Pet-Store Database

Mary Barsoum, Brooke Podlipec

Abstract:
We aim to To help pet stores manage both the commercial and animal welfare sides of their business. We want to create a one-stop system for pet stores to view all information related to their animals such as food, toys, and living spaces, as well as information about employees. We have implemented our database using MySQL, and have connected it to a frontend React project using an Express server. Both our React project and our SQL database were thoroughly tested. The project was developed using the Visual Studio Code IDE and the code was shared between partners using Github. Our project is capable of adding, updating, and deleting records in the database, as well as listing certain sets of records using Select. Overall, we successfully built a managerial system for a Pet Store by implementing an ERD, converting it into a relational schema, and using MySQL to create a database.

Keywords: MySQL, React, SQL, Express, Relational Schemas, Aggregation, NodeJS, SQL, Pet Store, Management System, Data Relations.

Overview

Pet stores are very diverse, not only in the types of animals present but also in the items they sell for each type of animal. There are many facets to a pet store: the animals available, the types of food and toys for each animal, and the employees. It is important to keep all of that data organized and accessible.

On top of that, a good database could provide a lot of insight into how the company is currently run, and provide methods for improving upon one's business in the future. Inventory data is important for the maintenance of a store's stock, and more advanced data analytics could even provide forecasting for future supply and demand (Importance of Inventory Databases for Retail). Employee data is also important for company analytics, as well as for figuring out how to improve company culture and employee retention (7 Best Practices For Employee Data Management) Through careful analysis of both the Petco and PetSmart website, we pieced together what features that a good pet Store database would need, both in terms of managing its employees and through the features commonly found on both sites.

We aim to To help pet stores manage both the commercial and animal welfare sides of their business. We want to create a one-stop system for pet stores to view all information related to their animals, such as food, toys, living spaces, and employee information. Having a platform as such would make pet store management much more efficient and could provide benefits to the customers as well.

As a team, our goal is to understand how complex retail environments such as pet stores handle their data. On top of this, we wish to provide a relational database application that makes life easier for pet stores! We also think that this type of project would be a great opportunity to fully understand the capabilities of SQL and a database management system.

Background Material

- Beginners CRUD Tutorial - ReactJS, MySQL, NodeJs: https://www.youtube.com/watch?v=re3OIOr9dJI
- Beginners CRUD Tutorial/Update and Delete - ReactJS, MySQL, NodeJS: https://www.youtube.com/watch?v=AohARsUlwQk
- The Basics of MySQL in 23 Easy Steps: https://www.youtube.com/watch?v=Cz3WcZLRaWc
- Create data table using MUI: https://dev.to/cubejs/react-data-table-with-material-ui-and-a-spark-of-joy-50o1
- React Docs: https://reactjs.org/docs/getting-started.html
- MySQL Docs: https://dev.mysql.com/doc/mysql-getting-started/en/
- React Data Table Tutorial Series: https://www.youtube.com/watch?v=YwP4NAZGskg&list=PLC3y8-rFHvwgWTSrDiwmUsl4ZvipOw9Cz
- Gliffy: https://www.gliffy.com/
- Lucidchart Relational Schema Formatting: https://www.lucidchart.com/pages/examples/database-design-tool
- Performing database tests on SQL databases: https://circleci.com/blog/relational-db-testing/
- Bootstrap CSS Framework - Full Course for Beginners: https://www.youtube.com/watch?v=-qfEOE4vtxE
- React to MySQL tutorial: https://www.bezkoder.com/react-node-express-mysql/
- Create a Database in mysql: https://www.mysqltutorial.org/mysql-create-database/

- Importance of Inventory Databases for Retail: https://smallbusiness.chron.com/importance-inventory-databases-retail-40269.html

- Petco: https://www.petco.com/shop/en/petcostore?cm_mmc=PSH%7cGGL%7cCCY%7cCCO%7c0%7c0%7c8NKj5o5xHVGnrqzKb8DLi2%7c58700008123842969%7cpetco%7c0%7c0&gclid=CjwKCAjwjMiiBhA4EiwAZe6jQ9S8cx-y87JY026_T4VSOH9JSY3So6bP1E3xmyZSvkP3g1e8oJtYFBoCs78QAvD_BwE&gclsrc=aw.ds
- PetSmart:https://www.petsmart.com/?gclsrc=aw.ds&gclid=CjwKCAjwjMiiBhA4EiwAZe6jQx8U6T2FcjXbD0NTGEP2I_nZMafaR2htX_k0pBHIJCGWHXIX9gK1phoCIEUQAvD_BwE&gclsrc=aw.ds
- 7 best Practices For Employee Data Management: https://engagedly.com/best-practices-for-employee-data-management/
-

We followed the tutorials listed to create a React project, set up a MySQL database, and create an Express server to connect the two. We used tutorials for Bootstrap.js and MUI to create a componentized front-end interface. We used Gliffy to create our ERD.

