**ANSWERS**

**ANSWER 1**

MongoDB is a NoSQL database program which uses JSON-like documents with schemas. It is free and open-source cross-platform database. MongoDB, top NoSQL database engine in use today, could be a good data storage alternative when analysing large volume data. The first thing that we are going to do is we are going to check some of the drivers available for the R Lang to connect with mongodb.

There are two R packages providing the interface with MongoDB, namely RMongo and rmongodb. While RMongo package is very straight-forward and user-friendly, it did take me a while to figure out how to specify a query with rmongodb package.

Example of rmongo:

library(RMongo)

mg1 <- mongoDbConnect('db')

print(dbShowCollections(mg1))

query <- dbGetQuery(mg1, 'test', "{'AGE': {'$lt': 10}, 'LIQ': {'$gte': 0.1}, 'IND5A': {'$ne': 1}}")

data1 <- query[c('AGE', 'LIQ', 'IND5A')]

summary(data1)

Example of rmongodb:

library(rmongodb)

mg2 <- mongo.create()

print(mongo.get.databases(mg2))

print(mongo.get.database.collections(mg2, 'db'))

buf <- mongo.bson.buffer.create()

mongo.bson.buffer.start.object(buf, 'AGE')

mongo.bson.buffer.append(buf, '$lt', 10)

mongo.bson.buffer.finish.object(buf)

mongo.bson.buffer.start.object(buf, 'LIQ')

mongo.bson.buffer.append(buf, '$gte', 0.1)

mongo.bson.buffer.finish.object(buf)

mongo.bson.buffer.start.object(buf, 'IND5A')

mongo.bson.buffer.append(buf, '$ne', 1)

mongo.bson.buffer.finish.object(buf)

query <- mongo.bson.from.buffer(buf)

cur <- mongo.find(mg2, 'db.test', query = query)

age <- liq <- ind5a <- NULL

while (mongo.cursor.next(cur)) {

  value <- mongo.cursor.value(cur)

  age   <- rbind(age, mongo.bson.value(value, 'AGE'))

  liq   <- rbind(liq, mongo.bson.value(value, 'LIQ'))

  ind5a <- rbind(ind5a, mongo.bson.value(value, 'IND5A'))

  }

mongo.destroy(mg2)

data2 <- data.frame(AGE = age, LIQ = liq, IND5A = ind5a)

summary(data2)

**ANSWER 2**

MongoDB is a document-oriented NoSQL database used for high volume data storage. MongoDB is a database which came into light around the mid-2000s. It falls under the category of a NoSQL database. Its key features are:

1. Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. The size and content of each document can be different from each other.
2. The document structure is more in line with how developers construct their classes and objects in their respective programming languages. Developers will often say that their classes are not rows and columns but have a clear structure with key-value pairs.
3. The rows (or documents as called in MongoDB) doesn't need to have a schema defined beforehand. Instead, the fields can be created on the fly.
4. The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.

Data storage hierarchy of MongoDB is as follows:

**Database:** In simple words, it can be called the physical container for data. Each of the databases has its own set of files on the file system with multiple databases existing on a single MongoDB server.

**Cursor** – This is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results

**Collection:** A group of database documents can be called a collection. The RDBMS equivalent to a collection is a table. The entire collection exists within a single database. There are no schemas when it comes to collections. Inside the collection, various documents can have varied fields, but mostly the documents within a collection are meant for the same purpose or for serving the same end goal.

**Document:** A set of key–value pairs can be designated as a document. Documents are associated with dynamic schemas. The benefit of having dynamic schemas is that a document in a single collection does not have to possess the same structure or fields. Also, the common fields in a collection’s document can have varied types of data.

**Field** - A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases.

**ANSWER 4**

The schema of a database describes the structure of the data to be stored. In a relational database, the schema defines its tables, its fields in each table and the relationships between each field and each table. The data stored needs to comply with the structure defined (tables, columns, data types and relations). So, every register in a table have the same number of columns and format.

Let us first define what a schema-less database is.

1. A schema-less database does not require conformation to a rigid schema (database, schema, data types, tables etc.) that one is required to live up to through the life of a system.
2. Does not enforce data type limitations on individual values pertaining to one single column type
3. Models the business usage and not a database schema, application or product.
4. Can store structured and unstructured data.
5. Eliminates the need to introduce additional layers (ORM layer) to abstract the relational model and expose it in an object-oriented format.

Every RDBMS database will possess a structure with respect to persisting the data. So, when the database needs to be modified, say a **column** of particular table needs to be modified (changing the datatype) or adding/removing a column from a table, we have to mention, what type of data this column will hold.

But MongoDB is a document-based database, which does not use the concept of tables and columns, instead of which it uses the concept of documents and collections. All the referential data with respect to different modules will be stored as one collection. More over the BSON data structure used by MongoDB can easily have varying sets of data and fields with different types.

When we say **schemaless**, we actually mean **dynamically typed schema**, as opposed to statically typed schemas as available in RDBMS(SQL) databases.

JSON is a completely schema free data structure, as opposed to XML which allows you to specify XSD if you need.

Whenever we store something in MongoDB, we know what we are storing, it’s a String, Int or a custom Object in JSON format, and as soon as we try to store it in a collection, MongoDB accepts it as a JSON formatted data.

And when you read somewhere that data migration is easy in NoSQL databases, it’s also not absolutely true. There are certain scenarios where document-based databases like MongoDB proves to be better, but there are scenarios where SQL based databases have the upper hand. For example: data processing while migrating, takes longer in case of NoSQL due to data duplication, while in a Normalised SQL database, minimal data duplication is there.

**ANSWER 5**

Syntax to create collection:

db.createCollection**(name, options)**

Creates a new collection or [view](https://docs.mongodb.com/manual/core/views/). Because MongoDB creates a collection implicitly when the collection is first referenced in a command, this method is used primarily for creating new collections that use specific options. For example, you use [db.createCollection()](https://docs.mongodb.com/manual/reference/method/db.createCollection/#db.createCollection) to create a [capped collection](https://docs.mongodb.com/manual/reference/glossary/#term-capped-collection), or to create a new collection that uses [document validation](https://docs.mongodb.com/manual/core/schema-validation/).

[db.createCollection()](https://docs.mongodb.com/manual/reference/method/db.createCollection/#db.createCollection) is a wrapper around the database command [create](https://docs.mongodb.com/manual/reference/command/create/#dbcmd.create).

Syntax to drop a collection:

db.collection.drop**(<options>)**

Removes a collection or [view](https://docs.mongodb.com/manual/core/views/) from the database. The method also removes any indexes associated with the dropped collection. The method provides a wrapper around the [drop](https://docs.mongodb.com/manual/reference/command/drop/#dbcmd.drop) command.

For Example:

Creating a database mydata with a collection hello.

* Create db mydata
* Create a collection hello:> db.createCollection("hello")
* Show all the collections there:> db.getCollectionNames()
* Insert some dummy data:> db.hello.insert({'a':'b'})
* Make sure it was inserted:> db.hello.find()
* Delete the collection and make sure it is not present any more:> db.hello.drop()