NoSQL and MongoDB

Prepared by referencing the following

https://docs.mongodb.com/manual/tutorial/

- A database Management System provides the mechanism to store and retrieve the data. There are different kinds of database Management Systems:
  1. RDBMS (Relational Database Management Systems)
  2. OLAP (Online Analytical Processing)
  3. NoSQL (Not only SQL)

introduction to NoSQL

- A **NoSQL** originally referring to non SQL or non relational is a database that provides a mechanism for storage and retrieval of data. This data is modeled in means other than the tabular relations used in relational databases.
- NoSQL databases are different than relational databases like MySql. In relational database you need to create the table, define schema, set the data types of fields etc before you can actually insert the data. In NoSQL you don't have to worry about that, you can insert, update data without creating schema.

NoSQL

NoSQL is a non-relational DMS

Does not require a fixed schema, avoids joins, and is easy to scale.

The purpose of using a NoSQL database is for distributed data stores

NoSQL is used for Big data and real-time web apps

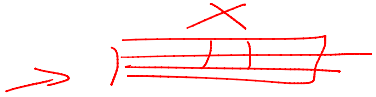
For example, companies like Twitter, Facebook, Google collect terabytes of user data per day

NoSQL database stands for "Not Only SQL" or "Not SQL.“

Traditional RDBMS uses SQL syntax to store and retrieve data for further insights

NoSQL can store structured, semi-structured, unstructured and polymorphic data (in one collection we have many versions of document schema)

Features of NoSQL



Non-relational

- NoSQL databases never follow the relational model
- Never provide tables with flat fixed-column records
- Work with self-contained aggregates or BLOBs
- Doesn't require object-relational mapping and data normalization
- No complex features like query languages, query planners, referential integrity joins, ACID

Schema-free

- NoSQL databases are either schema-free or have relaxed schemas
- Do not require any sort of definition of the schema of the data
- Offers heterogeneous structures of data in the same domain

Types of NoSQL Databases

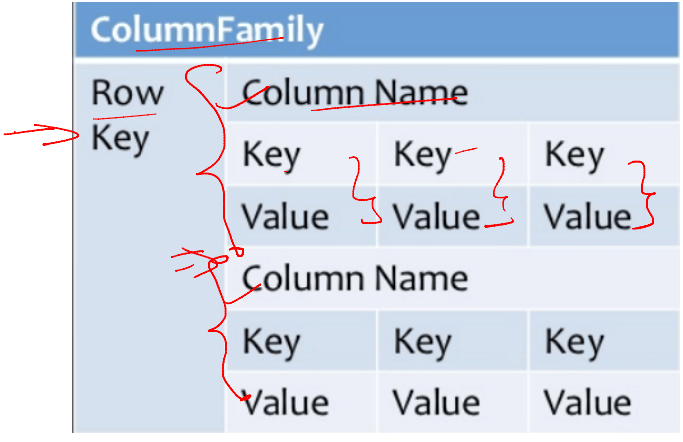
- Key-value Pair Based
- Column-oriented Graph
- Graphs based
- Document-oriented = MongoDB

- **Types of NoSQL database:**
  Types of NoSQL databases and the name of the databases system that falls in that category are:
- MongoDB falls in the category of NoSQL document based database.
- **Key value store:** Memcached, Redis, Coherence
- **Tabular:** Hbase, Big Table, Accumulo
- **Document based:** MongoDB, CouchDB, Cloudant
- Graph based: AllegroGraph, Amazon Neptune etc