Our Approach

Approach and Design

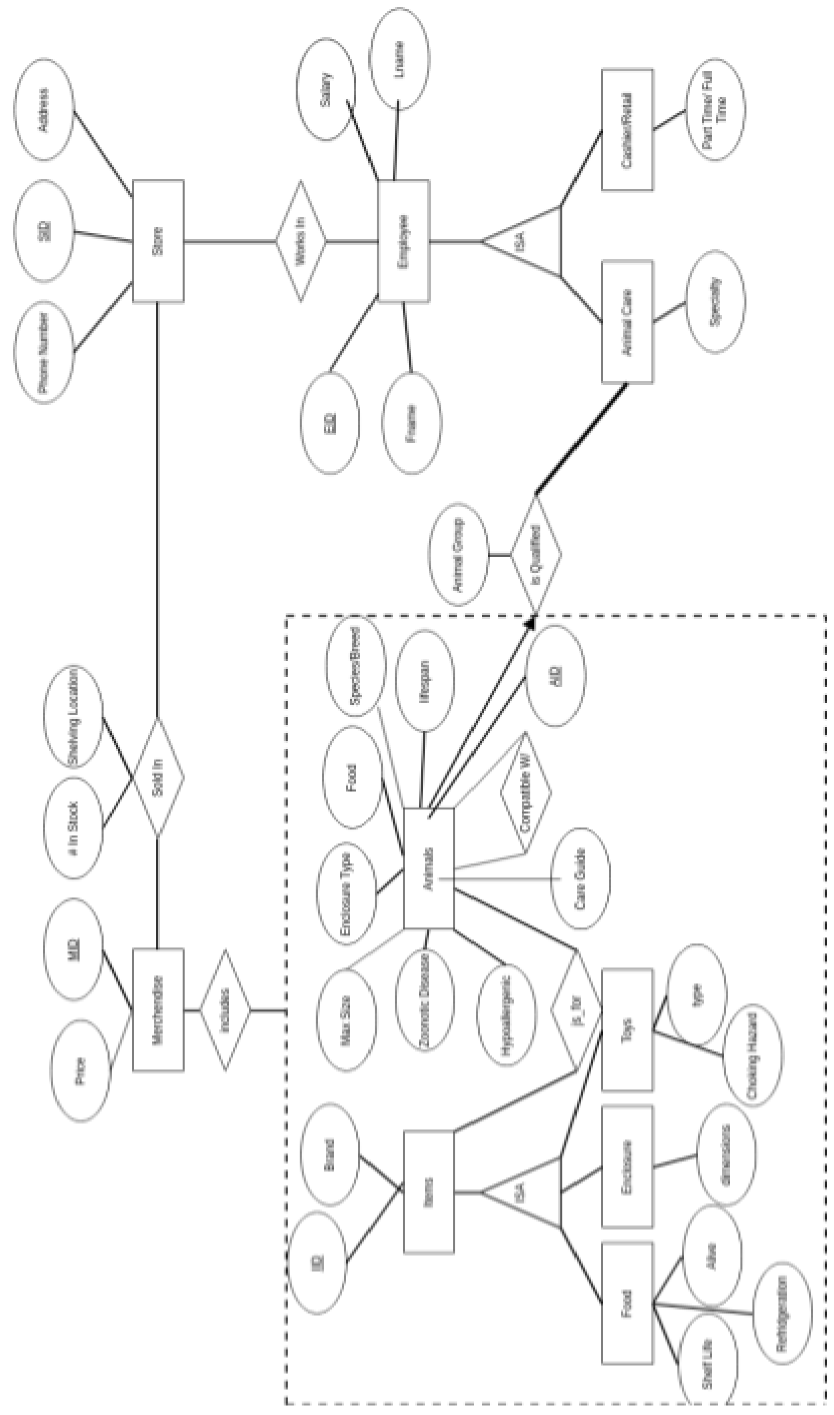


Figure 1. Pet store ERD

d

Table 1. Relational Schema, adapted from ERD in figure 1.

Store(SID: string, Phone Number:string, Address:String)
Primary Key: SID
No Candidate Key
No Foreign Keys

Works_In(SID: string, EID:string)
Primary Key: (SID, EID)
No Candidate Keys
Foreign Key: SID references Store(SID)
Foreign Key: EID references Employee(EID)

Employee(EID:string, Salary:real, Lname:string, Fname:string,)
Primary Key: EID
No Candidate Key
No Foreign Key

Animal_Care(EID:string, Salary:real, Lname:string, Fname:string, Specialty:string)
Primary Key: EID
No Candidate Key
Foreign Key: EID reference Employee (EID)

Cashier/Retail(EID:string, Salary:real, Lname:string, Fname:string, Part Time/Full Time:string)
Primary Key: EID
No Candidate Key
Foreign Key: EID References Employees (EID)

Sold_in(#in Stock: number , Shelving_Location: string, MID: string, SID:string)
Primary Key: (MID, SID)
No Candidate Key
Foreign Key: MID, SID

Merchandise(Price:real, MID: string)
Primary Key: Mid
No Candidate Key
No Foreign Key

Includes(MID:string, ,IID:string, AID:string)
Primary Key: (MID, IID,AID)
No Candidate Key
Foreign Key:MID, (IID, AID)

Animals(Hypoallergenic:Boolen, Max_Size: Real, Enclosure_Type: String, Food: String, Species: String, Lifespan: string, <u>AID: string</u>, Care Guide: string)
Primary Key: AID
No Candidate Key
No Foreign Key

Items(<u>IID: string</u>, Brand: string)
Primary Key: IID
No Candidate Key
No Foreign Key

Food(<u>IID: string</u>, Brand: string, Shelf_Life: string, Refrigeration: Bool, Alive: Bool)
Primary Key: IID
No Candidate Key
Foreign Key: IID References Items

Enclosure(<u>IID: string</u>, Brand: string, dimensions: string)
Primary Key: IID
No Candidate Key
Foreign Key: IID references Items

Toys(<u>IID: string</u>, Brand: string, type: string, choking_hazard: bool)
Primary Key: IID
No Candidate Key
Foreign Key: IID references Items

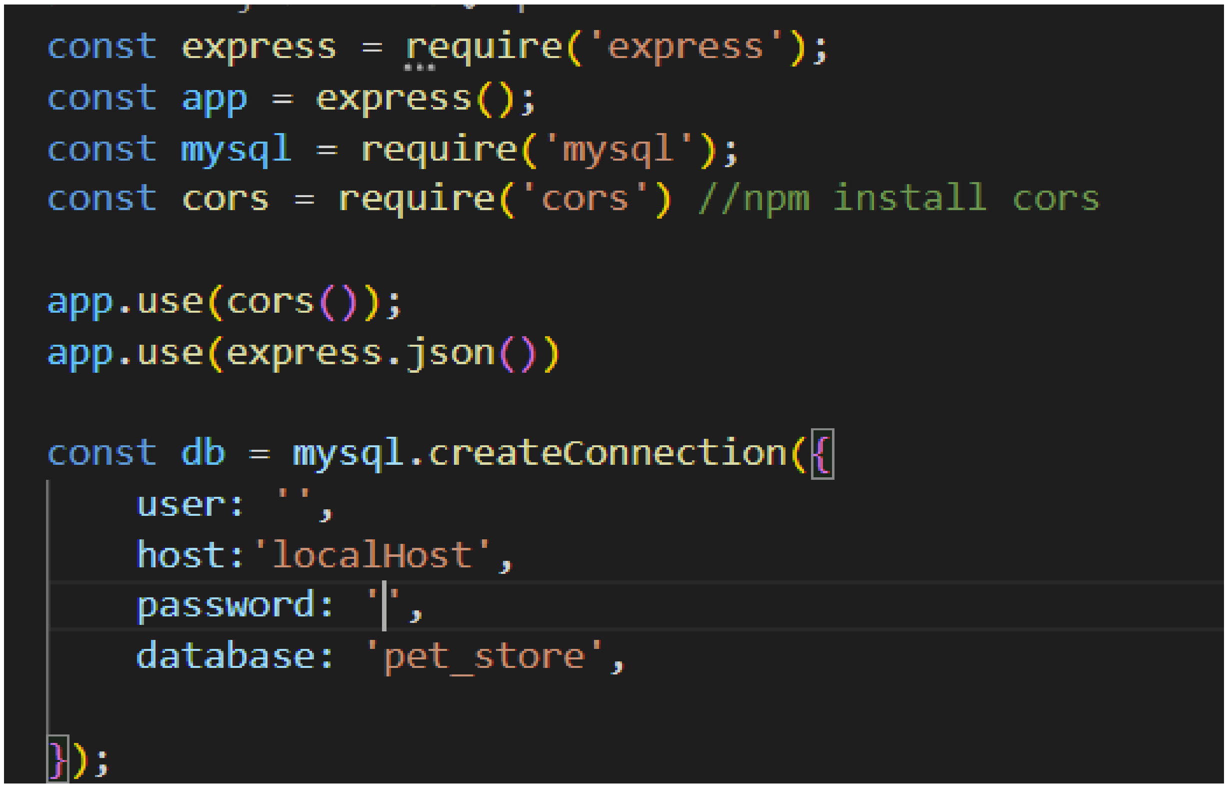
Is_For(<u>IID: string</u>, <u>AID: string</u>)
Primary Key: (IID, AID)
No Candidate Key
Foreign Key: IID references Items
Foreign Key: AID references Animals

