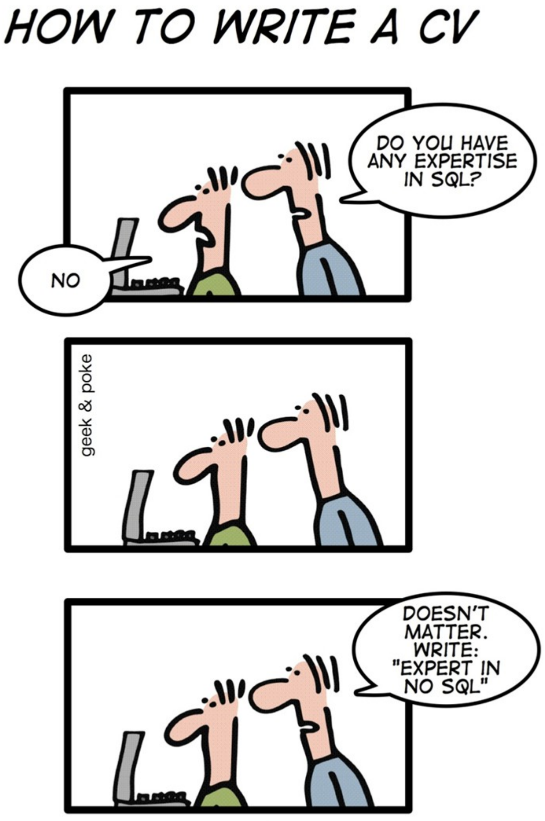
Sistemi NoSQL

Corso di Basi di Dati
Corso di Laurea in Informatica

Kristian Reale
Dipartimento di Matematica
e Informatica
DLVSystem S.r.l.

NoSQL!

NoSQL databases are currently a hot topic in some parts of computing, with over a hundred different NoSQL databases.


HOW TO WRITE A CV
DO YOU HAVE ANY EXPERTISE IN SQL?
NO
geek & poke
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"

Leverage the NoSQL boom

RDBMS Characteristics

- Data stored in columns and tables
- Relationships represented by data
- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Transactions
- Abstraction from physical layer
  - Conceptual Schema and Logical Schema

No SQL?

- NoSQL stands for:
  - No Relational
  - No RDBMS
  - **Not Only SQL**
- NoSQL is an **umbrella term** for **all databases** and data stores that don’t follow the RDBMS principles
  - A class of products
  - A collection of several (related) concepts about data storage and manipulation
  - Often related to large data sets

Where does NoSQL come from?

- Non-relational DBMSs are not new
- But NoSQL represents a new incarnation
  - Due to massively scalable Internet applications
  - Based on distributed and parallel computing
- Development
  - Starts with Google
  - First research paper published in 2003
  - Continues also thanks to Lucene's developers/Apache (Hadoop) and Amazon (Dynamo)
  - Then a lot of products and interests came from Facebook, Netflix, Yahoo, eBay, Hulu, IBM, and many more

NoSQL and Big Data

- NoSQL comes from Internet, thus it is often related to the “big data” concept
- How much big are “big data”?
  - Over few terabytes **enough** to start spanning multiple storage units
- Challenges
  - Efficiently **storing and accessing large amounts of data is difficult**, even more considering fault tolerance and backups
  - Manipulating large data sets involves running **immensely parallel processes**
  - Managing continuously *evolving schema* and metadata for *semi-structured and un-structured* data is **difficult**

How did we get here?

