University of London

BSc Computer Science

Databases and Advanced Data Techniques **CM3010**

SECTION A

Candidates should answer ALL of Question 1 in Section A.

Question 1

(a) What is missing from the following set of commands?

Choose ONE option. [4]

```
START TRANSACTION;
UPDATE Account SET Balance = Balance-100 WHERE AccNo=21430885;
UPDATE Account SET Balance = Balance+100 WHERE AccNo=29584776;
SELECT SUM(Balance) FROM Account;
```

i. `ROLLBACK;`
ii. `INSERT INTO Account VALUES (100);`
iii. `END TRANSACTION;`
iv. `COMMIT;`
v. `UPDATE Account SET Balance = Balance+100 WHERE AccNo=21430885;`

(b) The following query should return the name of the city of Christiano Ronaldo's birth. Why doesn't it?

Choose ONE option. [4]

```
SELECT DISTINCT *
WHERE
{
  "Cristiano Ronaldo"@en dbo:birthPlace
    [
       a            dbo:City ;
       rdfs:label   ?cityName
    ] .
  FILTER ( LANG(?cityName) = 'en' )
}
```

i. The city is not in England, so the filter removes it.
ii. `"Cristiano Ronaldo"@en` is a string, not a URL. It can't be the subject of a triple. The WHERE clause should begin `[] rdfs:label "Cristiano Ronaldo"@en ;`
iii. The first part of the WHERE clause is a duple, not a triple. The object is missing.
iv. Ronaldo's place of birth is not in Wikipedia in a way that dbpedia can access.

(c) The text below is derived from Sir Tim Berners-Lee's 'v-card', an online document that gives contact information about a person. How many predicates does this extract contain?

Choose ONE option. [4]

```
card:I a :Male
       foaf:family_name "Berners-Lee";
       foaf:givenname "Timothy";
       foaf:title "Sir".
```

i. 4
ii. 7
iii. 5
iv. 8
v. 9
vi. 1
vii. 12
viii. 2
ix. None

(d) Given this XML (which is an extract of the library file of a compact disk catalogue), how many results does the following query select?

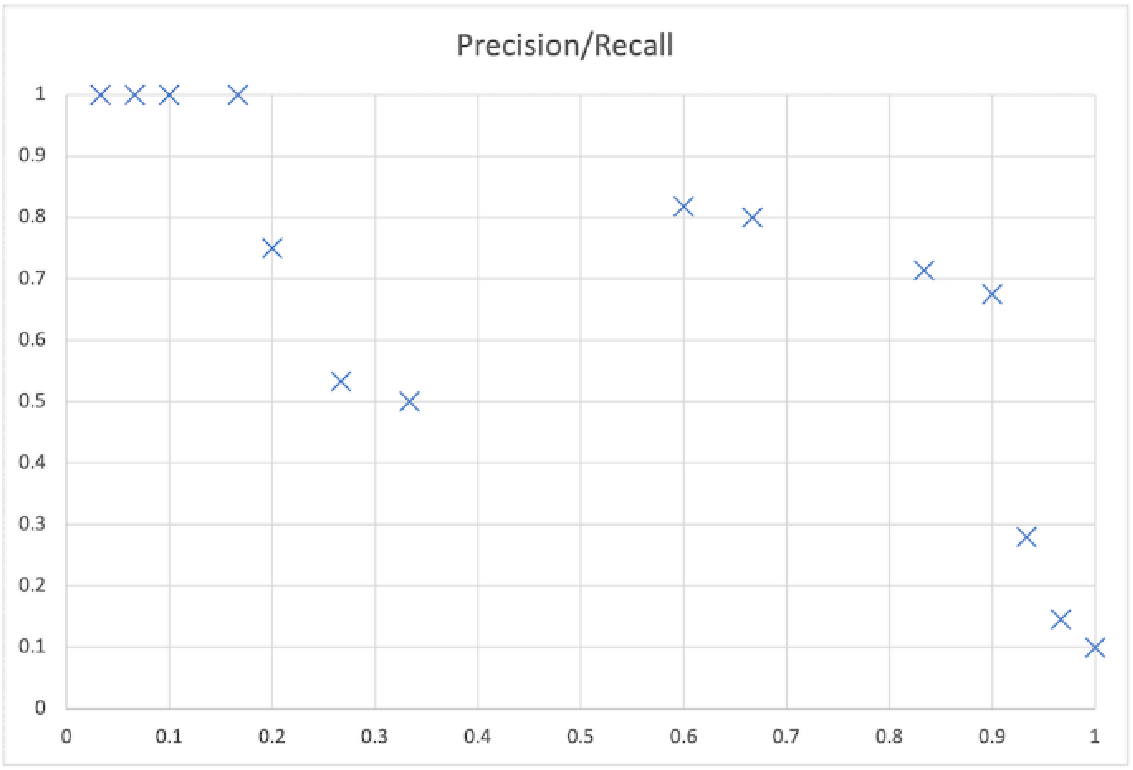
Choose ONE option. [4]

```
<collection>
  <disk xml:id="1847336">
    <title>The Greatest Hits Ever: Volume 123</title>
    <tracks>
      <track no="1" duration="193">
        <title>What is wrong with parsley?</title>
        <artist>Herbal Reasoning</artist>
      </track>
      <track no="2" duration="167">
        <title>Love threw me a googly</title>
        <artist>Botham and the Fielders</artist>
      </track>
      <track no="3" duration="121">
        <title>Comedy farm</title>
        <artist>Just weird</artist>
      </track>
    </tracks>
  </disk>
</collection>
```

