WolfHospital Management System

For a hospital in North Carolina

CSC 540 - Database Systems

Project Report #3

Project Team U
Niharika Acharya, Ankit Manendra, Hasham Mukhtar, Vaibhav Nagvekar

Revisions

We have given revisions for report 2. These are at the end of the report after Questions 4 (High Level Design Decisions)

Transactions (Question 3)

Transactions:
In our application code, we have implemented transactions for two APIs.

1) releaseBed API
2) releasePatient API

Transaction for releaseBed API:

This transaction has been implemented in:
Class name - Beds
File name - Beds.java
Method name - releaseBed

1) This API the following two operations:
    a) Updates the bed table to set reserved=0,patient_id=NULL
    b) Updates the patients table to set ward_id=NULL and in_ward="no"
2) If any of the two operations results in an exception but the other one commits successfully then it will leave the database in an inconsistent state.
3) Hence we have grouped these two statements in a transaction by setting autoCommit=false.
4) If only one of the statements executes successfully but the other one throws an exception, it is caught and the transaction is rolled back by the conn.rollback() method of the connection object.
5) After roll back, the database is again left in a consistent state. In the finally block, we again set autocommit=true.

```
private static void releaseBed(){
      Connection conn = login.connection;
        try{

            System.out.println("Enter Bed Id: ");
            Integer bedID = sc.nextInt();
            String sqlCheckBed = "select * from beds where reserved=1 and id=" +
bedID ;
            ResultSet resCB = login.result;
            resCB = login.statement.executeQuery(sqlCheckBed);
            if (!resCB.isBeforeFirst()) {
                System.out.println("Bed already vacant");
                System.out.println("\n\nPress Enter To Continue");
```

```
                sc = new Scanner(System.in);
                sc.nextLine();
            } else {
                 conn.setAutoCommit(false);  //set auto commit to false
                String patID="";
                while (resCB.next()) {
                    patID = resCB.getString("patient_id");
                 }
                String sqlUpdateBed = "UPDATE beds SET reserved = 0,patient_id
=NULL,patient_ssn=NULL WHERE id =" + bedID;
                int id = login.statement.executeUpdate(sqlUpdateBed);
                if (id > 0) {
                    System.out.println("Bed Released!");
                    //update wards id in patients
                    if (patID!="") {
                        String sqlUpPatWard = "UPDATE patients SET
ward_id=NULL,in_ward='"+ "no" +"' where id=" + patID;
                        int idPat = login.statement.executeUpdate(sqlUpPatWard);
                    }
                } else {
                System.out.println("Bed not released!");
                System.out.println("\n\nPress Enter To Continue");
                sc = new Scanner(System.in);
                sc.nextLine();
                conn.commit();  // if both the updates are successful then commit
the statements together
                }
            }
        }catch (SQLException ex){
            ex.printStackTrace();
            if(conn!=null)
            {
                  try {
                        System.out.println("Transaction is being rolled back");
                        conn.rollback(); //if any exception occurs rollback the
transaction
                  }catch(SQLException se) {
                        se.printStackTrace();
                  }
            }
        }
        finally {
            try {
```

```
            conn.setAutoCommit(true); // set auto commit to true
            } catch (SQLException e) {
                  e.printStackTrace();
            }
        }
}
```

Transaction for releasePatient API:

This transaction has been implemented in:
Class name: PatientSQL
File name: PatientSQL.java
Method name: releasePatient

