Database LayerCMPUT 391 Report

This project report describes the CMPUT391 Radiology Project. The report is divided in three sections based on the 3-tier architecture: Database Layer, Business Layer, and Presentation Layer.

# Database Layer

Our database layer is specifed in html/php/setup.sql, a modified version of setup.sql provided in the eclass project page. That being said, once the setup.sql is executed via sqlplus or other database client, the Databse Layer is specified. SQL statements in setup.sql are compose of schema definitions and function definitions. Schema definitions consists of the following:

* **persons**: To store personal information.
* **users**: log-in information.
* **family\_doctor**: to indicate who is whose family doctor.
* **radiology\_record**: to store the radiology records.
* **radiology\_image**: used in place of the pacs image table. Used to store the images in base64.
* **pacs\_images**: to store the pacs images
* **radiology\_record\_rt**:is a row type of radiology\_record schema. This is done so we can have radiology\_record as parameter in various functions, and row type of radiology\_record\_rt\_t.
* **radiology\_record\_rt\_t**: is a table of radiology\_record\_rt. This is done so we can return table of radiology\_record\_t.
* **persons\_rt**: persons row type. This is done so we can have persons tuple as a paremter to various functions.
* **ft01\_t**: fact table row type for the Data Cube function.
* **ft01\_t\_t**: is a table type of ft01\_t. This allows getDataCube01 return a data cube.

Function definitions consists of the following:

* **InsertRadiologyRecord**: this function is created for inserting radiology record. This is used instead of an ordinary insert to automate ID generation if second paramter autoID is set to true.
* **InsertPerson**: Same as insertRadiologyRecord but for persons schema.
* **GetRadiologyRecords**: given a userName, the function will return the accessible records based on the user associatd with the userName.
* **SearchWithKeywordsByRank**: given a userName, and keywords returns a table of radiology\_records, ordered by rank function.
* **SearchWithKeywordsByTime**: Given userName and a 'T' or 'F' string for descending, returns a table of radiology\_records, ordered by test\_date. This is descending if second paramter is 'T'.
* **SearchWithPeriodByTime**: Given a userName, lower bound date D1 and upper bound date D2, returns a table of radiology records that are taken [D1, D2]. Furthermore, it is descending (recent to least recent) if descending paramter is set to 'T'.
* **SearchWithKPByRank**: given a userName, keywords, and time boundary, returns result in order of the rank function.
* **SearchWithKPByRank**: given a userName, keywords, and time boundary, returns result in order of time.
* **GetDataCube01**: returns data cube, in weekly, monthly or annual.

All the Search\* functions aggregate GetRadiologyRecords for modularity. Thus the user of Search functions are assured to have restricted radiology\_record based on the given userName's user class.

Other SQL statement that are act as auxillary are the following:

* **persons\_seq**: aids in generating automated id for persons schema.
* **records\_seq**: aids in generating automated id user schema.

To allow ranking, the following indices are created:

* **recordIndexTestType**: index on test type of radiology\_record.
* **RecordIndexDiagnosis**: index on diagnosis of radiology\_record.
* **RecordIndexDescription**: index on description of radiology\_record.
* **PersonsIndexFirstName**: index on first\_name of persons schema.
* **PersonsIndexLastName**: index on last\_name of persons schema.

Now that we have discussed the Database Layer, we can turn on the Business Layer where the Database Layer is utilize and tested.

# Business Layer

Most of our business layer is found in html/php folder. That being said, the business layer is implemented in php. Also to ensure correctness in both database layer and business layer, tests are also implemented to modules in database layer.

### Database

To connect Business Layer and Database layer, *Database* class in Database.php acts as a Facade for the underlying database and singleton for easier access. It's a facade because Database class abstracts away from the underlying database. By having an interface that is apathetic to the underlying underlying database, it makes the switching to different database system minimal in code change. The following method makes this possible:

* createConnection(userName, password, connectionString).
* DestroyConnection().
* executeQuery(sqlStmt).
* executeQueryWithBindings(sqlStmt, array inBinding, array &$outBinding).

By overriding the following methods, one can switch to different database system in minimal code change.

### Database Schema Proxies

Database Schema Proxies are a bunch of place holder classes for the schema defined in the Database. The following are the proxy classes: Person, User, FamilyDoctor, RadiologyRecord, and PacsImages. All of these classes contains the same attributes as their Databse equivalent.

### Date

The Date class is another proxy, but not a schema proxy. It is a proxy class that represent the Date object built-in oracle. It also standardized our Date format to further reduce programming errors.

### Search

Encapsulate the Search module as demanded in the requirements. It provides various searches that meets the requiements while restricting output based on the given userName.

### UserManagement

A module for managing users that is only available for admin user class. It contains all the user, person, and record manipulation methods that is specified in the requirements.

### Login/Logout

Module that acts as an interface to Presentation Layer concerning logging in and out.

### Tests

Located in html/php/test. Modules found in that directory allows us to correctly verify our program is up to specification base on eclass site. It contains a bunch of unit tests, and a few integration testing. This is handy since we can verify correctness whenever we do refactoring or when debugging errors that resulted in the merging of Presentation Layer and Business Layer. Last but not least, we we were also able to test our Database layer, which is an easier option than having SQL Developer around.

# Presentation Layer