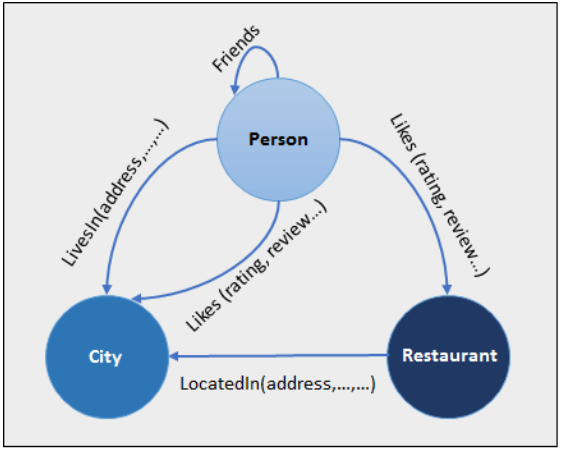
Key Value Pair Based

Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175cm
Weight	77kg

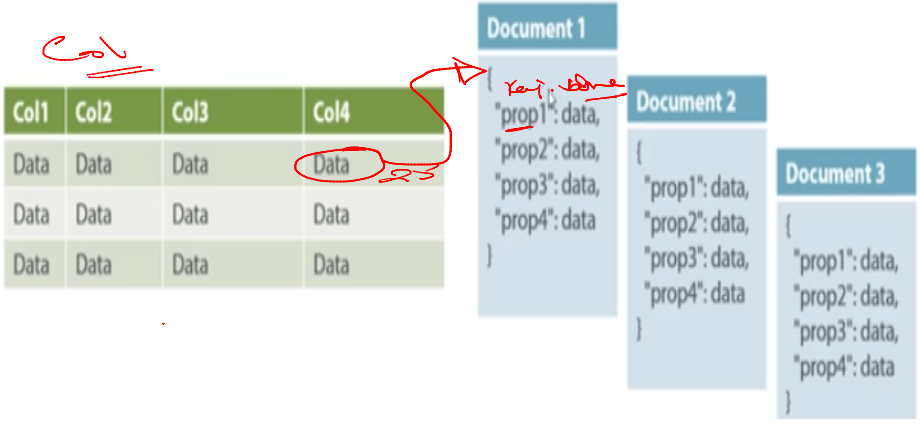
Column-oriented Graph



Graphs based



Document-oriented



SQL	NOSQL
RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)	Non-relational or distributed database system.
These databases have fixed or static or predefined schema	They have have dynamic schema
These databases are not suited for hierarchical data storage.	These databases are best suited for hierarchical data storage.
These databases are best suited for complex queries	These databases are not so good for complex queries
Verticlly Scalable	Horizontally scalable

Difference between RDBMS and NoSql

RDBMS

- **Stands for Relational Database Management System**
- It is completely a structured way of storing data.
- The amount of data stored in RDBMS depends on physical memory of the system or in other words it is vertically scalable.
- In RDBMS schema represents logical view in which data is organized and tells how the relation are associates.
- It is a mixture of open and closed development models, like oracle, apache and so on.
- RDBMS databases are table based databases This means that SQL databases represent data in form of tables which consists of n number of rows of data
- RDBMS have predefined schemas.
- For defining and manipulating the data RDBMS use structured query language i.e. SQL which is very powerful.
- **RDBMS database examples:** MySql, Oracle, Sqlite, Postgres and MS-SQL.
- RDBMS database is well suited for the complex queries as compared to NoSql.
- If we talk about the type of data then RDBMS are not best fir for hierarchical data storage
- **Scalability:** RDBMS database is vertically scalable so to manage the increasing load by increase in CPU, RAM, SSD on a single server.
- RDBMS is best suited for high transactional based application and its more stable and promise for the atomicity and integrity of the data.
- RDBMS support large scale deployment and get support from there vendors.
- **Properties:** ACID properties(Atomicity, Consistency, Isolation, Durability).

NoSql

- **Stands for Not Only SQL**
- It is completely a unstructured way of storing data.
- While in Nosql there is no limit you can scale it horizontally.
- Work on only open source development models.
- NoSQL databases are document based, key-value pairs, graph databases or wide-column stores.whereas NoSQL databases are the collection of key-value pair, documents, graph databases or wide-column stores which do not have standard schema definitions which it needs to adhered to.
- NoSql have dynamic schema with the unstructured data.
- It uses UnQL i.e. unstructured query language and focused on collection of documents and vary from database to database.
- **NoSQL database examples:** MongoDB, BigTable, Redis, RavenDb, Cassandra, Hbase, Neo4j and CouchDb
- NoSql is note well suited for complex queries on high level it dose not have standard interfaces to perform that queries.
- NoSql is best bit for hierarchical data storage because it follows the key-value pair way of data similar to JSON. Hbase is the example for the same.
- **Scalability:** as we know Nosql database is horizontally scalable so to handle the large traffic you can add few servers to support that.
- NoSql is still rely on community support and for large scale NoSql deployment only limited experts are available.
- **Properties:** Follow Brewers CAP theorem(Consistency, Availability and Partition tolerance).

- **Advantages of NoSQL:**
  There are many advantages of working with NoSQL databases such as MongoDB and Cassandra. The main advantages are high scalability and high availability.
- **High scalability –**
  NoSQL database use sharding for horizontal scaling. Partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved is sharding . Vertical scaling means adding more resources to the existing machine whereas horizontal scaling means adding more machines to handle the data. Vertical scaling is not that easy to implement but horizontal scaling is easy to implement. Examples of horizontal scaling databases are MongoDB, Cassandra etc. NoSQL can handle huge amount of data because of scalability, as the data grows NoSQL scale itself to handle that data in efficient manner.
- **High availability –**
  Auto replication feature in NoSQL databases makes it highly available because in case of any failure data replicates itself to the previous consistent state.

