School of IT
Faculty of SEBE
Deakin University

SIT322 Distributed Systems

Assignment 2

All 3 assignments in this unit involve creating and building upon a Customer Relationship Management (CRM) system for an Australian logistics company. In Assignment 2 the CRM system will have additional membership functionality with an associated database.

Assignment 2 involves development of a marketing analysis component where statistical results are derived based on a salary database. More specifically, an engineering company provides its employment information in a compressed database dump named as either an SQL database, or within CSV files, which consists of a few database tables such as department, titles, salary and so on. Your task is to categorize the employees into one of three membership classes:

- **Gold:** those who earn at least as much as the top 10% of salaries in the employee database
- **Silver:** those who earn at least as much as the top 30% of salaries in the employee database
- **Loyalty:** those who been with the company for 25 years or more (unless they are already a Gold or Silver member)
- **Regular:** are any remaining employees who do not qualify for the other memberships

Database Creation

The first task is to create a local MDF database as you did in Prac 6 Section 4. Your database should contain at least one table. This table should consist of at least the following columns:

- Membership ID (in plaintext)
- First name (in plaintext)
- Last name (in plaintext)
- Membership class (in plaintext)
- Date joined (in plaintext)
- Date-of-birth (in ciphertext)
- Salary (in ciphertext)
- Gender (in ciphertext)

You should insert several dummy entries before you encrypt the sensitive information. How the encryption is implemented is not important, you may use in-built database encryption, you may use encryption classes provided by the .NET framework and store/retrieve the data already encrypted, or you may even use your own encryption scheme (it does not need to be a secure scheme for this assignment, e.g. a Caesar cipher).

School of IT
Faculty of SEBE
Deakin University

Web interface

To manage memberships, you will need to create an ASP.NET website with one web form (an *incomplete example* of the UI can be seen in Appendix A). Remember that a good UI must be intuitive, therefore make it very clear what format you expect certain data to be in (e.g. is your date-of-birth meant to be year/month/day? Let the user know). On this web form (e.g. "Default.aspx"), there should be at least the following controls:

- SQL database controls for displaying membership information including membership ID, names, membership class and so on.
- "TextBox" controls for updating member's membership ID, names, membership class and so on.
- "Label" controls to indicate error messages
- A CSS component (keep it clean and simple)
- And a few "Button" controls to enable the database functionalities (as listed below)

In the C# file associated with the form (e.g. "Default.aspx.cs"), you should write code to implement the following actions:

- **View members:** functionality to view all current members and their details in your database
- **Migrate member:** functionality to migrate members from the provided employee database to your own database via their employee ID
- **Add member:** for adding a person not already in the provided employee database, so additional fields must be filled out, e.g. first/last name
- **Remove member:** for removing a person from your database, perhaps by their ID, or their full name
- **Update member:** for updating member details such as name, and salary (Note: it makes sense that fields such as date of birth and member ID should not be editable). Additionally, a check will need to be made to see if a member is still eligible for their current membership class (i.e. salary is still in a valid range).
- **Search member:** functionalities according to given member ID, first name and last name

Additionally, in order to demonstrate the correctness of your implementation of the business rules, you must include the screenshots of the following items:

1. A salary histogram of all your members
2. An age histogram of all members
3. A list of the members who:
    - Earn above the *average* salary of Gold class
    - Top 5% who have been with the company the longest

These outputs must be conducted on *your* database (and not the supplied database). Your database should include a list of dummy members that you have added in during the development period, as well as the first 500 employees from the suppled database that must also be imported into your own database (see below). This ensures that everybody should have roughly uniform outputs.

Database Migration

The last task is to utilize the employees data provided into your ASP.NET solution, by migrating a list of the *first 500* employees by their *employee ID* number. The list of employees that must be migrated into your own database can be found in the *employees_to_migrate.csv* file under Resources. These records must also be utilized for other components of the assignment, such as the Migrate Member feature, and the generating of the histograms.

Provided to you is the employee database records in two formats:

- **CSV files** (*employees.csv*, *salaries.csv, etc*): Comma Separated Values (csv) files are platform and language independent, making them ideal for parsing by your own application by manually translating the table structures and inserting the data. (Hint: You might have to make use of "BULK INSERT" when reading the data from the csv format).

- **MySQL database dump:** The employee tables originated from a MySQL database file (note that MySQL and MS SQL are not the same!). There are ways for .NET environments to read other database types, however additional plugins or libraries may be required. Third-party software may assist in a direct parsing from MySQL to MSSQL as well.

Prototype

A prototype of the assignment has been provided on CloudDeakin. This prototype is of the front end and is given to clarify how users are expected to interface with the assignment. This prototype does not have all features required of the assignment and does not contain any back-end component. Assignments may use any functional aesthetic and layout, the aesthetic and layout of the prototype is not a requirement (Remember that your assignment needs a CSS component as well).

Assignment total: 15 marks
Assignment 2 Due: April 24, 2016.
Assignment submission: Upload the compressed (zipped) solution folder containing source codes and necessary auxiliary files to assignment dropbox via CloudDeakin.
Special requirements: When submitting your solution please ensure to *delete* any of the database dump files that were provided to you (e.g. employees.sql, salaries.csv etc). They may be in the App_Data folder, or even in the bin and obj folders. This is to drastically reduce the size of your submission file.

