Group 2

The Employee Management System

Software Design Document

Name (s): Jarhud Augustin,
Linh Pham,
Marie Ezoua,
Vibi Chari.

Date: 07/25/2024

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of The Employee Management System. It acts as a reference for developers, providing them with detailed information on how to implement the various components of the system. The intended audience is developers, project managers, administrators, and stakeholders who have a vested interest in the project's success.

1.2 Scope

The employee database management service is intended for administrators to access key employee information. The administrator should be able to access the employee information and pay statement history databases, and be able to modify these tables using simple commands without any prior software knowledge. The benefits of this project are that administrators can perform high-level tasks related to employee records by using simple commands, without needing to either learn MYSQL or other software.

1.3 Overview

This document is split up into multiple sections. The System Overview section provides a general description of the software functionality and design of the program. The System Architecture section explains all of the classes and methods in the program, shows how all of the software is connected, and the design rationale choices that were made while creating the project. The Data Design section describes how the major data or system entities are stored, processed, and organized. The Component Design section summarizes all of the components of the program in pseudocode. The Human Interface Design section shows an overview of the user interface and shows pictures of how the interface looks when an administrator is using the application. The Requirement Matrix section shows the relationship between software requirements and the code that completes these requirements.

1.4 Reference Material

Java JDBC API. (n.d.). https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/

1.5 Definitions and Acronyms

Acronyms:

- JDBC: Java Database Connectivity API, allows the software to access the MYSQL databases.
- SSN: Social Security Number, a nine-digit number issued to United States citizens, these are used to identify individual employees.

- UI: User interface, allows the user to interact with the software through graphical icons and visual indicators.
- EMPID: Identification, a unique number that identifies individual employees.

Definitions:

- MYSQL: An open-source relational database management system.

2.0 SYSTEM OVERVIEW

The employee database management service functionality includes:

- Database Access: View the database in its entirety.
- Employee Information Access: Search, update, and add new employees to the database.
- Pay Statement History Access: Search for all pay statements for a specific employee.
- Salary Updating: Update employee salary based on a percentage increase and give a minimum and maximum salary.
- View Total Pay: Views the total sum of each company division or job title given a specific month in the current calendar year.
- Add Column to database: Allows access to update the existing database by adding a column.

The software is designed for organizations seeking to enhance their employee data management processes. It is particularly useful in contexts where the administrators managing the system may not have technical expertise in databases or software, necessitating an intuitive and user-friendly interface that requires no technical skills.

3.0 SYSTEM ARCHITECTURE

3.1 Architectural Design

The ClientInteration class will be the main way the customer can interact with the software. It allows the user to access different functions of the software.

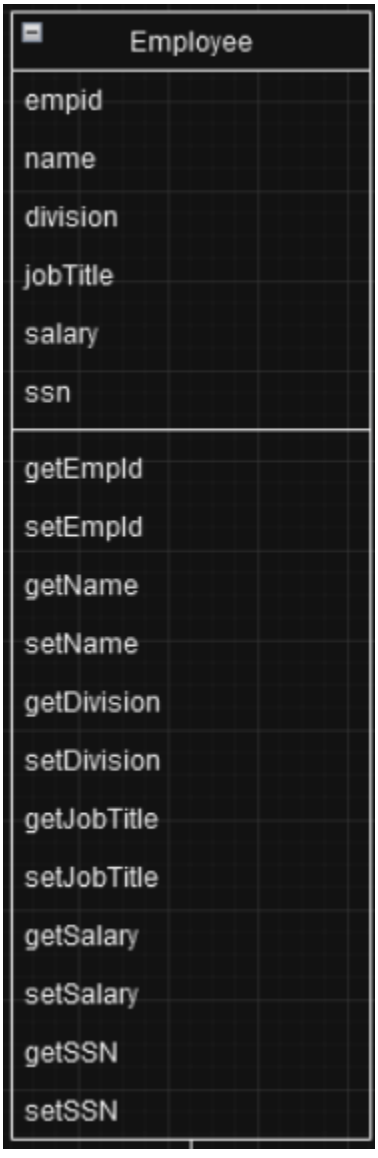
The employee class is used to store data that the software pulls from the Employee table.

The DatabaseTableFunctionality class, implemented from DatabaseFunctionalityInterface interface, houses the functions that interact with tables in the database.

