**Petopia Management System**

**A Programming (Java) NCIII Program Presented to**

**Mr Jareth Ronquillo**

**Trainer**

**JOYSIS Tech Voc Inc.**

**#1 Inda Maria Rd. Potrero, Malabon City**

**In Partial Fulfillment of the Requirements for the Course Java Programming NC III**

**Submitted by:**

**Daffodil Jorda**

**Wendell Centeno**

**Clarisse Joy Sevilla**

**Henessy Hernaez**

**John Adam Ramos**

**Dominick Aquitania**

**Table of Contents**

**Page**

**Title i**

**Table of Contents ii**

**I. Introduction 3**

**II. Case Diagram 4**

**III. Database Design and Table Design 5-7**

**IV. Program Photos 8-15**

**V. Source Code 16-40**

**Introduction**

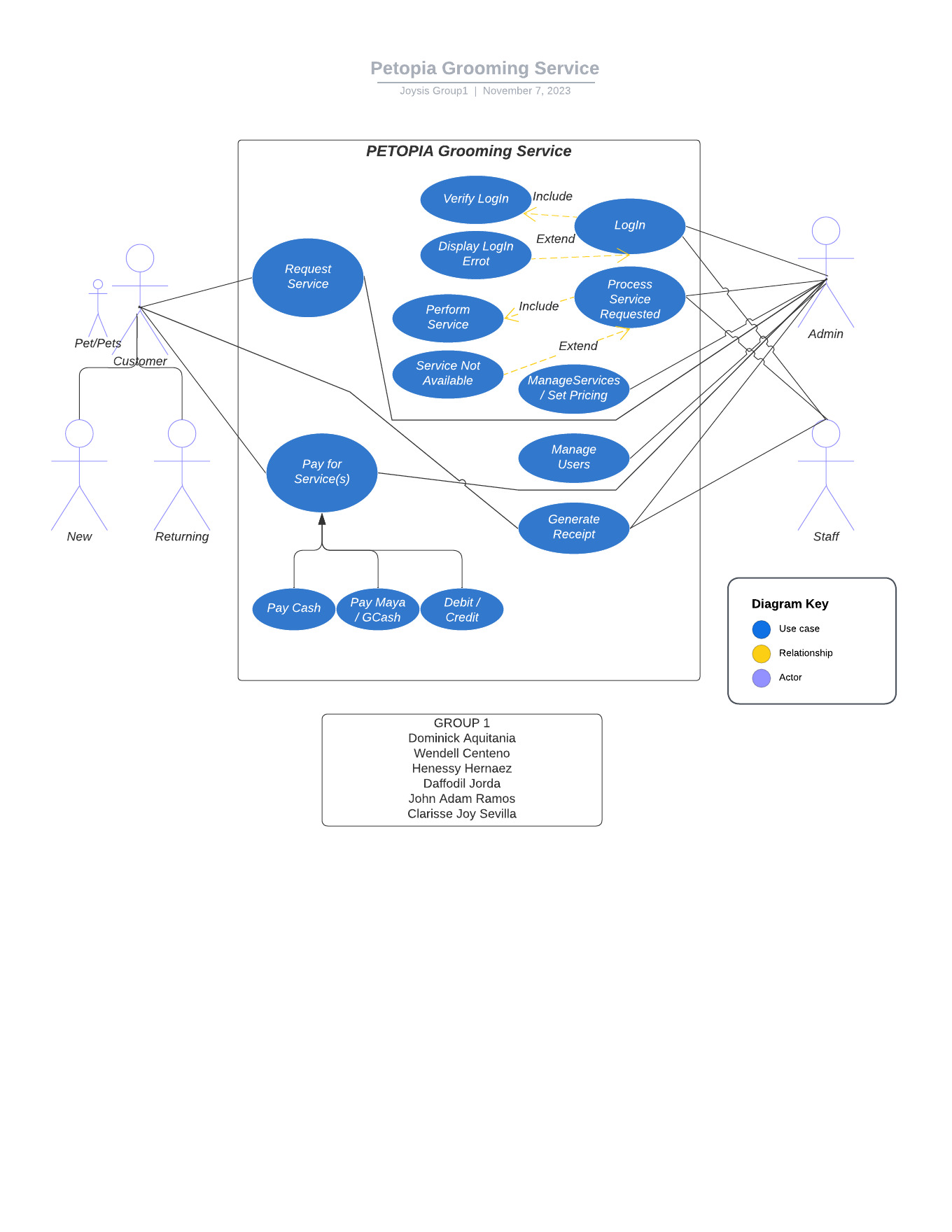
Pets have strengthened their place as beloved family members, close friends, and loyal companions in a world growing more interconnected by the day. Human-animal relationships have developed into deep emotional bonds that go beyond simple ownership. This change has led to the growth of the pet care industry, where businesses and pet owners work together to give our fluffy and furry animals the greatest care and services available.

This increasing need gives rise to the Petopia Management System, which brings in a new era of pet care management. Petopia aims to transform pet ownership and service delivery by using a complete and creative strategy.

Petopia Management System is a system that allows grooming services and pet owners to have a smooth transaction and to efficiently manage pet appointments, grooming sessions, and a variety of other pet-related tasks. This system can offer inventory-tracking for pet-related services such as whichever employee handled a specific pet, who is the owner of the pet, and their availed services.

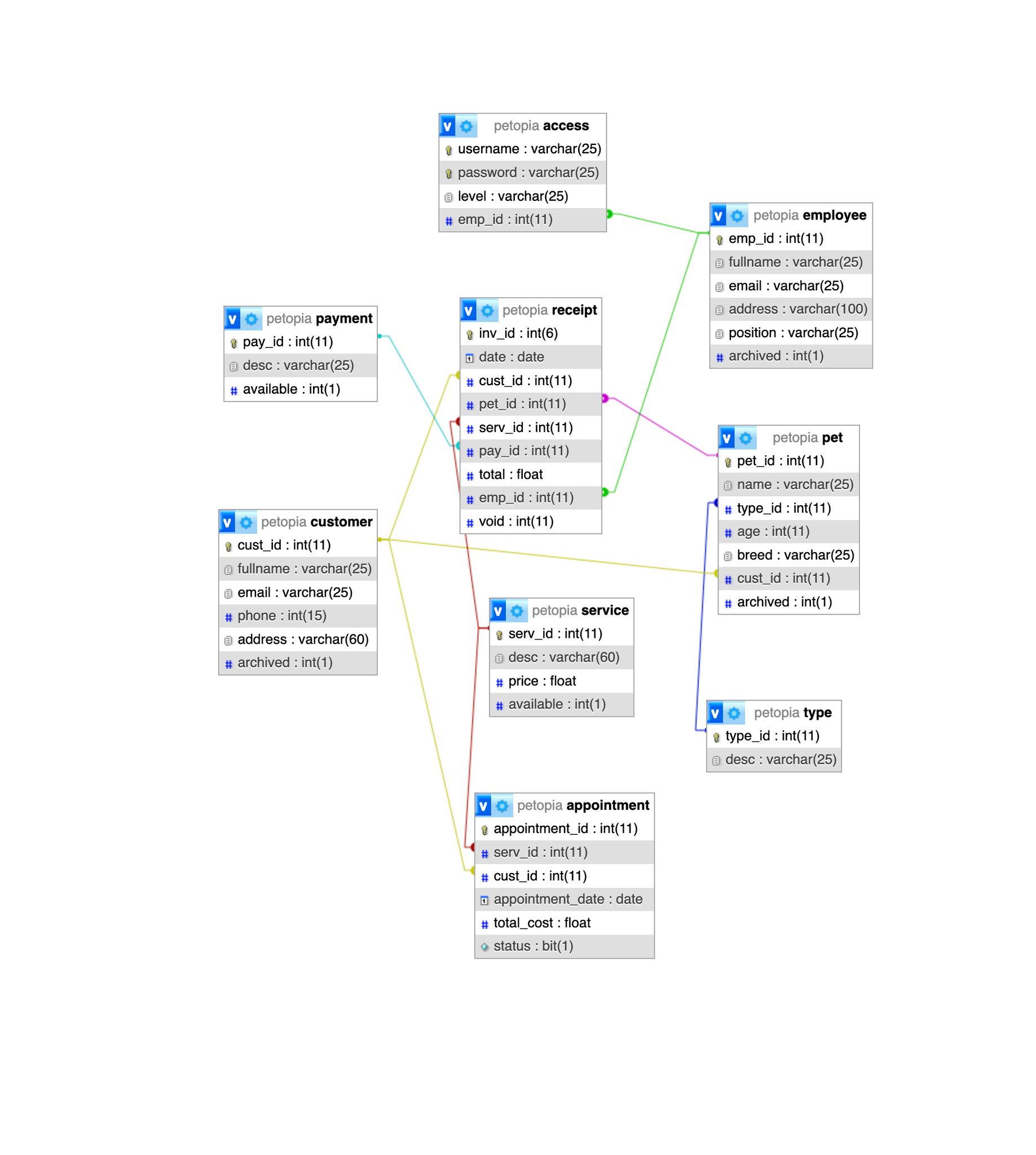
The key advantages of this approach are record keeping and the comfort of business owners and fur moms/dads as they entrust their pets to the grooming business.