Compatible_With(<u>AID: string</u>)
Primary Key: AID
No Candidate Key
Foreign Key: AID references Animals

Is_Qualified (AID: string, AnimalCareID:string)
Primary Key: (AID:string, AnimalCareID:string)
No Candidate Key
Foreign Key: Animal Care ID references Animal_Care(EID)
Foreign Key: AID references Animals

Figure 1 and Table 1 show our ERD and relational schema, respectively, that map out the pet store database for our project. In the ERD A*nimals* and *Items* are aggregated with the *Include* relation under *Merchandise* as it is a flexible way to model the relationships between them. *Employee* has an ISA with two subclasses *Cashier/Retail* and *Animal Specialist* which represent different types of workers at the store. *Items* is also an ISA with three subtypes of goods typically sold at a pet store: *Food*, *Enclosure*, and *Toys*. *Items* relate to *Animals* with an *Is_For* relation that allows one to determine which *Items* are for which *Animals*. The self-relation *Compatible_With* allows for the specification of what *Animals* should or should not be kept together (e.g. a larger predatory fish placed in the same tank as a small bottom feeder). The *Is_Qualified* relation specifies which *Employees* are qualified to take care of what *Animals* (e.g. fish specialists). The *Sold-In* relation specifies what *Merchandise* is sold in what *Stores*. Likewise, the *Works_In* relation specifies what *Employees* work in what *Stores*.

Figure 2. How the Database was Connected to the React Project



```
const express = require('express');
const app = express();
const mysql = require('mysql');
const cors = require('cors') //npm install cors

app.use(cors());
app.use(express.json())

const db = mysql.createConnection({
    user: '',
    host:'localHost',
    password: '',
    database: 'pet_store',

});
```

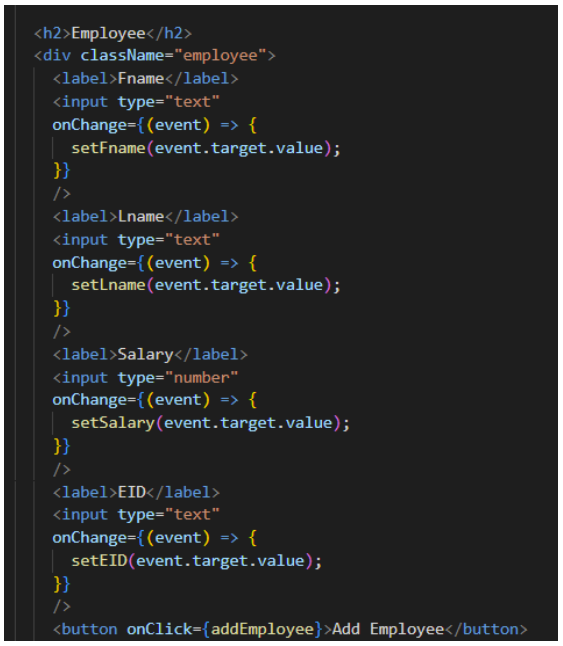
After the ERD and the resulting relational schema were created, the schema was implemented in SQL in a MySQL database. This MySQL database was then connected to our React project through an Express server (figure 2). On the front end, add, delete, update, and select functionalities were implemented, and work was started on designing a better UI. Moreover, both the SQL file and React Project were tested to make sure that all functionalities of our project worked as expected.

Implementation

Our team utilized MySQL Workbench and Visual Studio Code to develop our application, which was shared via a GitHub repository. We used a MySQL database to implement the SQL code, and the front end was created using React.js and Node.js. To connect the React project to the MySQL database, we utilized an Express server. We chose these tools due to their popularity and the abundance of resources available, as well as the real-time updating capabilities of React.js. The UI was designed using HTML, CSS, and JavaScript, and we made an attempt to transition to a cleaner UI using Material UI (MUI) near the end of the project. However, due to time constraints and limited team members, we prioritized the functionality of the application, and the new UI was left incomplete. To package our JSONs during POST and GET requests, we utilized Axios.

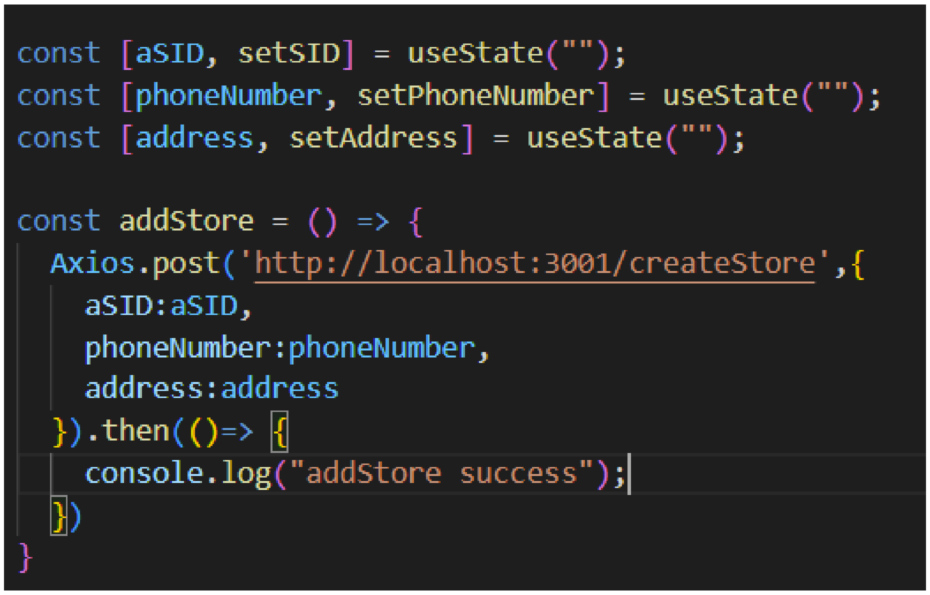
Add

Figure 3. Example of an Add Button (Specifically Add Employee)



```
<h2>Employee</h2>
<div className="employee">
  <label>Fname</label>
  <input type="text"
  onChange={(event) => {
    setFname(event.target.value);
  }}
  />
  <label>Lname</label>
  <input type="text"
  onChange={(event) => {
    setLname(event.target.value);
  }}
  />
  <label>Salary</label>
  <input type="number"
  onChange={(event) => {
    setSalary(event.target.value);
  }}
  />
  <label>EID</label>
  <input type="text"
  onChange={(event) => {
    setEID(event.target.value);
  }}
  />
  <button onClick={addEmployee}>Add Employee</button>
```

