Introduction

This project is the implementation of a Radiology Information System (RIS). It is a 3 tier web application used to store and process radiology information with the additional capability to store images. In this report we will describe the implementation of the modules used in this application as well as information on the coding strategies and underlying architecture and design patterns.

Language and Technologies

Given the group members' history with and knowledge of the Java programming language, we chose build our application with Java Servlet and JavaServer Pages (JSP) technologies. For database access, our application makes use of the Java Database Connectivity (JDBC) API in communications with the Oracle database system hosted on the provided servers of the University of Alberta's Department of Computing Science. Our application was developed and debugged with the Eclipse IDE and designed for deployment on an Apache Tomcat web server. In order to match the conditions of the Computing Science lab machines, our system was developed using Java SE 6 and Tomcat 6.0.

Model-View-Controller Architecture

While certain modules make use of the simplified practice of connecting to and displaying information from the database directly through JSP, other modules in our application make use of a model-view-controller (MVC) architecture. The idea in these cases was to keep JSP for display purposes only in order to maintain readability within the HTML views. Java servlets acted as controllers, handling form processing and making requests to and retrieving information from the model when database information is required. Model consistency was handled by a single DataSource class (explained in detail below) which controllers could make requests from and submit information to. The separate datasource class allows our application to be used with a different database system (i.e. other than Oracle) if need be by simply reimplementing this single class.

The DataSource Class and Data Objects

The DataSource class represents our application's attempt to separate the concerns of database consistency and implementation from the concerns of displaying and working

with the data itself. The DataSource class was designed to act as a straightforward API for communicating with the database, allowing users of the DataSource to retrieve application-specific information without needing to worry about how the information was retrieved, sent or handled. Consequentially the DataSource class also works to convert database specific implementation objects (i.e. Result Set objects) to generic application-usable Java objects with appropriate names and attributes.

The java object representations of the database columns allows the information to be bundled up and passed to and from different application classes. Their information is accessible through simple-to-read, well-named getter methods and they are directly constructible from the sources that create them (i.e. form requests and result set objects). The goal with the data objects was to have our application easy to read and follow from the perspective of an object oriented developer. The result is more files, but with more readable code.

Methods of the DataSource Class

For brevity, this report contains only the methods regarding a single type of database object, in this case “person”. The actual implementation includes similar methods for “user”, “family doctor”, etc.

getNextPersonId()

Method for generating the next available Person ID in the person table of the database. Assumes the highest ID is the last entered ID. This method uses "`SELECT MAX(person_id) AS id FROM persons`".

submitPerson(Person)

Method for inserting a person object into the database. This method uses “`INSERT INTO persons VALUES (?,?,?,?,?,?)`”

getPersonList()

Method to retrieve all rows from the persons table and turn them into a list of Java person objects. This method uses "`SELECT * FROM PERSONS`".

getPersonById(Integer)

Method to retrieve a single row from the person database with the matching person ID and turn it into a single java person object; uses "`SELECT * FROM persons WHERE person_id = ?`".

updatePerson(Person)

Method for updating a single person row in the database based on the information of the provided java person object. If a person row exists with a matching ID, all columns are updated to match the information of the given person object; uses "`UPDATE persons SET first_name = ?, last_name = ?, address = ?, email = ?, phone = ? WHERE person_id = ?`".

deletePerson(Integer)

Method for deleting a single row from the person database. "`DELETE FROM persons WHERE person_id = ?`".

Login Module

Login Authentication

All registered users are able to login to the app with proper privileges and they can also modify their own personal information and change the password.

The login page gets the user's input of username and password. Once submitted, a Java servlet checks the correct password from the user table in database. If the username does not match with the password in the database, the system will reject the access request and redirect to the login page with a warning displaying the error description. If the authentication succeeds, user will be redirected to the proper homepage with certain privileges according to the user type stored in the database. We have 4 home pages corresponding to 4 user types: Admin, Patient, Doctor and Radiologist.

The following SQL statements are used to conduct database query:

Retrieve corresponding password to the input username: "`select password from users where user_name = '"+userName+"'"`

Retrieve corresponding user type to the input username: "`select class from users where user_name = '"+userName+"'"`

Retrieve corresponding person id to the input username: `select person`*`id from users where user`*`name = '"+userName+"'"`

This information is stored in the session after retrieved.

Modify profile and change password

Once the user successfully logs in, he will find a button called "Edit My Profile and Password" on his homepage. By clicking it, the user will be redirected to the profile and password editing page.

First of all, the personid will be retrieved from the users table. And then, all the personal information will be retrieved from the persons table using the personid.