**Case Diagram**

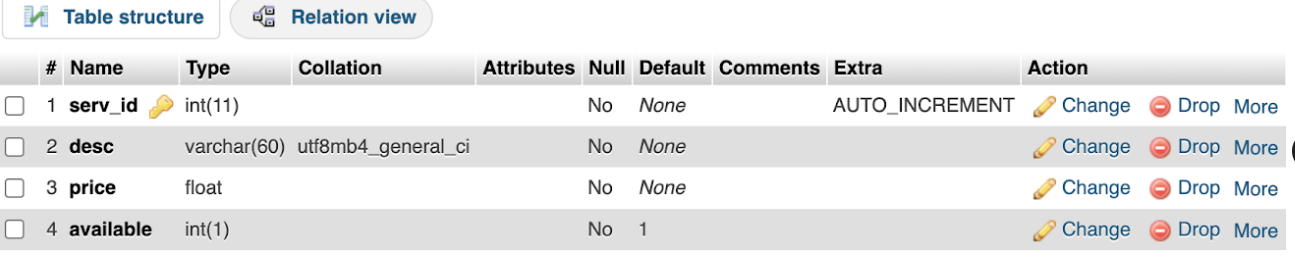
****

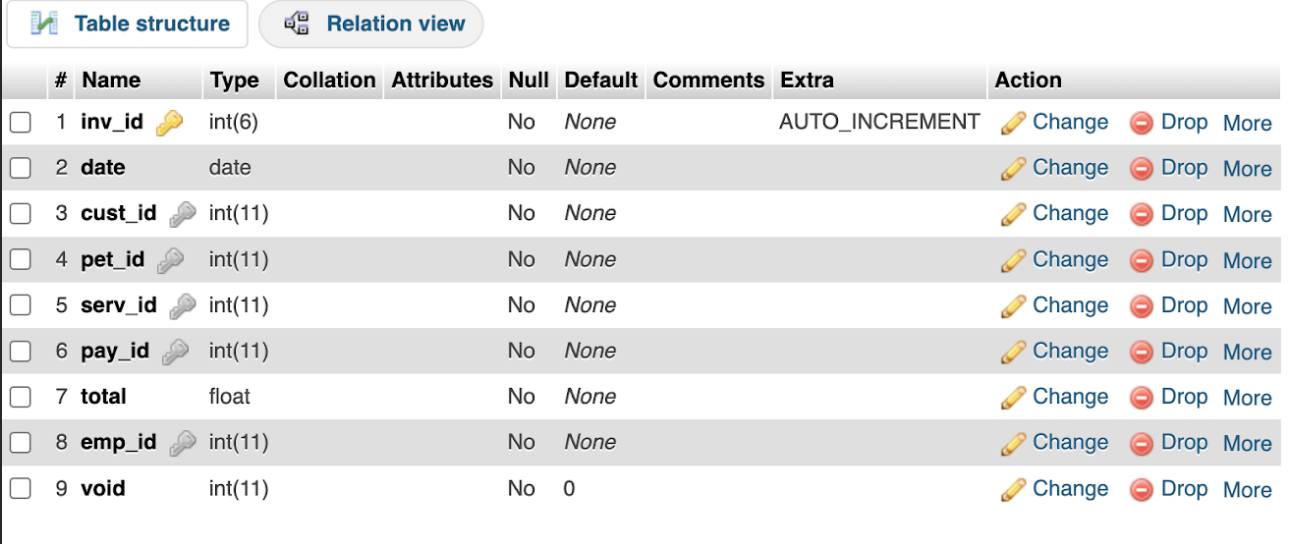
**Database Design and Table Design**

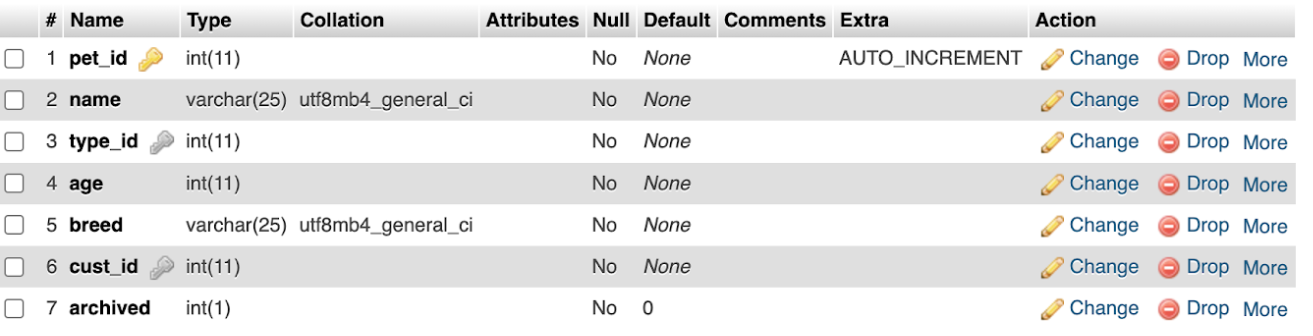
**Database Design**

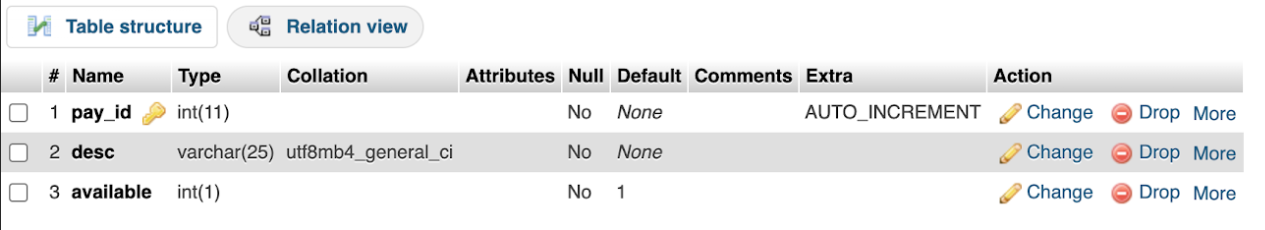
****

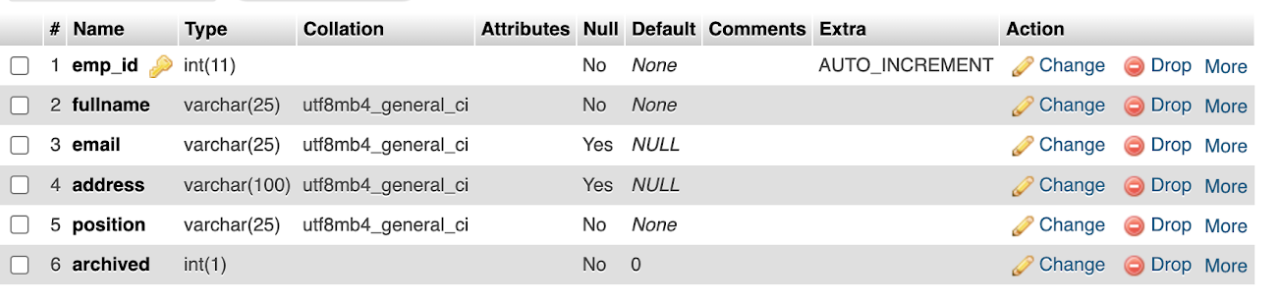