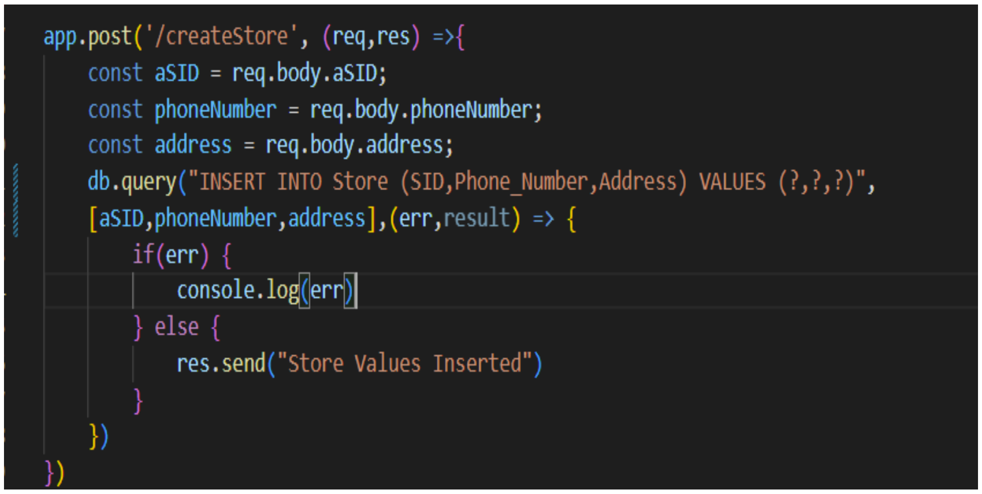
Figure 4. Example of an Axios.post request for Add (Specifically Add Employee)



```
const [aSID, setSID] = useState("");
const [phoneNumber, setPhoneNumber] = useState("");
const [address, setAddress] = useState("");

const addStore = () => {
  Axios.post('http://localhost:3001/createStore',{
    aSID:aSID,
    phoneNumber:phoneNumber,
    address:address
  }).then(()=> {
    console.log("addStore success");
  })
}
```

Figure 5. Example of a INSERT request (Specifically Add Employee)

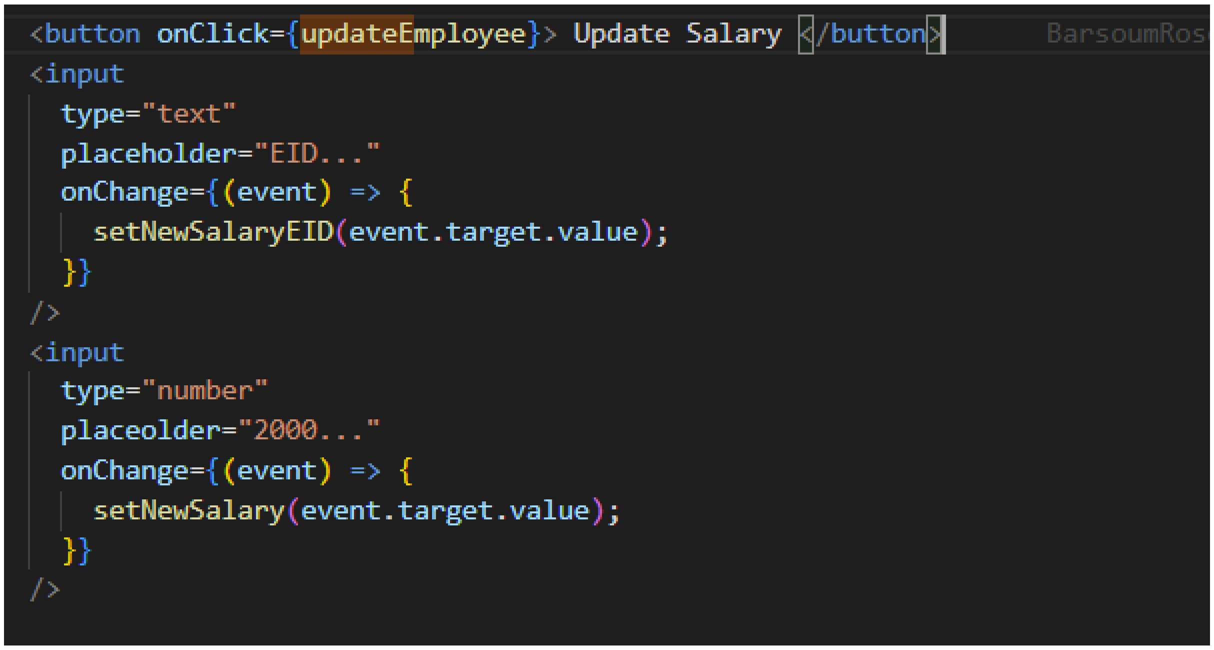


```
app.post('/createStore', (req,res) =>{
    const aSID = req.body.aSID;
    const phoneNumber = req.body.phoneNumber;
    const address = req.body.address;
    db.query("INSERT INTO Store (SID,Phone_Number,Address) VALUES (?,?,?)",
    [aSID,phoneNumber,address],(err,result) => {
        if(err) {
            console.log(err)
        } else {
            res.send("Store Values Inserted")
        }
    })
})
```

The functionality to add a record was incorporated into our project for every table and relation in the database. Our React application is designed to allow the user to input all the attributes for a record and then add that record to the database through the press of the corresponding add button. This was implemented in three sections of the code of our React app. Firstly, a button onClick event is associated with onChange events corresponding to every attribute in the record to be added. These onChange events capture and store the values of the attributes inputted by the user when the add button is pressed (figure 6). Then an Axios.post request is sent to the backend of our React project (figure 7). There, an INSERT query with the attributes of the record to be added is sent over to the MySQL database (figure 8).

