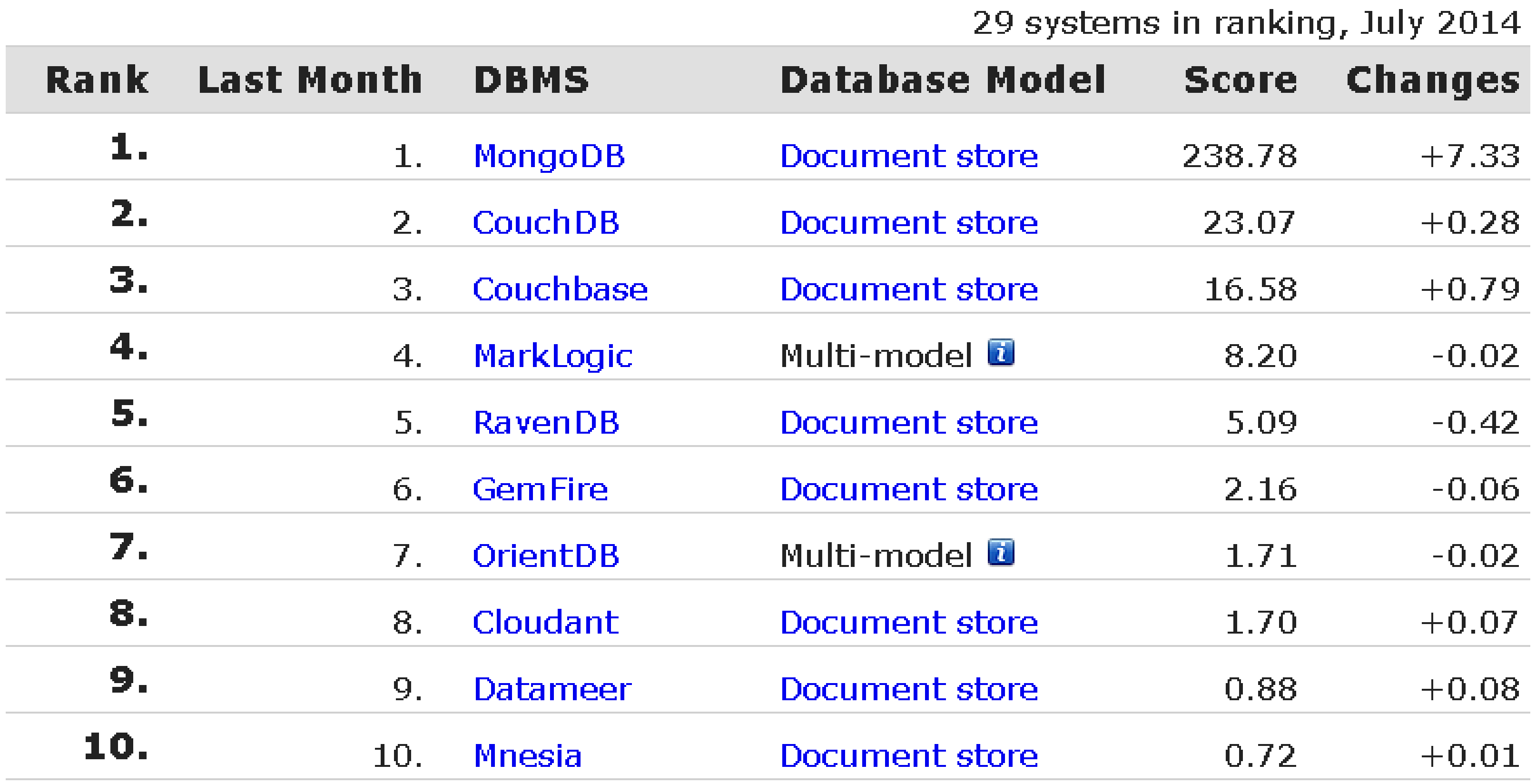
    DML

~~5. Data structure creation within application system~~

    ~~JDBC resultset to e.g. Java objects~~

NoSQL Database: Major Players

- Too many document NoSQL databases to name a few distinct ones



29 systems in ranking, July 2014

Rank	Last Month	DBMS	Database Model	Score	Changes
1.	1.	MongoDB	Document store	238.78	+7.33
2.	2.	CouchDB	Document store	23.07	+0.28
3.	3.	Couchbase	Document store	16.58	+0.79
4.	4.	MarkLogic	Multi-model ℹ	8.20	-0.02
5.	5.	RavenDB	Document store	5.09	-0.42
6.	6.	GemFire	Document store	2.16	-0.06
7.	7.	OrientDB	Multi-model ℹ	1.71	-0.02
8.	8.	Cloudant	Document store	1.70	+0.07
9.	9.	Datameer	Document store	0.88	+0.08
10.	10.	Mnesia	Document store	0.72	+0.01

Key Benefit of NoSQL: O(1) Lookup

- Fast lookup
  - No joining required
  - All data about one domain concept in one document
- Direct programming language representation
  - No mapping or ‘ORM’ layer required
- JSON library
  - Direct result representation and manipulation
  - JavaScript: representation in language data types directly
  - E.g., check out MongoDB node.js driver

Key Problem of NoSQL: No Join Operator

- Many NoSQL databases do not implement a join query operator
  - If you need to join data, then you have to do it in the application system layer
- But, wait a moment ...
  - Is it ever necessary to join data in NoSQL databases?
  - Some claim: not necessary due to support of
    - Sub-documents
    - Arrays (lists)
- Let's look at an example
  - Supplier - Parts

Key Problem of NoSQL: No Join Operator

- Example
  - Supplier - Parts relationship (N:M)