**Table Design**

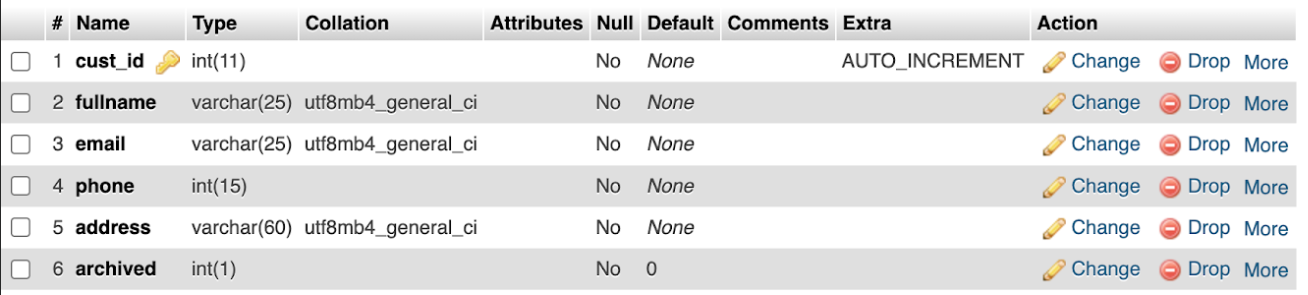




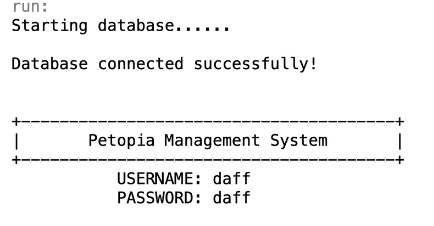


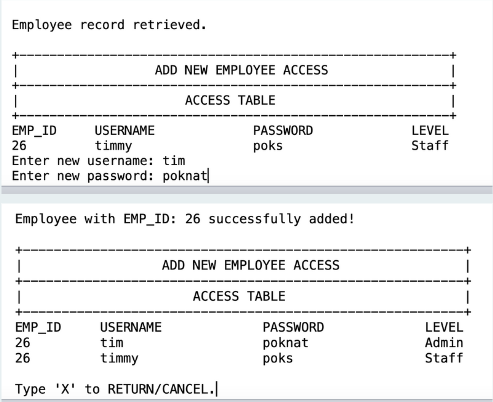


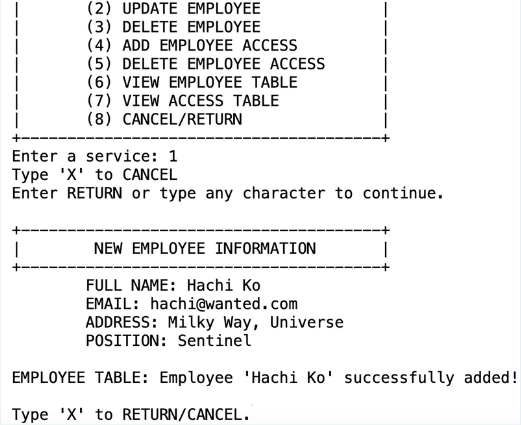


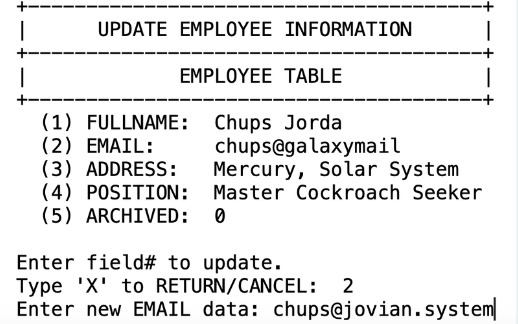
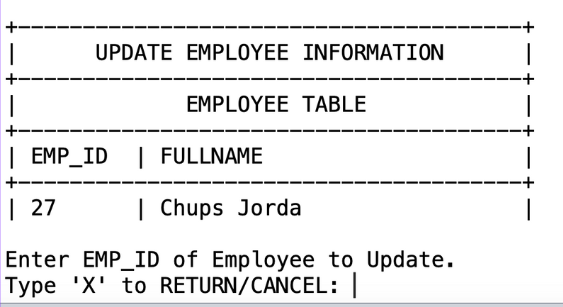