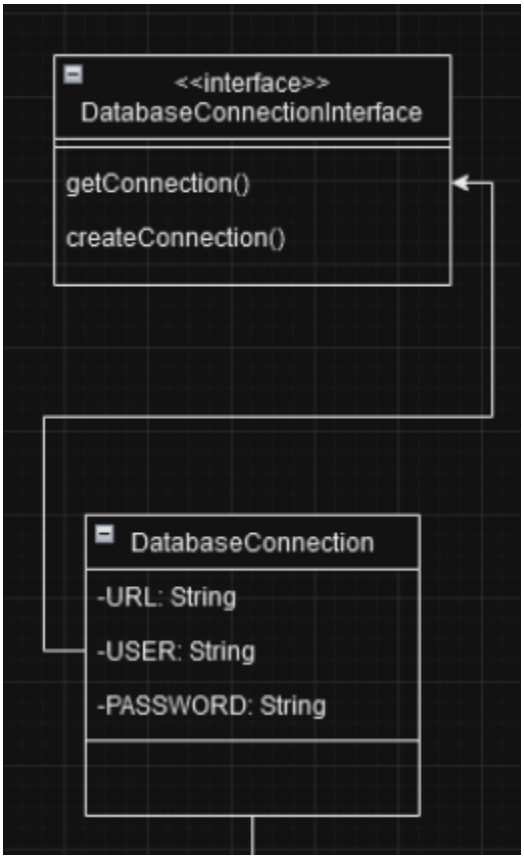
The DatabaseConnection class, implemented from the DatabseConnectionInterface interface, creates the connection between the software and the MySql database.

The EmployeeDao class houses functions that interact with the Employee table.
The PayStatementDao class contains functions that interact with the PayStatement table.
The Main class is where the software is started.

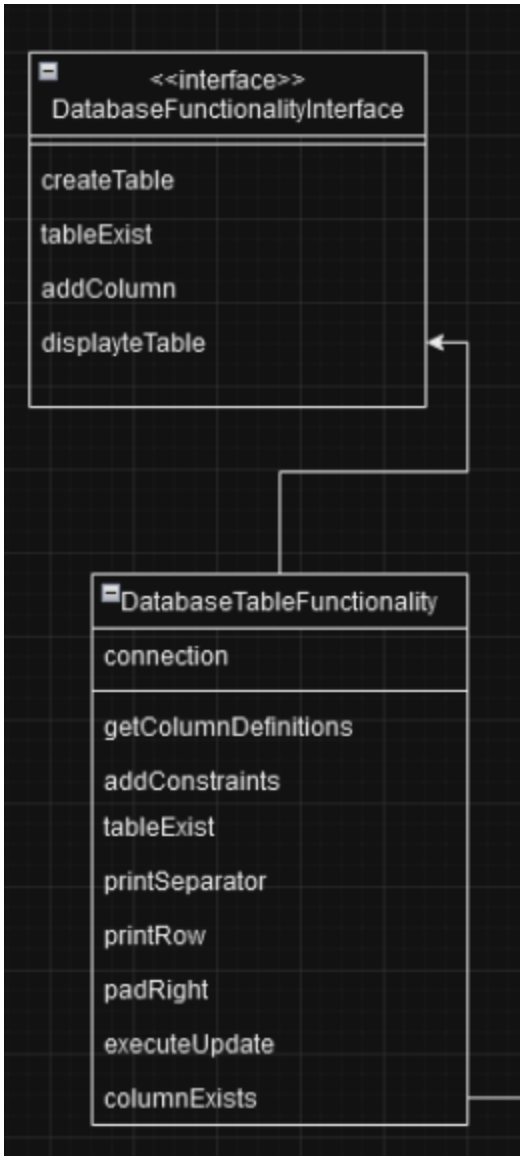
3.2 Decomposition Description



The Employee class provides a way for the data from the Employee table to be stored and manipulated. It has private variables that can be used to store individual elements of the table. Since the variables are private, we have getter and setter functions so the software is able to access the variables.



The DatabaseConnection class implements functions from the DatabaseConnectionInterface interface. This class has the location of the database, username, and password. It allows the software to connect to the database.



