5200 Project Report
Group Name: MaY Name: Yun Ma

README section

Table of Contents

Overview

This is an Java-based GUI application designed to interact with an SQL database.
It allows managing level users to access the Aquarium Management System to perform CRUD (Create, Read, Update, Delete) operations through a user interface.
The Aquarium Management Database System includes entities: animals, staff, tank, care tasks (i.e. feeding, cleaning, training), schedule (scheduled time and staff for each task), and aquarium events (i.e. feeding time or performances session).

Features

* Connect to MySQL database
* Perform CRUD Operations
 * Create new rows when clicking create button
 * Read and display data by default
 * Update existing data by typing new data in input box and clicking update button
 * Delete row according to table id

Technical Specifications

* Programming Language: Java
* GUI Framework & libraries: Java Swing, AWT
* Database: MySQL
* Database Connector: JDBC
* Software: MySQL Workbench, IntelliJ

Setup Instructions

Prerequisites
1. Set up a SQL database (MySQL) with the necessary schema and data.
2. Install Java (version 11 or higher)
3. Download the JDBC database driver (mysql-connector-j-9.1.0)
Steps
1. download mysql-connector-java-9.1.0.jar and add the jar file as database connector library to the project's classpath
2. Update database connection details (URL, USER, PASSWORD) in DatabaseConfig class
3. Run the application from IDE

Lessons Learned

1. Technical expertise

I learned how to create a database from scratch: designing schemas, defining tables and relationships using MySQL workbench. I also understood the importance of normalizing databases to eliminate redundancy and improve query efficiency. By creating the database, I practiced configuring various database components such as primary key, foreign key and composite primary key to ensure the integrity of the data.

I also learned how to implement CRUD operations as stored procedures in MySQL and utilize them through JDBC. By writing modular stored procedures in MySQL, I learned how to reuse SQL code and reduce direct SQL execution from the application layer. I also learned how to use JDBC callable statements to interact with stored procedures in Java which reduce SQL codes in the application as well. By implementing error handling for the application, I learned how to detect exceptions and roll back transactions.

Front end wise, I learned how to use Java Swing and AWT frameworks to design a graphic UI that incorporates table presentation and utilizing buttons to realize CRUD operations.

2. Insights

Time management wise, I learned the importance of prioritizing database design. From the feedback I got from the project proposal, I modified the origin database design to ensure the current design is specific to a type of user and doesn't contain redundant relationships and tables. Spending time understanding the purpose of designing the database helps writing the SQL scripts later on. I learned the advantages of modularizing SQL procedures. By wrapping the CRUD operations in stored procedures, it is much easier to call when integrating with the front end component.

It cost more time than debugging when not able to detect the error is located in SQL script or front end. I learned to spend more time to test each stored procedure using simple SQL commands before integrating with the front end component.

Data domain wise, I learned how to prevent errors and reliance on frontend validation by implementing constraints for each field in the data layer.

3. Alternative design / approaches to the project

An alternative approach to the database component of the project is to use MongoDB for database schema design. Instead of representing as relational tables, MongoDB represents documents as JSON-like objects. And all the related data will be stored together within one document. To integrate that with the front end, I could use MongoDB's java driver for connection. MongoDB functions such as collection.find() and collection.insertOne() can be used to handle CRUD operations.