- **Disadvantages of NoSQL:**
  NoSQL has the following disadvantages.
- **Narrow focus –**
  NoSQL databases have very narrow focus as it is mainly designed for storage but it provides very little functionality. Relational databases are a better choice in the field of Transaction Management than NoSQL.
- **Open-source –**
  NoSQL is open-source database. There is no reliable standard for NoSQL yet. In other words two database systems are likely to be unequal.
- **Management challenge –**
  The purpose of big data tools is to make management of a large amount of data as simple as possible. But it is not so easy. Data management in NoSQL is much more complex than a relational database. NoSQL, in particular, has a reputation for being challenging to install and even more hectic to manage on a daily basis.
- **GUI is not available –**
  GUI mode tools to access the database is not flexibly available in the market.
- **Backup –**
  Backup is a great weak point for some NoSQL databases like MongoDB. MongoDB has no approach for the backup of data in a consistent manner.
- **Large document size –**
  Some database systems like MongoDB and CouchDB store data in JSON format. Which means that documents are quite large (BigData, network bandwidth, speed), and having descriptive key names actually hurts, since they increase the document size.

Introduction to MongoDB

- **MongoDB** is a cross-platform open-source document-oriented **database** program
- **MongoDB** database's has the ability to scale up with ease and hold very large amounts of data.
- **MongoDB** stores documents in collections within databases.
- **MongoDB** uses JSON-like documents
- In MongoDB, each document stored in a collection requires a unique _id field that acts as a primary key.
- BSON is a binary serialization format used to store documents on MongoDB

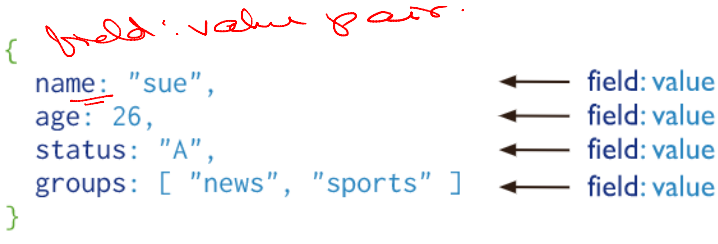
SQL to MongoDB Mapping Char

SQL Terms/Concepts	MongoDB Terms/Concepts
database	database
table	collection
row	document or BSON document
column	field
index	index
table joins	$lookup, embedded documents
primary key	primary key
Specify any unique column or column combination as primary key.	In MongoDB, the primary key is automatically set to the _id field.

Database executables

	MongoDB	MySQL	Oracle	Informix	DB2
Database Server	mongod	mysqld	oracle	IDS	DB2 Server
Database Client	mongo	mysql	sqlplus	DB-Access	DB2 Client

- Document Database
- A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents



- The advantages of using documents are:
- Documents (i.e. objects) correspond to native data types in many programming languages.
- Embedded documents and arrays reduce need for expensive joins.
- Dynamic schema supports fluent polymorphism.

- **Collections/Views/On-Demand Materialized Views**
- MongoDB stores documents in collections. Collections are analogous to tables in relational databases.
- In addition to collections, MongoDB supports:
- Read-only Views (Starting in MongoDB 3.4)
- On-Demand Materialized Views (Starting in MongoDB 4.2).


MongoDB Web Shell
Click to connect
Connecting...
MongoDB shell version v4.2.0
connecting to: mongodb://127.0.0.1:27017/?
authSource=admin&compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("007c14c9-a959-4272-a6a0-1cbd81945877") }
MongoDB server version: 4.2.0
type "help" for help
>>> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
>>> show collections
>>>

MongoDB CRUD Operations

C-create/insert
R-query
U- update
D-delete

Create or insert operations add new documents to a collection.
If the collection does not currently exist, insert operations will create the collection.
MongoDB provides the following methods to insert documents into a collection:

- db.collection.insertOne() *New in version 3.2*
- db.collection.insertMany() *New in version 3.2*

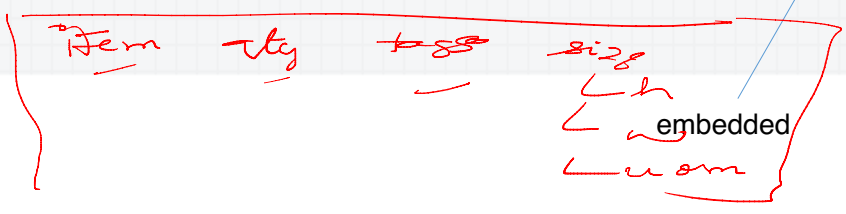
Insert a Single Document

`db.collection.insertOne()` inserts a *single* document into a collection.

