1)

a)

MATCH (u:Umpire)

WHERE u.role = “field”

RETURN u;

Alternative (non-typed):

MATCH (m)-[:umpired\_in]->(u)

RETURN u;

b)

MATCH (t)-[:played]->(m)

WHERE t.name = “Ireland” AND m.date = “February 25 2015”

RETURN count(m) > 0;

c)

MATCH (t)-[:played]->()-[:played\_at]->(v)

WHERE t.name = “India”

RETURN DISTINCT v;

d)

MATCH (p)-[:includes]->(t)-[:played]->(m)-[:played\_at]->(v), (p)-[:includes]->(t2)-[:played]->(m)

WHERE v.name = “Hagley Oval, Christchurch”

RETURN m;

e)

MATCH (m)-[:played\_at]->(v)

RETURN v, count(m)

ORDER BY count(m) DESC

LIMIT 1;

f)

MATCH (m:Matches)

WHERE m.date = “8 March 2015”

RETURN m;

Alternative (non-typed):

MATCH (t) -[:played]-> (m)

WHERE m.date= “8 March 2015”

g)

MATCH (p) -[:INCLUDES]-> () -(:PLAYED)-> (m) -[:PLAYED\_AT]-> (v),

(m) -[:UMPIRED\_IN]-> (u)

WHERE p.name = “A”

AND m.date = “March 8 2015”

AND venue.name = “Hagley Oval, Christchurch”

RETURN u;

h)

MATCH (p)-[:includes]->(t)-[:played]->(m), (p)-[:includes]->(t2)-[:played]->(m)

WHERE t.name = “India” AND t2.name = “Pakistan”

RETURN count(m) > 0;

i)

MATCH (t)-[:played]->(m)-[:umpired\_in]->(u), (t2)-[:played]->(m)

WHERE t.name = “India” AND t2.name = “Pakistan” AND u.role = “Field”

RETURN u;

j)

MATCH (o)<-[:played|played\_at|umpired\_in\*]-(t)

WHERE t.name = “India”

RETURN o;

2)

a)

db.restaurant.find();

b)

db.restaurant.find({“borough” : “Bronx”});

c)

db.restaurant.find({“grades.score” : {$gt : 90}});

d)

db.restaurant.find({“address.coord.1” : {$lt : -95.754168}});

e)

db.restaurant.find({“name” : {$regex : /^Wil/}},

{“restaurant\_ID” : 1, “name” : 1, “borough” : 1, “cuisine” : 1});

f)

db.restaurant.find({$and : [

{“borough” : “Bronx”},

{“cuisine” : {$in : [“American”, “Chinese”]}}

]});

db.restaurant.find({{“borough” : “Bronx”, “cuisine” : {$in : [“American”, “Chinese”]}}});

g)

db.restaurants.find({“address.coord.1” : {$gt : 42, $lt : 52}})

.sort({“name” : 1, “address” : 1, “address.coord.1” : 1});

h)

db.restaurants.sort({“cuisine” : 1, “borough” : -1});

i)

db.restaurants.find(“address.coord” : {$type : “double”});

(although, I think the above takes the question too literally since it is actually of type array, we want to see if any of the entries are double.)

db.restaurant.find(“address.coord” : {$elemMatch : {$type : “double”}});

j)

db.restaurants.aggregate([

{ $lookup :

{ from : “boroughs”,

localField : “borough”,

foreignField : “borough”,

as : “borough\_info”

}

}

]);

3)

Assumptions about question wording:

2000 tracks/surface

50 sectors/track

a)

Track capacity: 50\*512 = 25600 bytes/track

Surface capacity: 25600 \* 2000 = 51200000 bytes

Disk capacity: 51200000\*(5\*2) = 512000000 bytes

Number of blocks in disk:

Disk capacity/block size = 500,000

b)

1024/100 = 10.24 records per block. No block span so 10 records per block.

100000records/ 10 records per block = 10,000 blocks

Wasted space = 24bytes\*10000blocks = 240,000bytes

c)

Disk has 500,000 blocks. Assuming again that we don’t allow records to span blocks we get 500,000blocks\*10 records per block = 5,000,000 records

d)

Time to read 10,000 blocks sequentially.

Blocks per track = 25600/1024 = 25 blocks per track.

We have 10\*25 = 250 blocks in a cylinder.

10000/250 = 40 cylinders we need to read.

Read time = transfer time+seek time+rotational delay

Transfer time = 0.001secs per block \* 10000 blocks = 10 secs

Avg seek time = 0.1 secs. Total seek time = 40 cylinder\*0.1 secs = 4 secs

Rotational delay = 0.005 (just one of this cause sequential access).

Total read time = 10+4+0.005 secs = 14.005 secs

If all heads can read in parallel

Transfer rate would be 0.001 secs per 10 blocks(cause all 10 heads could read at the same time)

So transfer time = 0.0001secs per block\*10000blocks = 1 sec

Total read time = 1 + 4 + 0.005 = 5.005 secs

e)

Flash-only OLTP

Flash-aided business intelligence (OLAP)

Logging on flash / HDD

Random writes slower because data must be overwritten, this is done by deleting, thus applying a voltage to a block. However, the voltage applied is never precise enough so data must be copied and rewritten which takes more time than sequential writes.

Random writes are the limit of durability as they are the bottleneck in terms of read/write speeds for SSDs.