The DatabaseTableFunctionality class implements functions from the DatabaseTableFunctionality interface.
createTable() creates a table in the database
tableExist() verifies the existence of a table
addColumn() add a new column to a table
displayTable() showing a table with its current data

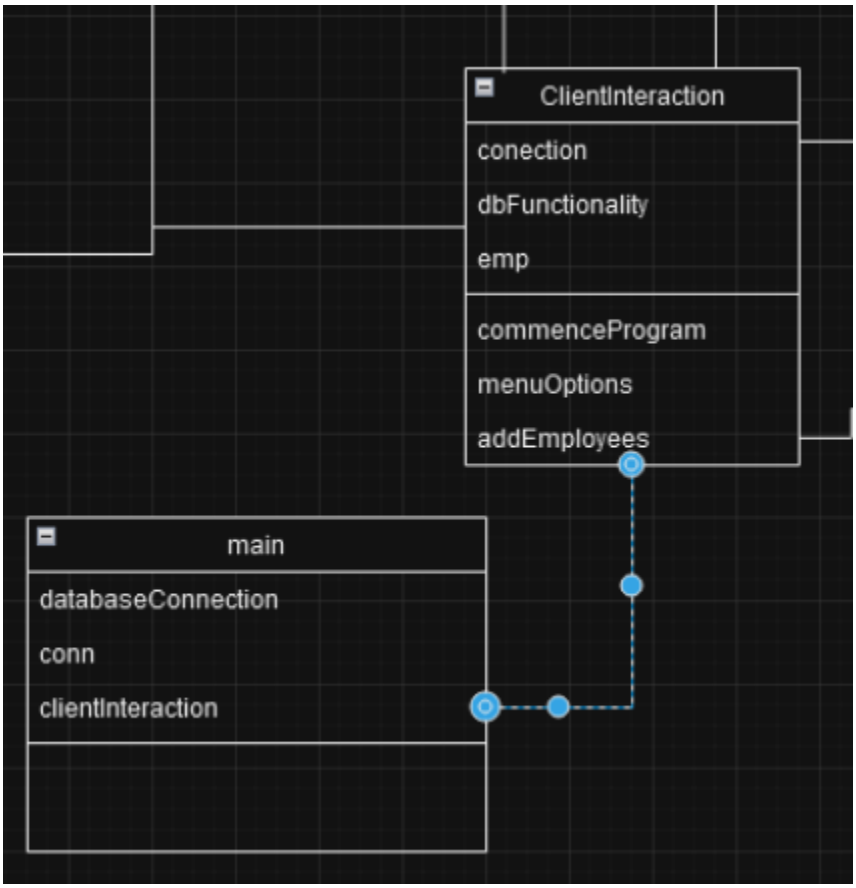
The DatabaseTableFunctionality class also has other unique functions:
getColumnDefinitions() stores the default parameters of some tables that the user can use to create tables

addConstraints() alter the PayStatement table to ensure the table meets the requirements.
printSepator() creates an outline of the table when displayed, increases the visibility of the table

padRight() increases the visibility of the table

executeUpdate() a function that creates a connection with the database, then uses JDBC to send a SQL command to the database.

columnExists() checking if a table has a specific column

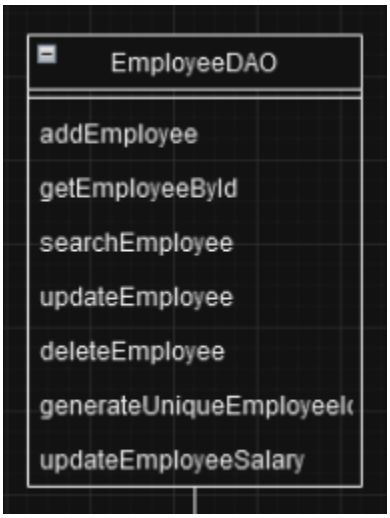


The Main class is where the software is run. It creates a connection to the database. It also calls the ClientInteraction class.
The ClientInteraction class is where the UI is created and shown via the console. It allows the user to interact with the software and its functions.

commenceProgram() displaying options for the user to choose. It also verifies the inputs.

menuOptions() taking the input from commenceProgram() to direct the user to the appropriate function

addEmployee() prompts the user to enter the necessary information. An Employee class is then created and sent to EmployeeDAO.addEmployee() for processing.