- Explosion of social media sites (Facebook, Twitter) with large data needs
- Rise of cloud-based solutions such as Amazon S3 (simple storage solution)
- Just as moving to dynamically-typed languages (Python, Ruby, Groovy), a shift to dynamically-typed data with frequent schema changes
- Open-source community

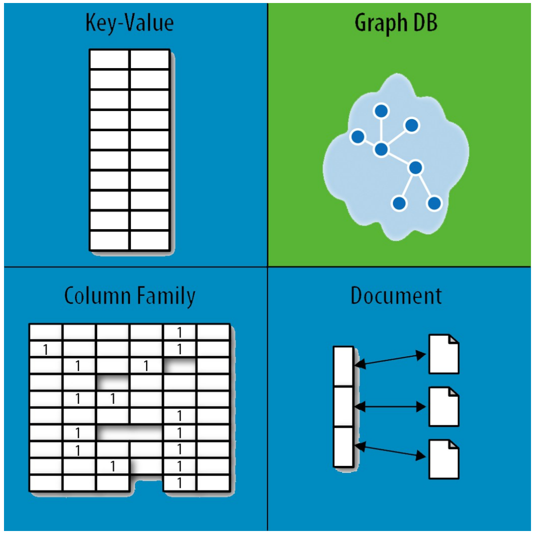
Why are RDBMS not suitable for Big Data

- The context is Internet
- RDBMSs assume that data are
  - Dense
  - Largely uniform (structured data)
- Data coming from Internet are
  - Massive and sparse
  - Semi-structured or unstructured
- With massive sparse data sets, the typical storage mechanisms and access methods get stretched

NoSQL Distinguishing Characteristics

- Large data volumes
  - Google's "big data"
- Scalable replication and distribution
  - Potentially thousands of machines
  - Potentially distributed around the world
- Queries need to return answers quickly
- Mostly query, few updates
- Asynchronous Inserts & Updates
- Schema-less

NoSQL Database Types



Discussing NoSQL databases is complicated because there are a variety of types:

- **Sorted ordered Column Store**
  - Optimized for queries over large datasets, and store columns of data together, instead of rows
- **Document databases:**
  - pair each key with a complex data structure known as a document.
- **Key-Value Store :**
  - are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value.
- **Graph Databases :**
  - are used to store information about networks of data, such as social connections.

Document Databases (Document Store)

- **Documents**
  - Loosely structured sets of key/value pairs in documents, e.g., XML, JSON, BSON
  - Encapsulate and encode data in some standard formats or encodings
  - Are addressed in the database via a unique key
  - Documents are treated as a whole, avoiding splitting a document into its constituent name/value pairs
- Allow documents retrieving by keys or contents
- Notable for:
  - **MongoDB** (used in FourSquare, Github, and more)
  - **CouchDB** (used in Apple, BBC, Canonical, Cern, and more)

Document Databases (Document Store)

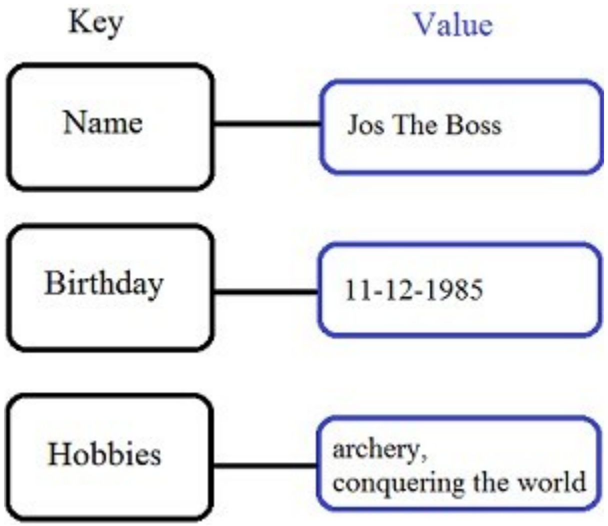
- The central concept is the notion of a "document“ which **corresponds to a row** in RDBMS.
- A document comes in some standard formats like JSON (BSON).
- Documents are addressed in the database via a **unique *key*** that represents that document.
- The database offers an API or query language that retrieves documents based on their contents.
- Documents are schema free, i.e., different documents can have structures and schema that differ from one another. (An RDBMS requires that each row contain the same columns.)