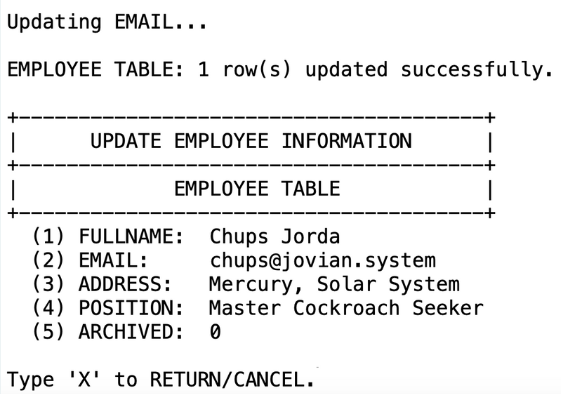
**Program Photos**

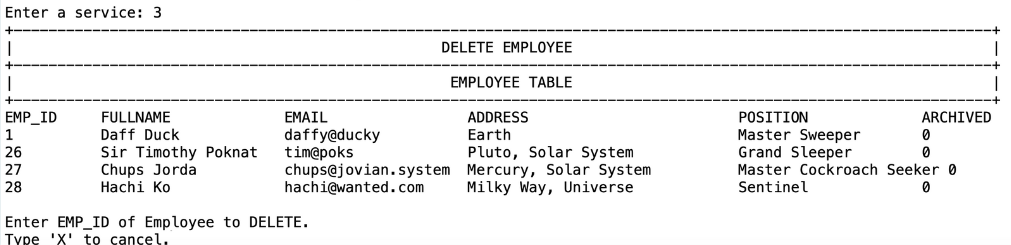
**Log in**

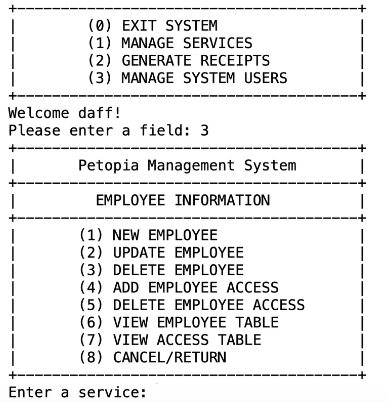
**Add new employee access**

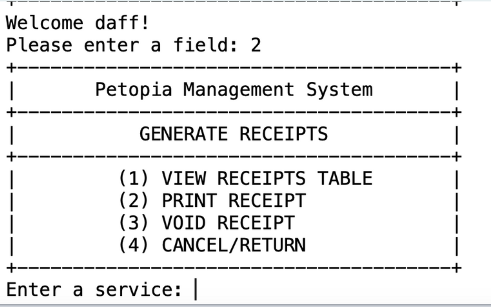
**Add new employee**

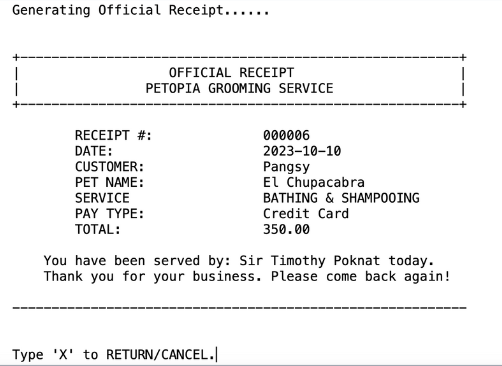
**Update employee information**

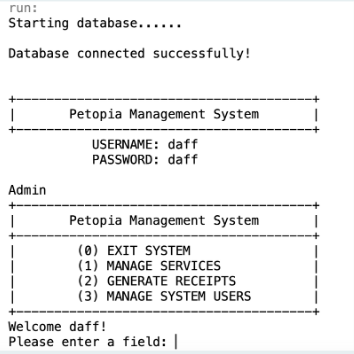


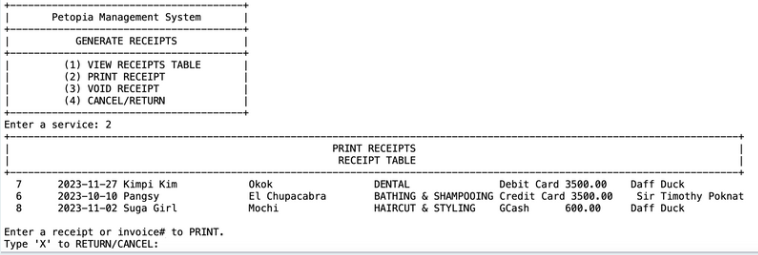
**Delete employee**

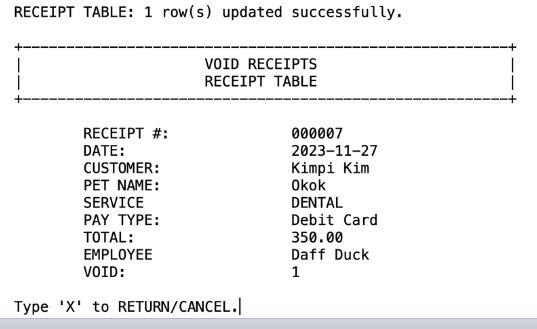
 **Employee fields**

**Generate receipts**

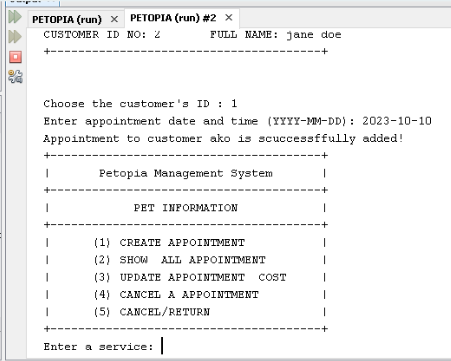
**Official receipt printed**

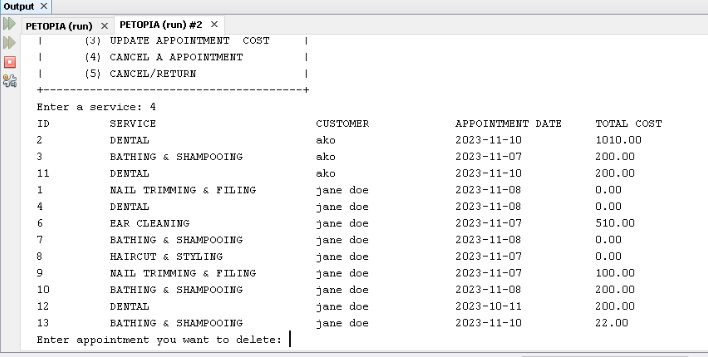
**Petopia Management System Log in**

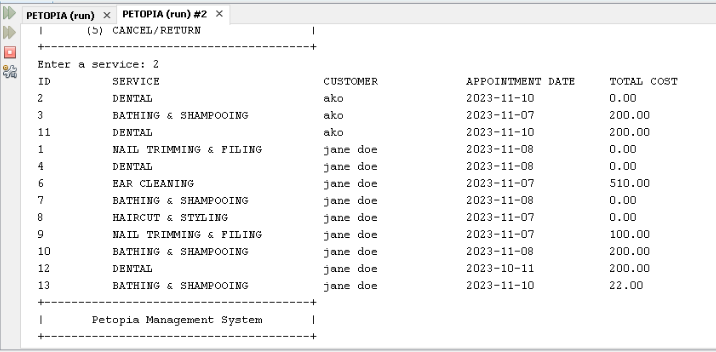
**Print receipts**

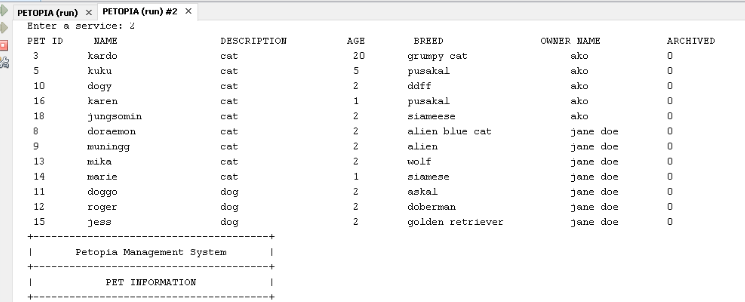
**Void receipts**

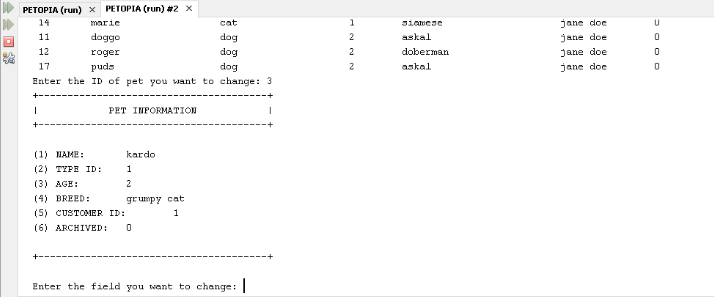
**Pet and Appointment**

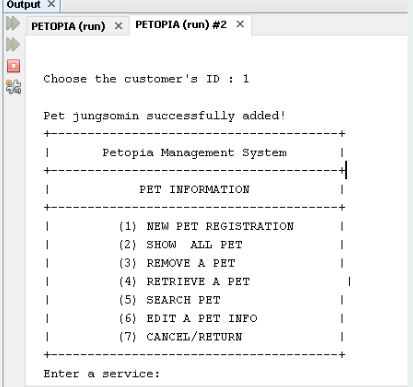
**Adding an appointment**

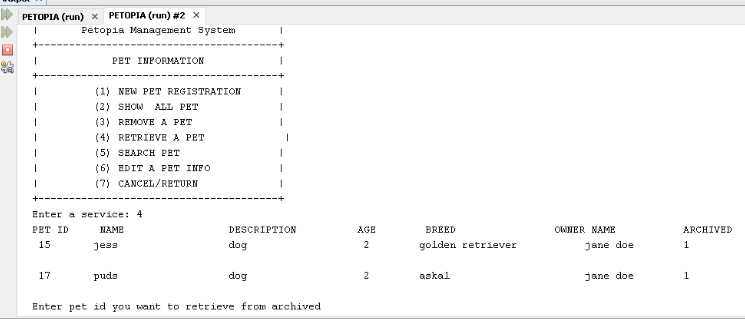
**Cancel appointment**

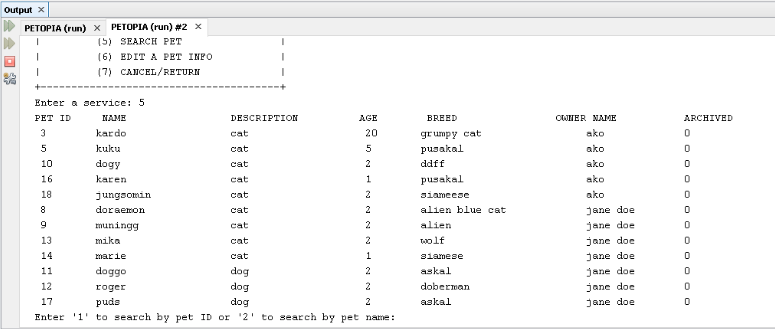
**Display all appointment**

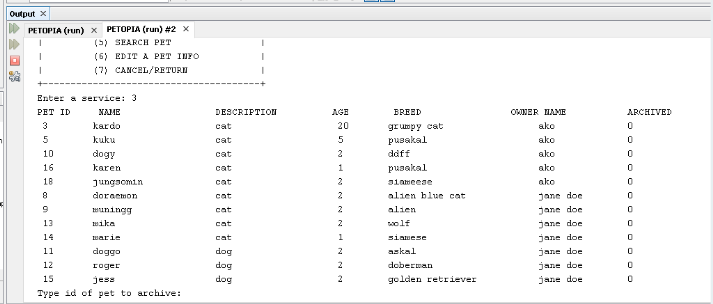
**Display all pets**

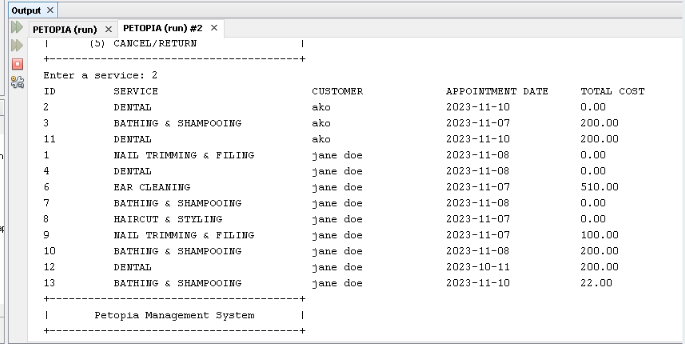
**Edit pet field value**

**Inserting pet**

**Retrieve pet**

**Search pet by ID or name**

**Soft delete**

**Update total cost**

**Source Code**

Appointment info

package com.petopia;

import com.petopia.api.AppointmentQuery;

import com.petopia.api.PetQuery;

import com.petopiables.PetopiaHeader;

import java.util.Scanner;

public class AppointmentInfo extends PetopiaHeader{

Scanner sc = new Scanner(System.in);

PetQuery petQuery = new PetQuery();

Services services = new Services();

public void appointmentInfo(){

//instantiations

AppointmentQuery apptQuery = new AppointmentQuery();

petopiaHeader();