  - Each supplier supplies many parts
  - Each part supplied by many suppliers
- Relational DBMS
  - “Supplier” table
  - “Part” table
  - “Supplies” relationship in table

Key Problem of NoSQL: No Join Operator

Supplier - Part - Supplies

```
| Supplier |      | Part |     | Supplies |
+------------+      +------------+    +------------------------+
| id | name|    | id | name| |supplier_id|part_id |
+----+---------+  +----+-------+  +------------+----------+
| 10 | Supp1 |      | 20 | Part1 |   | 10           |       20      |
| 11 | Supp2 |      | 21 | Part2 |   | 10           | 21        |
                                         | 11         | 20       |
```

Key Problem of NoSQL: No Join Operator

Supplier - Supplies – Part

```
{ "id": 10,
"name": "Supp1",
"supplies": [{"id": 20,"name":"Part1"},
              {"id": 21, "name":
"Part2"}]}

{ "id": 11,
"name": "Supp2",
"supplies": [{"id": 20,"name":"Part1"}]}
```

Supplier - Supplies – Part

```
{ "id": 10,
"name": "Supp1",
"supplies": [20, 21]}
{ "id": 10,
"name": "Supp1",
"supplies": [20, 21]}

{"id": 20, "name": "Part1"}
{"id": 21, "name": "Part2"}
```

Why use MongoDB?

- MongoDB stores data in Objects
- Uses BSON (Binary JSON)
- No Joins
- No Complex Queries
- Embedded Documents and arrays reduce the need for joins
- No multi-document transactions

Where to use MongoDB ?

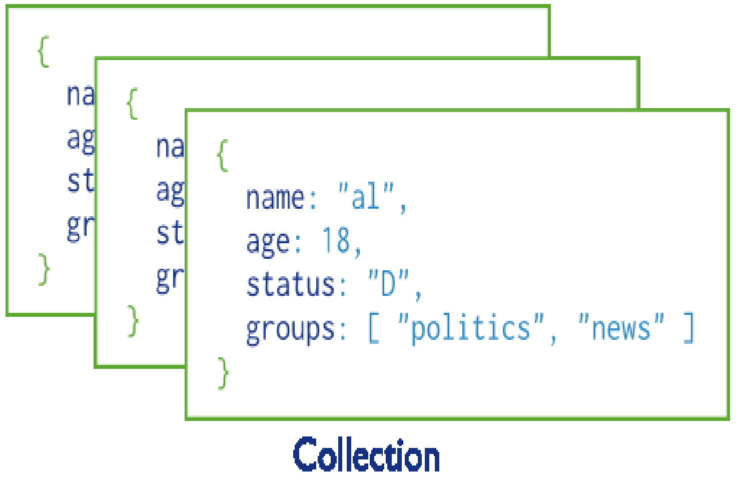
- Ideal for Web Applications
- Applications containing semi-structured data and need flexible schema management
- Caching and High Scalability
- Scenarios where **data availability** and **size of data** are priorities over the **transactions** of data