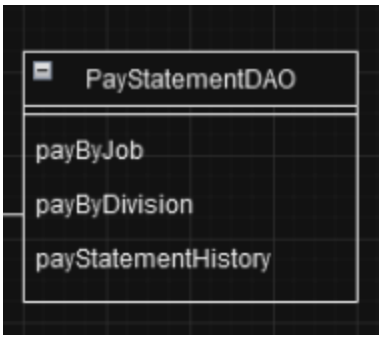
The EmployeeDAO class contains all of the functions that can interact with the Employee table:
addEmployee() taking information from the user to create an INSERT SQL command. Then send it to the database using JDBC.
getEmployeeById() taking an input of an Employee ID. Create a SELECT SQL command. Then send it to the database using JDBC. Finally, returning all of the information of that Employee in a class.

searchEmployee() user can input either ssn, id, or name of an employee. The input will select the appropriate input and create a SELECT SQL command. Then send it to the database using JDBC. The function will print out all of the information of the target Employee.

updateEmployee() receives an Employee class from the user. Extract the information via getters from the Employee class. Create an UPDATE SQL command. Then send it to the database using JDBC.

deleteEmployee() The program will ask the user to input the target Employee ID number.
generateUniqueEmployeeId() generates a unique number for an individual employee. Generate a SQL SELECT command with the new number. Then send it to the database using JDBC.

updateEmployeeSalary() prompting the user for the percentage the salary will increase to. The user can then input the range of salary that will be affected. All of the information will be used to create an UPDATE SQL command. Then send it to the database using JDBC.



The PayStatementDAO class contains functions that can view and display information from the PayStatement table via the console

payByJob() The user can input a specific month. The function will create a SELECT SQL command based on the date. Then send it to the database using JDBC. Finally, the function will print out the Job Title and the Total Amount the company has paid in the specific month.

payByDivision() The user can input a specific division name. The function will create a SELECT SQL command based on the division name. Then send it to the database using JDBC. Finally, the function will print out the Division name and the Total Amount the company has paid for that division.

payStatementHistory() The user can input an Employee ID. The function will create a SELECT SQL command based on the id. Then send it to the database using JDBC. Finally, the function will print out the Employee information along with their Payment information.

3.3 Design Rationale

This software was designed to be modular. Using classes, the software is divided into different modules, each one targeting a specific aspect of the database. By segregating the codes, each function is isolated, making it easier to code and debug. Modularity can also help in the reusability of the code. Different programmers were able to implement functions that they didn't create. Finally, the code is scalable, which means it will be very easy for the code to be changed and modified based on the growth of the company.

4.0 DATA DESIGN

4.1 Data Description

The information domain of the employee management system is transformed into structured data stored in MySQL. The major data entities include EmployeesDAO, Pay StatementDAO. These entities are organized into corresponding tables in the database. The data is processed using SQL queries to perform various operations such as retrieving pay statements, calculating total pay for job titles or divisions, adding columns, updating employee records, and searching for employee information.

4.2 Data Dictionary

1. Employee

- Attributes:
  - **empId INT PRIMARY KEY**: Identifier for the employee
  - **name VARCHAR(100)**: Name of the employee
  - **division VARCHAR(50)**: Division of the employee
  - **jobTitle VARCHAR(50)**: Job role of the employee
  - **salary DECIMAL(10, 2)**: current salary of the employee
  - **ssn CHAR(9)**: Social Security Number of the employee

- Methods
  - **void addEmployee(Connection conn, String tableName, Employee employee)**: Adds employee
  - **void updateEmployee(Connection conn, Scanner inputScanner, String tableName)**: Update Employee data
  - **getEmployeeById(String empId, Connection conn, String tableName)**: Get employee by the ID
  - **searchEmployee(Connection conn, Scanner inputScanner, String tableName)**: search for Employee
  - **deleteEmployee(Connection conn, Scanner inputScanner, String tableName)**: Delete employee
  - **int generateUniqueEmployeeId(Connection conn, String tableName):**generate Unique Employee Id

- **void updateEmployeeSalary(Connection conn, Scanner inputScanner, String tableName):** update Employee Salary

2. PayStatementDAO
    - Attributes:
        - **payStatementId INT PRIMARY KEY :** Identifier for the Pay Statement
        - **empId INT:** Identifier for the Pay Statement
        - **payDate DATE :** Date of the pay statement
        - **amount DECIMAL(10, 2)**: the amount of the Pay statement

          FOREIGN KEY (empId) REFERENCES Employee(empId)

    - Methods
        - **payByJob(Connection conn, Scanner inputScanner)**: Total pay for a month by job title
        - **void payByDivision(Connection conn, Scanner inputScanner):** Total pay for month by division
        - **void payStatementHistory(Connection conn, Scanner inputScanner):** History of the pay statements