System.out.format("| PET INFORMATION |%n"); System.out.format("+---------------------------------------+%n"); System.out.format("| (1) CREATE APPOINTMENT |%n"); System.out.format("| (2) SHOW ALL APPOINTMENT |%n"); System.out.format("| (3) UPDATE APPOINTMENT COST |%n"); System.out.format("| (4) CANCEL AN APPOINTMENT |%n"); System.out.format("| (5) CANCEL/RETURN |%n"); System.out.format("+---------------------------------------+%n");

try {

System.out.print("Enter a service: ");

int choice = sc.nextInt();

sc.nextLine();

switch (choice) {

case 1:

// INSERT INTO PET VALUES....

// PETID WILL BE AUTOGENERATED

petQuery.displayOwnerName();

//validate and make use the user input is an int

int custId = 0;

boolean validCustomerInputId = false;

do {

System.out.print("Choose the customer's ID : ");

try {

custId = Integer.parseInt(sc.nextLine());

validCustomerInputId = true;

} catch (NumberFormatException e) {

System.out.println("Invalid input. Please enter a valid number for the customer ID.");

}

} while (!validCustomerInputId);

System.out.print("Enter appointment date and time (YYYY-MM-DD): ");

String appointmentDate = sc.nextLine();

double totalCost = 0;

int status = 0;

apptQuery.addNewAppointment(custId, appointmentDate, totalCost, status);

// ENTER ALL FIELDS IN THE DATABASE

this.appointmentInfo();

break;

//DIPLAY APPOINTMENTS

case 2:

apptQuery.displayAppointmentsUsingInnerJoin();

this.appointmentInfo();

break;

//CALCULATE THE TOTAL COST OF APPOINTMENT SERVICE AND UPDATE ITS TOTAL COST FIELD

case 3:

apptQuery.displayAppointmentsUsingInnerJoin();

System.out.println("Enter appointment id: ");

int appTId = sc.nextInt();

sc.nextLine();

boolean addMoreServices = true;

double total = 0.0;

//loop thru so the use has an option to add more service do {

apptQuery.displayServiceDescription();

