School of Computing and Information Technology
University of Wollongong

Session: **Autumn 2020**
Lecturer: **Janusz R. Getta**

CSCI235/CSCI835 Database Systems
Assignment 3
24 April 2020

Scope

This assignment includes the tasks related to implementation of database transactions.

The outcomes of the laboratory work are due by **Saturday 13 June, 2020, 11.00 pm (sharp).**

Please read very carefully information listed below.

This assignment contributes to 20% of the total evaluation in a subject CSCI235 and it contributes to 17% of the total evaluation in a subject CSCI835.

A submission procedure is explained at the end of specification.

This assignment consists of 4 tasks and specification of each task starts from a new page.

It is recommended to solve the problems before attending the laboratory classes in order to efficiently use supervised laboratory time.

A submission marked by Moodle as "late" is treated as a late submission no matter how many seconds it is late.

A policy regarding late submissions is included in the subject outline.

A submission of compressed files (zipped, gzipped, rared, tared, 7-zipped, lhzed, … etc) is not allowed. The compressed files will not be evaluated.

All files left on Moodle in a state "`Draft(not submitted)`" will not be evaluated.

An implementation that does not compile due to one or more syntactical errors scores no marks.

It is expected that all tasks included within **Assignment 3** will be solved **individually without any cooperation** with the other students. If you have any doubts, questions, etc. please consult your lecturer or tutor during lab classes or office hours. Plagiarism will result in a **FAIL** grade being recorded for the assessment task.

Prologue

If VirtualBox is not installed yet then install it on your system first. It is explained at

https://documents.uow.edu.au/~jrg/115/cookbook/e1-1-frame.html

how to install and how to use VirtualBox.

Use one of the links given below to download `Ubuntu18.04-64bits-MongoDB-4.2.2-08-JAN-2020.ova` file with an image of virtual machine running `MongoDB 4.2.2` on `Ubuntu 18.04`. You can also use the links published in a section `OTHER RESOURCES` on Moodle.

OneDrive:
https://uowmailedu-my.sharepoint.com/:u:/g/personal/jrg_uow_edu_au/EYCB0u8xnORCkApv9kaYgMsBC3MyOkAdH8Kyl_J8OrVfJw?e=wVlLbB

GoogleDrive:
https://drive.google.com/open?id=1-kzGEb521LS3HCYatgoVKU1hMhlDG89h

CloudStor: (When downloading from CloudStor you must use Firefox browser)
https://cloudstor.aarnet.edu.au/plus/s/qg8J3vE4SoxRzKo

Start VirtualBox and import a virtual machine `Ubuntu18.04-64bits-MongoDB-4.2.2-08-JAN-2020.ova.`

Next, start the virtual machine and login as an Ubuntu Linux user `CSCI235` with a password `csci235`.

Next, start `Terminal` program and within a `Terminal` window start `MongoDB` server in the following way.

```
mongod –dbpath DATA –port 4000
```

A server displays a lot of messages. A successful start of `MongoDB` server is confirmed with a message like

```
…
… I NETWORK  [initandlisten] waiting for connections on port 4000…
…
```

pretty well hidden somewhere at the end of a long list of other messages issued by the starting server.

Note, that the `Terminal` window you use just now becomes a console of the running `MongoDB` server and you cannot use it any more. Do not close the `Terminal` window ! Just minimize it.

Open a new `Terminal` window. To create a `BSON` collection `orders` use a command

```
cd CSCI235
```

to move to `CSCI235` folder with the scripts.

Next, to start a command line client `mongo` process the following command.

```
mongo -port 4000
```

To create a collection `orders` and to load the documents into the collection, process the scripts `employees.js`, `suppliers.js`, and `customers.js` at `>` prompt in the following way.

```
load("employees.js");
load("suppliers.js");
load("customers.js");
```

Next, you can use the methods

`db.orders.find().count()` and
`db.orders.find().pretty()`

to count the total number of the documents in a collection `bookshop` and to list all documents in a pretty format.