Document Databases, JSON

```
{
        _id: ObjectId("51156a1e056d6f966f268f81"),
        type: "Article",
        author: "Derick Rethans",
        title: "Introduction to Document Databases with MongoDB",
        date: ISODate("2013-04-24T16:26:31.911Z"),
        body: "This arti…"
},
{
        _id: ObjectId("51156a1e056d6f966f268f82"),
        type: "Book",
        author: "Derick Rethans",
        title: "php|architect's Guide to Date and Time Programming with PHP",
        isbn: "978-0-9738621-5-7"
}
```

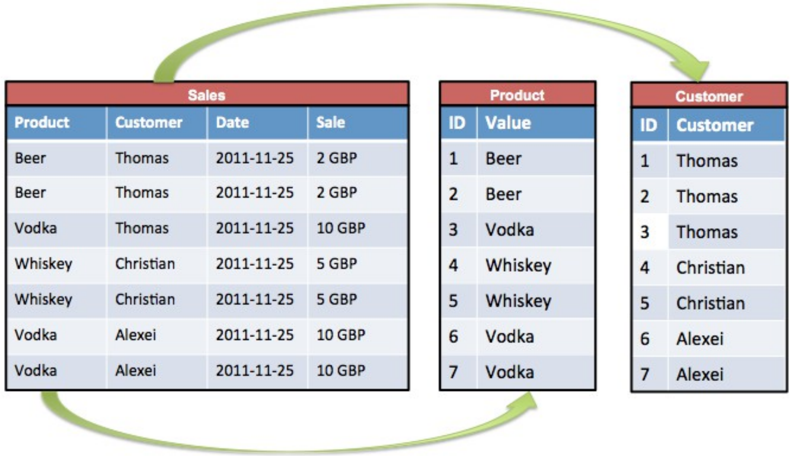
Key/Value stores

- Store data in a schema-less way
- Store data as maps
  - HashMaps or associative arrays
  - Provide a very efficient average running time algorithm for accessing data
- Notable for:
  - **Couchbase** (Zynga, Vimeo, NAVTEQ, ...)
  - **Redis** (Craiglist, Instagram, StackOverfow, flickr, ...)
  - **Amazon Dynamo** (Amazon, Elsevier, IMDb, ...)
  - **Apache Cassandra** (Facebook, Digg, Reddit, Twitter,...)
  - **Voldemort** (LinkedIn, eBay, ...)
  - **Riak** (Github, Comcast, Mochi, ...)



Sorted Ordered Column-Oriented Stores

- Data are stored in a **column-oriented** way
  - Data efficiently stored
  - Avoids consuming space for storing nulls
  - Columns are grouped in column-families
  - Data isn’t stored as a single table but is stored by column families
  - Unit of data is a set of key/value pairs
    - Identified by “row-key”
    - Ordered and sorted based on row-key
- Notable for:
  - **Google's Bigtable** (used in all Google's services)
  - **HBase** (Facebook, StumbleUpon, Hulu, Yahoo!, ...)
  - **MariaDB Columnstore**