5.0 COMPONENT DESIGN

addEmployee(connection, tableName, employee)
    Create SQL insert statement with tableName and employee details
    Prepare the statement
    Set employee details in the prepared statement
    Execute the update

getEmployeeById(empId, connection, tableName)
    Create SQL select statement with tableName and empId
    Prepare the statement
    Set empId in the prepared statement
    Execute the query
    If employee is found
        Create an Employee object and set its properties from the result set
        Return the Employee object
    Else
        Return null

searchEmployee(connection, inputScanner, tableName)
    Prompt for SSN, employee ID, and name (optional)
    Create SQL select statement with tableName and dynamic conditions based on inputs
    Prepare the statement
    Set inputs in the prepared statement if they are provided
    Execute the query
    For each result in the result set
        Print employee details

updateEmployee(employee, connection, tableName)
    Create SQL update statement with tableName and employee details
    Prepare the statement
    Set employee details in the prepared statement
    Execute the update

deleteEmployee(connection, inputScanner, tableName)
    Prompt for employee ID
    Create SQL delete statements for PayStatement and Employee tables with tableName and empId
    Prepare the statements
    Set empId in the prepared statements

Execute the updates
Print confirmation message

generateUniqueEmployeeId(connection, tableName)
Initialize random number generator
Loop until a unique ID is generated
Generate a random 9-digit employee ID
Create SQL select statement with tableName and empId
Prepare the statement
Set empId in the prepared statement
Execute the query
If no employee with the generated ID exists, break the loop
Return the unique employee ID

updateEmployeeSalary(connection, inputScanner, tableName)
Prompt for salary increase percentage, minimum salary, and maximum salary
Calculate the multiplier based on the percentage increase
Create SQL update statement with tableName and salary conditions
Prepare the statement
Set multiplier, minimum salary, and maximum salary in the prepared statement
Execute the update

payByJob(connection, inputScanner)
Prompt for pay month
Create SQL select statement to sum payments by job title for the given month
Prepare the statement
Set month in the prepared statement
Execute the query
For each result in the result set
Print job title and total pay

payByDivision(connection, inputScanner)
Prompt for pay month
Create SQL select statement to sum payments by division for the given month
Prepare the statement
Set month in the prepared statement
Execute the query
For each result in the result set
Print division and total pay

```
payStatementHistory(connection, inputScanner)
        Prompt for employee ID
        Create SQL select statement to get pay statement history for the given employee ID
        Prepare the statement
        Set employee ID in the prepared statement
        Execute the query
        For each result in the result set
                Print employee ID, full name, pay date, and amount

 createTable(tableName)
        if table does not exist
                get column definitions for the table
                if columns are null
                        throw IllegalArgumentException("Unknown table: " + tableName)
                build SQL create table statement
                execute the update and print the success message
                add constraints to the table if any
        else
                print "Table already exists"

 getColumnDefinitions(tableName)
        create a map for columns
        switch (tableName)
                case "Employee":
                        add employee column definitions to the map
                case "PayStatement":
                        add pay statement column definitions to the map
                case "TestTable":
                        add test table column definitions to the map
                default:
                        return null
        return columns

addConstraints(tableName)
        if tableName is "PayStatement"
                build SQL alter table statement to add primary and foreign keys
        if addConstraintsSQL is not null
                execute the update and print the success message

tableExist(tableName)
```

```
        initialize exists to false
        get table metadata
        if table metadata exists
                set exists to true
        return exists

addColumn(tableName, columnName, dataType)
        if column does not exist
                build SQL alter table statement to add column
                execute the update and print success message
        else
                print "Column already exists"

deleteColumn(tableName, columnName)
        if column exists
                build SQL alter table statement to drop column
                execute the update and print success message
        else
                print "Column does not exist"

displayTable(tableName)
        if table does not exist
                print "Table does not exist"
                Return
        build SQL select statement
        execute the query
        get metadata and column count
        determine column widths
        print table header
        print separator
        while there are rows in the result set
                print each row
        print separator
```

```
printSeparator(columnWidths)
        for each width in columnWidths
                print separator
        print new line

printRow(metaData, columnWidths)
        for each column in metadata
                print column name padded to width
        print new line

padRight(text, length)
        return text padded to length

executeUpdate(sql, successMessage)
        create a statement and execute the update
        print success message
        if there is an error
                print error message
                throw the error

columnExists(tableName, columnName)
        initialize exists to false
        get column metadata
        if column metadata exists
                set exists to true
        return exists

commenceProgram(employeeTable, payStatementTable, conn) {
        create a Scanner object for user input
        print welcome message

        boolean loopFlag = true

        while loopFlag {
                print menu options
                get user input
                if userInput is invalid {
                        handle invalid input
                } else {
                        call menuOptions(userInput, employeeTable, inputScanner, conn)
```

```
                ask if user wants to continue
                if no, set loopFlag to false and print exit message
            }
        }
}

menuOptions(userInput, employeeTable, inputScanner, conn) {
        switch (userInput) {
            case 1:
                display table
            case 2:
                call addEmployees(inputScanner, conn, employeeTable)
            case 3:
                call updateEmployee(inputScanner, conn, employeeTable)
            case 4:
                call EmployeeDAO.searchEmployee(conn, inputScanner,
        employeeTable)
            case 5:
                call EmployeeDAO.updateEmployeeSalary(conn, inputScanner,
        employeeTable)
            case 6:
                call PayStatementDAO.payByJob(conn, inputScanner)
            case 7:
                call PayStatementDAO.payByDivision(conn, inputScanner)
            case 8:
                call PayStatementDAO.payStatementHistory(conn, inputScanner)
            case 9:
                call EmployeeDAO.deleteEmployee(conn, inputScanner,
        employeeTable)
            case 10:
                add column to TestTable
            case 11:
                print exiting message and exit program
            default:
                print invalid option message
          }
}

updateEmployee(scanner, conn, tableName) {
        get employee ID from user
```

```
        if empId is empty, prompt user again

        try {
                get employee by ID from EmployeeDAO
                if employee not found, print not found message and return
                get and set employee attributes from user input
                call EmployeeDAO.updateEmployee(emp, conn, tableName)
                print success message
        } catch (SQLException e) {
                print error message
        }
}
```

