NoSQL and MongoDB
HOW TO WRITE A CV

DO YOU HAVE ANY EXPERTISE IN SQL?

NO

Leverage the NoSQL boom

DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"
Introduction to NoSQL

Based on a presentation by Traversy Media
What is NoSQL?

• Not only SQL
• SQL means
  ▪ Relational model
  ▪ Strong typing
  ▪ ACID compliance
  ▪ Normalization
  ▪ ...
• NoSQL means more freedom or flexibility
Relevance to Big Data

• Data gets bigger
• Traditional RDBMS cannot scale well
• RDBMS is tied to its data and query processing models
• NoSQL relaxes some of the restrictions of RDBMS to provide a better performance
Advantages of NoSQL

• Handles Big Data
• Data Models – No predefined schema
• Data Structure – NoSQL handles semi-structured data
• Cheaper to manage
• Scaling – Scale out / horizontal scaling
Advantages of RDBMS

• Better for relational data
• Data normalization
• Well-established query language (SQL)
• Data Integrity
• ACID Compliance
Types of NoSQL Databases

• Document Databases [MongoDB, CouchDB]
• Column Databases [Apache Cassandra]
• Key-Value Stores [Redis, Couchbase Server]
• Cache Systems [Redis, Memcached]
• Graph Databases [Neo4J]
• Streaming Systems [FlinkDB, Storm]
## Structured/Semi-structured

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Email</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jack</td>
<td><a href="mailto:jack@example.com">jack@example.com</a></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Jill</td>
<td><a href="mailto:jill@example.net">jill@example.net</a></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Alex</td>
<td><a href="mailto:alex@example.org">alex@example.org</a></td>
<td></td>
</tr>
</tbody>
</table>

**Document 1**

```json
{  "id": 1,  "name":"Jack",  "email": "jack@example.com",  "address": {"street": "900 university ave",  "city": "Riverside",  "state": "CA"},  "friend_ids": [3, 55, 123]}
```

**Document 2**

```json
{  "id": 2,  "name": "Jill",  "email": "jill@example.net",  "hobbies": ["hiking", "cooking"]}
```
Document Database
MongoDB
Document Data Model

- Relational model (RDBMS)
  - Database
    - Relation (Table) : Schema
      - Record (Tuple) : Data

- Document Model
  - Database
    - Collection : No predefined schema
      - Document : Schema+data

- No need to define/update schema
- No need to create collections

Document 1

```json
{ "id": 1, "name": "Jack", "email": "jack@example.com", "address": {"street": "900 university ave", "city": "Riverside", state: "CA"}, "friend_ids": [3, 55, 123]}
```
Document Format

• MongoDB natively works with JSON documents

• For efficiency, documents are stored in a binary format called BSON (i.e., binary JSON)

• Like JSON, both schema and data are stored in each document
How to Use MongoDB

Install: Check the MongoDB website
https://docs.mongodb.com/manual/installation/

Create collection and insert a document

```javascript
db.users.insert({name: "Jack", email: "jack@example.com"});
```

Retrieve all/some documents

```javascript
db.users.find();
db.users.find({name: "Jack"});
```

Update

```javascript
db.users.update({name: "Jack"}, {$set: {hobby: "cooking"}});
updateOne, updateMany, replaceOne
```

Delete

```javascript
db.users.remove({name: "Alex"});
deleteOne, deleteMany
```

https://docs.mongodb.com/manual/crud/
• You can still explicitly create collections and enforce schema validation

```javascript
db.createCollection("students", {
    validator: { $jsonSchema: {
        bsonType: "object",
        required: [ "name", "year", "major", "address" ],
        properties: {
            name: {
                bsonType: "string",
                description: "must be a string and is required" },
            ...
        }
    }
})
```

https://docs.mongodb.com/manual/core/schema-validation/
Indexing

• Like RDBMS, document databases use indexes to speed up some queries

```javascript
db.users.find({ score: { "$lt": 30 } }).sort({ score: -1 })
```

• MongoDB uses B-tree as an index structure

[Diagram](https://docs.mongodb.com/manual/indexes/)
Index Types

• Default unique _id index
• Single field index
  ▪ db.collection.createIndex({name: -1});
• Compound index (multiple fields)
  ▪ db.collection.createIndex( { name: 1, score: -1});
• Multikey indexes (for array fields)
  ▪ Creates an index entry for each value

https://docs.mongodb.com/manual/indexes/
Index Types

• Geospatial index (for geospatial points)
  ▪ Uses geohash to convert two dimensions to one dimension
  ▪ 2d indexes: For Euclidean spaces
  ▪ 2d sphere: spherical (earth) geometry
  ▪ Works with multikey indexes for multiple locations (e.g., pickup and dropoff locations for taxis)

• Text Indexes (for string fields)
  ▪ Automatically removes stop words
  ▪ Stems the words to store the root only

• Hashed Indexes (for point lookups)
Geohashes
Additional Index Features

• Unique indexes: Rejects duplicate keys
• Sparse Indexes: Skips documents without the index field
  ▪ In contrast, non-sparse indexes assume a null value if the index field does not exist
• Partial indexes: Indexes only a subset of records based on a filter.

```javascript
db.restaurants.createIndex(
    { cuisine: 1, name: 1 },
    { partialFilterExpression: { rating: { $gt: 5 } } }
)
```
Comparison of data types

- Min key (internal type)
- Null
- Numbers (32-bit integer, 64-bit integer, double)
- Symbol, String
- Object
- Array
- Binary data
- Object ID
- Boolean
- Date, timestamp
- Regular expression
- Max key (internal type)

https://docs.mongodb.com/v3.6/reference/bson-type-comparison-order/
Comparison of data types

- Numbers: All converted to a common type
- Strings
  - Alphabetically (default)
  - Collation (i.e., locale and language)
- Arrays
  - <: Smallest value of the array
  - >: Largest value of the array
  - Empty arrays are treated as null
- Object
  - Compare fields in the order of appearance
  - Compare <name,value> for each field
Distributed Processing

• Two methods for distributed processing
  ▪ Replication (Similar to MySQL)
  ▪ Sharding (True horizontal scaling)

Replication
https://docs.mongodb.com/manual/replication/

Sharding
https://docs.mongodb.com/manual/sharding/
Distributed Index Structure

Log-structured Merge Tree (LSM)
Big Data Indexing

• Hadoop and Spark are good in scanning large files
• We would like to speed up point and range queries on big data for some queries
• HDFS limitation: Random updates are not allowed
• Log-structured Merge Tree (LSM-Tree) is adopted to address this problem.
RDBMS Indexing

New record → Log → Index
Index Update in RDBMS

New record

Randomly updated disk page(s)

Append a disk page
**LSM Tree**

- Key idea: Use the log as the index
- Regularly: Merge the logs to consolidate the index (i.e., remove redundant entries)

LSM in Big Data

- First major application: BigTable (Google)

Citations by year

First report from Google mentioning LSM
LSM in Big Data

• Buffer data in memory (memory component)
• Flush records to disk into an LSM as a disk component (sequential write)
• Disk components are sorted by key
• Compact (merge) disk components in the background (sequential read/write)
Conclusion

• MongoDB is a document database that is geared towards high update rates and transactional queries
• It adopts JSON as a data model
• It provides the flexibility to insert any kind of data without schema definition
• LSM Tree is used for indexing
• Weak types are handled using a special comparison method for all types