University of South Australia

Assignment 2 – Data Queries in MongoDB

Assessment Summary

Weighting: 25%
Due Date: 23:59PM Sun 16 Apr
Assignment type: group of three or less

Submission

Right-click on the folder **DBE-A2** to zip the folder and submit the zip file to the Assignment 2 submission area on the course website.

Assignment Overview

This assignment continuous your work on Assignment 1. It enables you to apply your knowledge of XML, XQuery, JSON, and Relational Databases to an application. The objectives include the following.

- Data transformation from the json model to the relational model
- Query composition in MongoDB
- Contrast of query composition in MongoDB and in the relational database
- Understanding of modelling power of different models

Before starting, you should complete all the exercises on *MongoDB*.

If a group has less than three members, the group still completes all tasks. If you work in a group, you elect one group member to submit. Carefully edit the **people.txt** file to include the username of all group members. Note that this file is important, especially when there is a dispute among group members. Generally, the marker will give all group members the same marks. However, when the work quality is significantly inconsistent or in the case of a dispute, you may be given marks based on how well your tasks are done.

To start:

Unzip the DBE-A2.zip file to a convenient location and this will give you the **DBE-A2** folder. Here is a note about the location. A good option for the location is a cloud storage like OneDrive or Google drive which makes sharing easy and has automatic backup functions. After you make progress, you zip your work and send it to your UniSA outlook email or another email account on a third-party email server like Gmail. This leaves a trail of evidence of your effort. In the case of a laptop issue, or a dispute among group members, the email backups are your support for marks.

In the DBE-A2 folder, you see the data files and some empty files as shown below. You will gradually put your answers in these files.

```
people.txt
.basex
DBE-A2-markreport.xlsx
nasaproj.json
query1.js          --- MongoDB query
A2-report.docx   --- the report file
nasaproj.xml       --- input file to get-csv.xql
```

```
get-csv.xql
*.csv
create-tables.sql
query2.sql        --- SQL query
direct-query.py   --- optional
```

Edit the prople.txt to include group member information.

Application and requirements

You work on the data file **nasaproj.json** you used previously.

Task 1: Import data to MongoDB and identify a query requirement. (3 marks)
Task 2: Compose a query for the query requirement in MongoDB. (8 marks)
Task 3: Transform necessary data to csv and upload csv to Oracle. (10 marks)
Task 4: Compose a SQL Query for the same query requirement in Oracle. (4 marks)
Task 5 (optional, bonus): Write a python program to compute the query. (4 bonus marks)

The detailed requirements of the tasks are given below.

Task1. Import data to MongoDB and identify a query requirement.

1.1) Create a data collection called 'nasaproj' in a MongoDB database and import data in nasaproj.json to the collection.

MongoDB's import requires a file to contain only an array of objects, not an object. Here an object means a dictionary.

You edit the nasaproj.json file so that you only import the releases array. This means that you need to remove parent elements of 'releases' and the 'releases' field name so that you keep only the array of release objects in the file. After you import of the file to MongoDB, you should see 1294 documents.

In the report called **A2-report.docx**, include a screenshot to show that you have successfully imported the data.

1.2) Identify a query requirement for you to work on in the following tasks.

The complexity of your query requirement is measured by a factor. The factor reflects not only the complexity of the answer to the requirement, but also the workload. It will be applied to your solutions in all parts by multiplication. The complexity factor of your query requirement is determined by the number of the following operators needed in its answer:

group-by, the number of complex-subs.

Here, a 'sub' means the need to use the information in a sub array of values and a 'complex-sub' is one that uses the combination of array and object. Examples of simple-subs are tags, contact, and sti_keywords_passed_thresholds. Examples of complex-subs are permissions, contributors, and sti_keywords. Features is special, has the semantics of an array, and, as a result, is a complex-sub. Multiple subs imply joins of the subs.

Example query requirements and their complexity factors are:

- Find the contributor who linked to the maximal number of features through project participation among all the contributors in all projects.
  Complexity factor is 1.0. Answer requires group-by and two complex subs.
- Find all sti_keywords_passed_thresholds that match a feature field in the same project.
  Complexity factor is 0.9. Answer requires two complex subs.
- Find the contributor who worked on the maximal number of projects among all contributors.
  Complexity factor is 0.9. Answer requires one complex-sub and group-by.
- Find the release (project) that has the maximal number of features.
  Complexity factor is 0.8. Answer requires one complex-sub.
- Find the release (project) that has the maximal number of contributors.
  Complexity factor is 0.7.

You are encouraged to create your own requirement. If you pick up a requirement from the above list, your get a minor deduction of 0.7 marks. You should go through the json document to find a few releases with more data to identify your requirement.

Describe your query requirement in the report.

Include one of these two sentences after the description:

This query requirement is created by our group.
This query requirement is selected from the given list.

Task2. Write a MongoDB query to compute answer for the query requirement.