1) This API the following operations:
    a) Updates the end_date in the billing_accounts table
    b) Updates the accommodation fee in the biiling_accounts table
    c) Updates the end_date in the medical_records table
    d) Updates the end_date in the check_in_info table
    e) Updates the beds table and sets reserved=0 and patient_id=NULL
    f) Updates the patients table and sets the ward_id, processing_treatment_plan, completing_treatment_plan to NULL and in_ward to 0
2) If any of the operations results in an exception but the other one commits successfully then it will leave the database in an inconsistent state.
3) Hence we have grouped these statements in a transaction by setting autoCommit=false.
4) If only one of the statements executes successfully but the other throws an exception, it is caught in the catch block and the transaction is rolled back by the connection.rollback() method of the connection object.
5) After roll back, the database is again left in a consistent state. In the finally block, we again set connection.autocommit=true.

```
static void releasePatient(int pid, String startDate, String endDate)
            throws ParseException, ClassNotFoundException {

        int id = 0;
        PreparedStatement ps1 = null;
        try {
```

```
            connection.setAutoCommit(false);      //set auto commit to false

            // Sets end date for billing account for this patient
            ps1 = connection.prepareStatement(
                    "UPDATE billing_accounts SET end_date = ?, settled = 1 WHERE
patient_id= ? AND start_date = ?;");
            ps1.setString(1, endDate);
            ps1.setInt(2, pid);
            ps1.setString(3, startDate);
            id = ps1.executeUpdate();

            if (id > 0) {
                System.out.println("Billing Account Updated");
            } else {
                System.out.println("Billing Account Not Updated");

            }

            // Updates accommodation fee for this patient and start date
            BillingAccount.updateAccomodationFee(pid, startDate);

            // sets end date for medical records for this patient
            ps1 = connection.prepareStatement(
                    "UPDATE medical_records SET end_date = ?, active = 0 WHERE
patient_id= ? AND start_date = ?;");
            ps1.setString(1, endDate);
            ps1.setInt(2, pid);
            ps1.setString(3, startDate);
            id = ps1.executeUpdate();

            if (id > 0) {
                System.out.println("Medical Record Updated");
            } else {
                System.out.println("Medical Record Not Updated");

            }
            // sets end date for check in info for this patient
            ps1 = connection
                    .prepareStatement("UPDATE check_in_info SET end_date = ?
WHERE patient_id= ? AND start_date = ?;");
            ps1.setString(1, endDate);
            ps1.setInt(2, pid);
            ps1.setString(3, startDate);
```

```
            id = ps1.executeUpdate();

            if (id > 0) {
                System.out.println("Check In Info Updated");
            } else {
                System.out.println("Check In Info Not Updated");

            }

            // Removes patient from the bed they are currently assigned to
            ps1 = connection.prepareStatement("UPDATE beds SET reserved = 0,
patient_id = NULL WHERE patient_id= ?;");
            ps1.setInt(1, pid);
            id = ps1.executeUpdate();

            if (id > 0) {
                System.out.println("Patient Removed From Bed");
            } else {
                System.out.println("Patient Not removed from bed");

            }
            // Sets ward id to null and in ward to no in the patient table for
this patient
            ps1 = connection.prepareStatement(
                    "UPDATE patients SET ward_id = NULL,
processing_treatment_plan = 0, completing_treatment = 0, in_ward = 'no' WHERE id=
?;");
            ps1.setInt(1, pid);
            id = ps1.executeUpdate();

            if (id > 0) {
                System.out.println("Patient Updated");
            } else {
                System.out.println("Patient Not Updated");

            }

            ps1.close();
            connection.commit();
            /*if both the statements execute successfully then commit the
transaction*/
        } catch (SQLException oops) {
            System.out.println("Connection Failed! Check output console");
```

```
            oops.printStackTrace();
            if (connection != null)
            {
                try
                {
                    System.out.println("Transaction is being rolled back");
                    connection.rollback();    // if any exception occurs rollback
the transaction
                } catch(SQLException se) {
                    se.printStackTrace();
                }
            }
        }
        finally {
            try {
                        connection.setAutoCommit(true);       // set auto commit
to true
                  } catch (SQLException e) {
                        e.printStackTrace();
                  }
        }
    }
```

High Level Design Decisions (Question 4)

Compiling and Running Code

- Download/extract code
- Make sure mariadb jar file is in project dependencies
- Run code with:
    - javac *.java
    - java login
- Use below credentials to log into system:
    - Username: admin
    - Password: admin

Code Structure

We separated all of our jave files and make them public and static so they can be reused among each other. We felt this would reduce the number of duplicate code and copying and pasting in the program. This also significantly cut the total number of lines in each java file and made each of them more

manageable. We took all the APIs from the previous reports and added them to the java files that would affect those certain database tables (ex: assignBed API would go in the Bed.java file). Below is a list of all the files we created and general description of what they are for.

- We have a java file that takes a username and password to login in to the system. This file also holds the JDBC connection objects (Connection, ResultSet, Statement) that the rest of the application will call. This is done by making these objects public and static.
    - login.java
- We created a java file that lists all the tables that user has access to in the system
    - HomeMenu.java
- We created separate java files that has a menu for each table in our database
    - PatientMenu.java, StaffMenu.java, CheckInMenu.java, Beds.java, BillingAccount.java, MedicalRecord.java, reportMenu.java, Test.java, Treatment.java, TestForPatients.java, Wards.java
- We created a separate java file that contains common actions for each table like View, Edit, Delete, Add. This reduced copying and pasting of code between menu files. If there was a table that needed more options, it would be put in the code after calling this file.
    - CommonActionMenu.java
- We created a java file that calls the home menu and table menus and is the main user navigation hub
    - UsersInterface.java
- We also created a class that checks for space in the hospital
    - CheckHospitalSpace.java
- For some of the menus, we created separate java files that handles the SQL requests for those tables. Separating them allows them to be reused among other menus if need be
    - PatientSQL.java, StaffSQL.java, CheckInSQL.java, CheckHospitalSpaceSQL.java
- For the rest of the menus, the SQL code was put in the same menu files
    - Beds.java, BillingAccount.java, MedicalRecord.java, reportMenu.java, Test.java, Treatment.java, TestForPatients.java, Wards.java

Team Roles

Everyone on the team contributed when it came to database design, writing documentation for the reports, and writing code. We split the work evenly among the four of us, but there were people in charge of certain roles that made sure the work was being done (prime and backup) during each part (1 to 3). This is listed below:

Report 1

Ankit:

- Prime Database Designer / Administrator

Niharika:

- Database Designer

Vaibhav:

- Database Designer

Hasham:

- Backup Database designer

Report 2

Ankit:

- Database designer
- Backup application programmer

Niharika:

- Prime Database Designer / Administrator

Vaibhav:

- Backup database designer

Hasham:

- Database designer
- Prime application programmer

Report 3

Ankit:

- Application programmer
- Software engineer
- Backup software engineer

Niharika:

- Application programmer
- Software engineer
- Prime application programmer
- Backup test plan engineer

Vaibhav:

- Application programmer
- Software engineer
- Backup application programmer

Hasham:

- Application programmer
- Prime Software engineer
- Prime Test Plan Engineer