1. How would you add a new movie to the document database? (You can write code if you want, or just describe it in words.) Compare this with the relational database.

   In the document database you would navigate to the part of the database which you wished to insert the movie into. (so navigate to the movies section) then define attributes of the movie and insert it.
   In a relational database you would add a new record and fill in the fields.
   IE: Insert into people (NI, name, DOB) values ('AQ340893L', 'John Smith', '09-04-1960');
   Overall, (for insertion) the main difference is that for the relational database the attributes are pre-defined, while for the document oriented database, you define when inserting the entity – and can insert multiple attributes of the same type.

2. How would you compute Bacon Numbers using the document database? Compare this with the graph database Neo4j.

   (a) Create a list (visited) and a queue (todo).

   (b) Navigate to the person Kevin Bacon. Add a new attribute called "bacon_number". Give Kevin Bacon the bacon_number 0.

   (c) Look which movies Kevin Bacon has acted in. Enqueue them onto the queue and add it onto the visited queue.

   (d) Dequeue the first movie and iterate through all the actors who have acted in that movie. For each actor,

      - search the people document for the actor.
      - Once the actor has been found: if they do not have a bacon number then create a bacon_number attribute and give them a bacon_number one larger than the Bacon Number assigned at the previous iteration.
      - Search the movies document: and for each movie; if the actor acted in it and the movie is not already in the visited list, enqueue it and add it to the visited list.

   (e) Repeat 2d until the queue is empty.

   (f) Now that the queue is empty, every actor who has a non-Null bacon_number has been given a bacon_number.

   In Neo4j; you would

   (a) Start at Kevin Bacon. Give Kevin Bacon the attribute bacon_number = 0.

   (b) Add every movie Kevin Bacon has starred in onto a stack.

   (c) For every movie on the stack; go to every plays_role node for someone who plays a role in that movie. Then navigate to the person.

   (d) If the person does not have a bacon_number then set their bacon_number to one larger than the largest set bacon_number and add every film which they have played a role in onto the stack. If they do have a bacon_number then pass.

   (e) Repeat 2c and 2d until an iteration does not give anyone a bacon_number.

   So it is obviously much more efficient to compute bacon_numbers in neo4j than in a document oriented database - this is because in graph oriented databases the edges link the nodes directly to each other and so queries about transistive closure are far more efficient.

3. Suppose that we are presented with only the document database containing movies and people. Could we use this database to help us reconstruct an Entity-Relationship model of the data? Could you even consider automating this process?

   In this case, you could use this database to reconstruct an Entity-Relationship model of the data. However, it would be difficult to create a schema to which **every** movie and person in the database conformed to. This strict enforcement could lead to lots of very small databases. IE movie_id as a key and every separate attribute in a different table.

   A more realistic approach would be to accept that some movies and people will have null fields and allow Null values in some places. In either case; although it is possible to recreate the original data, the schema would inevitably be very poor.

   You could not automate this process. The program would have no way of knowing what the keys were. IE it may decide that the combination of release date and director in this database happens to be unique and hence a composite key containing release date and director is an appropriate primary key (despite obviously not being the best choice). A fully automated version of the database would also be unable to tell whether entities should be foreign keys or not and how to normalise (for example, a poor choice could lead to release date coincidentally being a primary key of the people table. Should the release date of movies be a foreign key linking to the actors table? A fully automated version would be unable to tell (despite the fact that any human could see that it should not be).

   Overall, I don't think that you could automate reconstructing a (good or appropriate) relational database from a document oriented database.

1 2016 Paper 4 Question 6

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2016p4q6.pdf

This question deals with the variety of approaches to database design.

(a) What is meant by the term on-line transaction processing (OLTP)?

   Optimising a database for write and update queries. This involves creating minimal redundancy so that updates and writes and updates do not have to lock databases. This allows for more concurrent updates, removal of the possibility of race conditions, a reduction in the size of the database and faster overall write and update operations. However, often even simple queries require many joins. This means that reading from the database often takes a large amount of time and computing power. As a result, often databases with OLTP are often ONLY written and updated. Typically, data in databases optimised for OLTP is archived frequently so that the database only contains recent data.

(b) What is meant by the term on-line analytic processing (OLAP)?

   Optimising a database to deal with large amounts of complicated read and join queries compared to write and update queries.
   This usually means precomputing many operations (such as joins) and creating redundancy so that read operations take less time (ie indices to change lookups from O(n) operations to O(1)).
   This can often be at the cost of the ability to update the database at all. IE having one database which is only written and updated and then intermittently precomputing values to create a database optimised for OLAP.
   OLAP is usually used on either **very** large databases where even simple joins take too much time to be computed live or in systems where a real time response is required.

(c) Compare and contrast the approach to schema design for OLTP and OLAP databases.

OLTP databases are normalised and have very low redundancy and lots of foreign keys mapping to other databases. This enables them to be updated rapidly without the need to lock many records. However, many reads require join operations and hence take a long time. The schema is designed to enable writes and update operations with almost no reads.

OLAP databases are not normalised, have a lot of redundancy and few keys (since often databases are pre-joined – and this removes a key). They are optimised for reads.

(d) Compare OLAP with the so-called NoSQL approach to database design.

OLAP databases can only store structured data, while NoSQL is often used to store semi-structured or unstructured data. NoSQL is often used to store **very** large databases while OLAP databases are usually smaller (although still large).

NoSQL databases can support more complicated (ie graph oriented databases which are notorious for enabling complicated operations [around transistive closure] which would require an unmanageable joins in SQL). OLAP databases are optimised for reads and so also enable a lot of these operations – however the relational schema which they are based on does not enable this intrinsically.

Both NoSQL databases and OLAP databases often have a fairly "loose" schema which allows for redundancy and null fields.

NoSQL databases are easy to write to – while OLAP databases are very difficult to write to due to the amount of redundancy (and so usually are never written to).

(e) Give an example of a set of requirements whose solution would need to combine OLAP, OLTP and NoSQL. Describe an architecture integrating these elements in the system design.

This situation would arise when storing large amounts of data – which is often added to but has historic elements which are not (or are rarely) updated – and has a portion which is semi-structured or unstructured.

An example of this would be search engine indexing.

Websites are unstructured and so their data cannot be easily stored in a relational database. They would be best stored in a document-oriented database.

The information about websites (ie which links websites have to other websites, the traffic which passes through the websites, the times that users spend on the website or the data at which the website was last updated or created) would be better stored in a relational database; while the website itself could be stored in a NoSQL database (document-oriented).

However, much of this data will not change rapidly, changes do not need to be immediately reflected and using this data would require expensive join operations. So it would make more sense to have a OLTP database which is updated with new information (such as visits today, new changes etc). A OLAP database could be remade daily with the updated data. The NoSQL database could be used to store the structures of the websites themselves – Database Oriented databases are particularly suited to this since they match HTML well.