Next try few simple queries.

For example, to list information about a hierarchy of customers submitting orders that consist of products process a method

```
db.orders.find({"CUSTOMER":{$exists:true}}).pretty();
```

For example, to list information about a customer who has a contact name `Maria Anders` and the orders submitted by the customer process a method

```
db.orders.find({"CUSTOMER.contact name":"Maria Anders"}).pretty();
```

For example, to list information about an order that has order id 325 process a method

```
db.orders.find({"CUSTOMER.submits.ORDER.order id":325}).pretty();
```

The conceptual and logical schemas of a collection `orders` are available in the files `dbschema-bson.bmp` and `bsonschema.bmp` in section `SAMPLE DATABASES` on Moodle. It is strongly recommended to make yourself familiar with the conceptual and logical schemas of a sample database.

No report is expected from implementation of the actions included in **Prologue** section.

<u>Tasks</u>
Task 1 (5 marks)
Data manipulations

Download and unzip a file `solution1.zip`. You should get a file `solution1.js`. The file contains the specifications of the following 10 data manipulation operations on a collection `orders`.

(1) Append a new product `Changde Noodles` that belongs to a category `Noodles` to a list of products supplies by a supplier located in a city `Zaandam`. All other information is unknown at the moment. Display the names of products supplied by a supplier located in a city `Zaandam`.

(2) Remove information about a product `Longlife Tofu` supplied by a supplier `Tokyo Traders`. Display the names of products supplied by a supplier `Tokyo Traders`.

(3) Increase a `unit price` of a product `Flotemysost` by 100%. Display the `product name` and the changed `unit price` in a pretty format.

(4) Rename a key `submits` to a key `sends` in the orders submitted by a customer `FAMIA`. Display all information about a customer `FAMIA`.

(5) An order with `order_id` equal to `310` is now handled by an employee with `employee id` equal to `7`. Update the database. After update display `order_id` and `employee id` in a pretty format.

Implement the data manipulations listed above in a data manipulation language of MongoDB. Write your solutions into the empty slots following a specification of each data manipulation in a file `solution1.js`. Do not remove the specifications of the data manipulations and semicolons following the specifications.

Implementation of each data manipulation is worth 1 mark.

When ready create a report from processing of the data manipulations in the following way.

Use `gedit` editor to open a file `solution1.js` with the specifications and implementations of the data manipulations.

Select the entire contents of the file and Copy it into a buffer.

Open a new `Terminal` window and start `mongo` client in the following way.

```
mongo -port 4000
```

Paste the contents of the buffer copied earlier from `gedit` window in front of > prompt of `mongo` client. You may have to press `Enter` key to process the last data manipulation in a case when it is not followed by a newline control character.

Select the entire contents of the `Terminal` window and Copy&Paste it into a file `solution1.lst`. Save a file `solution1.lst`.

Deliverables
A file `solution1.lst` with a report from processing of MongoDB script `solution1.js` with the implementation of the data manipulations listed above.

And again, please remember that:
- a report without the specifications of the data manipulations and listings of the processed data manipulations scores no marks,
- a report that contains any kind of processing errors scores no marks.

Task 2 (7.5 marks)
Query processing and data transformation with aggregation framework

Download and unzip a file `solution2.zip`. You should get a file `solution2.js`. The file contains the comments with the specifications of the following 5 queries and data transformations.

(1) Save all information about the names of products supplied by a supplier `Gai paturage` into a collection `products1`. Display in a pretty format without document identifiers all documents in a collection `products1`.

(2) Save all information about the names of products supplied by a supplier `Gai paturage` into a collection `products2` that consists of the documents like `{"product name": a-name-of-product}`. Display in a pretty format without document identifiers all documents in a collection `products2`.

(3) Find the total number of products in a collection `orders`. Display a result in a format `{"total number of products":integer-value}`.

(4) List in the ascending order the ids of the first 3 employees who handled at least one order. Display the results in a format `{"employee id":a-value-of employee-id}`. List only distinct values.

