M149: Database Systems **Winter 2020**

NoSQL Databases: December 2020

Lecturer: Panagiotis Liakos

1 Drawbacks of Relational Databases

- Static Schemas
  - Relational databases require that schemas be defined before you can add data.
  - This fits poorly with agile development approaches, because each time you complete new features, the schema of your database often needs to change.
  - If the database is large, this is a very slow process that involves significant downtime.
- Horizontal Scaling
  - Relational databases usually scale vertically; a single server has to host the entire database to ensure acceptable performance for cross-table joins and transactions.
  - 'Sharding' a database across many server instances can be achieved with SQL databases, but usually is accomplished through SANs and other complex arrangements for making hardware act as a single server.

2 Motivation for NoSQL Databases

NoSQL encompasses a wide variety of different database technologies that were developed in response to the demands presented in building modern applications:

- Massive volumes of new, rapidly changing data types — structured, semi-structured, unstructured and polymorphic data.
- Agile Development
- Services must be always-on, accessible from many different devices and scaled globally to millions of users.
- Organizations are now turning to scale-out architectures using open source software, commodity servers and cloud computing instead of large monolithic servers and storage infrastructure.

Relational databases were not designed to cope with the scale and agility challenges that modern applications face, nor were they built to take advantage of the commodity storage and processing power available today.

3 Features of NoSQL Databases

- NoSQL databases are built to allow the insertion of data without a predefined schema. That makes it easy to make significant application changes in real-time, without worrying about service interruptions – which means development is faster, code integration is more reliable, and less database administrator time is needed.
- NoSQL databases usually support auto-sharding, meaning that they natively and automatically spread data across an arbitrary number of servers, without requiring the application to even be aware of the composition of the server pool. Data and query load are automatically balanced across servers, and when a server goes down, it can be quickly and transparently replaced with no application disruption.
- Most NoSQL databases also support automatic database replication to maintain availability in the event of outages or planned maintenance events. More sophisticated NoSQL databases are fully self-healing, offering automated failover and recovery, as well as the ability to distribute the database across multiple geographic regions to withstand regional failures and enable data localization. Unlike relational databases, NoSQL databases generally have no requirement for separate applications or expensive add-ons to implement replication.
- Many NoSQL database technologies have excellent integrated caching capabilities, keeping frequently-used data in system memory as much as possible and removing the need for a separate caching layer. Some NoSQL databases also offer fully managed, integrated in-memory database management layer for workloads demanding the highest throughput and lowest latency.

4 Comparison of Database Solutions

- Relational Databases
  - PostgreSQL, Oracle, MariaDB, SQLite
  - Tables
  - SQL
  - Vertical Scaling, Limited support for clustering and manual partitioning
  - Transactions, Normalized data model
- Key-Value Stores
  - Redis, Memcached
  - Key-Value
  - Redis cluster shards the keys to numerous nodes. Memcached converts all available RAM into a single, monolithic datastore
  - Caching, Web ops
- Column Stores
  - Cassandra, HBase
  - Eventually consistent, multinode distribution for high availability and easy failover
  - Column stores are very efficient at data compression and/or partitioning.
  - Due to their structure, columnar databases perform particularly well with aggregation queries.
- Document Databases

  - MongoDB, CouchDB
  - Documents (JSON, BSON)
  - MongoDB 4.0 supports multi-document transactions.
  - Powerful query engines and indexing features that provide fast and efficient querying capabilities.

- Graph Databases
  - Neo4j
  - Nodes, Edges
  - Several orders of magnitude improvements in performance for applications involving intensive data relationship handling.

5 MongoDB

5.1 Document

Where relational databases have tables, MongoDB has collections. Relational databases keep their data in tables of rows, while MongoDB keeps its data in collections of documents. In a sharded setup, the concept of chunks also exist. A chunk is a a group of documents clustered by values on a field.

A document is a set of property names and their respective values. The values can be simple data types, such as strings, numbers, and dates. But these values can also be arrays and even other documents. These latter constructs permit documents to represent a variety of rich data structures. In the example of Figure 1 we can see various data types. Fields *title* and *url* are strings. Field *vote_count* is a number. Field *image* is a JSON document it self. Finally, *tags* is an array of strings and *comments* is an array of JSON documents. Figure 2 shows the equivalent schema in a relational database. Representing the various one-to-many relationships requires multiple tables.

Embedded data models allow applications to store related pieces of information in the same database record. As a result, applications may need to issue fewer queries and updates to complete common operations.

In general, use embedded data models when:

- you have "contains" relationships between entities.
- you have one-to-many relationships between entities. In these relationships the "many" or child documents always appear with or are viewed in the context of the "one" or parent documents.