4. Document any code not working in this section

To ensure the data is updated when the CRUD function in front end is called, JTable components must have the necessary structure. The original design that failed had JTable components added before the data was fully updated and defined. The current design creates the table dynamically and adds it to the JTabbedPanel later in the code so that UI is in response to the user actions.

```
//public class AquariumGUI extends JFrame {
//  // db connection
//  private static final String URL = "jdbc:mysql://localhost:3306/AquariumSystem_MaY";
//  private static final String USER = "root";
//  private static final String PASSWORD = "my1998";
//
//  // tables
```

```
//  private JTable staffTable;
//  private JTable tankTable;
//  private JTable animalTable;
//  private JTable careTaskTable;
//  private JTable scheduleTable;
//  private JTable eventTable;
//
//  public AquariumGUI() {
//    JFrame frame = new JFrame("Aquarium GUI");
//    frame.setSize(1650, 1080);
//    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//    JTabbedPane tPane = new JTabbedPane();
//
//    JPanel p1 = new JPanel(new BorderLayout());
//    staffTable = new JTable();
//    JScrollPane scrollPane1 = new JScrollPane(staffTable);
//    p1.add(scrollPane1, BorderLayout.CENTER);
//    tPane.addTab("Staff Table", p1);
//
//    JPanel p2 = new JPanel(new BorderLayout());
//    tankTable = new JTable();
//    JScrollPane scrollPane2 = new JScrollPane(tankTable);
//    p2.add(scrollPane2, BorderLayout.CENTER);
//    tPane.addTab("Tank Table", p2);
//
//    // Animal table tab
//    JPanel p3 = new JPanel(new BorderLayout());
//    animalTable = new JTable();
//    JScrollPane scrollPane3 = new JScrollPane(animalTable);
//    p3.add(scrollPane3, BorderLayout.CENTER);
//    tPane.addTab("Animal Table", p3);
//
//    frame.add(tPane);
//    frame.pack();
//    frame.setVisible(true);
//
//    // default content
//    displayTable("GetStaff", new String[]{"ID", "Name", "Role", "Contact"}, staffTable);
//
//    tPane.addChangeListener(e -> {
//      int selectedIndex = tPane.getSelectedIndex();
//      switch (selectedIndex) {
//        case 0 -> displayTable("GetStaff", new String[]{"staff_id", "staff_name", "staff_role", "contact_info"}, staffTable);
//        case 1 -> displayTable("GetTank", new String[]{"tank_id", "location", "water_type", "temperature", "capacity"},
tankTable);
//        case 2 -> displayTable("GetAnimal", new String[]{"animal_id", "scientific_name", "common_name", "tank_id",
"age", "diet"}, animalTable);
//        case 3 -> displayTable("GetCareTask", new String[]{"task_id", "task_description", "frequency",
"special_requirement"}, careTaskTable);
//        case 4 -> displayTable("GetSchedule", new String[]{"schedule_id", "task_id", "staff_id", "animal_id",
"scientific_name", "scheduled_date", "scheduled_time"}, scheduleTable);
//        case 5 -> displayTable("GetAquariumEvent", new String[]{"event_id", "event_name", "event_date",
"event_time", "event_location", "event_description", "staff_id", "animal_id"}, eventTable);
//      }
```

```
//    });
//  }
//  private void displayTable(String procedureName, String[] columnNames, JTable table) {
//    try (Connection connection = DriverManager.getConnection(URL, USER, PASSWORD)) {
//      CallableStatement stmt = connection.prepareCall(STR."{call \{procedureName}()}",
//            ResultSet.TYPE_SCROLL_INSENSITIVE,
//            ResultSet.CONCUR_READ_ONLY);
//      ResultSet rs = stmt.executeQuery();
//
//      // count number of rows
//      rs.last();
//      int rowCount = rs.getRow();
//      rs.beforeFirst();
//
//      // prep data for table
//      String[][] tableData = new String[rowCount][columnNames.length];
//      int i = 0;
//      while (rs.next()) {
//        for (int j = 0; j < columnNames.length; j++) {
//          tableData[i][j] = rs.getString(j + 1);
//        }
//        i++;
//      }
//
//      // update table model with retrieved data
//      table.setModel(new javax.swing.table.DefaultTableModel(tableData, columnNames));
//
//    } catch (SQLException e) {
//      e.printStackTrace();
//    }
//  }
//  public static void main(String[] args) {
//    SwingUtilities.invokeLater(AquariumGUI::new);
//  }
//}
//
```

Future work

1. Planned uses of the database

There are some functionalities that can be realized to assist managing level staff to better monitor the aquarium system. Utilizing the current database application as the foundation, we can add monitoring functionalities to monitor animals' health, tank environment, staff performance or event preparation. By accessing the required data from the database using Read operation and generating monitor reports. The monitor data can be used for behavioral research of the animals or to improve animals' conditions.
The database can also be used to provide automated notification. For example, staff can receive alerts when there are oncoming care tasks or health monitoring notification. The database can also be used to manage visitor booking, creating a system for visitors to book reservations and being able to see which animals are available to be visited.