System.out.print("CHOOSE SERVICE (ID) NO: ");

int serviceId = sc.nextInt();

sc.nextLine();

double servicePrice = apptQuery.getServicePrice(serviceId);

// Add the service price to the total cost total

+= servicePrice;

System.out.println("Service added successfully.");

// Prompt user to add another service

System.out.print("Do you want to add another service? (y/n): ");

String addMore = sc.nextLine();

if (addMore.equalsIgnoreCase("n")) {

addMoreServices = false; }

}

while (addMoreServices);

// Store the total cost in the other table

apptQuery.updateTotalAmouintService(total, appTId);

System.out.println("");

this.appointmentInfo();

break;

//CANCEL APPOINTMENT

case 4:

apptQuery.displayAppointmentsUsingInnerJoin();

System.out.print("Enter appointment you want to delete: ");

int apptIdToArchived = sc.nextInt();

sc.nextLine();

apptQuery.cancelAppointment(apptIdToArchived);

this.appointmentInfo();

break;

case 5:

services.services(LogIn.qUsername, LogIn.qLevel);

// cancel

// return break;

default: System.out.println("Invalid Input! Enter choice enclosed in ().");

this.appointmentInfo();

}

} catch (Exception e) { System.out.println(e);

}

//end of try catch

}

}

**Customer Info**

package com.petopia;

import com.petopia.api.CustomerQuery;

import com.petopiables.PetopiaHeader;

import java.util.Scanner;

public class CustomerInfo extends PetopiaHeader{

public void customerInfo(){

//instantiations

Scanner sc = new Scanner(System.in);

ManageServices manageServices = new ManageServices();

CustomerQuery customerQuery = new CustomerQuery();

petopiaHeader();

System.out.format("| CUSTOMER INFORMATION |%n"); System.out.format("+---------------------------------------+%n");

System.out.format("| (1) ENTER NEW CUSTOMER |%n"); System.out.format("| (2) VERIFY CUSTOMER |%n"); System.out.format("| (3) UPDATE CUSTOMER |%n"); System.out.format("| (4) DELETE CUSTOMER |%n"); System.out.format("| (5) CANCEL/RETURN |%n"); System.out.format("+---------------------------------------+%n");

System.out.print("Enter a service: ");

int choice = sc.nextInt(); switch (choice)

{

case 1:

// INSERT INTO CUSTOMER TABLE VALUES...

customerQuery.addCustomer();

customerInfo();

break;

case 2:

customerQuery.verifyCustomer();

customerInfo();

break;

case 3:

// UPDATE CUSTOMER () SET VALUES ()

customerQuery.updateCustomer();

customerInfo();

break;

case 4:

// DELETE CUSTOMER WHERE CUST\_ID = ...

// Implement the code for deleting a customer based on CUST\_ID.

break;

case 5:

manageServices.manageServices();

default:

// Invalid input

System.out.println("Invalid input. Please enter a valid option (1-5).");

customerInfo();

break;

}

}

}

**Employee info**

package com.petopia;

import com.petopia.api.EmployeeQuery;

import com.petopiables.PetopiaHeader;

import com.petopiables.Functions;

import java.util.InputMismatchException;

import java.util.List;

import java.util.Scanner;

public class EmployeeInfo extends PetopiaHeader{

public static int empId;

public static Scanner sc = new Scanner(System.in);

public EmployeeQuery empQuery = new EmployeeQuery();

public Functions function = new Functions();

void employeeInfo()

{

//instantiations Services services = new Services();

LogIn logIn = new LogIn();

petopiaHeader();

System.out.format("| EMPLOYEE INFORMATION |%n"); System.out.format("+---------------------------------------+%n");

System.out.format("| (1) NEW EMPLOYEE |%n");

System.out.format("| (2) UPDATE EMPLOYEE |%n"); System.out.format("| (3) DELETE EMPLOYEE |%n"); System.out.format("| (4) ADD EMPLOYEE ACCESS |%n"); System.out.format("| (5) DELETE EMPLOYEE ACCESS |%n"); System.out.format("| (6) VIEW EMPLOYEE TABLE |%n"); System.out.format("| (7) VIEW ACCESS TABLE |%n"); System.out.format("| (😎 CANCEL/RETURN |%n"); System.out.format("+---------------------------------------+%n");

try {

System.out.print("Enter a service: ");

String choiceInput = sc.nextLine();

if (function.isNumber(choiceInput)){

int choice = Integer.parseInt(choiceInput);

if (choice > 0 && choice < 9) {

if (choice == 1){

System.out.print("Type 'X' to CANCEL\nEnter RETURN or type any character to continue.");

String input = sc.nextLine();

if (input.equalsIgnoreCase("X")) { this.employeeInfo();

} else {

System.out.format("\n+---------------------------------------+%n");

System.out.format("| NEW EMPLOYEE INFORMATION |%n"); System.out.format("+---------------------------------------+%n"); System.out.print("\tFULL NAME: ");

String fullname = sc.nextLine();

System.out.print("\tEMAIL: ");

String email = sc.nextLine();

System.out.print("\tADDRESS: ");

String address = sc.nextLine();

System.out.print("\tPOSITION: ");

String position = sc.nextLine();

// ARCHIVED IS DEFAULT TO 0 FOR NEW EMPLOYEE

empQuery.addNewEmployee(fullname, email, address, position);

this.xToCancel(); }

}else if (choice == 2) { //update emp info

this.searchName();

this.xToCancel();

} else if(choice == 3){

//DELETE EMPLOYEE INFO FROM EMPLOYEE AND ACCESS TABLE

this.delFormat();

empQuery.displayAllEmployee();

//view employee table

this.deleteEmployee();

} else if (choice == 4) {

//add emp access

this.lineFormat();

System.out.format("|%35s %31s |%n", "", "ADD EMPLOYEE ACCESS");

empQuery.displayAllEmployee();

//view employee table

this.addEmpAccess();

} else if (choice == 5){

//delete emp access

this.delAccess();

} else if (choice==6){

//

this.lineFormat();

empQuery.displayAllEmployee();

//view employee table this.xToCancel();

} else if (choice == 7){

empQuery.displayAllAccess();

this.viewAccess();

}else {

//cancel/return

services.services(logIn.qUsername, logIn.qLevel);

}

} else {

System.out.println("Invalid input. Enter an integer in ().");

this.employeeInfo();

}

} else {

System.out.println("Invalid input. Enter an integer in ().");

this.employeeInfo();

}

} catch (Exception e) {

System.out.println("\nInvalid Input! Enter choice enclosed in ().");

this.employeeInfo();

} } void delFormat() { this.lineFormat();

System.out.format("|%35s %28s |%n", "", "DELETE EMPLOYEE");

} void lineFormat(){ System.out.format("+-------------------------------------------------" + "---------------------------------------------------------------+%n");

} void deleteEmployee()

{ do

{ try {

System.out.print("\nEnter EMP\_ID of Employee to DELETE." + "\nType 'X' to cancel.");

String empIdString = sc.nextLine();

if (empIdString.equalsIgnoreCase("X"))

{ this.employeeInfo();

} else {

if (function.isNumber(empIdString))

{ int delEmpId = Integer.parseInt(empIdString);

//will delete user access from table access first for foreign key constraint requirement