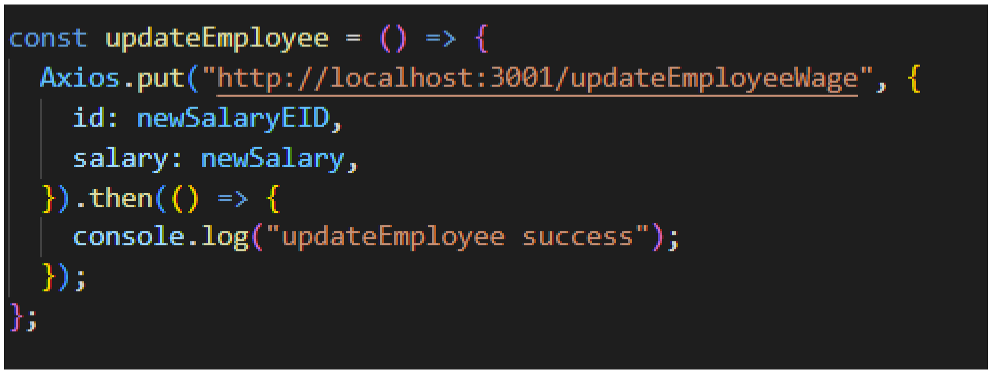
Update

Figure 6 Example of an Update Button (Specifically Update Employee Wage)



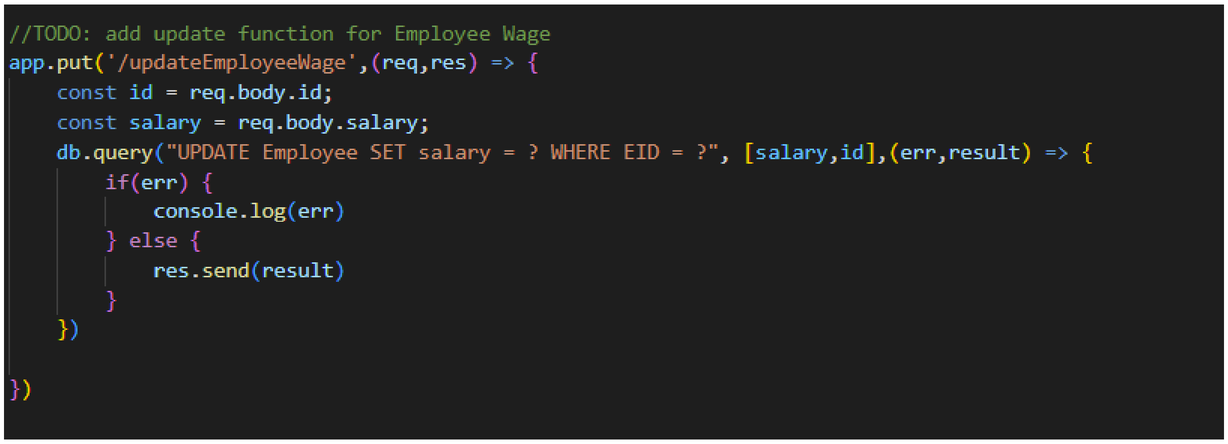
```
<button onClick={updateEmployee}> Update Salary </button>        BarsoumRos
<input
  type="text"
  placeholder="EID..."
  onChange={(event) => {
    setNewSalaryEID(event.target.value);
  }}
/>
<input
  type="number"
  placeolder="2000..."
  onChange={(event) => {
    setNewSalary(event.target.value);
  }}
/>
```

Figure 7 Example of an Axios.put Request for Update (Specifically Update Employee Wage)



```
const updateEmployee = () => {
  Axios.put("http://localhost:3001/updateEmployeeWage", {
    id: newSalaryEID,
    salary: newSalary,
  }).then(() => {
    console.log("updateEmployee success");
  });
};
```

Figure 8. Example of an UPDATE Request (Specifically Update Employee Wage)



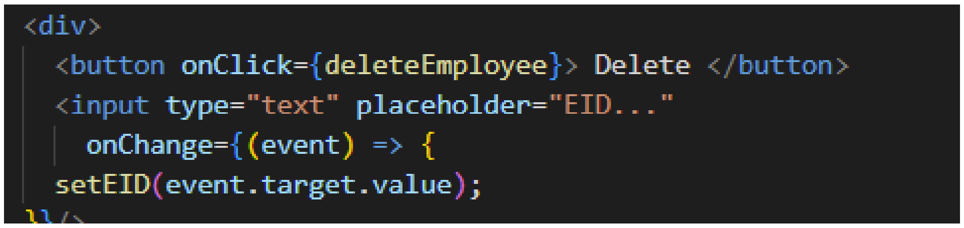
```
//TODO: add update function for Employee Wage
app.put('/updateEmployeeWage',(req,res) => {
    const id = req.body.id;
    const salary = req.body.salary;
    db.query("UPDATE Employee SET salary = ? WHERE EID = ?", [salary,id],(err,result) => {
        if(err) {
            console.log(err)
        } else {
            res.send(result)
        }
    })

})
```

The functionality to update a record was incorporated into our project for the select tables and relations in the database where it is useful to modify specific attributes. Our React application is designed to allow the user to input the primary key/s and the attribute to be updated, and update that record in the database through the press of the corresponding update button. This was implemented in three sections of the code of our React app. Firstly, a button onClick event is associated with the onChange events for both the record's primary keys and the updated value. These onChange events capture and store the value of the primary key/s and the updated attribute inputted by the user when the update button is pressed (figure 6). Then an Axios.put request is sent to the backend of our React project (figure 7). There it is sent to the database in the form of an UPDATE query with the primary key/s of the record to be updated (figure 8).

Delete

Figure 9 Example of a Delete Button (Specifically Delete Employee)



```
<div>
  <button onClick={deleteEmployee}> Delete </button>
  <input type="text" placeholder="EID..."
    onChange={(event) => {
  setEID(event.target.value);
}}/>
```

Figure 10 Example of an Axios.delete Request (Specifically Delete Employee)

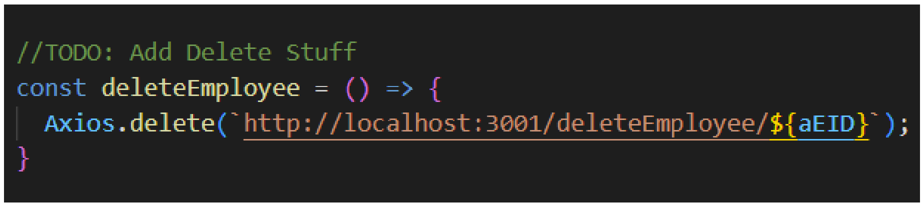
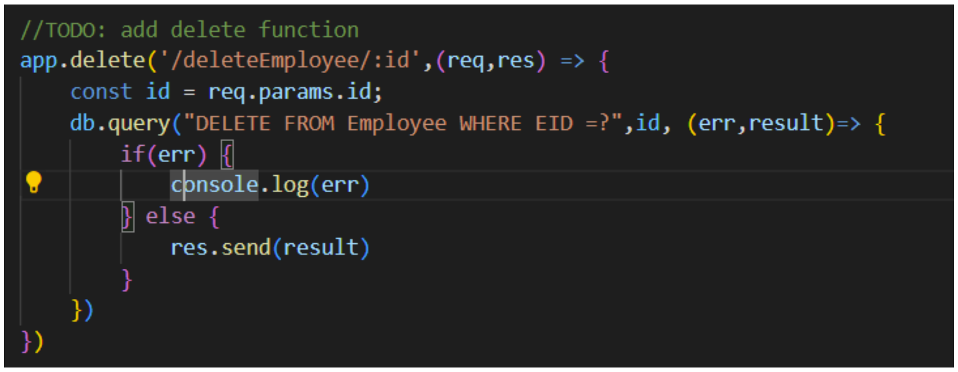

```
//TODO: Add Delete Stuff
const deleteEmployee = () => {
  Axios.delete(`http://localhost:3001/deleteEmployee/${aEID}`);
}
```