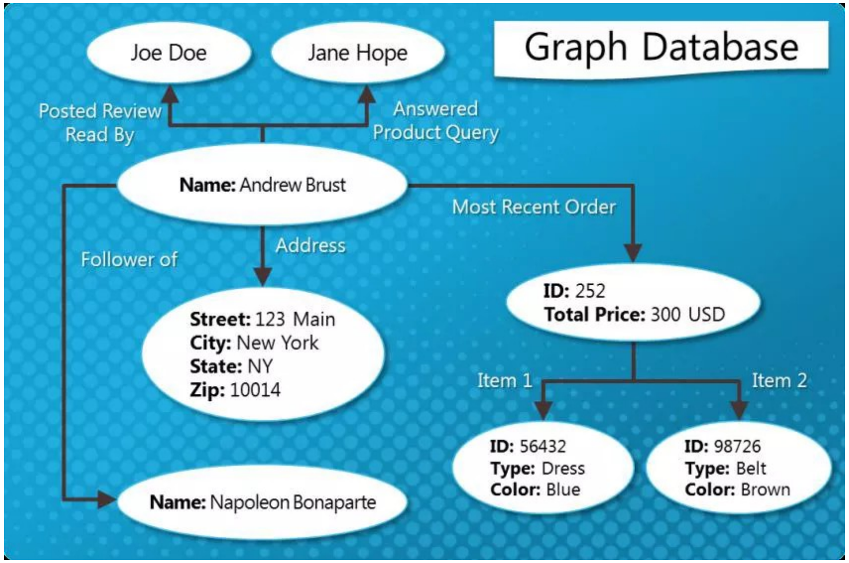
Sales			
Product	Customer	Date	Sale
Beer	Thomas	2011-11-25	2 GBP
Beer	Thomas	2011-11-25	2 GBP
Vodka	Thomas	2011-11-25	10 GBP
Whiskey	Christian	2011-11-25	5 GBP
Whiskey	Christian	2011-11-25	5 GBP
Vodka	Alexei	2011-11-25	10 GBP
Vodka	Alexei	2011-11-25	10 GBP

Product	
ID	Value
1	Beer
2	Beer
3	Vodka
4	Whiskey
5	Whiskey
6	Vodka
7	Vodka

Customer	
ID	Customer
1	Thomas
2	Thomas
3	Thomas
4	Christian
5	Christian
6	Alexei
7	Alexei

Graph Databases

- Graph-oriented
- Everything is stored as an edge, a node or an attribute.
- Each node and edge can have any number of attributes.
- Both the nodes and edges can be labelled.
- Labels can be used to narrow searches.



Dealing with Big Data and Scalability

- Issues with scaling up when the dataset is just too big
- **RDBMS were not designed to be distributed**
- Traditional DBMSs are best designed to run well on a “single” machine
    - Larger volumes of data/operations requires to upgrade the server with faster CPUs or more memory known as ‘scaling up’ or ‘Vertical scaling’
- NoSQL solutions **are designed to run on clusters or multi-node** database solutions
    - Larger volumes of data/operations requires to add more machines to the cluster, Known as ‘**scaling out**’ or ‘**horizontal scaling**’
    - Different approaches include:
        - Master-worker
        - Sharding (partitioning)

Scaling RDBMS

- Master-Worker
  - All writes are written to the master. All reads performed against the replicated worker databases
    - Reading is more frequent than writing
  - Critical reads may be incorrect as writes may not have been propagated down
  - Large data sets can pose problems as master needs to duplicate data to
- Sharding
  - Any DB distributed across multiple machines needs to know in what machine a piece of data is stored or must be stored
  - A sharding system makes this decision for each row, using its key

Performance

- There is no perfect NoSQL database
- Every database has its advantages and disadvantages
  - Depending on the type of tasks (and preferences) to accomplish
- NoSQL is a set of concepts, ideas, technologies, and software dealing with
  - Big data
  - Sparse un/semi-structured data
  - High horizontal scalability
  - Massive parallel processing
- Different applications, goals, targets, approaches need different NoSQL solutions

Where would I use it?

- Where would I use a NoSQL database?
- Do you have somewhere a large set of uncontrolled, unstructured, data that you are trying to fit into a RDBMS?
  - Log Analysis
  - Social Networking Feeds (many firms hooked in through Facebook or Twitter)
  - External feeds from partners
  - Data that is not easily analyzed in a RDBMS such as time-based data
  - Large data feeds that need to be massaged before entry into an RDBMS

First example:

What is MongoDB?