2. Potential areas for added functionality

To implement functionality to monitor entity condition, first access the careTask table and related Animal table data, we can monitor the animal's care conditions. We can use JOIN between careTask and Animal

tables and get related data. By accessing the relevant data, we can generate reports on status of the animals and track changes over time.

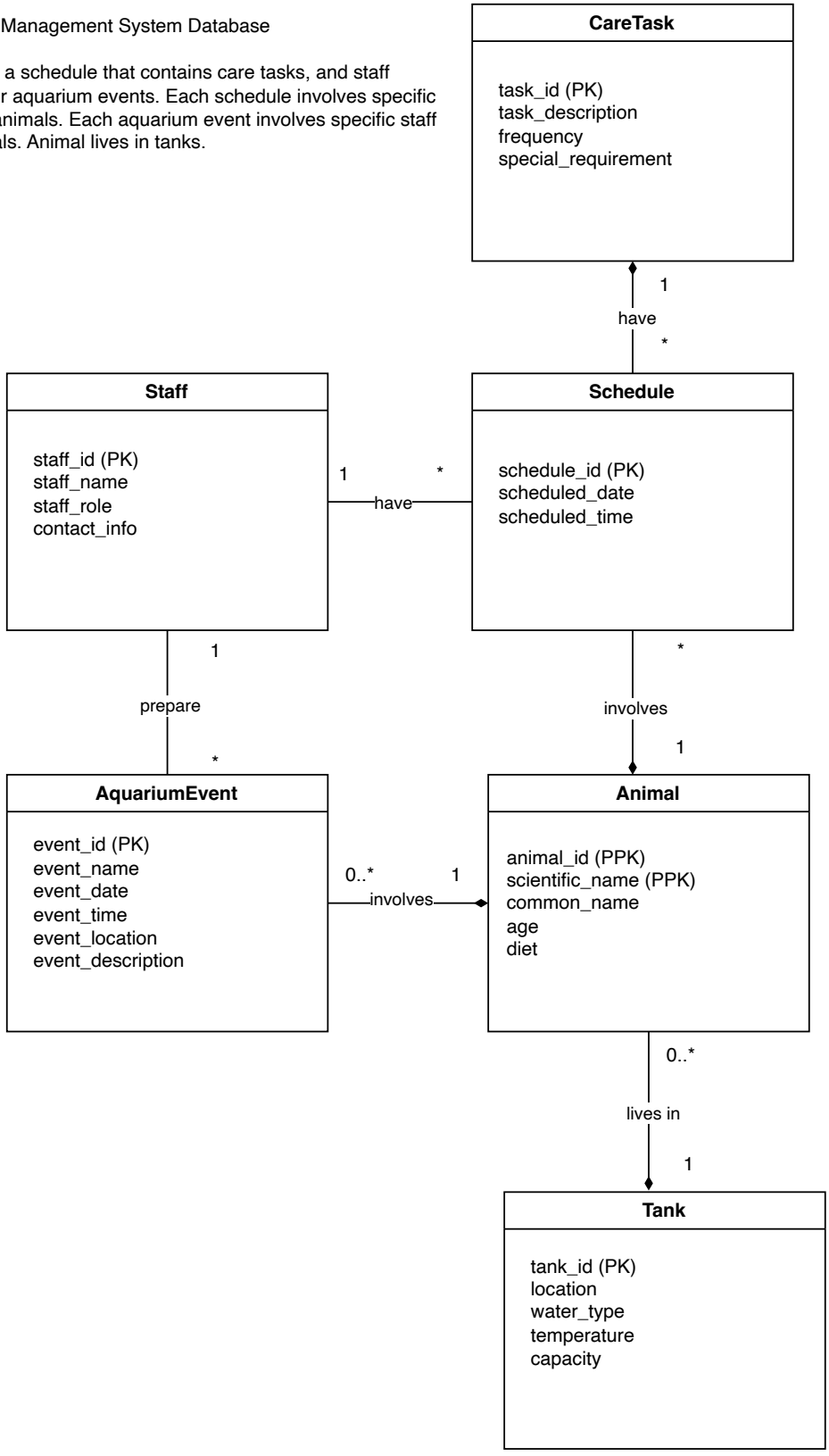
Based on the current Schedule table, we can also implement a more thorough schedule system for staff to rotate based on the care tasks. By extending the current Schedule table, we associated a care task type with each ro, a rotation status, and a staff member assigned. Also we need to define the rotation logic in the staff table.

