Приложение написано на C#, с использованием **SQLite** и **ORM** **Entity FrameWork**.

Базовая логика и функции по заданию сначала были предварительно реализованы

в виде консольного приложения.

Для реализации графического интерфейса и дополнительных функций решил первый раз использовать WPF.

Для взаимодействия с БД и формирования сущностей, а на их основе классов, использовался **Entity Framework** и подход **Database First**,

так как **Code First** с **SqLite** работать не получилось.

**Схема программы:**

БД состоит из 2х основных (Employee и User) связанных таблиц для реализации функций по работе с сотрудниками и из учетными записями.

Имеется также вспомогательная таблица (ExceptionNote) для мониторинга и записи исключений в програмном коде.

По заданию, сотрудники могут быть трех групп, соответвенно, логично было их разделить на классы, соответвующие каждой группе. В итоге, имеется три класса сотрудников -

**Employee**, **Manager** и **Salesman**, причем последние два наследуются от базового класса **Employee**. Дополнительных свойств они не имеют (все свойства одинаковы для всех классов и описаны в классе

**Employee**), но для каждого класса уникален метод **FullSalary** - который рассчитывает полную з/п сотрудника по группе на основании данных задания. В зависимости от определенных условий и задач,

классы **Manage**r и **Salesman** можно было бы сделать наследуемыми друг от друга, но для текущей задачи это можно не имплементировать.

Для реализации такого разделения на классы и наследования в БД изначально была создана таблица **Employee**.

[ID] int NOT NULL

[Name] text NOT NULL

[BaseSalary] int NOT NULL

[HireDate] bigint NOT NULL

[ChiefID] int NULL

[Discriminator] text NOT NULL - запись этого поля определяла группу работника

CONSTRAINT [sqlite\_autoindex\_Employee\_1] PRIMARY KEY ([ID])

, FOREIGN KEY ([ChiefID]) REFERENCES [Employee] ([ID]) ON DELETE NO ACTION ON UPDATE NO ACTION

);

На основе ее EF создал сущность и partial класс **Employee** со свойствами. Чтобы была возможность разделить сотрудников по классам, вручную были созданы сущности **Manager** и **Salesman**, для которых

базовой является сущность **Employee**. При этом дополнительных полей в этих сущностях нет. Таким образом имеется три сущности**(Employee,Manager,Salesman)**

для которых EF автоматически сгенерировал partial-классы. В настройках маппирования было указано условие, что в зависимости от класса сотрудника EF меняет

значение поля **Discriminator**. Однако, непосредственно работать с этим полем в программе невозможно, т.к. для корректной работы это свойтво удаляется из сущности **Employee** на этапе проектирования диаграммы сущностей.

Условие задачи, что у работников могут быть подчиненные, условно перефразировалось в то, каждый работник может иметь начальника из числа сотрудников.

Ограничение на то, что начальником не может быть сотрудник класса Employee реализовано логически в коде - при выборе начальника из списка не показываются работники класса Employee.

В БД для имплементации возможности назначить начальника для сотрудника в таблице **Employees** используется внешний ключ (**ChiefID**), который ссылается на эту же самую таблицу **(ID)**

Возможной реализацией данного условия также могло бы быть использование дополнительных свойств в классах **Manager** и **Salesman** для хранения данных о подчиненных.

Таблица User

[ID] int NOT NULL

[Login] text NOT NULL

[Password] text NOT NULL

[Role] int NOT NULL

CONSTRAINT [sqlite\_autoindex\_Users\_1] PRIMARY KEY ([ID])

FOREIGN KEY ([ID]) REFERENCES [Employee] ([ID]) ON DELETE NO ACTION ON UPDATE NO ACTION

Таблица **User** используется для хранения данных учетных записей работников. Имеется внешний ключ **(ID**), который ссылается на таблицу **Employees (ID).** Используется отношение 1-to-0...1, т.е у каждого сотрудника может быть учетная запись.

Таблица **ExceptionNotes** используется для мониторинга исключений, которые могут возникнуть в коде программы. Является служебной и необязательной.

[Id] int NOT NULL

[Date] bigint NOT NULL

[Source] text NOT NULL

[Details] text NOT NULL

CONSTRAINT [sqlite\_autoindex\_ExceptionNotes\_1] PRIMARY KEY ([Id])

Таким образом, в схеме программы используется 3 таблицы и 5 сущностей (**Employee,Manager,Salesman,User,ExceptionNote**) на основе которых были сгенерированы одноименные partial-классы.