- Developed by 10gen
  - Founded in 2007
- A document-oriented, NoSQL database
  - Hash-based, *schema-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

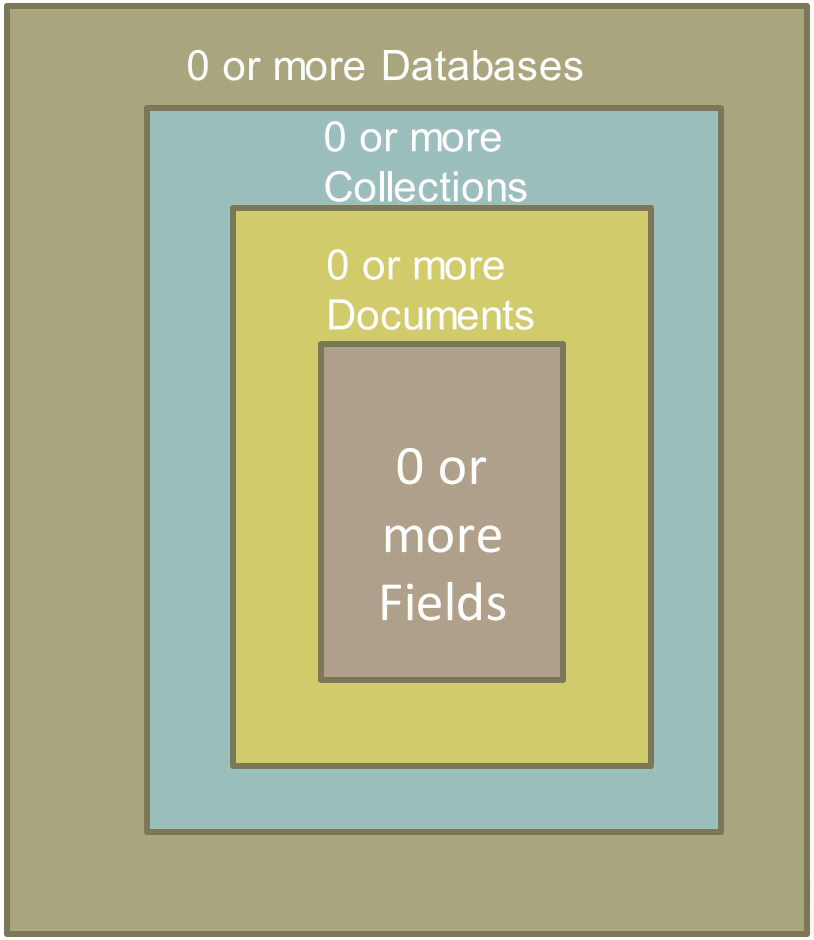
- Dynamic schema
  - No DDL
- Document-based database
- Secondary indexes
- Query language via an API
- Atomic writes and fully-consistent reads
  - If system configured that way
- Master-slave replication with automated failover (replica sets)
- Built-in horizontal scaling via automated range-based 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: 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