Terminology

• Mysql	• MongoDB
• Table	• Collection
• Row	• Document
• Column	• Field
• Joins	• Not Recommended ($lookup)
• Group By	• Aggregation

Collections in MongoDB

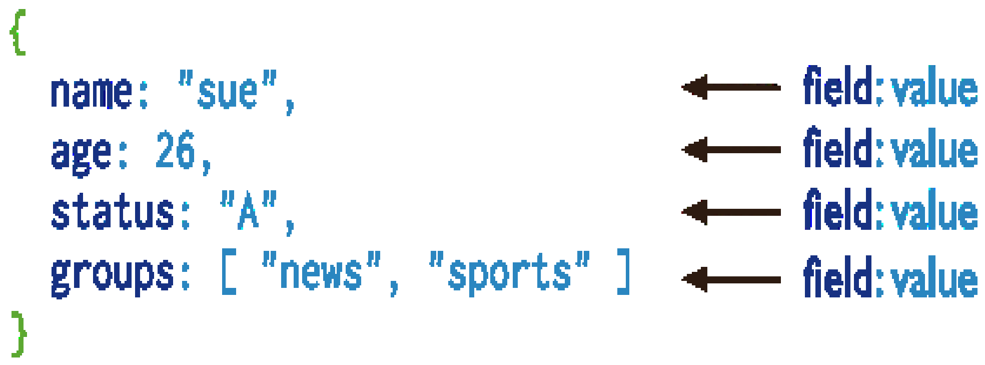
- MongoDB stores all data in Collections
- It is schema – less and contains a group of related documents
- Created on-the-fly when referenced for the first time



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

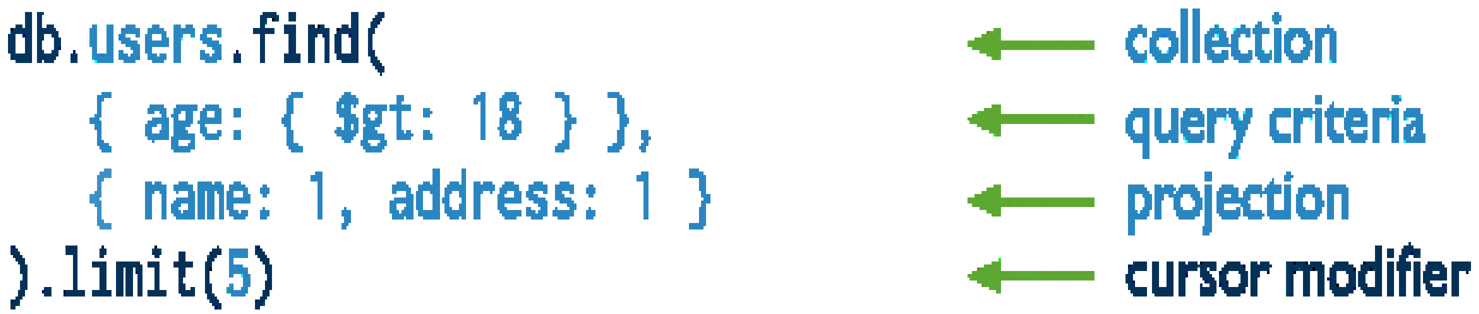
Document in MongoDB

- Stored in Collections
- Has **_id** field – works like Primary keys in Relational databases
- Sample document containing name, age, status and groups