Логика программы:

Папка **LogicMethods** - необходимые методы для работы с датой.

Папка **Model** - реализация метода расчета полной з.п для каждого partial-класса сотрудников.

Папка **WindowModels** - основные классы окон в программе

В программе имеется несколько окон.

При старте запускается окно **Login**, где нужно ввести данные учетной записи из таблицы Users. Каждая уз имеет свою роль (поле **Role**, 1- для **Employee**, 2-**Chief**, 3- для Админа).

Строгой привязки к классам сотрудников нет, но условно подразумевается, что 1 - для простых работников, 2- для Manager и **Salesman**. Админом (роль 3) может быть любая уз.

После успешной авторизации, в зависимости от роли уз открывается окно одного из трех классов (**EmployeeWindow,ManagerWindow,AdminWindow**), соответственно для ролей 1,2 и 3.

Изначальным окном, где происходила настройка интерфейса было окно класса MainWindow.

Схема наследования классов: EmployeeWindow: MainWindow, ManagerWindow:EmployeeWindow , AdminWindow:ManagerWindow

**MainWindow** - использовалось для настройки интерфейса. Програмной логики для данного окна немного - описание инумиратора и метода для записи исключений в БД.

**EmployeeWindow: MainWindow** - фактически базовое окно с базовым функционалом работника уровня Employee. В не входит: Просмотр своих данных, просмотр коллег(сотрудников с таким же начальником) и расчез полной

з.п. на фиксированную дату. Имеется возможность сменить пароль свой учетной записи, пароль хешируется в md5-формат и сохраняется в БД.

**ManagerWindow:EmployeeWindow** - имеется функционал EmployeeWindow, плюс добавлена возможность просматривать своих подчиненных, их данные, редактировать их з/п. Имеется возможность получить информацию о полной з/п

всех подчиненных на фиксированную дату, с указанием доп. данных(максимальная,минимальная и средняя з/п)

**AdminWindow:ManagerWindow** - функционал предыдущих окон, плюс возможность смотреть статистику з.п. сразу по всем сотрудникам, возможность полного редактирования данных, создания,удаления работников и их учетных записей.

Для создания сотрудника или его учетной записи, а также редактирования данных используются окна:

**NewEmployee** - базовый класс окна для создания нового сотрудника

**EditWindow : NewEmployee** - для редактирования данных сотрудника. В зависимости от типа у.з. меняются возможности редактирования.

**AccountAddWindow : NewEmployee** - для добавления новой у.з сотрудника

**AccountEditWindow : AccountAddWindow** - для редактирования данных у.з

**ChangePasswordWindow : AccountAddWindow** - для изменения пароля у.з.

**Плюсы данного приложения:**

- Возможность добавления разных наследуемых друг от друга классов сотрудников в одну таблицу(упрощение схемы)

- Расширенный функционал просмотра и редактирования данных сотрудников и у.з

- Использование (хотя, и не в полную меру) преимуществ наследования классов для описания логики программы (работа с разными классами окон).

- Графический интерфейс

**Минусы**

-Из-за того, что поле Descriminator недоступно из кода программы, для определения класса сотрудника приходилось использовать избыточный код(Сотрудник класса менеджер может определяться

как сотрудник базового класса Employee, приходится делать проверку сначала на принадлежность к наследуемому классу). Особенно это заметно при расчете полной з.п. сотрудника.

-Избыточный код в логике наследуемых классов

-Возможные неучтенные ошибки

-Недоработки интерфейса - новые окна открываются где попало. Использование MessageBox для информирования.

**Спорные моменты.**

-Использование одной таблицы для сотрудников всех классов. Можно было разделить - каждому классу своя таблица. Тогда бы, возможно, исчезла бы проблема избыточного кода при определении класса сотрудника.

-Нет строгой привязки роли у.з к классу сотрудника

-При создании сотрудника не требуется создавать его у.з. У.з создается админом в разделе Manage Accounts

-Класс MainWindow избыточен, базовым можно было сделать EmployeeWindow

-При определении полной зарплаты всех сотрудников или подчиненных на конкретную дату учитывается дата принятия сотрудника на работу. Если расчетная дата раньше даты приятия на работу, то данный сотрудник не учитывается

в расчете полной зарплаты, но из списка сотрудников в листе не удаляется.

У.З. АДМИНА

Commando

123456