(5) Find the company names of suppliers together with the total number of supplied products by each company. Display the results in a format `{"total products":integer-value,"company name":a-company-name}`.

Use the methods `aggregate()` and `pretty()` to implement the queries and data transformations and to display the results. Note, that you may need two or more statements to implement a single task.

Implementation of each query/data transformation is worth 1.5 mark.

When ready create MongoDB script file `solution2.js` with the implementations of your queries and create a report from processing of the data manipulations in the following way.

Use `gedit` editor to open a file `solution2.js` with the specifications and implementations of the data manipulations.

Select the entire contents of the file and Copy it into a buffer.

Open a new `Terminal` window and start `mongo` client in the following way.

```
mongo –port 4000
```

Paste the contents of the buffer copied earlier from `gedit` window in front of > prompt of `mongo` client. You may have to press `Enter` key to process the last data manipulation in a case when it is not followed by a newline control character.

Select the entire contents of the `Terminal` window and Copy&Paste it into a file `solution2.lst`. Save a file `solution2.lst`.

Deliverables
A file `solution2.lst` with a report from processing of MongoDB script `solution2.js` with the implementation of the data manipulations listed above.

Please remember that:
- a report without the specifications of the queries and data manipulations and listings of the processed queries and data manipulations scores no marks,
- a report that contains any kind of processing errors scores no marks.

Task 3 (4 marks)
Implementation of indexing

Download and unzip a file `solution3.zip`. You should get a file `solution3.js`.

Consider the documents included in a collection `orders` and the queries consistent with the following query templates.

(1) Find the company name, contact name, and contact title of the customers with a given company name.
(2) Find the company name of the customers who submitted an order with a give date.
(3) Find the names of products ordered by the customers living in a given city.
(4) Find the names of countries the customers live in.

Repeat the implementations of the following four steps for each one of the query patterns listed above.

Step 1 Create an index that speeds up processing of a query consistent with a pattern.

Step 2 Apply a method `getIndexes()` to list all existing indexes, for example `db.collection.getIndexes()`.

Step 3 Apply a method `explain()` to verify whether the system plans to use the indexes created for processing of a query consistent with a pattern, for example `db.collection.find({"country":"Sweden", "greetings":"Tjenare"}).explain()`.
The constants used in a query are up to you.

Step 4 Drop an index created in Step 1 with a method `dropIndex()`, e.g. `db.collection.dropIndex("index_name")`.
You can find a name given to an index by the system from the results of Step 2.

Write your solutions into a file `solution3.js` in the empty slots following a specification of each problem. Do not remove the comments with the specifications of queries and semicolons following the comments !

Implementation of each index, displaying query processing plans and dropping an index is worth 1 mark.

When ready create a report from processing of the queries in the following way. Use `gedit` editor to open a file `solution3.js` with the specifications of the queries and implementations of the queries.

Select the entire contents of the file and Copy it into a buffer.
Open a new `Terminal` window and start `mongo` client in the following way.

```
mongo -port 4000
```

Paste the contents of the buffer copied earlier from `gedit` window in front of > prompt of `mongo` client. You may have to press Enter key to process the last query in a case when it is not followed by a newline control character.

Select the entire contents of the `Terminal` window and Copy&Paste it into a file `solution3.lst`. Save a file `solution3.lst`. Examine the contents of a file `solution3.lst` for possible errors.

Deliverables
A file `solution3.lst` with a report from processing of MongoDB script `solution3.js` with the implementation of indexing, listing the indexes and query processing plans, and dropping the indexes.

And again, please remember that:
- a report without the listings of applied methods and feedback messages issued by MongoDB scores no marks,
- a report that contains any kind of processing errors scores no marks.

Task 4 (3.5 marks)
Implementation of validation with JSON schema

Download and unzip a file `solution4.zip`. You should get a file `solution4.js`.