In general, embedding provides better performance for read operations, as well as the ability to request and retrieve related data in a single database operation. Embedded data models make it possible to update related data in a single atomic write operation.

With a relational database, you store rows in a table. Each table has a strictly defined schema specifying which columns and types are permitted. If any row in a table needs an extra field, you have to alter the table explicitly. MongoDB groups documents into collections, containers that don't impose any sort of schema. In theory, each document in a collection can have a completely different structure; in practice, a collection's document will be relatively uniform.

```
{
  _id: ObjectID('4bd9e8e17cefd644108961bb'),
  title: 'Adventures in Databases',
  url: 'http://example.com/databases.txt',
  author: 'msmith',
  vote_count: 20,
  tags: ['databases', 'mongodb', 'indexing'],
  image: {
    url: 'http://example.com/db.jpg',
    caption: 'A database.',
    type: 'jpg',
    size: 75381,
    data: 'Binary'
  },
  comments: [
    {
      user: 'bjones',
      text: 'Interesting article.'
    },
    {
      user: 'sverch',
      text: 'Color me skeptical!'
    }
  ]
}
```

Figure 1: A JSON document.

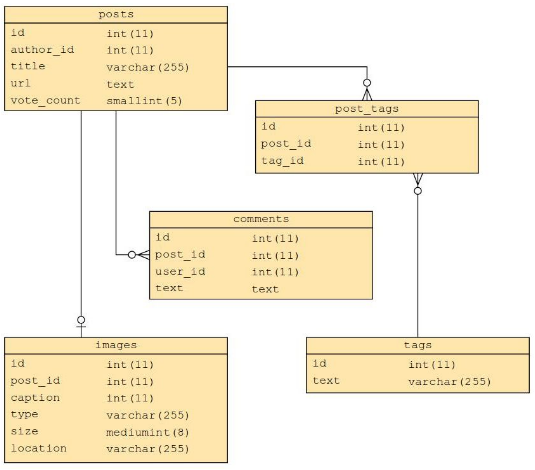

Figure 2: Representing the various one-to-many relationships in a relational database would require multiple tables.

5.2 Inserting Documents

To insert the a book in your collection issue the folliwing:

```
db.books.insert( { name: 'Ulysses', publication_year: 1922 } )
```

The resulting document will also include an id:

```
{ '_id' : ObjectId('5063114bd386d8fadbd6b004'), 'name' : 'Ulysses', 'publication_year'
```

5.3 Querying Documents

To search for this document you can issue the following queries:

If you know the _id:

```
db.books.find( { _id: ObjectId('5063114bd386d8fadbd6b004') } )
```

If you want to search by name:

```
db.books.find( { name: 'Ulysses' } )
```

If you are aware of when the book was published:

```
db.books.find( { publication_year: { $gte: 1920, $lt: 1925 } } )
```

5.4 Updating Documents

If you want to update a document you can issued the following query:

```
db.books.update(
   { name: 'Ulysses' },
   {
      name: 'Ulysses',
      publication_year: 1922,
      author: 'James Joyce'
   },
   { upsert: true }
)
```

Note, that this will update the entire document. Using *upsert* you make sure that the document will be inserted if it does not exist.

If you want to replace fields you can use $inc and $set:

```
db.books.update(
   { name: 'Ulysses' },
   {
     $inc: { publication_year: 1 },
     $set: {
       author: 'Joyce, James'
     }
```

```
    }
)
```

If you want to remove a field you can use $unset.

```
db.books.update( { name: 'Ulysses' }, { $unset: { author: 1 } } )
```

5.5 Sharding

A sharded cluster consists of shards, mongos routers, and config servers.

To allow for partitioning of an individual collection, MongoDB defines the idea of a chunk, which is a logical grouping of documents, based on the values of a predetermined field or set of fields called a shard key. It's the user's responsibility to choose the shard key.

MongoDB's sharding is range-based; this means that each "chunk" represents a range of shard keys. When MongoDB looks at a document to determine what chunk it belongs to, it first extracts the values for the shard key and then finds the chunk whose shard key range contains the given shard key values.

- Shards store the application data. In a sharded cluster, only the mongos routers or system administrators should be connecting directly to the shards. Like an unsharded deployment, each shard can be a single node for development and testing, but should be a replica set in production.
- mongos routers cache the cluster metadata and use it to route operations to the correct shard or shards.
- Config servers persistently store metadata about the cluster, including which shard has what subset of the data.

5.6 Limitations of MongoDB

- No referential integrity
- High degree of denormalization means updating something in many places instead of one
- Lack of predefined schema is a double edged sword
  - You must have a model in your app
  - Objects within a collection can be completely inconsistent in their fields

5.7 Resources

You can find a curated list of MongoDB resources, libraries, tools and applications in the following URL: `https://github.com/ramnes/awesome-mongodb`