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

Queries in MongoDB

- MongoDB provides **db.collection.find()** method
- This method accepts both query criteria and projections
- 

```
db.users.find(
    { age: { $gt: 18 } },
    { name: 1, address: 1 }
).limit(5)
```

Projections - Queries in MongoDB

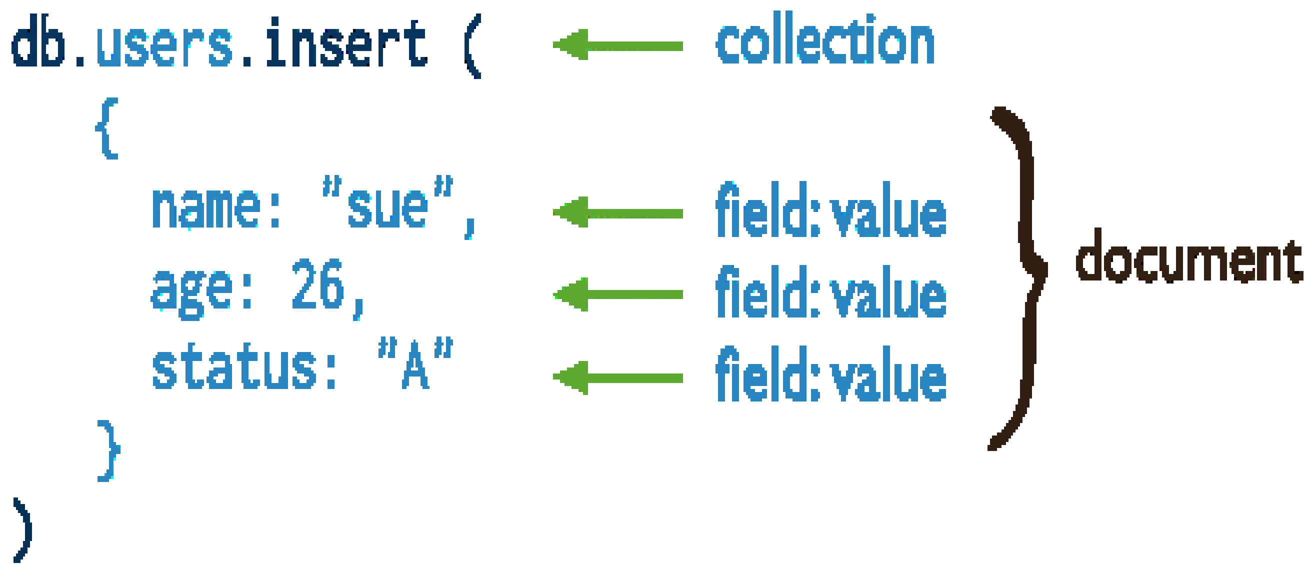
- If you include 1 –it returns the value
- If you include 0 –it eliminates it from the result

```
db.records.find( { "user_id": { $lt: 42} }, { "_id": 0, "name": 1 , "email": 1 } )
```

- _id – always included in results. Specify “**_id : 0**” to exclude it from results

Insert Operation

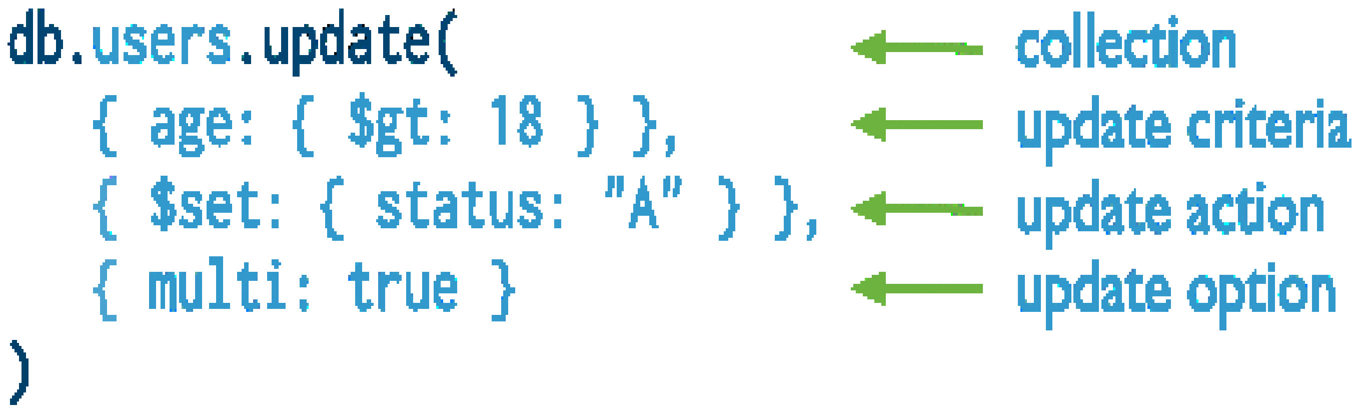
- In MongoDB, **db.collection.insert()** method adds new documents to collections



```
db.users.insert (
   {
     name: "sue",
     age: 26,
     status: "A"
   }
)
```

