Question 1

A)i)

MATCH (a)-[:wrote]->(b)

WHERE a.name = “Mark Needham”

RETURN count(b)

ii)

MATCH (b) -[:sold|sold\_by]->(s)

WHERE b.name = “Introduction to Neo4J”

RETURN collect(s.name)

iii) match (r)-[:frequents]->(s)-[:located\_in]->(l) where r.name=”Mark” return s.name, l.name

iv)   
match (a)-[:wrote]->(b)-[:published\_by]->(p), (e)-[:edits]->(b)   
return p.name, a.name, count(b), collect(e.name)

v)   
match (r)-[:reads]->()-[:published\_by]->(p) <-[:works\_for]- (e)  
where r.name=”Mark” AND NOT EXISTS {  
match (a)-[:wrote]->()<-[:edits]-(e)  
}  
return e.name

Alternative: pls could someone check?

MATCH (e) - [:works\_for] -> (p) <- [:published\_by] - (b) <- [:reads] - (r: Reader{Name:”Mark”})

WHERE NOT EXISTS (e) - [:edits] -> (b)

RETURN e.name

B)

db.people.find({})

db.people.find({'address.postcode': 'SW35HF'})

db.people.aggregate([

{$geoNear: {

near: { type: "Point", coordinates: [45, 47] },

maxDistance: 1000

distanceField: 'calculated\_dist'

}}

])

Alternative:  
db.people.find({ “location”:  
{“$near”: [45,47], “$maxDistance”:1000}  
})

NB: The question here isn’t clear whether they want the user to find people who own both or own one of the two types so assumption should be stated.

db.people.find({cars: {$and: [{$elemMatch: {model: "Bentley"}}, {$elemMatch: {model: "Rolls Royce"}}] } })

alternative using $and (don't need to use elemMatch here, see <https://docs.mongodb.com/manual/tutorial/query-array-of-documents/>):

db.people.find({$and: [{"cars.model": "Bentley"}, {"cars.model": "Rolls Royce"}]})

//Below should work as well

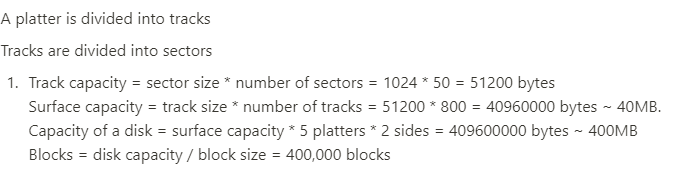
db.people.find({“cars.model” :{$all : ["Bentley", "Rolls Royce"]}})

db.people.find({cars: {$elemMatch: {value: {$gt: 100000}} }})

without elemMatch: db.people.find({"cars.value": {$gt: 100000}})

Question 2 (Really Not so sure)

A)



Track Capacity = sector size \* track size = 1024 \* 50 = 51,200  
Surface Capacity = track capacity \* platter size = 51200 \* 800 = 40,960,000  
Disk Capacity = surface capacity \* #surfaces = 40960000 \* 5\*2 = 409,600,000  
Blocks per disk = disk capacity / block size = 409600000 / 1024 = 400,000

B)

Records per block = floor(block size/record size) = 1024/200 ~ 5  
Blocks for file = #records / records per block = 10000/5 = 2000  
Waste per block = block size – records per block \* record size = 1024 - (5 \* 200) = 24  
Waste for file = waste per block \* blockss for file = 24 \* 2000 = 48000  
  
Yes, there will be waste depending on memory setup. We could also request that records do not span 2 logical partitions (blocks) for better alignment, and thus better speeds. But if memory is a constraint, we could allocate the minimum amount of memory needed to store the record, and thus not waste space.  
Waste per record = minimum data to move one record – record size = 60\*4 – 200 = 40  
Waste for file = waste per record \* #record = 40 \* 10000 = 400000

C) 3D XPoint 🤓. Unlike main memory, it is non-volatile and can thus store data after the power is turned off. It is also cheaper than RAM. However, while faster than flash SSD memory, it is still not as fast as main memory.XML

D)   
1. Balance writes across every flash page to improve flash overall lifetime  
2. Avoids making updates wait for an erase by writing to a new page and diverting all subsequent reads to that address  
3. Keep a list of all old invalid pages, and erasing them later

Question 3

A)

* One option is just to store the document as a blob, this pushes a lot of work into the application as we can’t make use of any querying
* We could build a schema by looking for every field across all the documents, this would lead to a very sparse database and be less compact than the document storage model

B)

* Embedding: inline data into a field
* Linking: put reference to data in field instead
* Push to application: retrieve documents separately and combine in application

C)

* We can think of the nested tags of an XML document as a tree structure
* We can represent the tree in a graph databases with edges for parent/child relationships
* The edge can include the ordinal number so the file can be exactly reconstructed
* It is possible to convert a graph database into a XML document if the graph is non-cyclic. However, cyclic graphs cannot be represented in a XML document // cycle by reference in XML

D)

* Graph and document databases both avoid needing an intermediate table, and extra joins, required for many to many relationships.
* For some relationships document databases designers may choose to embed documents within each other to join them. In some cases this might require duplicating some data, which would require extra space but may make querying more convenient. In the case of one-to-many relationships however this is quite compact
* Alternative:
  + Graph databases don’t need joins. The relationship between specific nodes is already represented and otherwise unintuitive joins over separate tables are represented as paths in the graph database.