Following SQL statements are used to conduct database query:

Retrieve corresponding person id to the input username: "SELECT person*id FROM users WHERE user*name="'+userName+""

Retrieve personal information according to the given person id: "SELECT * FROM persons WHERE person_id="+personId

Once user confirmed the changes by clicking the button "Update", ChangeProfileProcess.jsp will be called to write the updated personal information to the database. After that , user will be redirected to the corresponding home page according to his user type.

Following SQL statements are used to conduct database update: SELECT "some column" FROM persons WHERE person*id="person*id" FOR UPDATE.

Report Generating Module:

Any system administrator can get a list of all patients with a specified diagnosis for a given time period. For each patient, the list includes the name, address and phone number of the patient, and testing date of the first radiology record that contains the specified diagnosis. This module let administrators be able to get a report from the input diagnosis and time period.

Following SQL statements are used to conduct database query:

```
"SELECT firstname, lastname, address, phone, min(testdate) FROM
persons p, radiologyrecord r WHERE r.patientid = p.personid AND
r.diagnosis = ? AND r.testdate >= todate(?,'MM/DD/YYYY') AND
r.testdate <= todate(?,'MM/DD/YYYY') Group by patientid,
firstname, last_name, address, phone"
```

Upload Module

Implemented with the MVC architecture, the upload module uses JSP to retrieve and display information to the user, servlets to process the information and the Java object representations of the tables to pass information between them. The upload module is broken into two separate servlets: one for record uploading and editing and one for picture uploading. Once uploaded the pictures can be retrieved by passing a GET query string with the id of the image to "`/images/[size]?id=[image_id]`" where [size] can be regular, full size or thumbnail. All SQL statements are handled by the datasource class described earlier where instead of the person objects, PACS image and radiology record objects are passed to and retrieved from the data source.

User Management Module

The user management module controllers are broken into 3 sets of 3 types of servlets: one for each of "add", "edit" and "display" for each of "person", "user" and "family_doctor". The display servlets retrieve the relevant list of objects from the data source and pass them as a request parameter to the view which iterate through the objects, retrieving data from them and displaying that data. The add servlet renders an upload form whose information is handled by a post request to the display servlet. The edit servlet retrieves a data object of the corresponding row to be edited and passes the information to the edit view which populates a form with data corresponding to the row in the database. Submitting the form will update the object in the database to match the form.

Search Module

This module is used by all users to search the database for a list of all relevant records and to view the images that are related to the results. Any image will display its full size by clicking via the underlying image handler. Radiology records can be searched by a list of keywords and/or a specified time period. The results can be sorted in order of newest first, oldest first, or by a ranking system.

The file that handles all searches is the file *search.jsp* the file will create a graphical interface that the user can type in any keywords/date as well as select the

ordering option. Upon clicking the fetch button *search.jsp* will then extract all of the parameters entered by the user and parse them into an SQL query which will appear as followed

```
SELECT r.record_id, r.patient_id, r.doctor_id, r.radiologist_id, image_id, CONCAT(p1.first_name, CONCAT(' ', p1.last_name)) AS patient_name, CONCAT(p2.first_name, CONCAT(' ', p2.last_name)) AS doctor_name, CONCAT(p3.first_name, CONCAT(' ', p3.last_name)) AS radiologist_name, test_type, TO_DATE(prescribing_date, 'DD-MM-YY') AS prescribing_date, TO_DATE(test_date, 'DD-MM-YY') AS test_date, diagnosis, description FROM radiology_record r LEFT OUTER JOIN pacs_images i ON r.record_id = i.record_id, persons p1, persons p2, persons p3 WHERE p1.person_id = r.patient_id AND p2.person_id = r.doctor_id AND p3.person_id = r.radiologist_id
```

Then the program will extract the users ID

```
AND r.radiologist_id = 'ID'
```

The program will then parse any keywords input into the sql statement (if any)

```
AND (CONTAINS(p1.first_name, '" + keywords[0], 1) > 0 OR CONTAINS(p1.last_name, keywords, 2) > 0 OR CONTAINS(r.diagnosis, keywords,3) > 0 OR CONTAINS(r.description, keywords, 4) > 0)
```

Then we want to order the results by what the user wants

```
ORDER BY
```

If we want to order by the newest we add

```
test_date, record_id DESC
```

Similarly if we want to order by the oldest we add

```
test_date, record_id ASC
```

and if we want ranking we add

```
(6*(SCORE(1)+SCORE(2)) + 3*SCORE(3) + SCORE(4)),record_id DESC
```

The completed Query will allow us to find and display the results the user requested.