System.out.println("Deleting Employee Access first for \nforeign key constraint fulfillment.");

empQuery.deleteAccess(delEmpId);

int rowsDeleted = empQuery.deleteEmployee(delEmpId);

//deletes from employee table if (rowsDeleted > 0){ this.delFormat();

empQuery.displayAllEmployee();

} else {

System.out.println("EMP\_ID is non-existent.");

this.deleteEmployee();

}

} else {

System.out.println("Invalid Input. Enter an integer.");

this.deleteEmployee();

}

}

}

catch (Exception e) {

System.out.println("Invalid input. Please enter a valid integer.");

}

}

while (true);

} void addEmpAccess() {

String level = "";

System.out.print("Enter EMP\_ID. Type 'X' to RETURN/CANCEL: ");

String searchIdString = sc.nextLine();

try {

if (searchIdString.equalsIgnoreCase("X")){

this.employeeInfo();

} else {

if (function.isNumber(searchIdString)){ int searchId = Integer.parseInt(searchIdString);

int rowsRetrieved = empQuery.selectEmployeeFieldsRet(searchId);

//searches employee table first if (rowsRetrieved == 1)

{

System.out.println("\nEmployee record retrieved.");

newEmpAccessFormat();

//PetopiaHeader

empQuery.selectAccess(searchId);

//searches access table next

System.out.print("Enter new username: ");

String newUser = sc.nextLine();

System.out.print("Enter new password: ");

String newPass = sc.nextLine();

do {

System.out.print("Enter access level \n(1) 'Admin' \n(2) 'Staff': ");

String access = sc.nextLine();

try { int accessLevel = Integer.parseInt(access);

if (accessLevel == 1 || accessLevel == 2) { level = (accessLevel == 1) ? "Admin" : "Staff";

empQuery.insertAccess(newUser, newPass, level, searchId);

newEmpAccessFormat();

empQuery.selectAccess(searchId);

this.xToCancel();

break;

// This will exit the loop

} else {

System.out.println("Invalid input. Please enter 1 for Admin or 2 for Staff.");

}

} catch (NumberFormatException e) {

System.out.println("Invalid input. Please enter a valid integer.");

}

} while (true);

} else {

System.out.println("EMP\_ID is non-existent or it must have been deleted.");

this.xToCancel();

}

}

}

}

catch (NumberFormatException e) {

System.out.println("Invalid input. Please enter a valid integer.");

}

}

void delAccessByEmpId (){

System.out.print("Enter EMP\_ID: ");

String empIdInput = sc.nextLine();

if (function.isNumber(empIdInput)){

int empId = Integer.parseInt(empIdInput);

empQuery.deleteAccess(empId);

empQuery.displayAllAccess();

} else {

System.out.println("Invalid input. Enter an Integer.");

this.delAccessByEmpId();

}

}

**Void delete access**

void delAccessByUsernameAndPassword() {

System.out.print("Enter USERNAME: ");

String username = sc.nextLine();

System.out.print("Enter PASSWORD: ");

String password = sc.nextLine();

empQuery.deleteAccessByUserPass(username, password);

empQuery.displayAllAccess();

} void delAccess()

{ try {

System.out.print("DELETE BY: \n(1) EMP\_ID or\n(2) USERNAME & PASSWORD \nType 'X' to RETURN/CANCEL: ");

String input = sc.nextLine();

if (input.equalsIgnoreCase("X")) { this.employeeInfo();

} else {

if (function.isNumber(input)){

int delChoice = Integer.parseInt(input);

if (delChoice > 0 && delChoice ❤) {

//1 - 2 System.out.format("\n+---------------------------------------------------------+%n");

System.out.format("|%10s %30s |%n", "", "DELETE EMPLOYEE ACCESS");

empQuery.displayAllAccess();

if (delChoice == 1) {

//del by emp\_id delAccessByEmpId();

} else {

//user and pass delAccessByUsernameAndPassword();

}

} else {

System.out.println("Select 1 or 2.");

this.delAccess();

}

}

}

this.delAccess();

} catch (java.util.InputMismatchException e) {

System.out.print("Invalid input. Please enter a valid integer value " + "for EMP\_ID.");

sc.nextLine();

// Clear the input buffer this.delAccess();

}

} public void xToCancel(){

System.out.print("\nType 'X' to RETURN/CANCEL.");

String input = sc.nextLine();

if (input.equalsIgnoreCase("X")) {

this.employeeInfo();

} else {

System.out.println("Invalid input."); this.xToCancel();

}

}

**Void view access**

void viewAccess(){

try {

this.xToCancel();

} catch (java.util.InputMismatchException e) {

System.out.print("Invalid input. Please enter a valid integer value " + "for EMP\_ID.");

sc.nextLine();

// Clear the input buffer this.delAccess();

}

} void searchName(){

System.out.print("Enter first 1 - 3 letter(s) of Employee's name: ");

String name = sc.nextLine();

boolean containsLetters = false;

for (char c : name.toCharArray()) {

if (Character.isLetter(c)) {

containsLetters = true;

break;

}

} if (containsLetters) {

this.empSearch(name);

} else {

System.out.println("\nInvalid input! Enter characters.\n");

this.searchName();

sc.nextLine();

}

} void empSearch(String name){

List <Integer> likeList = empQuery.searchEmployeeLike(name);

//SQL LIKE

try {

System.out.print("\nEnter EMP\_ID of Employee to Update.\nType 'X' to RETURN/CANCEL: ");

String input = sc.nextLine();

if (input.equalsIgnoreCase("X")) {

// Cancel the update operation

this.employeeInfo();

} else {

if(function.isNumber(input)){

int empId = Integer.parseInt(input);

System.out.println(empId);

boolean found = false;

for (Integer like : likeList)

{ if (like == empId)

{

//if the empId is equal to an item in the likeList

then proceed found = true;

empQuery.selectEmployeeFields(empId);

this.updateEmployee(empId);

break;

}

} if (!found) {

System.out.println("EMP\_ID not in the retrieved list");

this.empSearch(name);

}

} else {

System.out.println("Invalid input. enter an integer.");

this.empSearch(name);

}

}

} catch (java.util.InputMismatchException e) {

System.out.print("\nInvalid input.\nPlease enter a valid " + "integer value \nfor EMP\_ID.\n\n"); sc.nextLine();

// Clear the input buffer

this.empSearch(name);

}

}

**Void update employee**

void updateEmployee(int empId) {

int updateField;

String newData;

String fieldToUpdate, field = "";

try {

System.out.print("\nEnter field# to update. \nType 'X' to RETURN/CANCEL: ");

String updateFieldInput = sc.nextLine();

if (updateFieldInput.equalsIgnoreCase("X")) {

this.employeeInfo();

return;

} else {

if (function.isNumber(updateFieldInput)){

try { updateField = Integer.parseInt(updateFieldInput);

if (updateField > 5){

//fields up to archive is 5

System.out.println("Field count is only 5.");

this.updateEmployee(empId);

} else {

field = this.switchUpdateField(updateField);

boolean isUpdated = false;

while (!isUpdated) {

System.out.print("Enter new " + field + " data: " );

newData = sc.nextLine();

switch (updateField) {

case 1:

System.out.println("\nUpdating FULLNAME...");

fieldToUpdate = "fullname";

break;

case 2:

System.out.println("\nUpdating EMAIL...");

fieldToUpdate = "email";

break;

case 3:

System.out.println("\nUpdating ADDRESS...");

fieldToUpdate = "address";

break;

case 4:

System.out.println("\nUpdating POSITION...");

fieldToUpdate = "position";

break;

default:

//ARCHIVE

try {

int newDataInput = Integer.parseInt(newData);

fieldToUpdate = "archived";

if (newDataInput >= 0 && newDataInput < 2) {

// 0 and 1 only

empQuery.updateEmployee(fieldToUpdate, newData, empId);

isUpdated = true;

// Set the control variable to true

} else {

System.out.println("Invalid input. Enter only 1 or 0.");

}

} catch (NumberFormatException e)

{ System.out.println("Invalid input for 'Archived' field." + " \nPlease enter 1 or 0.");

continue;

// Continue the loop to prompt for input again

}

} empQuery.updateEmployee(fieldToUpdate, newData, empId);

isUpdated = true;

// Set the control variable to true

}

}

} catch (NumberFormatException e) {

System.out.println("Invalid input for field number. Please enter a number.");

return;

}

} else {

System.out.println("Invalid Input. Enter an integer.");

this.updateEmployee(empId);

}

}

} catch (InputMismatchException e) {

System.out.print("\nInvalid input.\nPlease enter a valid integer value for EMP\_ID.\n");

sc.nextLine();

// Clear the input buffer

this.updateEmployee(empId);

}

}

**Switch update field**

String switchUpdateField(int updateField){

String field;

switch (updateField) {

case 1:

field = "FULLNAME";

break;

case 2:

field = "EMAIL";

break;

case 3:

field = "ADDRESS";

break;

case 4:

field = "POSITION";

break;

default: field = "ARCHIVED";

break;

}

return field;

}

}

**Generate receipt**

package com.petopia;

import com.petopia.api.GenerateReceiptQuery;

import com.petopiables.PetopiaHeader;

import com.petopiables.Functions;

import java.util.List;

import java.util.Scanner;

public class GenerateReceipt extends PetopiaHeader{

public Scanner sc = new Scanner(System.in);

public static GenerateReceiptQuery genQuery = new GenerateReceiptQuery();

public Functions function = new Functions();

public void generateReceipt(){

//instantiations Services services = new Services();

LogIn logIn = new LogIn();

petopiaHeader();

System.out.format("| GENERATE RECEIPTS |%n");

System.out.format("+---------------------------------------+%n");

System.out.format("| (1) VIEW RECEIPTS TABLE |%n"); System.out.format("| (2) PRINT RECEIPT |%n");

System.out.format("| (3) VOID RECEIPT |%n");

System.out.format("| (4) CANCEL/RETURN |%n"); System.out.format("+---------------------------------------+%n");

try {

System.out.print("Enter a service: ");

String choiceInput = sc.nextLine();

if (function.isNumber(choiceInput)){

Integer choice = Integer.parseInt(choiceInput);

if (choice > 0 && choice < 5) {

if (choice == 1){

//VIEW RECEIPTS TABLE whether void or not

this.viewReceipt();

this.xToCancel();

} else if (choice == 2) {

//print receipt this.printReceipt();

this.xToCancel();

} else if (choice == 3) {

//void receipts this.voidReceipt();

this.xToCancel();

} else {

services.services(logIn.qUsername, logIn.qLevel);

}

} else {

System.out.println("Enter an integer in ().");

this.generateReceipt();

}

} else {

System.out.println("Enter an integer in ().");

this.generateReceipt();

}

} catch (Exception e) {

System.out.println("Invalid Input! Enter choice enclosed in ().");

this.generateReceipt();

}

} public void xToCancel(){

System.out.print("\nType 'X' to RETURN/CANCEL.");

String input = sc.nextLine();

if (input.equalsIgnoreCase("X")) {

this.generateReceipt();

} else {

System.out.println("Invalid input.");

this.xToCancel();

}

} void viewReceipt(){

this.lineFormat();

System.out.format("|%35s %32s |%n", "", "VIEW RECEIPTS");

genQuery.selectAllReceipts();

boolean validInput = false;

int inv = 0;

while (!validInput) {

System.out.print("Enter Invoice # to VIEW. Type 'X' to RETURN/CANCEL: ");

String input = sc.nextLine();

if (input.equalsIgnoreCase("X")) {

this.generateReceipt();

this.xToCancel();

return;

} else {

if (function.isNumber(input)) {

inv = Integer.parseInt(input);

validInput = true;

break;

// exit the loop after setting

validInput to true

} else {

System.out.println("Invalid input. Enter an integer.");

this.printReceipt(); }

}

} genQuery.selectVoidReceipt(inv);

} void printReceipt(){

this.lineFormat();

System.out.format("|%35s %32s |%n", "", "PRINT RECEIPTS");

List <Integer> invList = genQuery.selectReceiptsId();

genQuery.selectAllReceiptsNotVoid();

//only prints non-void receipts boolean

validInput = false;

int inv = 0;

while (!validInput) {

System.out.print("\nEnter a receipt or invoice# to PRINT.\nType 'X' to RETURN/CANCEL: ");

String input = sc.nextLine(); if (input.equalsIgnoreCase("X")) { this.generateReceipt();

this.xToCancel();

return;

} else {

if (function.isNumber(input)) {

inv = Integer.parseInt(input);

validInput = true;

break;

// exit the loop after setting validInput to true

} else {

System.out.println("Invalid input. Enter an integer.");

this.printReceipt();

}

}

} if (!invList.contains(inv)) {

System.out.println("Invalid input. Select an integer ().");

return;

// return from the method if input doesn't match any invoice numbers

}

System.out.println("\n\nGenerating Official Receipt......\n\n");

System.out.format("+--------------------------------------------------------+%n"); System.out.format("| OFFICIAL RECEIPT |%n"); System.out.format("| PETOPIA GROOMING SERVICE |%n"); System.out.format("+--------------------------------------------------------+%n\n"); genQuery.selectReceipt(inv);

System.out.format("\n----------------------------------------------------------%n\n");

} void voidReceipt() {

this.lineFormat();

System.out.format("|%35s %32s |%n", "", "VOID RECEIPTS");

List <Integer> invList = genQuery.selectReceiptsId();

genQuery.selectAllReceiptsNotVoid();

boolean validInput = false;

int inv = 0;

while (!validInput) {

System.out.print("\nEnter a receipt or invoice# to VOID.\nType 'X' to RETURN/CANCEL: ");

String input = sc.nextLine();

if (input.equalsIgnoreCase("X")) {

this.generateReceipt();

this.xToCancel();

return;

} else {

if (function.isNumber(input)) {

inv = Integer.parseInt(input);

validInput = true;

break;

// exit the loop after setting validInput to true

} else {

System.out.println("Invalid input. Enter an integer.");

this.voidReceipt();

}

}

} if (!invList.contains(inv)) {

System.out.println("Invalid input. Select an integer ().");

return;

// return from the method if input doesn't match any invoice numbers

}

System.out.println(inv);

this.lineFormat();

System.out.format("|%35s %32s |%n", "", "VOID RECEIPTS");

genQuery.voidReceipt(inv);

} public void lineFormat(){

System.out.format("+--------------------------------------" + "------------------------------------------------------------------------------------+%n");

}

}