Update Operation

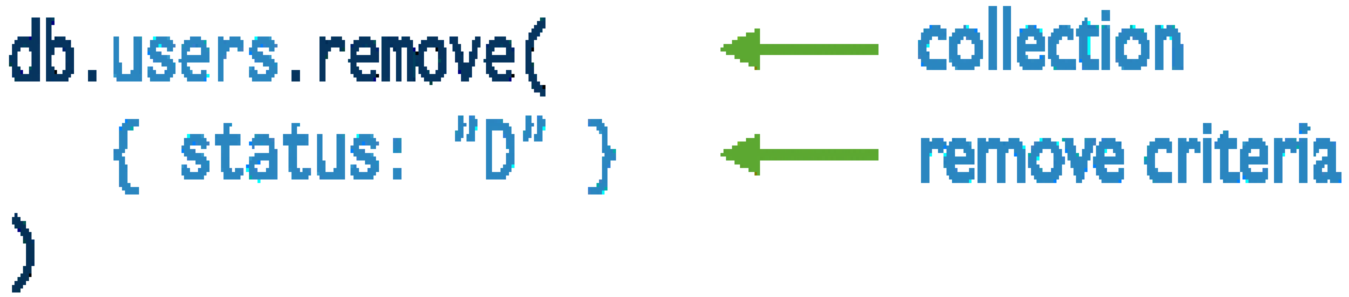
- In MongoDB, **db.collection.update()** method modifies existing documents in a collection



```
db.users.update(
   { age: { $gt: 18 } },
   { $set: { status: "A" } },
   { multi: true }
)
```

Remove Operation

- In MongoDB, **db.collection.remove()** method deletes document from the collection



```
db.users.remove(
   { status: "D" }
)
```