//disk[xml:id="1847336]/track[@duration>150]/*

i. 5
ii. 4
iii. 1
iv. 6
v. 3
vi. None
vii. 2

(e) An archive has created a new search tool for their collection. Experiments on their data suggest the precision/recall behaviour shown in this graph (as is usual, precision is shown on the y-axis, recall on the x-axis):



The archive has about 50,000 items, of which about 30 are relevant to me.

Without the tool, it would take me 15 minutes to find each item (so if an item is missed from the search results, I can find it in 15 minutes). Irrelevant results waste 30 seconds of my time before I realise they are not relevant.

Assuming each data point represents a parameter setting for the tool, which is likely to be best (in the sense that I spend the least time on the task)?

Choose ONE option. [4]

i. Just right of the centre of the graph, where precision goes up again. This will give me about 12 false negatives and 5 false positives, which will take very little time to fix.
ii. To the right of the graph, before it drops – with 68% precision and 90% recall, I'd expect to spend about 6 minutes on maybe 12 false positives and about 45 minutes on the c. 3 missing records.
iii. Do not use this tool. Finding each resource manually will be quicker.
iv. To the left of the graph – I can get 100% precision with 17% recall. The rest of the records I can find manually, spending 15 minutes on each.
v. To the left of the graph – I can get 100% precision with 17% recall. The rest of the records I can find manually, spending 30 seconds on each, taking about 13 minutes in total. There will be no (or very few) false positives with 100% precision.

(f) Look at the following extract from a data table:

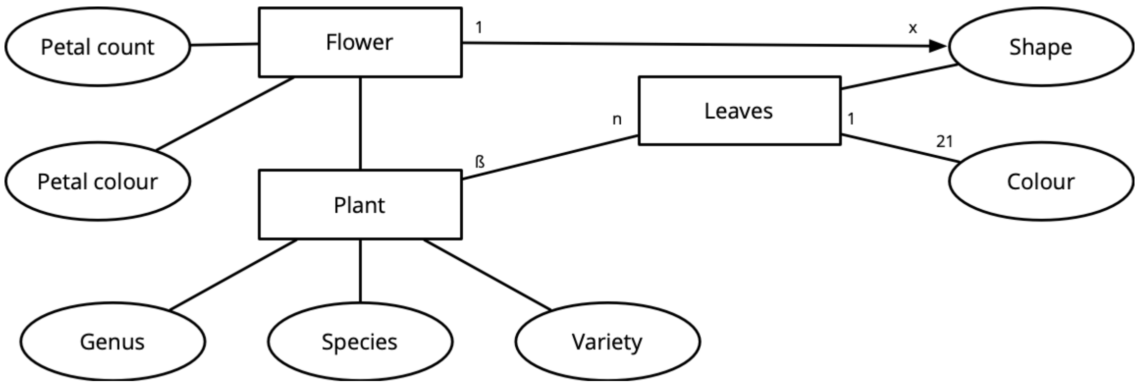
Music Singles					
Chart	Date	Position	Title	Artist	Date of Birth
RIAS	2022-04-14	1	As It Was	Harry Styles	1994-02-01
UK Singles Chart	2012-04-08	4	Starships	Nicki Minaj	1982-12-08
Billboard Hot 100	2022-04-22	1	First Class	Jack Harlow	1998-03-13
SNEP	1993-11-20	5	Il me dit que je suis belle	Patricia Kaas	1966-12-05

Which normal forms does this table satisfy?