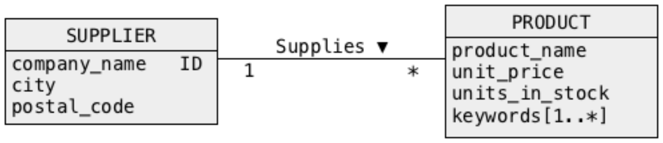
Start a command line interface mongo to MongoDB database server and process the following statements.

```
db.adminCommand( {setFeatureCompatibilityVersion:"4.2"} )
use test;
```

The first statement sets an appropriate compatibility level for application of JSON schema validator and the second statement sets an appropriate default database to be used with JSON Schema validator.

No report is expected from the actions listed above.

Consider the following conceptual schema of a database that contains information about suppliers and products.



An objective of this task is to create a new collection of documents `task4` that contains information represented by a conceptual schema above and such that the collection is validated with JSON schema validator.

Use a method `createCollection()` to create a collection of documents `task4` and use JSON schema validator to enforce the following constraints on the collection.

(1) Information about products must be nested within information about suppliers.
(2) Information about keywords describing products must be nested within information about products.
(3) The values associated with the key names `company name`, `city`, (within `SUPPLIER` class), `product name`, `keywords` (within `PRODUCT` class) must be of type string. All values are mandatory.
(4) The values associated with a key `postal code` (within `SUPPLIER` class must be of type integer in a range `0..9999`.
(5) The values associated with a key `units in stock` (within `PRODUCT` class) must be of type integer and must be positive.
(6) The values associated with a key `unit price` must be of type double and must be positive and less than `100.00`.
(7) A key `company name` is mandatory.

Next, insert into a collection `task4` two sample documents. The first document must pass all validations of the constraints listed above. The second document must fail a validation of only one of the constraints listed above. Provide information why a document fails a validation.

Write your solutions into a file `solution4.js` in the empty slots following a specification of each problem. Do not remove the comments with the specifications of queries and semicolons following the comments !

When ready create a report from processing of the queries in the following way.

Use `gedit` editor to open a file `solution4.js` with the specifications of the queries and implementations of the queries.

Select the entire contents of the file and Copy it into a buffer.

Open a new `Terminal` window and start mongo client in the following way.

```
mongo –port 4000
```

Paste the contents of the buffer copied earlier from `gedit` window in front of > prompt of `mongo` client. You may have to press Enter key to process the last query in a case when it is not followed by a newline control character.

Select the entire contents of the `Terminal` window and Copy&Paste it into a file `solution4.lst`. Save a file `solution4.lst`. Examine the contents of a file `solution4.lst` and make sure that it does not contain any errors.

Deliverables
A file `solution4.lst` with a report from processing of MongoDB script `solution4.js` with an implementation and testing of the validations.

And again, please remember that:
- a report without the listings of applied methods and feedback messages issued by MongoDB scores no marks,
- a report that contains any kind of processing errors except failed validation of the second document scores no marks.

Submission

Submit the files **solution1.lst**, **solution2.lst**, **solution3.lst** and **solution4.lst** through Moodle in the following way:

(1) Access Moodle at **http://moodle.uowplatform.edu.au/**
(2) To login use a **Login** link located in the right upper corner the Web page or in the middle of the bottom of the Web page
(3) When logged select a site **CSCI835/CSCI235 (S120) Database Systems**
(4) Scroll down to a section **SUBMISSIONS**
(5) Click at a link **In this place you can submit the outcomes of Assignment 3**
(6) Click at a button **Add Submission**
(7) Move a file **solution1.lst** into an area **You can drag and drop files here to add them**. You can also use a link **Add**...
(8) Repeat a step (7) for the files **solution2.lst**, **solution3.lst**, and **solution4.lst**.
(9) Click at a button **Save changes**
(10) Click at a button **Submit assignment**
(11) Click at the checkbox with a text attached: **By checking this box, I confirm that this submission is my own work,** ... in order to confirm the authorship of your submission.
(12) Click at a button **Continue**

End of specification