Web Service Assignment Report

Name: Rory McMahon

ID: 16167317

Module: CE4208 Distributed Systems

Date: 14/05/2020

Sections:

1. Database Design
2. Relationship Implementation
3. 3NF Justification
4. Web Service Design
5. How to Run the application

1. Database design

For this project I add 3 tables to the database, STUDENTS, MODULES and SCORES.

The table properties can be seen below.

STUDENTS		
STUDENT_ID (P)	FNAME	LNAME
INTEGER	VARCHAR	VARCHAR

MODULES			
MODULE_ID (P)	MODULE_NAME*	LECTURER	EXAM_DATE
INTEGER	VARCHAR	VARCHAR	DATE

SCORES			
SCORE_ID (P)	STUDENT_ID (F)	MODULE_ID (F)	SCORE
INTEGER	INTEGER	INTEGER	REAL

(P) – Primary key

(F) – Foreign key

The tables store student details, module details, and scores that students got in their exams for said modules. The SCORE table has 2 foreign keys, STUDENT_ID and MODULE_ID, which depend on the STUDENT and MODULES tables respectively. All data types that were requested are used in the tables, INTEGER, VARCHAR, DATE and REAL.

STUDENT_ID, MODULE_ID and SCORE_ID are all part of indexes which start at a specific number and are automatically incremented by 1 for every new entry.

* = The MODULE_NAME column is a UNIQUE column, as no two modules can have the same name.

2. Implementing One-to-One, One-to-Many and Many-to-Many

One to One

The one to one relationship can be seen in the MODULES table. One MODULE_ID can only have one MODULE_NAME associated with it, and vice versa. It is therefore a one to one relationship.

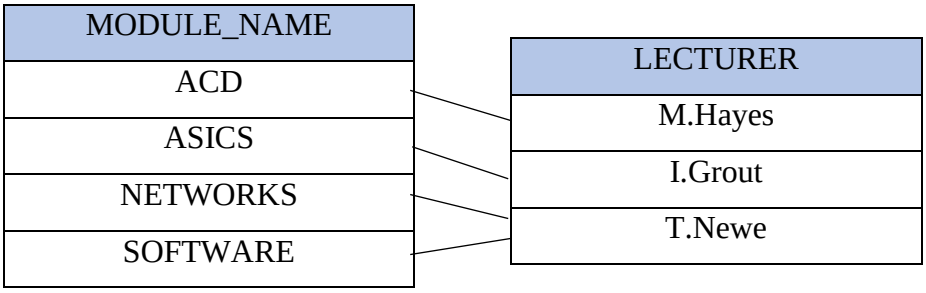
MODULE_ID	MODULE_NAME
4000	ACD
4001	ASICS
4002	NETWORKS
4003	SOFTWARE

When creating the MODULES table, I made MODULE_ID the primary key and MODULE_NAME unique so that there could be no duplicate entries. Therefore one ID can only have one Name and vice versa.

```
stmt.executeUpdate("CREATE TABLE MODULES"
        + "(MODULE_ID INTEGER PRIMARY KEY "
        + "GENERATED ALWAYS AS IDENTITY"
        + "(START WITH 4000, INCREMENT BY 1), "
        + "MODULE_NAME VARCHAR(255) NOT NULL UNIQUE, "
        + "LECTURER VARCHAR(255), "
        + "EXAM_DATE DATE)");
```

One to Many

The one to many relationship can be seen in the MODULES table, as one LECTURER can teach multiple modules, but one module cannot have multiple lecturers. As we can see from the example below, T.Newe is associated with 2 different modules. It is therefore a one to many relationship.