Using MongoDB with Node.js

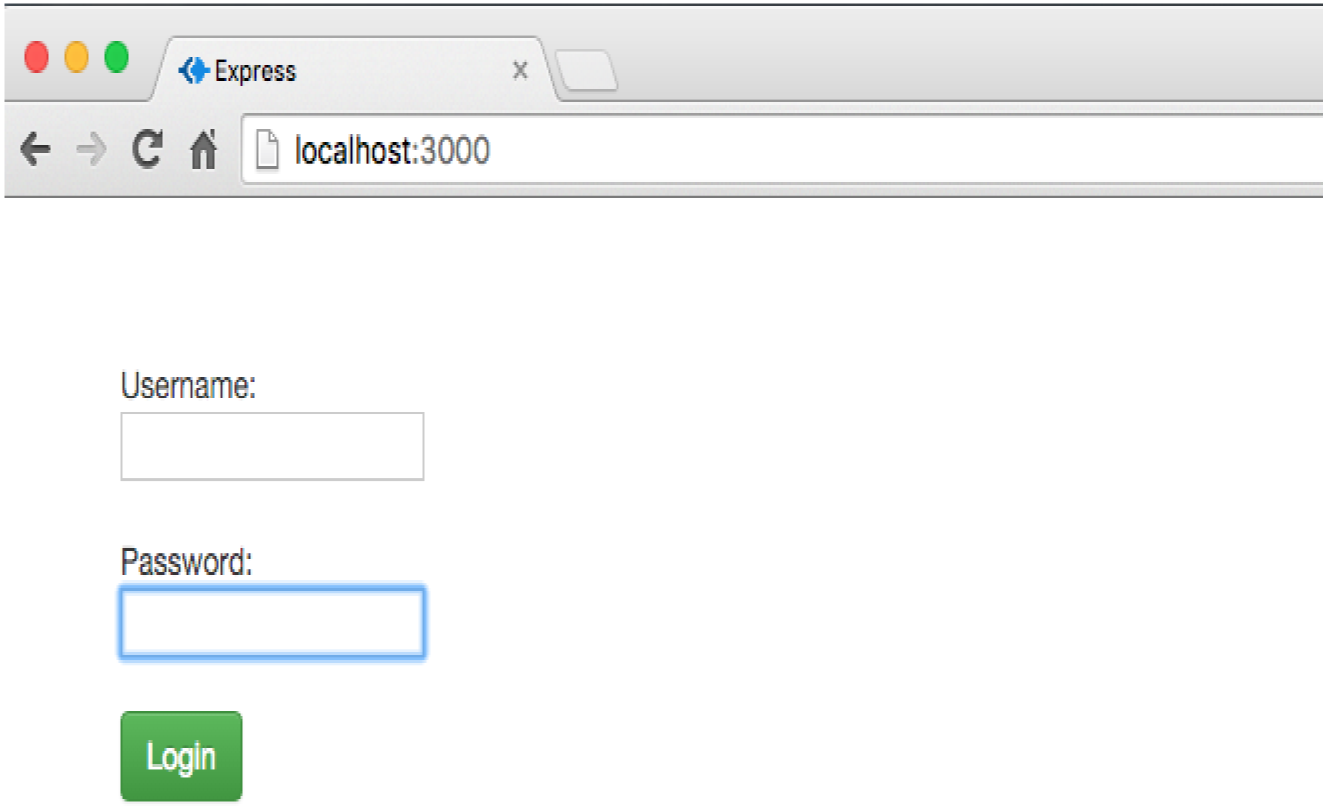
- **Install MongoDB Node.js Module**

```
npm install mongodb
```

Example

- Login Application
- Access MongoDB to authentic the user
- Use Mongo Store to store sessions in MongoDB

Example – Login Page



Exercise

- Create a Login Application
- Should have option to sign up the user
- Login with the same user
- Show the details of the signed in user
- Use MongoDB to store the data
- Use Passport with Express session for authentication

References

- SQL vs NoSQL - https://www.mongodb.com/nosql-explained
- MongoDB Introduction - http://docs.mongodb.org/manual/core/crud-introduction/
- Installing MongoDB (Mac) - https://www.youtube.com/watch?v=_WJ8m5QHvwc
- Installing MongoDB (Windows) - https://www.youtube.com/watch?t=1&v=sBdaRlgb4N8

When to not use MongoDB?

- ACID properties are important for storage
- Highly Transactional Applications (Banking domain, Security)
- Problems and applications requiring Joins and complex queries

Key Problem of NoSQL: No Database-Enforced Consistency

- Not enforced
    - Primary key
    - Foreign key
    - Enumeration
    - Cascading delete
    - etc.
- Enforcement can be accomplished
    - When
        - reading or writing
    - In application system code
    - In self-implemented database access layer
    - In separate consistency check process
    - Not at all

How does MongoDB Store data?

- Stores data in form of Documents
- JSON like field – value pair
- Documents analogous to structures in programming languages with key – value pair
- Documents stored in **BSON (Binary JSON)** format
- BSON is JSON with additional type information

NoSQL: Key Insights

- Specialized data models
  - Not universal, but optimized towards special cases
- Specialized query access
  - Not universal, but optimized towards special cases
- Different / absent consistency supervision
  - Relaxed constraints

- Trade-off
  - Gain through specialization
  - Implementation of missing functionality outside of database