Figure 11. Example of an DELETE Request (Specifically Delete Employee)


```
//TODO: add delete function
app.delete('/deleteEmployee/:id',(req,res) => {
    const id = req.params.id;
    db.query("DELETE FROM Employee WHERE EID =?",id, (err,result)=> {
        if(err) {
            console.log(err)
        } else {
            res.send(result)
        }
    })
})
```

The functionality to delete a record was incorporated into our project for every table and relation in the database. Our React application is designed to allow the user to input the primary key/s for a record and then delete that record from the database through the press of the corresponding delete button. This was implemented in three sections of the code of our React app. Firstly, a button onClick event is associated with one or more onChange events depending on the number of primary keys for a table . These onChange events capture and store the value of the primary key/s inputted by the user when the delete button is pressed (figure 9). Then an Axios.delete request is sent to the backend of our React project (figure 10). There it is sent to the database in the form of a DELETE query where the primary key/s of the record to be deleted are inputted into the query sent over (figure 11).

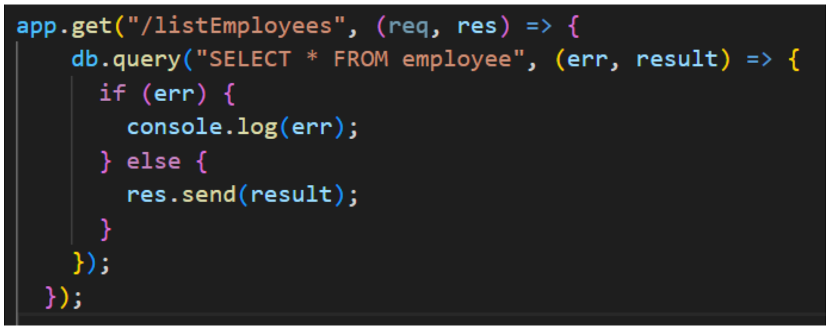
Select

For our project, we found the following two select types to be useful: Select * and Select based on a condition. It was often beneficial for us to list all the items in a table. For example, from a manager's perspective, it would be nice to see a full list of all the employees, or a full list of all the merchandise that is sold by the company in general. An example query would be as follows:

SELECT * FROM employee;

This type of select used a GET request, because we were requesting information from the server, without any input. An example of a select GET request can be seen below. As seen in Figure 12, the GET request uses the query shown above.

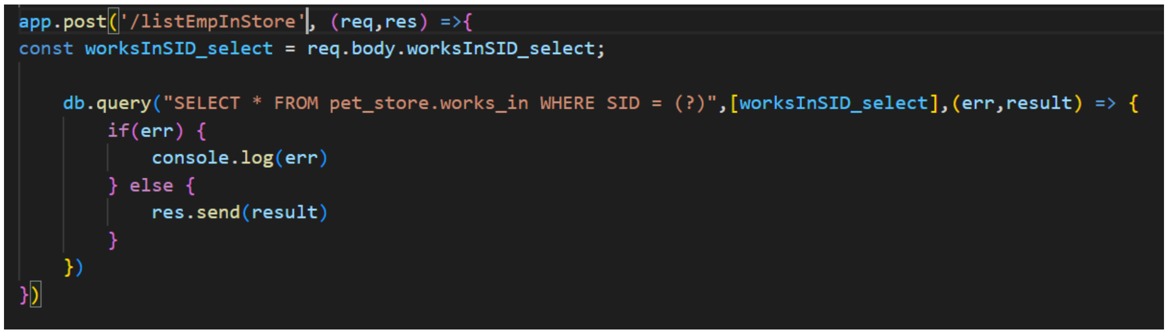
Figure 12. Example of a select* GET request



```
app.get("/listEmployees", (req, res) => {
    db.query("SELECT * FROM employee", (err, result) => {
      if (err) {
        console.log(err);
      } else {
        res.send(result);
      }
    });
  });
```

For our other select type, we selected based on a condition. For example, a manager may want to see what employees work in a given store, or what store each type of merchandise is sold in. This type of select used a POST request, because we had to send a value, used as the condition in our query, to the database. An example of this type of select can be seen in figure 13. To list what employees work in a given store, we post the SID to the server through the variable **worksInSID_select**, and pass that into our query.

Figure 13. Example of a select with a condition POST request



```
app.post('/listEmpInStore', (req,res) =>{
const worksInSID_select = req.body.worksInSID_select;

    db.query("SELECT * FROM pet_store.works_in WHERE SID = (?)",[worksInSID_select],(err,result) => {
        if(err) {
            console.log(err)
        } else {
            res.send(result)
        }
    })
})
```

Problems and Challenges

We experienced many unexpected problems and challenges during this project.
We started the project very inexperienced with the tools we were using and had to deal with the learning curve that came with them. When using MySQL, one of our teammates had trouble implementing a local database and the other teammate came into the project with no react.js experience.

With the limited time allocated to us, we decided to stick with using local databases for our project instead of setting up an online server. However, this, unfortunately, led to a difference between our SQL schemas that was only discovered later on in the project when we attempted to merge our sections. However, when that was addressed, functionality was fine.

We also had trouble merging our code when it came time to put our functionalities together, especially because neither team member had extensive experience using git.

In general, we also ran into time management issues and struggled to implement all of the functionalities necessary. In part, we attribute this to us only being a team of two, as well as having minimal frontend/backend experience, and no experience with SQL. With a third person on our team, we feel that we would have been able to complete more in our allotted time.