Data Analysis Module

This module is used by an administrator to generate an OLAP report for data analysis. This module allows the administrator to choose to display the number of images for each patient , test type, and/or period of time. It will generate a data cube information of the number of records for all the combinations of three columns, i.e., patient_name, test_type, and time(according to the values of column test_date). The module uses a very simple interface, The user will select either No, ALL or a specific

patient or test type as well as a style for the time ie: Weekly/Monthly/Yearly. The user is also able to select a specific year if desired. After pressing the fetch button the user is redirected to a view of the results, the user may change what is shown by pressing the back button and selecting the new parameters to be found.

This module uses a number of SQL commands to accomplish its function. First we create a view of the patients

```
CREATE OR REPLACE FORCE VIEW PATIENT (PATIENT_ID) AS SELECT distinct patient_id FROM radiology_record
```

Followed by a view of the test types

```
CREATE OR REPLACE FORCE VIEW TEST_TYPE (TEST_TYPE) AS SELECT distinct TEST_TYPE FROM radiology_record
```

and then a view of the time

```
create or replace VIEW time_id as SELECT ROW_NUMBER() OVER (ORDER BY year) as time_id,week,month,year as year FROM (select distinct to_char(TEST_DATE, 'IW') as week,to_char(TEST_DATE, 'MON') as month,to_char(TEST_DATE, 'YYYY') as year from radiology_record)
```

and a view of the images

```
create or replace view patient_number_image as select distinct p.PATIENT_ID,t.TEST_TYPE,COUNT(pi.image_id) as num,ti.time_id FROM PATIENT p, TEST_TYPE t,RADIOLOGY_RECORD rr, PACS_IMAGES pi,TIME_ID ti where p.PATIENT_ID = rr.PATIENT_ID AND t.TEST_TYPE = rr.TEST_TYPE AND pi.record_id = rr.record_id AND ti.week = to_char(rr.TEST_DATE,'IW') AND ti.year = to_char(rr.TEST_DATE,'YYYY') GROUP BY p.PATIENT_ID,t.TEST_TYPE,ti.time_id
```

We then create a table to store the information first drop it if it exists

```
DROP TABLE PATIENT_NUM_IMAGE_TABLE
```

then create it and insert the relevant info

```
CREATE TABLE PATIENT_NUM_IMAGE_TABLE(PATIENT_ID varchar(24),TEST_TYPE varchar(24),TIME_ID int, NUM int default 0) INSERT INTO PATIENT_NUM_IMAGE_TABLE (PATIENT_ID,TEST_TYPE,TIME_ID) SELECT p.PATIENT_ID,t.TEST_TYPE,ti.TIME_ID FROM PATIENT p,TEST_TYPE t,TIME_ID ti
```

We then merge the views into our table

```
Merge into PATIENT_NUM_IMAGE_TABLE p2 USING PATIENT_NUMBER_IMAGE p1 ON (p2.TEST_TYPE = p1.TEST_TYPE AND p2.PATIENT_ID =
```

```
p1.PATIENT_ID AND p2.TIME_ID = p1.TIME_ID) WHEN MATCHED THEN
UPDATE SET NUM = P1.NUM
```

As for viewing the data the SQL query is very sensitive to what the user wants to see however as an example if the user wished to see ALL patients, ALL test types with a weekly time period the SQL query will look like

```
SELECT CONCAT(p.first_name, CONCAT(' ', p.last_name)) as
name,TEST_TYPE.TEST_TYPE,TIME_ID.WEEK AS week,TIME_ID.YEAR AS
year,SUM(PATIENT_NUM_IMAGE_TABLE.NUM) FROM
PATIENT_NUM_IMAGE_TABLE,TIME_ID,PERSONS p,PATIENT p2 ,TEST_TYPE
WHERE p.person_id = p2.PATIENT_ID AND p2.PATIENT_ID =
PATIENT_NUM_IMAGE_TABLE.PATIENT_ID AND TEST_TYPE.TEST_TYPE =
PATIENT_NUM_IMAGE_TABLE.TEST_TYPE AND TIME_ID.TIME_ID
=PATIENT_NUM_IMAGE_TABLE.TIME_ID GROUP BY CONCAT(p.first_name,
CONCAT(' ', p.last_name)),TEST_TYPE.TEST_TYPE,week,year ORDER BY
year,week
```

In conclusion this module allows the administrator to view the system as an OLAP report.

Conclusion

This application is a simple Radiology Information System (RIS) that provides a computer information system that stores and processes the information for a radiology lab, including all the data regarding patients, and all medical images generated by various diagnostic and therapeutic equipments. The system is complemented with a Picture Archiving and Communication System (PACS) and provides a 3 tier system which is separated by the database server, the web server, and clients are running on different computer systems that are connected via the Internet.