The following example inserts a new document into the `inventory` collection. If the document does not specify an `_id` field, MongoDB adds the `_id` field with an ObjectId value to the new document. See Insert Behavior.

copy

```
db.inventory.insertOne(
   { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } }
)
```



Insert Multiple Documents

New in version 3.2.

`db.collection.insertMany()` can insert *multiple* documents into a collection. Pass an array of documents to the method.

The following example inserts three new documents into the `inventory` collection. If the documents do not specify an `_id` field, MongoDB adds the `_id` field with an ObjectId value to each document. See Insert Behavior.

copy

```
db.inventory.insertMany([
  { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } },
  { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },
  { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm"
])
```

Query documents

Select All Documents in a Collection

- To select all documents in the collection, pass an empty document as the query filter parameter to the **find method**. The query filter parameter determines the select criteria:

```
db.inventory.find({})
```

- This is equivalent to ***select * from inventory***

Specify Equality Condition

To specify equality conditions, use <field>:<value> expressions in the query filter document

db.inventory.find ({ **status : "D"** })

This is equivqlent to
Select * from inventory where status="D";

Specify Conditions Using Query Operators

The following example retrieves all documents from the inventory collection where status equals **either "A" or "D"**

```
db.inventory.find ( { status :{ $in : [ “A” , ”D” ] } } )
```

You can express it using $or operator also

```
db.inventory.find ( { status :{ $or : [ “A” , ”D” ] } } )
```

Specify AND Conditions

A compound query can specify conditions for more than one field in the collection's documents. Implicitly, a logical AND conjunction connects the clauses of a compound query so that the query selects the documents in the collection that match all the conditions.

The following query finds all the documents in the inventory whose status is "D" and quantity is less than 30

```
db.inventory.find ( {  status :"D", qty: { $lt:30 } } )
```

Specify OR Conditions

Using the `$or` operator, you can specify a compound query that joins each clause with a logical OR conjunction so that the query selects the documents in the collection that match at least one condition.

The following example retrieves all documents in the collection where the `status` equals `"A"` or `qty` is less than (`$lt`) `30`:

parameter for 'OR'

```
db.inventory.find( { $or: [ { status: "A" }, { qty: { $lt: 30 } } ] } )
```

copy

O.R.

The operation corresponds to the following SQL statement:

```
SELECT * FROM inventory WHERE status = "A" OR qty < 30
```

copy

Specify AND as well as OR Conditions

In the following example, the compound query document selects all documents in the collection where the `status` equals `"A"` and *either* `qty` is less than (`$lt`) `30` *or* `item` starts with the character `p`:

```
db.inventory.find( {
     status: "A",
     $or: [ { qty: { $lt: 30 } }, { item: /^p/ } ]
} )
```

The operation corresponds to the following SQL statement:

```
SELECT * FROM inventory WHERE status = "A" AND ( qty < 30 OR item LIKE "p%")
```

NOTE:

MongoDB supports regular expressions `$regex` queries to perform string pattern matches.

Query on Embedded/Nested Documents

- Populate the collection

copy

```
db.inventory.insertMany( [
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
```

Match an Embedded/Nested Document

To specify an equality condition on a field that is an embedded/nested document, use the query filter document `{ <field>: <value> }` where `<value>` is the document to match.

For example, the following query selects all documents where the field `size` equals the document `{ h: 14, w: 21, uom: "cm" }`:

```
db.inventory.find( { size: { h: 14, w: 21, uom: "cm" } } )
```

Equality matches on the whole embedded document require an *exact* match of the specified `<value>` document, including the field order. For example, the following query does not match any documents in the `inventory` collection:

```
db.inventory.find(  { size: { w: 21, h: 14, uom: "cm" } }  )
```

Query on Nested Field

To specify a query condition on fields in an embedded/nested document, use dot notation (`"field.nestedField"`).

> NOTE:
> When querying using dot notation, the field and nested field must be inside quotation marks.