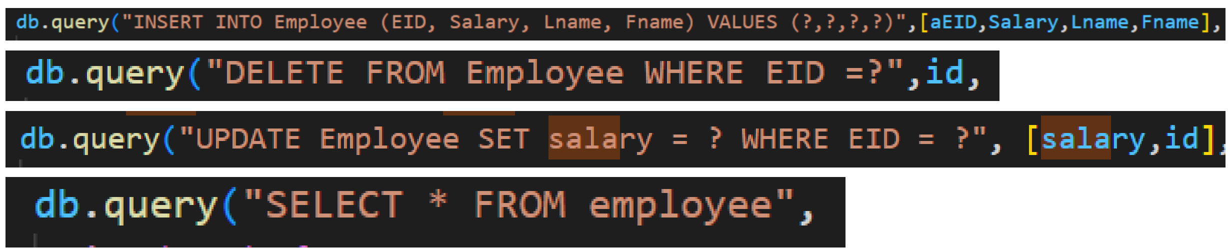
Validation

To verify that we had our complete functionality, we tested each of them through the frontend. For each section in our react project, we would complete each functionality, and verify that it worked on the backend by looking at our database to see if it updated correctly. For example, when we look at the employee's tab, we have the option to add an employee, delete an employee, update an employee salary, and list all the employees. We were able to test each one of these functionalities successfully through the frontend and use the backend to verify it. Table 3 shows all our test cases.

Also, to debug throughout our project, we used the web console and the server console to view error messages and adjust accordingly.

All of the core functionalities that we set out to complete have been completed. We implemented major Add, Delete, Update, and Delete Functionalities. Examples of our backend queries can be seen in figure 14.

Figure 14. Example Backend queries for Add employee, Delete employee, Update salary, and Select all list employees, respectively



```
db.query("INSERT INTO Employee (EID, Salary, Lname, Fname) VALUES (?,?,?,?)",[aEID,Salary,Lname,Fname],
db.query("DELETE FROM Employee WHERE EID =?",id,
db.query("UPDATE Employee SET salary = ? WHERE EID = ?", [salary,id],
db.query("SELECT * FROM employee",
```

Table 2. All the functionalities that we tested through the frontend.

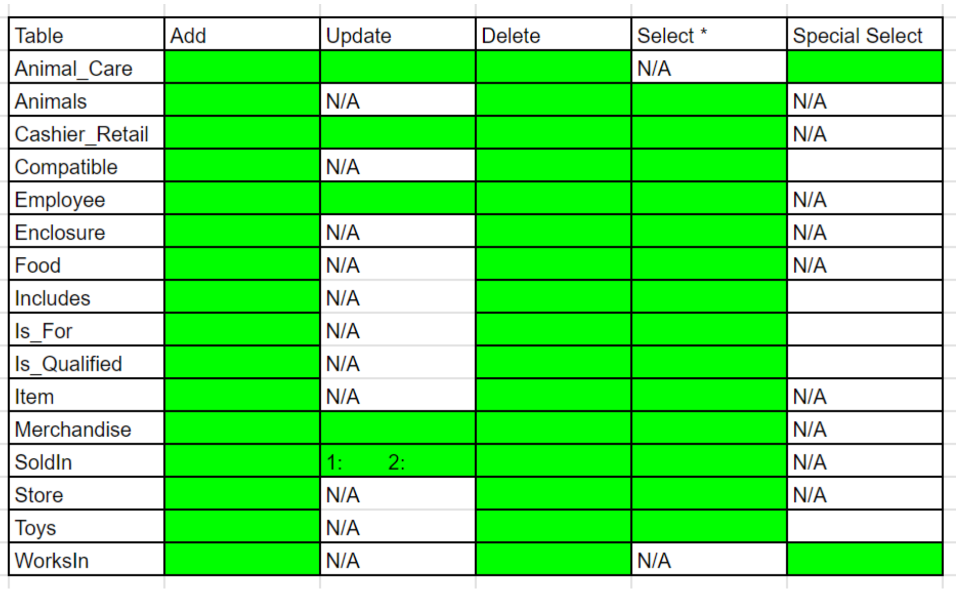


Table	Add	Update	Delete	Select *	Special Select
Animal_Care	🟩	🟩	🟩	N/A	🟩
Animals	🟩	N/A	🟩	🟩	N/A
Cashier_Retail	🟩	🟩	🟩	🟩	N/A
Compatible	🟩	N/A	🟩	🟩	
Employee	🟩	🟩	🟩	🟩	N/A
Enclosure	🟩	N/A	🟩	🟩	N/A
Food	🟩	N/A	🟩	🟩	N/A
Includes	🟩	N/A	🟩	🟩	
Is_For	🟩	N/A	🟩	🟩	
Is_Qualified	🟩	N/A	🟩	🟩	
Item	🟩	N/A	🟩	🟩	N/A
Merchandise	🟩	🟩	🟩	🟩	N/A
SoldIn	🟩	🟩 1: 2:	🟩	🟩	N/A
Store	🟩	N/A	🟩	🟩	N/A
Toys	🟩	N/A	🟩	🟩	
WorksIn	🟩	N/A	🟩	N/A	🟩

Table 3. List of Test Cases Validated

Test Case	Input	Expected Output	Pass/Fail
Add Store	User enters SID, Phone Number, Address Users presses the "Add Store" Button	Store is added to the store table	Pass
Delete Store	User enters store SID to be deleted User presses the "Delete" button	Store is deleted from the store table	Pass
List Stores	User clicks the button "List Stores"	All stores from the store table are listed	Pass
Add Works_in relation	User enters SID and EID. User clicks "Add works in relation" button	Works in relation is added to the works_in table	Pass
Delete Works_in relation	User enters SID and EID to be delete	Works_in relation is deleted from the Works_in table	Pass

	User presses the “delete” button		
List employees who work in given store	User enters SID User presses “List Employees in Store” button	EIDs of employees who work in given SID are listed	Pass
Add employee	User enters First name, Last name, Salary, and EID User presses “add employee” button	Employee is added to the employee table	Pass
Update Salary	User enters EID and new salary User clicks “update salary” button	Salary of given employee is updated in the employee table	Pass
List Employee	User clicks “List employee” button	List of employees from the employee table is listed	Pass
Add animal_care relation	User inputs EID and specialty User clicks “add animal care” button	Animal_care relation is added to the animal_care table	pass
Delete animal care relation	User inputs EID User presses “delete” button	Given animal relation is deleted from the animal_relation table	pass
Update specialty	User inputs EID and new specialty User clicks “update specialty: button	Given relation is updated in the animal_relation table	pass
List employees with specialty	User types specialty. User clicks “list employees with specialty button”	Employees with the given speciality are listed from the animal_care table	pass
Add cashier/retail employee	User inputs EID and checks if the employee	Employee is added to the cashier_retail table	pass

