Supervision 2 Questions

Question 1

1a

```
 match (p:Person)-[r:ACTED_IN]->(:Movie)
return p.name as name, count(*) as total
order by total desc, name
limit 10;
```

✓

For all `Person` nodes, count the number of arcs (relationships labelled `ACTED_IN`) which connect it to a `Movie` node.

1b

```
 match (:Movie {title: 'John Wick'}) <-[r2:ACTED_IN]- (p:Person) -[r1:ACTED_IN]-> (:Movie {title: 'The Matrix Reloaded'})
return p.name as name, r1.roles as roles1, r2.roles as roles2
order by name, roles1, roles2;
```

✓

Match a `Movie` node with the title "John Wick", in which a `Person` node has acted, where that `Person` node has also acted in a `Movie` node with title "The Matrix Reloaded".

1c

```
 match path=allshortestpaths(
          (m:Person {name : "Steven Spielberg"} ) -[:PRODUCED*]- (n:Person))
     where n.person_id <> m.person_id
     return length(path)/2 as spielberg_number,
            count(distinct n.person_id) as total
order by spielberg_number;
```

✓

Match all of the shortest paths between the `Person` with name "Steven Spielberg" and some other `Person` node, where they are related any number of arcs labelled `PRODUCED`. A path might go something like *Person -> Movie -> Person* so there are two arcs for every *coproducer* relationship, so divide the path length by 2 to get the Spielberg number. Count the number if distinct `Person` nodes which have this Spielberg number.

Question 2

`(a and b)` yields null when one of `a` or `b` is `null`, and the other one is either `true` or `null`.

a	b	a and b	a is null	b is null	a or (a is null)	b or (b is null)	(a is null) and (b or (b is null))	(b is null) and (a or (a is null))	((a is null) and (b or (b is null))) or ((b is null) and (a or (a is null)))	(a and b) is null
true	true	true	false	false	true	true	false	false	false	false
true	false	false	false	false	true	false	false	false	false	false
true	null	null	false	true	true	true	false	true	true	true
false	true	false	false	false	false	true	false	false	false	false
false	false	false	false	false	false	false	false	false	false	false
false	null	false	false	true	false	true	false	true	true	false
null	true	null	true	false	true	true	true	false	true	true
null	false	false	true	false	true	false	false	false	false	false
null	null	null	true	true	true	true	true	true	true	true

The last two columns are the same, showing that `(a and b) is null` is equivalent to `((a is null) and (b or (b is null))) or ((b is null) and (a or (a is null)))`

✓ Can also be deducted from the AND Truth table

Question 3

a	b	a and b	not a	not b	(not a) or (not b)	not (a and b)
true	true	true	false	false	false	false
true	false	false	false	true	true	true
true	null	null	false	null	null	null
false	true	false	true	false	true	true
false	false	false	true	true	true	true
false	null	false	true	null	true	true
null	true	null	null	false	null	null
null	false	false	null	true	true	true
null	null	null	null	null	null	null

The last two columns are the same, showing that `not (a and b)` is equivalent to `(not a) or (not b)` ✓

a	b	a or b	not a	not b	(not a) and (not b)	not (a or b)
true	true	true	false	false	false	false
true	false	true	false	true	false	false
true	null	true	false	null	false	false
false	true	true	true	false	false	false
false	false	false	true	true	true	true
false	null	null	true	null	null	null
null	true	true	null	false	false	false
null	false	null	null	true	null	null
null	null	null	null	null	null	null

The last two columns are the same, showing that `not (a or b)` is equivalent to `(not a) and (not b)` ✓

a	not a	a is null	a or (not a)	a or (not a) or (a is null)
true	false	false	true	true
false	true	false	true	true
null	null	true	null	true

When `a` is `null` the `a or (not a)` column is actually `null` rather than `true`. Instead, the extended axiom `a or (not a) or (a is null)` is always `true`. ✓

Question 4

Pros of a pre-defined schema

- You can't accidentally insert data in the wrong format ✓
- When selecting data, you can be certain of the output format ✓
- Allows for greater optimisations, especially for large tables ✓

Cons of a pre-defined schema

- You need to have a good idea of the structure of your data right at the start of the project ✓
- If you want to change your schema, you risk corrupting your existing data ✓
- The schema can be big and clunky so that it can handle edge cases for your data -- i.e. if you have some one-off data which doesn't quite fit the schema, you might have to make several extra tables to accommodate that. ✓

Question 5

One might migrate an SQL database to Cypher if their use cases include problems from graph theory such as exploring paths between two related entities, or clustering entities together. One might also choose cypher over SQL if the schema is expected to change over time (such as with updates to the application, etc.) because in Cypher, you wouldn't have to update all of the existing data to fit the new schema.

However, one might choose to stay with SQL over Cypher if there is a lot of data and it is important to query it quickly. The strongly structured nature of SQL allows for huge optimisations with features like indexes, which you cannot get in a graph database. On a more practical level, SQL is supported by a wide variety of database systems, so you'll perhaps have more choice, or better compatibility with other systems.

Yes but which one is old?

Question 6

a

A *safe query* is one which is guaranteed to return a finite number of tuples as its result.

b

✓ ✓

i. minimum: 0, maximum: rs ✓
ii. minimum: rs, maximum: rs ✓
iii. minimum: 0, maximum: min(r,s) ✓
iv. minimum: r, maximum: r ✗
v. minimum: max(r,s), maximum: r+s ✗

c

Query 3 is not equivalent, as shown in the following example.

R:

A	B
1	7
2	9

S:

B	C
7	3
6	6

take b=7.

Here, $\pi_A(R)$ will yield:

A
1
2

And $\sigma_{B=b}(S)$ will yield:

B	C
7	3

And so the cross product of these is:

A	B	C

A	B	C
1	7	3
2	7	3

And projecting columns A and C gives:

A	C
1	3
2	3

✓

Whereas queries 1 and 2 yield

A	C
1	3

✓