For the query requirement you identify in Task 1, write a MongoDB query to compute the answer.

Save your answer query in the file **query1.js**. The code in the file will be copy-pasted to Mongodb Shell to be tested. If you work in Mongo Compass, make sure that you merge the stages together to form a standard aggregate query (see example in the notes below) so that the final code can be executed in Mongo Shell.

In the report,

- include no more than two screenshots to show the run of the query in Mongo Shell and the output. If the output is too much, just a few lines at the top.
- write a paragraph to describe what you did to check the correctness of your query output.

Notes to help you in query writing:

1. Use a collection with a few documents. Otherwise, you will be lost in huge amount of data. For example, to test $project, $unwind and $function, a collection with the following only document is enough:
```
{"name" : "Beta: Text ", "tags" : [ "ARC", "Source" ],  "results" : { "code" :
200, "payload" : { "features" : { "chain" : "PROPN", "classi" : "NOUN" } } } }
```
2. Write your aggregate pipeline stage by stage.

For example, $project can be used to ignore data that is not needed for the query:

```
db.nasaproj3.aggregate([
  {$project:{"tags":1, "results.payload.features":1}}
  ])
```

Then, $unwind is used to flatten the tags array.

```
db.nasaproj3.aggregate([
  {$project:{"tags":1, "results.payload.features":1}},
  {$unwind: "$tags"}
  ])
```

3. The following is the full pipeline to test $project, $unwind, and $function. Try this example stage-by stage to get understanding. This query was tested on the MongoDB ver-6 server and with the standalone shell (ver 1.8.2) downloaded from https://www.mongodb.com/try/download/shell.

```
db.nasaproj3.aggregate([
  {$project:{"tags":1, "results.payload.features":1}},
  {$unwind: "$tags"},
  {$project:{ tags: 1,
              feat: {$function: { body: function(feat) { return
Object.keys(feat)},
                                  args:["$results.payload.features" ],
                                  lang: "js" } }   } },
  {$unwind: "$feat"}
  ])
```

Task 3: Transform necessary data to csv and uploading of csv to Oracle.

This task prepares data for the Oracle database on which you will compose a SQL query for the same requirement in the next step. Use your result for Task 4 of Assignment 1. Among the csv files in Assignment 1, you identify those that are necessary to answer the query.

Each identified file corresponds to a database table. Create statement to create the tables for the files. Upload the files into their corresponding tables in Oracle. Also, upload the files for all the parent tables of the necessary csv files/tables to the Oracle database.

For example, the query requirement “Find the contributor who linked to the maximal number of features through project participation among all contributors in all projects” requires the features table and the contributors table. You upload features.csv, contributors.csv, payload.csv, results.csv, and release.csv into the corresponding tables in the Oracle database. The csv files must have a header line and each header name must be a valid attribute name in Oracle, i.e., each header name must contain only letters and numeric characters.

You should improve your XQuery queries from Assignment 1 to generate your csv files to meet the above requirements.

In the report file, you list all the csv files that you upload to the oracle database.

Put your XQuery query generating these csv files in **get-csv.xql.** The code should run in basex without any needed change.

Include your well-formed xml file **nasaproj.xml** that is the input to the XQuery query in the DBE-A2 folder. The marker will run it to check if it works.

Include all the relevant csv files in the DBE-A2 folder.

Put your SQL table creation statements in **create-tables.sql**. The marker will copy them to sqldeveloper to test.

In the report, you write step-by-step instructions on how your data should be uploaded. At the same time, you use screenshots to show the first 10 lines of each table.

Task 4: Write a SQL Query for the query requirement in Oracle.

Write an SQL query for the query requirement identified in Task 1. Save your query in the file **query2.sql**.

Use a screenshot to show the query and its output.

Compare the output with the output obtained in Task 2 to ensure that they have the same information although they may be in different formats.

If they do not have the same information, you find out the reason and improve your queries in SQL and in MongoDB.

Task 5: (optional, bonus): Write a python program to compute the query.

This task is optional. However, its marks will be added to the marks of this assignment to make it perfect. Any remaining marks will be used to improve the marks of Assignment 1.

Write a python program called **direct-query.py** to compute the answer for the query requirement identified in Task1. The input of the program must be **nasaproj.json**. Your program will be executed from the DBE-A2 folder as the following:

```
python direct-query.py
```

Marking Criteria

Marking will consider filenames, correctness of results, their formats, and quality of writing.
If you attempt only some tasks and your results are correct, you get marks for the parts.

Extension and Late Submission Penalty

For each day of lateness, a 33% of deduction is applied unless an extension is given.

Extension requests based on workload reasons will **NOT** be approved to make the marks fair to everyone.

Extension requests must be submitted with **supporting evidence**. If you are not feeling well and you like to get an extension, go to see a doctor, and get a doctor's certificate as your evidence.