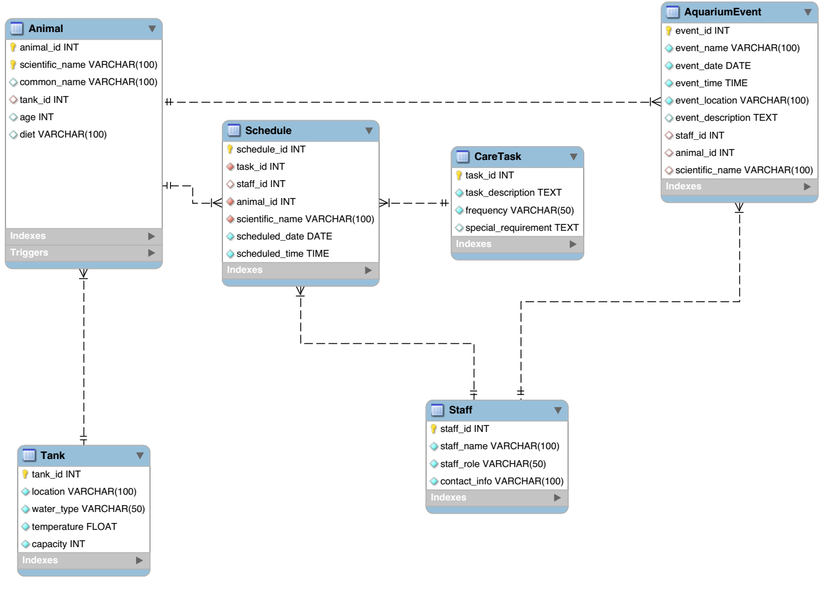
To add a staff notification function, we can add an alert class that relates to staff and careTask tables. And then use SQL commands to create the alert (i.e. INSERT INTO Alert(attributes) SELECT (attributes) FROM Schedule WHERE (specific schedule date time)).

To create a visitor booking system, we can create a table Reservation that allows visitors to schedule appointments on certain animals or events.To check availability, we can use LEFT JOIN on the animal/event table and the reservation table to book a specific animal or event.

Aquarium Management System Database

Staff have a schedule that contains care tasks, and staff prepare for aquarium events. Each schedule involves specific staff and animals. Each aquarium event involves specific staff and animals. Animal lives in tanks.




Animal
animal_id INT
scientific_name VARCHAR(100)
common_name VARCHAR(100)
tank_id INT
age INT
diet VARCHAR(100)
Indexes
Triggers
Schedule
schedule_id INT
task_id INT
staff_id INT
animal_id INT
scientific_name VARCHAR(100)
scheduled_date DATE
scheduled_time TIME
Indexes
CareTask
task_id INT
task_description TEXT
frequency VARCHAR(50)
special_requirement TEXT
Indexes
AquariumEvent
event_id INT
event_name VARCHAR(100)
event_date DATE
event_time TIME
event_location VARCHAR(100)
event_description TEXT
staff_id INT
animal_id INT
scientific_name VARCHAR(100)
Indexes
Tank
tank_id INT
location VARCHAR(100)
water_type VARCHAR(50)
temperature FLOAT
capacity INT
Indexes
Staff
staff_id INT
staff_name VARCHAR(100)
staff_role VARCHAR(50)
contact_info VARCHAR(100)
Indexes

Commands for user interaction:
Create new row -> create button
Read -> when clicking different table tab, display all data by default
Update selected row by table Id -> update button
Delete selected row by table Id -> delete button