6.0 HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

From the perspective of the user, the system begins by displaying a menu of eleven options that will allow the user to access various functions of the system.

The first option in the menu allows the user to view the employee database. This menu option prints out all the contents of the employee databases through a formatted table resembling an actual database table.

The second option available to the user is “Add New Employee(s).” This option allows the user to add one or more employees to the database. After the user inputs all of the information, they are then asked whether they would like to save these changes. If they do, the changes will be saved; if not, the program will discard all inputs.

Option 3, “Update Employee Info,” allows the user to update specific employee information. In the console, the different attributes associated with that employee will be printed one by one into the console allowing the user to edit them.

Option 4 allows the user to search for an employee using either their social security number (SSN), their employee ID, or their name; these options will be printed to the console one by one allowing the user to enter the information available to them.
When using the name option, if two or more employees share a name, they will all be displayed in the console for the user.

Option 5 allows the user to update the pay of employees within a defined range. The user is first prompted for the minimum salary to update and then the maximum salary. Following this, the user is prompted to enter the percent that salaries should be increased by.

Options 6 and 7 have similar functionalities. Option 6, “View Pay By Job,” displays pay according to the roles of employees at the company, whereas option 7, “View Pay By Division,” displays the pay according to the company’s divisions. In the console, the job title or division is displayed first followed by the pay amount.

The user is given the ability to view the entire pay statement history of an employee through option number eight. This menu option prompts the user for the employee’s ID, which is then used to fetch all pay statement history concerning that employee. In the console, the employee ID is displayed followed by their name, the date the payment was made, and the amount that was paid to the selected employee.

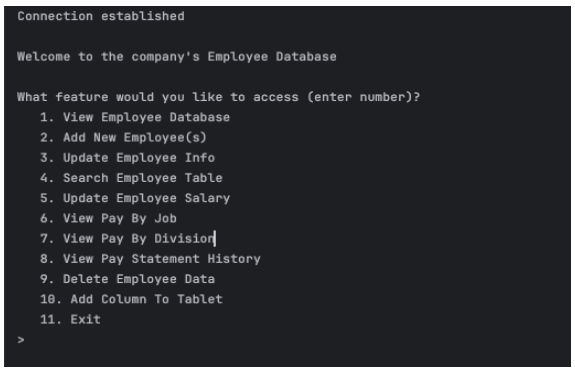
The ability to delete an employee's data is also given to the user through option nine. This option prompts the user for the employee ID and deletes the employee associated with that ID from the database then displays a confirmation message.

If the user wishes to expand the database table to include a new column of information, they can do this through number 10, "Add Column To Table."

Lastly, If the user does not want to perform any queries, they can exit the system using option eleven.

