1a)

Happy birthday inara mate

i)

MATCH (f)-[:Destination|Origin]->(a)

WHERE f.airline = “LX”

RETURN count(DISTINCT a)

ii)

MATCH (a)<-[:Origin]-(f)-[:Assign]->(t)

WHERE a.name = “Seattle” AND t.class = “business”

RETURN collect(t);

iii)

MATCH (seattle)<-[:Origin]-(f1)-[:Destination]->(s)<-[:Origin]-(f2)-[:Destination]->(sf)

WHERE seattle.name = “Seattle” AND sf.name = “San Francisco”

RETURN f1, f2;

iv)

MATCH (a)<-[:Origin]-(f)

RETURN DISTINCT(f.airline, a.name), count(f);

Can argue iv) not solvable as no information on cities in the data

v)

MATCH (seattle)<-[:Origin]-(f1)-[:Destination]->(s1), (s1)<-[:Origin]-(f2)-[:Destination]->(s2)

OPTIONAL MATCH (s2)<-[:Origin]-(f3)-[:Destination]->(s3)

OPTIONAL MATCH (s3)<-[:Origin]-(f4)-[:Destination]->(s4)

OPTIONAL MATCH (s4)<-[:Origin]-(f5)-[:Destination]->(s5)

WHERE seattle.name = “Seattle”

RETURN s2, s3, s4, s5;

This does not require that Origin and Destination permute so it probably won’t work:

MATCH (seattle)<-[:Origin]-(f1)-[:Destination]->(b:Airport)-[:Origin|Destination\*2..5]-(a:Airport)

WHERE seattle.name = “Seattle”

RETURN DISTINCT a;

1b)

i)

db.bakeware.find();

ii)

db.bakeware.find({“pricing.price\_per\_unit”: {$gt: 1}});

iii)

db.bakeware.find({“name”: /^C.\*/});

iv)

db.bakeware.find({$and:

[

{“type”: “donut”},

{“batters.batter”: {$elemMatch: {“type”: “Chocolate”}}},

{“toppings”: {$not: {$elemMatch: {“type”: “sugar”}}}}

]

})

v)

db.bakeware.find({$or:

[

{“batters.batter”: $elemMatch: {“type”: /.\*Chocolate.\*/}},

{“toppings”: $elemMatch: {“type”: /.\*Chocolate.\*/}}

]

})

Alternative: (As Figure 2 is just an example I don’t think this alternative contains all cases)  
db.bakeware.find({“$or”:

[

{“batters.batter”: {“$in”: [“chocolate”, “Devils food”]}},

{“toppings”: {“$in”: [“chocolate with sprinkles”, “chocolate”]}}

]

})

2)

a)

Track capacity = sector size \* track size

= 1024 \* 50 = 51200 = 51.2 KB

Surface capacity = surface size \* track capacity

= 400 \* 51200 = 20 480 000 = 20.48MB

Disk capacity = surface capacity \* number of surfaces

= 20 480 000 \* 10 = 204 800 000 = 204.8MB

Disk blocks = disk capacity / block size

= 204 800 000 / 1024 = 200 000 blocks

+ Say something about the disk architecture (i.e. sector is smallest unit of space on disk, track is composed of sectors, surface is composed of tracks and disk is composed of platters (with multiple surfaces). Blocks are unit of storage and can span multiple sectors. In this case it’s one block per sector.)

b)

records per block = floor(block size / record size)

= floor(1024 / 100) = 10

number of blocks required = number of records / records per block

= 10 000 / 10 = 1000 blocks

wasted space = ((block size / record size) - records per block) \* record size \* number of blocks required

= ((1024 / 100) - 10) \*100 \* 1000 = 0.24 \*100 \* 1000 = 24KB

Space is not wasted if the file is stored in memory as it can be stored contiguously.

50B of bandwidth is wasted. File size = 10 000 \* 100 = 1 000 000B. 1 000 000 / 75 = 13333.3 and 13333 \* 75 = 999 975 so there is 25B left to transfer, and thus 50B is wasted as 75 – 25 = 50.

c)

PCM may replace or complement SSD in the furute.

Benefits: Very good access time, good endurance (key benefit over SSD)

Drawbacks: Poor data density (too low)

Trends:

Not currently possible to tell what the trend in endurance will be.

Both access time and data density are likely to inrease which is good.

d)

Writes to SSD require a block to be read, copied (buffered in memory), erased and then rewritten. This is slower for random writes as we must do this from scratch, whereas with sequential write we can prefetch blocks into memory (so only need to perform the erase and rewrite steps.)

Also sequential writes are buffered in Internal Memory and then written out once the page is full.

The block is the smallest unit that can be erased on an SSD because it requires a larger voltage to erase than to read. It’s difficult to erase a small number of pages withouot erasing the surrounding ones.

3)

a)

Yes, a similar schema-free approach can be implemented for XML (It’s known as a model-mapping approach) on top of a RDB. Examples of this being done are Edge, Monet XParent and XRel.

Advantages of this are that it’ capable of supporting a range of XML applications that are either static (DTD does not change) or dynamic (DTD can change). Additionally, it doesn’t require extending the expressive power of DB models to support XML.

Some disadvantages of this are that query optimisation is made more difficult and some of the models don’t perform optimally for certain types of query / workload.

b)

Embedding is essentially a pre-computed join. It is convenient as the nested components can be accessed easily by the server and this doesn’t require any query optimisation.

Linking is a way of representing two joinable documents by keeping an identifier of one stored in the other. This allows for more flexibility but is more computationally expensive to process.

c) XRel is a node-oriented model-mapping approach, which creates tables for Path, Element Text and Attribute fields of an XML document.

XParent is edge oriented, using tables for LabelPath, DataPath, Element and Data.

Edge is also edge oriented, keeping a single table with information about all edges in an XML graph.

The performance of each is highly dependent upon the query workload. For example, Edge performs well for simple queries, whereas XRel and XParent outperform edge for more complex queries.

d) No longer examinable