Select ALL statements that apply. [4]

i. 2NF – no attribute is dependent on any non-key element or any subset of the primary key
ii. 3NF – table is in 2NF and any transitive dependencies depend on the primary key
iii. 5NF – every non-trivial join dependency is implied by the candidate keys
iv. 1NF – all rows are a single data type
v. 4NF – every non-trivial multivalued dependency depends on a superkey
vi. None of them
vii. SNCF – all other normal forms apply, and the data is timetable information
viii. BCNF – table is in 3NF and no redundancies based on functional dependency remain

(g) A client who wants to build a plant identification database has sent you the E/R diagram below to convert to a relational model and implement as a database.



This is not a good E/R diagram. Why?

Select ALL statements that apply. [4]

i. By convention, cardinality is only given between entities (not attributes).
ii. Entities are connected without explicit relationships.
iii. The arrow is meaningless.
iv. Spaces are not permitted in attribute names.
v. An attribute can’t be shared between entities.
vi. Cardinalities like ‘21’ are not allowed.
vii. There is a ternary relationship.
viii. Cardinalities like ß and x are inadvisable (use n or m).

(h) At a sales company, I want to find all staff members who have had interactions with a client called “Shug Avery”. The database query will begin:

```
SELECT Employee.givenName, Employee.familyName
```

How might it continue?

Select ALL statements that apply. [4]

i.
```
FROM Employee LEFT JOIN Client ON (Client.name="Shug Avery");
```

ii.
```
FROM Meeting LEFT JOIN Client ON (Meeting.ID = Client.ID)
             LEFT JOIN Employee ON (Meeting.ID = Employee.ID)
             WHERE Client.givenName LIKE "Shug"
             AND Client.familyName LIKE "Avery";
```

iii.
```
FROM Client
INNER JOIN Meeting ON (Meeting.ClientID = Client.ID)
INNER JOIN Employee ON (Employee.ID = Meeting.EmployeeID)
WHERE Client.givenName = "Shug";
AND Client.familyName = "Avery";
```

iv.
```
FROM Employee, Client, Meeting
WHERE Employee.ID = Meeting.EmployeeID
AND Client.ID = Meeting.ClientID
AND Client.givenName = "Shug"
AND Client.familyName = "Avery";
```

v.
```
FROM Client NATURAL JOIN Employee
WHERE Client.givenName LIKE "Shug"
AND Client.familyName LIKE "Avery"
```

(i) Which of these queries is likely to represent a successful MongoDB search for actors born before 1957?

Choose ONE option. [4]

i.
```
db.actors.findOne({
"dateOfBirth": {$lt: ISODate("1957-01-01")}
});
```

ii.
```
db.actors.findOne({
"dateOfBirth": {$lt: 1957}
});
```

iii.
```
db.actors.findOne({
"dateOfBirth": {"<": 1957}
});
```

iv.
```
db.actors.find({
"dateOfBirth": {"<": 1957}
});
```

v.
```
db.actors.find({
"dateOfBirth": {"<": ISODate("1957-01-01")}
});
```

vi.
```
db.actors.find({
"dateOfBirth": 1957
});
```

vii.
```
db.actors.find({
"dateOfBirth": {$lt: ISODate("1957-01-01")}
});
```

viii.
```
db.actors.findOne({
"dateOfBirth": {"<": ISODate("1957-01-01")}
});
```

(j) The Recipe Markup Language (RecipeML) can be used to specify recipes in a machine-readable way. Two lines from the language's .dtd file are as follows:

```
<!ELEMENT recipe (head, description*, equipment?, ingredients,
directions, nutrition?, diet-exchanges?)>
<!ATTLIST recipe
  %common.att;
  %measurement.att;>
```

Given this information, select ALL the true statements below. [4]

i. `<recipe>` must have one `<ingredients>` elements as a direct child.
ii. The `<ingredients>` element must come before the `<directions>` element.
iii. The order of children of `<recipe>` is not important.
iv. `<recipe>` can have one `<ingredients>` elements as a direct child.
v. `<recipe>` can have multiple `<ingredients>` elements as direct children.

END OF SECTION