	is part-time. User clicks “add cashier/retail employee” button		
Update part time/full time	User inputs EID to be updates and checks if they are part time. User presses the “update part time/full time button”	Employee part time/ full time status is updated in the cashier_retail table	pass
Delete cashier_retail relation	User inputs EID to be deleted User presses the “delete” button	Employee part time/full time relation is deleted from the cashier_retail table	pass
List Part time employee	User presses “list part time employees button”	Employees that are part time from the cashier_retail table are listed	pass
List full time employees	User presses “list full time employees button”	Employees that are full time from the cashier_retail table are listed	pass
Add sold in relation	User enters # in stock, shelving location, MID, and SID	Sold in relation is added to the sold-in table	pass
update in stock number	user enters MID , SID, in stock number user presses “update in stock number” button	Updated stock number is reflected in sold and table	Pass
Update shelving location	user enters MID, SID, and new shelving location User presses “update shelving location” button	Shelving location is reflected in sold in table	pass
Delete sold in relation	User enters MID, SID. user presses the “delete	Relation is deleted from the sold in table	pass

	button”		
Merchandise is listed in store	User enters SID User presses the “list merchandise and store” button	All the merchandise for given SID is listed from the sold_in table	pass
Stores with merchandise are listed	User enters MID User presses “list stores with merchandise”	All stores for given MID is listed from the sold_in table	pass
Add merchandise item	User enters price and MID User presses “add merchandise” button	Merchandise is added to the merchandise table	pass
Update price	User enters MID and new price User presses “update price” button	Merchandise price is updated in the merchandise table	pass
Delete merchandise	Users enters MID User presses “delete” button	Merchandise is deleted from the merchandise table	pass
List all merchandise	User presses “list all merchandise” button	All merchandise in the merchandise table is listed.	pass
Add includes relation	User inputs MID, IID, AID User presses “add includes relation” button	Relation is added to the includes table	pass
Delete includes relation	User inputs MID and IID and AID User presses the”list all includes relations” button	Relation is deleted from the includes table	pass
List all includes relation	User clicks “list all includes relations” button	All relations from the includes table is listed	pass

Add animal	User inputs max size, enclosure type, food, species, lifespan, AID, care guide, and checks if the animal is hypoallergenic, User presses “add animal button”	Animal is added to animal table	pass
Delete animal	User inputs AID User presses “delete “ button	Animal is deleted from the animal table	pass
List animals	User presses “list all animals”	All animals from animal table are listed	pass
Add item	User inputs IID and brand User presses “add items” button	Item is added to the items table	pass
Delete item	User inputs IID User presses “delete” button	Item is deleted from the item table	pass
List all items	User presses “list all items button”	All items from the item table is deleted	pass
Add food	User inputs IID, brand, shelf life, checks if food is alive or need refrigeration User presses “add food button”	Food is added to the food table	pass
Delete food	User inputs IID User presses the “delete” button	Food is deleted from the food table	pass
List all food items	User presses “list all food items” button	All food from the food table is listed	pass
Add enclosure	User inputs IID, brand, dimensions	Enclosure is added to the enclosure table	pass

	User presses “add enclosure button”		
Delete enclosure	User inputs IID User presses “delete” button	Enclosure is deleted from the enclosure table	pass
List all enclosures	User presses “list all enclosures” button	All enclosures from the enclosure table are listed	pass
Add toy	User inputs IID, brand, type, and checks if chocking hazard User presses “add toys” button	Toy is added to the toy table	pass
Delete toy	User inputs IID	Toy is deleted from the toy table	pass
List all toys	User presses “list all toys button”	All toys from the toys table are listed	pass
Add is for relation	User inputs IID and AID User presses “add is for relation button”	Relation is added to the is_for table	pass
Delete is for relation	User inputs IID and AId User presses “delete” button	Relation is deleted from the is_for table	pass
List items for given AID	User inputs AID User presses “List Items for Given AID” button	Items for given AID is listed from the is_for table	pass
Add compatible with relation	User inputs AID User presses “add compatible with relation”	Relation is added to the compatible_with table	pass
Delete compatible with relation	User inputs AID	Relation is deleted from the compatible_with	pass

	User presses “delete” button	table	
Adds is qualified relation	User inputs AID, animalCareEID User presses “add is qualified relation”	Relation is added to the is_qualifed table	pass
Delete is qualified relation	User inputs AID, EID User presses “delete” button	Relation is deleted from the is_qualified table	pass
List employees qualified for given AID	User inputs AID User presses “List employees qualified for given AID” button	All employees qualified for given AID are listed from the is_qualified table	pass

Lessons Learned

Looking back on our experience, we learned a lot. We learned how to use React.js and Express, something only one of our teammates had minimal experience with. We also learned how to create POST, GET, ADD, and DELETE requests.

In regards to Database Management, we learned how to design a database with constraints and implement it in SQL. Before this class, neither partner had experience with SQL or database design. Now we can say that we are both a lot more comfortable with the material taught in this class.

In terms of our project, looking back at our experience, we believe that it would have been beneficial for us to agree earlier on a UI design. We initially started the project with pure functionality in mind. However, had we discussed a minimal UI prior, we may have been able to implement more of it. For example, looking at our project, it can be noted that only some aspects of the UI are using MUI components. MUI is not hard to implement, and it could have been implemented from the beginning for cleaner and more professional work. However, due to starting the transition from the old UI to the new UI, towards the end of our project, we were unable to make the full transition to the new UI.

We also could have been better at task delegation and management. In hindsight, we just took on what we could at the time, instead of assigning individual tasks, and that caused some misunderstanding and technical difficulties down the line.

Conclusion

In conclusion, we successfully built a managerial system for a Pet Store by implementing an

ERD, converting it into a relational schema, and using mySQL to create a database. By connecting this database to a React.js CRUD app via an Express server, we created an interface for data management in which one could add, update (when reasonably updatable), delete, and list tables (some). Overall, the project demonstrated the practical application of database design concepts and provided the foundation of an application for managing a Pet Store's data.

Future Work

In the future, we hope to clean up the UI to look more professional, clean, and crisp. Currently, our UI showcases our SQL tables separately in a drop-down menu. Practically speaking, this would not be a very useful UI for a manager. Ideally, in the future, we would have a new front-end design that would allow the user to input values in a more efficient and practical way. We also hope to expand the functionality of our application to include a customer-facing front-end application. Some features we would like to implement on the customer-facing side of the application would be a user profile, shopping cart, and purchase history.

We would also like to limit variability in user inputs. For example, to currently add a specialty, the user has to type a string to specify what the employee specializes in. The user could input this in multiple ways (i.e. dogs, Dogs, DOGS, dog). To eliminate this variability and improve select functionality, we would like to implement a drop-down menu with preselected options for this user to pick from.

Moreover, we would like to expand upon our existing SQL constraints to help improve the functionality of our database. We would like to include better limitations of what data is allowed to be inputted by the user, as well as pop-ups warning against incorrect user-inputted data on the front-end.

We would also like to expand upon our update functionality so that in conjunction with select *, a user could see a list of what records are available to be updated and change them accordingly.