University of London

Assessment Coversheet

Complete this coversheet and read the instructions below carefully.

Candidate Number: KJ1932
Refer to your Admission Notice

Degree Title: BSc in Computer Science
e.g. LLB

Course/Module Title: Databases and Advanced Data Techniques
As it appears on the question paper

Course/Module Code: CM3010
This is in the top right corner of the question paper. If there is more than one code, use the first code.

Enter the numbers, and sub-sections, of the questions in the order in which you have attempted them:

2 a b c d e f g
3 a b c d e f g h i

Date: March 9, 2022

Instructions to Candidates

1. Complete this coversheet and begin typing your answers on the page below, or, submit the coversheet with your handwritten answers (where handwritten answers are permitted or required as part of your online timed assessment).
2. Clearly state the question number, and any sub-sections, at the beginning of each answer and also note them in the space provided above.
3. For typed answers, use a plain font such as Arial or Calibri and font size 11 or larger.
4. Where permission has been given in advance, handwritten answers (including diagrams or mathematical formulae) must be done on light coloured paper using blue or black ink.
5. Reference your diagrams in your typed answers. Label diagrams clearly.

The Examiners will attach great importance to legibility, accuracy and clarity of expression.

Begin your answers on this page

Question 2

(a) Two element names: `title`, `relationship`. Two attribute names: `rank`, `spouse`

(b) This expression will return the title element where the rank attribute matches “king” and the regnal attribute matches “VIII,” and the title element is a child of the royal whose name is “Henry.” It will return the first title element from the provided XML code.

(c) This query will return the royal grandchildren (as royal elements) of any royal whose rank is king or queen.

(d) I would use the following title element. It could be added as a child of Mary’s “royal” element. This would allow us to query all titles of a given royal easily. On the other hand, it results from a relationship, so another option would be to add it as a child of the relationship to Philip of Spain. However, I would prefer the former option (direct child element of the royal named “Mary.”

```
<title rank="queen_consort" territory="Spain" regnal="I"
from="1556-01-16" to="1558-11-17" />
```

(e) The strengths of this approach include the easy extensibility of the model to include complex relationships among royals. For example, additional types of relationships could be added to the schema. As royal families tend to be large with many descendants, this approach seems reasonable to model a family tree using the tree-like nature of XML. A further advantage is that these files are human-readable and easier to share. Discoveries can be added to past files and easily maintained in text format. The meaning of various objects (royals, their relationships, children, ranks, and territorial reigns) can easily be derived from the XML format. Disadvantages include the rigid tree structure that doesn’t always lend itself to the potential messy realities of royal relationships. For example, royals may have had children with multiple partners, often incestuously, and a graph more closely models these relationships than a tree. Further, ranks and titles often relate to relationships; hence it is unclear which level in a tree structure a title should be placed (it could be at the root level of a given royal or at the level of the relationship that resulted in the rank). The limitations get worse when faced with overlapping dates, multiple spousal relationships, or even children birthed out of wedlock, all of which would be hard to model in the current structure. Further, attributes are possibly free text (we don’t know the details of the schema) and are error-prone. No enforcement of data content is guaranteed. Data needs to be duplicated often.

(f) Both are valid approaches depending on the goal. RDF might be more beneficial for connecting data across multiple royal families' ontologies, possibly maintained across different countries by different historians. An RDBMS would enforce more centralized control but offer more flexible data modelling.

(g) I will choose the option of a relational database. It solved some of the problems stated above as follows. With tables for parent-child relationships, we would no longer have to nest these relationships. Still, we could model any parent-child relationship regardless of where in the family tree they occur. For example, this would allow a parent to have a child with one of their children, which is mappable in the database. Further, it would allow for standardized ranks, regionals, and territories. A relational database would further allow historicizing data, seeing how information changed over time. This is still possible with XML but is harder to model. An RDBMS can further allow the mapping of marriages more easily without duplicating data. For example, if Queen Mary was the consort of Spain, we could have both the King of Spain and Queen Mary in the same database once and link them with a marriage table. A weakness of the RDBMS approach is that it is no longer human-readable and requires a technical querying language to retrieve data. Further, the structure and meaning of the database need to be known to the querying agent/person.

Question 3

(a) This query will return a list of distinct instances of human persons that have their place of birth in New York City.

(b) The query assumes that there are records that have the attribute of being human recorded, along with their birthplace data being complete. The birthplaces are also present in the database with their specific names. It assumes that the birthplace is not more specific than "New York City" in this case, for example a hospital or other area that would technically count as New York City.

(c) This version of the query broadens the search scope of the birthplace to also consider any other birthplaces that fall in the administrative territory of New York City. The new criterion is given a "zero or more" quantifier. This version should result in more robust results.

(d) Our query is only asking for the URIs of the results, and not querying any specific attributes. Hence the results are just long lists of URIs to matching people.

(e)

```
select distinct ?item ?itemLabel ?itemDescription where {
    ?item wdt:P31 wd:Q5;
          wdt:P19/wdt:P131* wd:Q60;
    SERVICE wikibase:label { bd:serviceParam wikibase:language
"en" }
}
```

(f) The IMDB approach has a few advantages over what Wikidata does. Providing a URL structure to query data is more easily accessible, and provides a means for fuzzier querying - that is, you can provide any or alternate spellings and let the site handle the matching algorithm to provide meaningful results. This will give us a balance of precision and recall, as IMDB decides what to return on a given query. As such, the results are non-determined and potentially stochastic given any underlying algorithms used to surface data. The WikiData approach however relies on a much more robustly defined ontology and data model, making the results much more predictable, and focused on precision given the input query's defined expression of triples.

(g) We could use the power of linked data, the semantic web and web ontologies to build a graph of information that spans across both data sources. Using that ontology, we could map IMDB information about an actor to their corresponding data in WikiData. However, we would still need to centrally retrieve this data to make it available, and one approach might be to load this data into an RDBMS for easier querying. We would need to first map the data from IMDB and WikiData to a common schema.

(h) I would create a database with the following tables: Creatures, Birthplaces, Administrative Territorial Entities.

The query would be:

```
SELECT DISTINCT creatures.name, birthplaces.name
FROM creatures
LEFT JOIN birthplaces
ON birthplaces.id = creatures.birthplace
WHERE birthplaces.name = "New York City"
```

(i)

```
SELECT DISTINCT creatures.name, birthplaces.name,
territorial_entities.name
FROM creatures
LEFT JOIN birthplaces
ON creatures.birthplace = birthplaces.id
LEFT JOIN territorial_entities
ON birthplaces.territory = territorial_entities.id
WHERE birthplaces.name = "New York City" OR
birthplaces.territory = "New York City"
```