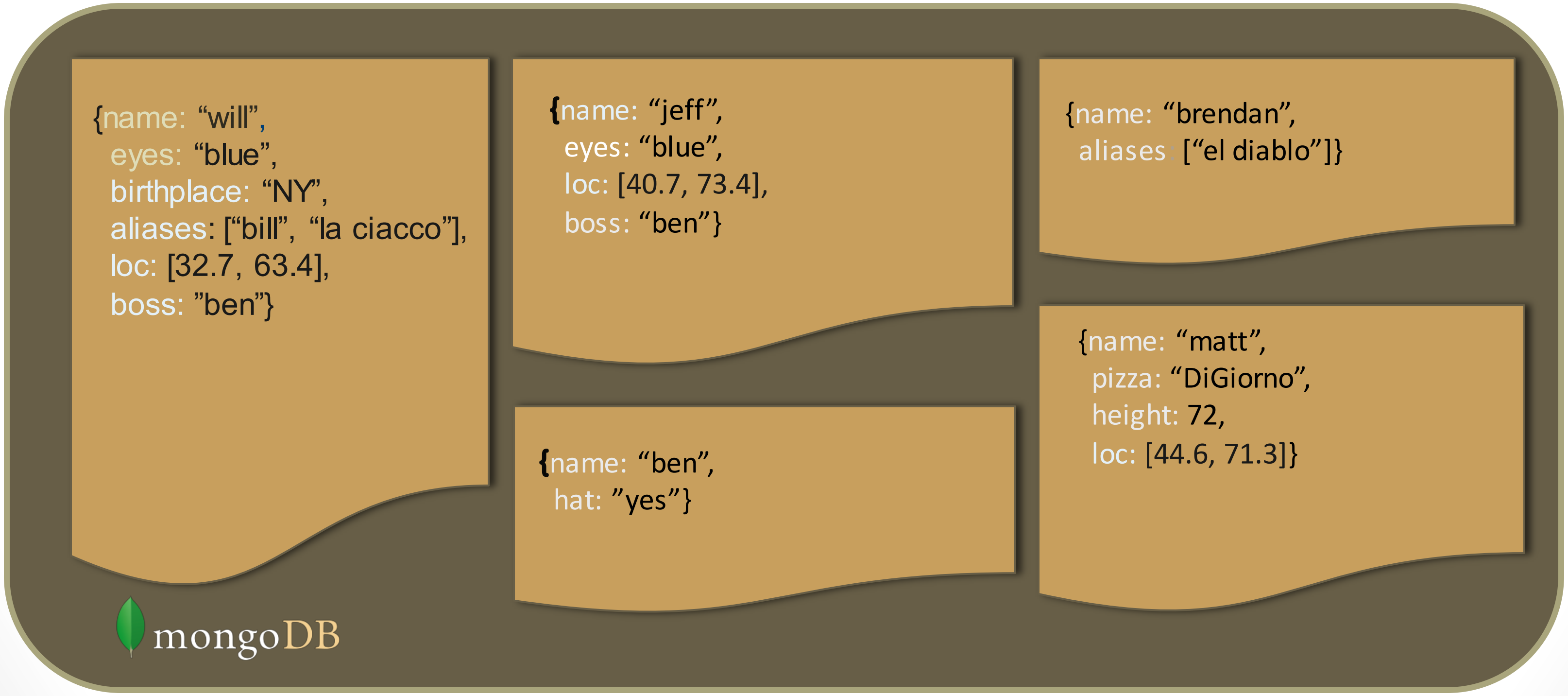
Embedded Document ?

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



```
{name: “will”,
  eyes: “blue”,
  birthplace: “NY”,
  aliases: [“bill”, “la ciacco”],
  loc: [32.7, 63.4],
  boss: ”ben”}
```

```
{name: “jeff”,
  eyes: “blue”,
  loc: [40.7, 73.4],
  boss: “ben”}
```

```
{name: “brendan”,
  aliases: [“el diablo”]}
```

```
{name: “ben”,
  hat: ”yes”}
```

```
{name: “matt”,
  pizza: “DiGiorno”,
  height: 72,
  loc: [44.6, 71.3]}
```

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: var PartsCursor = 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 loginal 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/
})
```

SQL vs. Mongo DB entities

My SQL

```
START TRANSACTION;
INSERT INTO contacts VALUES
   (NULL, ‘joeblow’);
INSERT INTO contact_emails
VALUES
   ( NULL, ”joe@blow.com”,
      LAST_INSERT_ID() ),
   ( NULL,
“joseph@blow.com”,
      LAST_INSERT_ID() );
COMMIT;
```

Mongo DB

```
db.contacts.save( {
    userName: “joeblow”,
    emailAddresses: [
      “joe@blow.com”,
      “joseph@blow.com” ] }
);
```

Similar to IDS from the 70’s
Bachman’s brainchild
DIFFERENCE:
MongoDB separates physical structure from logical structure

Designed to deal with large &distributed

Aggregated functionality

Aggregation framework provides SQL-like aggregation functionality

- Pipeline documents from a collection pass through an aggregation pipeline, which transforms these objects as they pass through
- Expressions produce output documents based on calculations performed on input documents
- Example db.**parts**.aggregate ({$group : {_id: type, totalquantity : { $sum: quanity} } })

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

LIVE DEMO

Querying sensors data with MongoDB