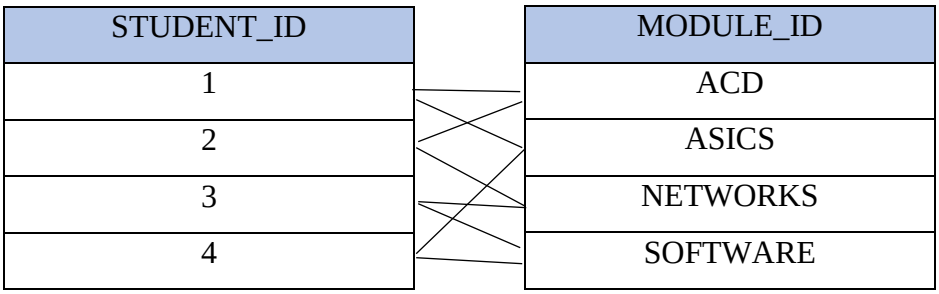


When creating the MODULES table, I made MODULE_ID the primary key and LECTURER not unique so that there could be multiple entries of LECTURER if they were associated with multiple modules.

```
stmt.executeUpdate("CREATE TABLE MODULES"
        + "(MODULE_ID INTEGER PRIMARY KEY "
        + "GENERATED ALWAYS AS IDENTITY"
        + "(START WITH 4000, INCREMENT BY 1), "
        + "MODULE_NAME VARCHAR(255) NOT NULL UNIQUE, "
        + "LECTURER VARCHAR(255), "
        + "EXAM_DATE DATE)");
```

Many to Many

The many to many relationship can be seen in the SCORES table, as one module can have multiple students associated with it, and one student can have multiple modules associated with him/her. It is therefore a many to many relationship.



When creating the SCORES table, I made SCORE_ID the primary key and STUDENT_ID and MODULE_ID the foreign keys that there could be multiple entries of STUDENT_ID and MODULE_ID if the student had multiple exams.

```
stmt.executeUpdate("CREATE TABLE SCORES"
        + "(SCORE_ID INTEGER PRIMARY KEY "
        + "GENERATED ALWAYS AS IDENTITY"
        + "(START WITH 1000, INCREMENT BY 1), "
        + "STUDENT_ID INTEGER, "
        + "MODULE_ID INTEGER, "
        + "SCORE REAL, "
        + "FOREIGN KEY(STUDENT_ID) REFERENCES STUDENTS(STUDENT_ID),"
        + "FOREIGN KEY(MODULE_ID) REFERENCES MODULES(MODULE_ID))");
```

3. Justification of 3NF

For a DB to be in the 3NF it must:

- Be in the 2nd Normal Form
- Contain columns that are non-transitively dependant on primary key

2NF:

The tables are in the 2NF, as the table stores columns that do not repeat, the primary key is used to identify each row, and all non-key columns are dependant on the primary key.

Non-Transitive Dependency:

To be non-transitive dependant, means that all columns are dependent on the primary key and no other columns in the table. This is true for each table I have created, as each row in each table is defined by its primary key and no other.

4. Web Service Design

I use REST for my web service. All web services can be found in the MyWebService.java file found in the Source Packages folder of the project. The services are implemented using the HTTP methods GET,POST,PUT and DELETE.

Below is a list of each method and the web service it implements:

Method	2.1 Web Service & Description
createTables()	Creates tables in derby. This method creates the 3 tables and adds some initial values to each table.
queryOneToOne()	Invokes one to one query. This method is used to invoke a query that selects module id's and module names from the MODULES table and displays them.
queryOneToMany()	Invokes one to many query. This method is used to invoke a query that selects module names and lecturers from the MODULES table and displays them.
queryManyToMany()	Invokes many to many query. This method is used to invoke a query that selects module id's and student id's from the SCORES table and displays them.
addOneToOne()	Adds to one to one table. This method is used to invoke a query that adds a module ID and a module name to the MODULES table based on the input from the user and displays the updated table.
addOneToMany()	Adds to one to many table. This method is used to invoke a query that adds a module ID, module name and a lecturer to the MODULES table based on the input from the user. It then displays the updated table.
addManyToMany()	Adds to many to many table. This method is used to invoke a query that adds a Score ID and Score to the SCORES table based on the input from the user and displays the updated table.

5. Running the application.

Running the web service

The web service can be run by right clicking on the project name *WebServiceApp* and clicking "Run". Instructions on how to choose which service to implement can be found on each page. The web service requires user input to go from service to service.

Running the application that invokes the web services

The application uses JSF and xhtml to invoke the web services. The application can be run by simply right clicking the *index.xhtml* file in the *Web Pages* folder and clicking "Run file".

The application requires no user input only a button click. Click the "Next" button to invoke each service one by one when running the application.

All web pages can be found in the *Web Pages* folder. The application steps follow this sequence: (all .xhtml files)

index

createTables

query1-1

query1-M

queryM-M

add1-1

add1-M

addM-M

finished