6.2 Screen Images

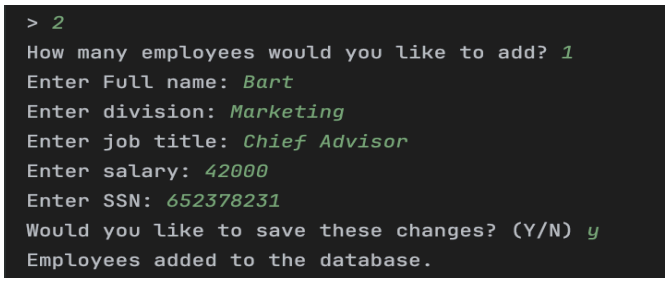
Introduction to the system and menu option from the User's perspective:



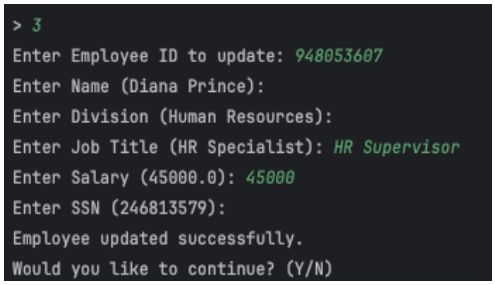
Users view of Employee database table displayed in through console (option 1):

empId	division	jobTitle	name	salary	ssn
102609667	Intelligence	Intelligence Analyst	Natasha Romanoff	70000.00	159753468
120884097	Legal	Policy Advisor	Julia Roberts	27405.00	489302567
170322558	Engineering	Chief Engineer	Tony Stark	90000.00	135792468
274503187	Human Resources	HR Specialist	Michael Johnson	45000.00	347802392
393384566	Legal	Lawyer	Lois Griffin	34012.19	809424584
472202073	Security	Security Guard	John Wick	23000.00	765849307
487713084	Research and Development	R&D Scientist	Wanda Maximoff	68000.00	753951486
516743903	Communications	Communications Director	Clark Kent	67000.00	321654987
581552625	Legal	Lawyer	Greg Martin	36115.72	849362942
601376829	Security	Security Specialist	Bruce Wayne	75000.00	789456123
655083760	Public Relations	PR Manager	Steve Rogers	55000.00	951753486
812122105	Marketing	Assistant Advisor	Viola Davis	31017.02	041820983
948053607	Human Resources	HR Specialist	Diana Prince	45000.00	246813579
990783579	Operations	Operations Manager	Sarah Connor	63240.00	987654321

User adding employee new employee to table (option 2)



User updating employee Diana Prince's information(option 3):



Users searching for a specific employee using their SSN (option 4):

```
> 4
Enter employee SSN or leave blank: 135792468
Enter employee ID or leave blank:
Enter employee name or leave blank:

Employee ID: 170322558
Full Name: Tony Stark
Division: Engineering
Job Title: Chief Engineer
Salary: 90000.0
SSN: 135792468
```

User increasing the pay of all employees within a specified range (option 5):

```
> 5
Enter salary increase % (EX: 3.2): 1.5
Enter minimum salary: 27000
Enter maximum salary: 34000
Would you like to continue? (Y/N)
```

User viewing pay by job position for the month of March (option 6, not complete output):

```
> 6
Enter pay month: 3
Job Title: Assistant Advisor
Total Pay: 2433.33

Job Title: Chief Engineer
Total Pay: 7500.0

Job Title: Communications Director
Total Pay: 5583.33

Job Title: HR Supervisor
Total Pay: 3750.0

Job Title: Intelligence Analyst
Total Pay: 5833.33
```

User viewing pay by division for March (option 7):

```
Enter pay month: 3
Division: Communications
Total Pay: 5583.33

Division: Engineering
Total Pay: 7500.0

Division: Human Resources
Total Pay: 3750.0

Division: Intelligence
Total Pay: 5833.33

Division: Legal
Total Pay: 2250.0

Division: Marketing
Total Pay: 2433.33

Division: Operations
Total Pay: 5166.67

Division: Public Relations
Total Pay: 4583.33

Division: Research and Development
Total Pay: 5666.67

Division: Security
Total Pay: 8166.67
```

The user viewing the pay statement history for a calendar year for an employee(option 8, not complete output):

```
> 8
Enter employee ID: 472202073
Employee ID: 472202073
Full Name: John Wick
Pay Date: 2023-01-01
Amount: 1916.67

Employee ID: 472202073
Full Name: John Wick
Pay Date: 2023-02-01
Amount: 1916.67

Employee ID: 472202073
Full Name: John Wick
Pay Date: 2023-03-01
Amount: 1916.67

Employee ID: 472202073
Full Name: John Wick
Pay Date: 2023-04-01
Amount: 1916.67
```