School of IT
Faculty of SEBE
Deakin University

Marking Rubrics

Criteria	Missing or unable to assess implementation 0 points	Below Expectation - Incorrect implementation 0.3 points	Satisfactory - Functional implementation with some errors 0.65 points	Proficient - Correct implementation with few errors 1 point
Creation of an MS SQL database with stored and migrated members and their details	Implementation was missing or could not be assessed.	Lacked fields necessary for storage of members and generation of reports, with mostly appropriate SQL datatypes and encryption. Members were migrated successfully, with documentation explaining how this was achieved (e.g. on your UI, in code, or in an attachment).	Has most fields necessary for storage of members and generation of reports, with appropriate SQL datatypes and encryption if required. All 500 members were migrated successfully, with documentation explaining how this was achieved (e.g. on your UI, in code, or in an attachment).	Has all fields necessary for storage of members and generation of reports, with appropriate SQL datatypes and encryption if required. All 500 members were migrated successfully, with documentation explaining how this was achieved (e.g. on your UI, in code, or in an attachment).
Demonstration of implementing the business rule based on membership class requirements	Implementation was missing or could not be assessed.	Few members were accurately assigned a membership class based on the specifications.	Most members were accurately assigned a membership class based on the specifications.	All members were accurately assigned a membership class based on the specifications.
Demonstration of obtaining the data required to create histograms of members' age and salary	Implementation was missing or could not be assessed.	Somewhat accurate histograms of all members' salary and ages were generated (within your ASP.NET solution or by third-party tools). Code demonstrating how the required data was generated can be clearly seen.	Mostly accurate and informative histograms of all members' salary and ages were generated (within ASP.NET solution or by third-party tools). Code demonstrating how the required data was generated can be clearly seen.	Accurate and informative histograms of all members' salary and ages were generated (within ASP.NET solution or by third-party tools). Code demonstrating how the required data was generated can be clearly seen.
Demonstration of generating a list of members who meet the required specification	Implementation was missing or could not be assessed.	Questionable accuracy of lists of members who meet requirements 3.1 and 3.2. Code that demonstrates how the required data was generated can be clearly seen.	Mostly accurate and informative lists of members who meet requirements 3.1 and 3.2. Code that demonstrates how the required data was generated can be clearly seen.	Accurate and informative lists of members who meet requirements 3.1 and 3.2. Code that demonstrates how the required data was generated can be clearly seen.
Implementation of "View Members" functionality	Implementation was missing or could not be assessed.	Somewhat accurate implementation allows Viewing of all members' details.	Mostly correct implementation allows accurate and informative Viewing of all members'	Correct implementation allows accurate and informative Viewing of all members'

			details.	details. Ideally we can limit how many members are viewed at a time.
Implementation of "Search Member" functionality	Implementation was missing or could not be assessed.	Somewhat accurate implementation allows Searching of members by their ID and/or their fullname.	Mostly correct implementation allows accurate and informative Searching of members by their ID and/or their fullname.	Correct implementation allows accurate and informative Searching of members by their ID and/or their fullname.
Implementation of "Add Member" and "Delete Member" functionality	Implementation was missing or could not be assessed.	Implementation somewhat allows for Adding of members however lacks required fields, or fails to avoid duplicate records and/or incorrect input data, and the Removing of existing members by their ID and/or fullname.	Mostly correct implementation allows accurate Adding of members with all required fields, avoiding duplicate records and incorrect input data, and/or the Removing of existing members by their ID and/or fullname.	Correct implementation allows accurate Adding of members with all required fields, avoiding duplicate records and incorrect input data, and the Removing of existing members by their ID and/or fullname.
Implementation of "Update Member" functionality	Implementation was missing or could not be assessed.	Implementation somewhat allows for Updating some fields of an existing member, and the membership class of that member may update to reflect the changes.	Mostly correct implementation allows for Updating the appropriate fields of an existing member, and the membership class of that member accurately updated to reflect the changes.	Correct implementation allows for Updating the appropriate fields of an existing member, and the membership class of that member accurately updated to reflect the changes.
Implementation of "Migrate Member" functionality	Implementation was missing or could not be assessed.	Implementation somewhat allows for Migrating of members from the supplied database into your own, with most of the required fields and correct parsing of datatypes.	Mostly correct implementation allows accurate Migrating of members from the supplied database into your own, with most of the required fields and parsing of datatypes. Duplicate records were avoided.	Correct implementation allows accurate Migrating of members from the supplied database into your own, with all required fields and correct parsing of datatypes. Duplicate records were avoided.
CSS layout, usability, and input validation in the webform	Implementation was missing or could not be assessed.	Questionable UI that could have been cleaner and more informative through labels and descriptive error messages.	UI was mostly clean, functional, and intuitive through informative labels and error messages. It communicated to the user the state of each operation (e.g. success, failure, more input required).	UI was clean, functional, and intuitive through informative labels and error messages. It communicated to the user the state of each operation (e.g. success, failure, more input required).
Overall Score	**Level 0** **0 or more**	**Level 1** **3 or more**	**Level 2** **6 or more**	**Level 3** **9 or more**

Appendix A – Example of the web UI

Here is a brief example of what the UI can look like. It is an incomplete example however as it lacks some of the functionality that we require, but it is still a good example of a professional and clean look. Take special care when requiring users to enter in data, as they may not know the exact format you wish for the data to be in (e.g. should Date of Birth be entered as dd/mm/yyyy or dd-mm-yyyy? When entering Salary should the user include commas or omit them?).