1) Explain, generally, what is meant by a NoSQL database.

Characteristics of NoSQL:

- **Non-relational**
  Which means that we don't divide the database into different tables of atomic values using normalization, as we do with SQL.
- **Mostly open-source**
  Almost all of what is characterized as NoSQL databases are open-source. There are a few exceptions, but they aren't relevant to us at the moment.
- **Cluster-friendly**
  Since there are no relations, NoSQL databases are easily scalable.
- **21st century web**
  All NoSQL databases come from the 21st century web culture.
- **Schema-less**

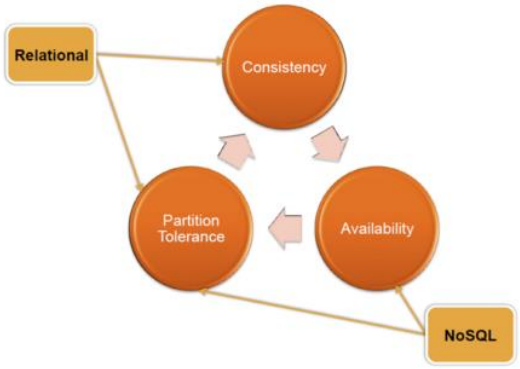
2) Explain Pros & Cons in using a NoSQL database like MongoDB as your data store, compared to a traditional Relational SQL Database like MySQL.

Pros:

- Simplicity of design
- Horizontal scaling
- Finer control over availability

Cons:

- Does not offer **ACID** guarantees:
  - **A**tomicity
  - **C**onsistency
  - **I**solation
  - **D**urability



Additionally the CAP theorem (model on the right) states:
It's theoretically impossible to have all 3 requirements met, so a combination of 2 must be chosen and this is usually the deciding factor in what technology is used.

Consistency:
All the servers in the system will have the same data so anyone using the system will get the same copy regardless of which server answers their request.

Availability:
The system will always respond to a request (even if it's not the latest data or consistent across the system or just a message saying the system isn't working).

Partition Tolerance
The system continues to operate as a whole even if Individual servers fail or can't be reached.

3) Explain how databases like MongoDB and redis would be classified in the NoSQL world.

MongoDB - Document

A document data model is a storage of documents, where each document is a complex data structure, usually represented in JSON. In a document model, you can retrieve whole documents or partial data of a document, update documents etc. And in contrast to key/value databases, document databases can see structure within the aggregate.
A document database has no schema (just like the other NoSQL databases), however you will need to use some kind of schema, in our case in the form of mongoose.

redis – Key/Value

Key/value store means that you have a key which is linked to a certain value in the database. The database doesn't know what the value is and it doesn't care. It could be an image, a complex document etc.
So Key-value databases like redis is like a hashmap but persistent in the disc.
It is also possible to store metadata about the value, which means that the difference between key-value databases and document databases isn't

4) Explain reasons to add a layer like Mongoose, on top on of a schema-less database like MongoDB.

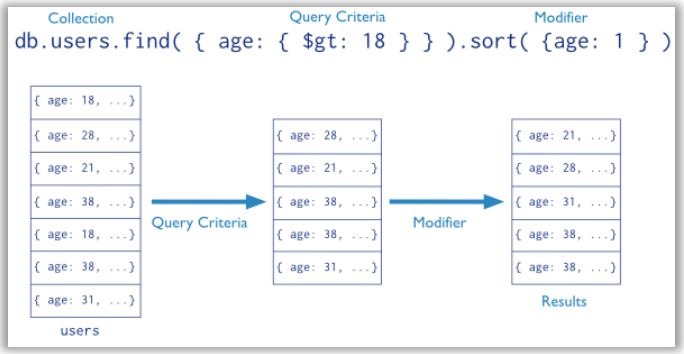
- Real data normally has some kind of structure, which we would like to mimic.
- Real data normally has types.
- To handle data as objects easier.

So it basically boils down to **being able to do more with less work**.

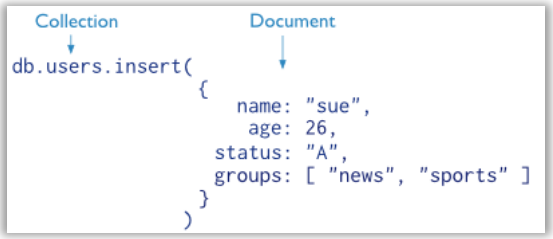
5) Explain, using relevant examples, the strategy for querying MongoDB.

In MongoDB a query targets a specific collection of documents. Queries specify criteria, or conditions, that identify the documents that MongoDB returns to the clients. A query may include a *projection* that specifies the fields from the matching documents to return. You can optionally modify queries to impose limits, skips, and sort orders.

Retrieving data:



Inserting data:



Documentation:
https://docs.mongodb.org/manual/core/crud-introduction/

6) Demonstrate, using a REST-API, how to perform all CRUD operations on a MongoDB

7) Explain the benefits from using Mongoose, and provide an example involving all CRUD operations

Mongoose provides a straight-forward, schema-based solution to modeling your application data and includes, out of the box:

- Schemas.
- Built-in type casting.
- Validation.
- Query building.
- Business logic hooks (middleware).

Retrieving data:

```
User.find(
  {'username' : 'Kurt Wonnegut'},
  function (err, users){
  if (!err){console.log(users);}
});
//LIKE 'Kurt' (regEx)
User.find(
  {'username' : /Kurt/i}, //Find
  function (err, users){
  if (!err){console.log(users);}
});
```

Inserting data:

```
var mongoose = require( 'mongoose' );
var User = mongoose.model("User"););

// create a new user
var newUser = User({
  username: 'Kurt Wonnegut',
  email: "kw@somewhere.dk"
});

// save the user
newUser.save(function(err) {
  if (err){
      throw err
  };
  console.log('User created!');
});
```

8) Explain how redis "fits" into the NoSQL world, and provide an example of how to use it.

redis is a Key/Value database, where we can store aggregates as the value and have a key or ID to access that data, but the database cannot see any structure within the aggregate. For more info see question 3.

Redis in terms of speed and persistence:

- The entire dataset needs to be able to exist in memory on the server to take advantage of the potential speed benefits.
- With respect to persistence, by default, redis snapshots the database to disk based on how many keys have changed. You configure it so that if X number of keys change, then save the database every Y seconds.

How it works:
Strings are the most basic kind of redis value. Redis Strings are binary safe, this means that a redis string can contain any kind of data.

Example:

```
set users:leto '{"name": leto, "planet": dune, "likes": ["spice"]}'
get users:leto
```

9) Explain, using a relevant example, how redis (or a similar) can increase scalability (drastic) for a server using server side sessions

10) Explain, using a relevant example, a full MEAN application including relevant test cases to test the REST-API (not on the production database)