User deleting employee from the table (option 9):

```
What feature would you like to access (enter number)?
   1. View Employee Database
   2. Add New Employee(s)
   3. Update Employee Info
   4. Search Employee Table
   5. Update Employee Salary
   6. View Pay By Job
   7. View Pay By Division
   8. View Pay Statement History
   9. Delete Employee Data
   10. Add Column To Tablet
   11. Exit
> 9
Enter employee ID to delete: 413451600
Employee deleted.
```

User adding new table column to the table (option 10):

```
> 10
Enter table name: employee
Enter column name: ssn
Enter data type: char(9)
Column ssn added to table employee
Column added successfully.
```

6.3 Screen Objects and Actions

Main Menu:
The main menu provides the user with a list of actions they are able to perform in the database which are:

1. View Employee Database
2. Add New Employee(s)
3. Update Employee Info
4. Search Employee Table
5. Update Employee Salary
6. View Pay By Job
7. View Pay By Division
8. View Pay Statement History
9. Delete Employee Data
10. Add Column To Table
11. Exit

1. **View Employee Database:** This menu option accesses the "displayTable" method found in the "DatabaseTableFunctionality.java" class. Relying on both Java and SQL, this menu option displays the contents of the employee database table in a formatted way specified by the method "displayTable".

2. **Add New Employee(s):** This menu option allows the user to add new employees via the "addEmployee" found in the "EmployeeDAO.java" class. This menu option gives the user the choice to add numerous employees to the database.

3. **Update Employee:** The user is able to use this menu option to update various details about an employee, such as their name in the case of a name change, their divisions and job title if the employee takes on a different role, and their SSN in case of a mistake. This menu option gets its functionality from the "updateEmployee" found in "EmployeeDAO.java."

4. **Search Employee Table:** This option allows the user to search for the employee based on either their SSN, employee ID or their name. When selected, this menu option will call on the method "searchEmployee" from "EmployeeDAO.java" which provides all the functionality. After the employee has been found, their information will be displayed in the console. If the user decides to use an employee's name, all employees with the same name will be listed.

5. **Update Employee Salary:** With this menu option, the employee is able to update the salaries of all employees within a range defined by the user. It gains its functionality from

“updateEmployeeSalary” found in the “EmployeeDAO.java.” After the employee has entered all the information, this menu option then updates the employee database.

6. **View Pay By Job:** This menu option allows the employee to view the sum of all payments issued in a specified month for all job roles. The menu option gets its functionality from the “payByJob” method found in “PayStatementDAO.” The report is then displayed in the console for the user.

7. **View Pay By Division:** This menu option allows the employee to view the sum of all payments issued in a specified month for all company divisions. The menu option gets its functionality from the “payByDivision” method found in “PayStatementDAO.” The report is then displayed in the console for the user.

8. **View Pay Statement History:** Through this menu option, the user is able to view all payments made to an employee using their employee ID. This is all done through the “payStatementHistory” method found in “PayStatementDAO.” The report is then displayed in the console for the user.

9. **Delete Employee Data:** The user is also given the ability to delete employees from the database using this menu option. This menu option passes the employee ID which was passed in by the user into the “deleteEmployee” method found in “EmployeeDAO.java.” A confirmation message is then displayed in the console.

10. **Add Column To Table:** The user is able to use this option to add a new column to the database via the “addColumn” method found in the “DatabaseTableFunctionality.java” class. To prevent duplicate tables, the method will return a message if the column already exists.

11. **Exit:** This menu option allows the user to exit the system if they do not wish to use any of the functionality provided to them through other menu options.

7.0 REQUIREMENTS MATRIX

insert data in Emp table	static void addEmployees(Scanner inputScanner, Connection conn, String tableName)
Change employee table; add column SSN	public void addColumn(String tableName, String columnName, String dataType)
Full-time employee information with pay statement history	public static void payStatementHistory(Connection conn, Scanner inputScanner)
Total pay for month by job title	public static void payByJob(Connection conn, Scanner inputScanner)
Total pay for month by Division	public static void payByDivision(Connection conn, Scanner inputScanner)
Search for an employee using name, SSN, empid to show their information	public static void searchEmployee(Connection conn, Scanner inputScanner, String tableName)
Update an employee's data	public static void updateEmployee(Employee emp, Connection conn, String tableName)
Update employee's salary for an increase of a particular percentage only for a salary amount range	public static void deleteEmployee(Connection conn, Scanner inputScanner, String tableName)
delete data in Emp table	public static void deleteEmployee(Connection conn, Scanner inputScanner, String tableName)