Specify Equality Match on a Nested Field

The following example selects all documents where the field `uom` nested in the `size` field equals `"in"`:

```
db.inventory.find( { "size.uom": "in" } )
```

copy

Specify Match using Query Operator

A query filter document can use the query operators to specify conditions in the following form:

```
{ <field1>: { <operator1>: <value1> }, ... }
```

copy

The following query uses the less than operator (`$lt`) on the field `h` embedded in the `size` field:

```
db.inventory.find( { "size.h": { $lt: 15 } } )
```

copy

Specify **AND** Condition

The following query selects all documents where the nested field `h` is less than `15`, the nested field `uom` equals `"in"`, and the `status` field equals `"D"`:

```
db.inventory.find( { "size.h": { $lt: 15 }, "size.uom": "in", status: "D" } )
```

copy

Query an Array

```
db.inventory.insertMany([
   { item: "journal", qty: 25, tags: ["blank", "red"], dim_cm: [ 14, 21 ] },
   { item: "notebook", qty: 50, tags: ["red", "blank"], dim_cm: [ 14, 21 ] },
   { item: "paper", qty: 100, tags: ["red", "blank", "plain"], dim_cm: [ 14, 21 ] },
   { item: "planner", qty: 75, tags: ["blank", "red"], dim_cm: [ 22.85, 30 ] },
   { item: "postcard", qty: 45, tags: ["blue"], dim_cm: [ 10, 15.25 ] }
]);
```

Match an Array

To specify equality condition on an array, use the query document `{ <field>: <value> }` where `<value>` is the exact array to match, including the order of the elements.

The following example queries for all documents where the field `tags` value is an array with exactly two elements, `"red"` and `"blank"`, in the specified order:

copy

```
db.inventory.find( { tags: ["red", "blank"] } )
```

If, instead, you wish to find an array that contains both the elements `"red"` and `"blank"`, without regard to order or other elements in the array, use the `$all` operator:

copy

```
db.inventory.find( { tags: { $all: ["red", "blank"] } } )
```

Query an Array for an Element

To query if the array field contains at least *one* element with the specified value, use the filter `{ <field>: <value> }` where `<value>` is the element value.

The following example queries for all documents where `tags` is an array that contains the string `"red"` as one of its elements:

```
db.inventory.find( { tags: "red" } )
```

copy

To specify conditions on the elements in the array field, use query operators in the query filter document:

```
{ <array field>: { <operator1>: <value1>, ... } }
```

copy

For example, the following operation queries for all documents where the array `dim_cm` contains at least one element whose value is greater than 25.

```
db.inventory.find( { dim_cm: { $gt: 25 } } )
```

copy

Query an Array with Compound Filter Conditions on the Array Elements

The following example queries for documents where the `dim_cm` array contains elements that in some combination satisfy the query conditions; e.g., one element can satisfy the greater than `15` condition and another element can satisfy the less than `20` condition, or a single element can satisfy both:

```
db.inventory.find( { dim_cm: { $gt: 15, $lt: 20 } } )
```

copy

Query for an Array Element that Meets Multiple Criteria

Use `$elemMatch` operator to specify multiple criteria on the elements of an array such that at least one array element satisfies all the specified criteria.

The following example queries for documents where the `dim_cm` array contains at least one element that is both greater than (`$gt`) `22` and less than (`$lt`) `30`:

copy

```
db.inventory.find( { dim_cm: { $elemMatch: { $gt: 22, $lt: 30 } } } )
```

Query for an Element by the Array Index Position

Using dot notation, you can specify query conditions for an element at a particular index or position of the array. The array uses zero-based indexing.

> **NOTE:**
> When querying using dot notation, the field and nested field must be inside quotation marks.

The following example queries for all documents where the second element in the array `dim_cm` is greater than `25`:

```
db.inventory.find( { "dim_cm.1": { $gt: 25 } } )
```

Query an Array by Array Length

Use the `$size` operator to query for arrays by number of elements. For example, the following selects documents where the array `tags` has 3 elements.

```
db.inventory.find( { "tags": { $size: 3 } } )
```

copy

operator : value.

Update methods

`db.collection.updateOne()`	Updates at most a single document that match a specified filter even thoug multiple documents may match the specified filter. *New in version 3.2.*
`db.collection.updateMany()`	Update all documents that match a specified filter. *New in version 3.2.*
`db.collection.replaceOne()`	Replaces at most a single document that match a specified filter even thou multiple documents may match the specified filter. *New in version 3.2.*
`db.collection.update()`	Either updates or replaces a single document that match a specified filter o updates all documents that match a specified filter. By default, the `db.collection.update()` method updates a **single** document. To update multiple documents, use the multi option.

Delete Methods

MongoDB provides the following methods to delete documents of a collection:

`db.collection.deleteOne()`	Delete at most a single document that match a specified filter even though mu documents may match the specified filter. *New in version 3.2.*
`db.collection.deleteMany()`	Delete all documents that match a specified filter. *New in version 3.2.*
`db.collection.remove()`	Delete a single document or all documents that match a specified filter.