Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
Факультет комп'ютерно-інформаційних систем і програмної інженерії
Кафедра програмної інженерії

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи

магістра

на тему: «Інформаційна система обліку ефективності робочого часу працівників підприємства на базі .NET»

Виконав: студент (ка) VI курсу, групи СПм-61

спеціальності (напряму підготовки) 121

Інженерія програмного забезпечення

(шифр і назва спеціальності (напряму підготовки))

(підпис) (прізвище та ініціали)

Керівник (підпис) (прізвище та ініціали)

Нормоконтроль (підпис) (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

м. Тернопіль – 2021

АНОТАЦІЯ

Атестаційна робота магістра на тему «Інформаційна система обліку ефективності робочого часу працівників підприємства на базі .NET”». Мальцева Віталя Анатолійовича. Тернопільський національний технічний університет імені Івана Пулюя, Факультет комп'ютерно-інформаційних систем і програмної інженерії, Кафедра програмної інженерії, група СПм–61, Тернопіль, 2021. С. – __, рис. – __, табл. – __, слайдів. – __, додат. – 4, бібліогр. – 6.

Мета проекту розробка інформаційної системи обліку робочого часу співробітників. Дана робота включає розробку програмного забезпечення на основі використання баз даних. Для розробки інформаційної системи, було використано СУБД MySql з яким здійснювалась взаємодія через технологію ADO.NET на мові програмування C#, зовнішній інтерфейс програми був розроблений з підсистемою WPF (Windows Presentation Foundation). З допомогою програми можна спостерігати за відвідуваністю співробітників завдяки обліку робочого часу. Можна переглядати поточну кількість співробітників, Також інформаційна система забезпечує можливість створення декількох відділень на роботі на які можна назначити менеджерів які будуть виконувати основні дії по видаленню да добавленню співробітників. Система дозволить зменшити витрати підприємства за рахунок підвищення мотивації співробітників та дослідженню часових витрат на виконання тої чи іншої задачі.

Ключові слова: ОБЛІК РОБОЧОГО ЧАСУ, ПІДПРИЄМСТВА, КОНТРОЛЬ ВИТРАТ, ADO.NET, ЕФЕКТИВНІСТЬ ВЕДЕННЯ БІЗНЕСУ, ПІДПРИЄМСТВА, WINDOWS PRESENTATION FOUNDATION, C#, MYSQL, СУБД.

ABSTRACT

Master's certification work on the topic "Information system for accounting for the efficiency of working time of employees of the enterprise on the basis of .NET". Maltsev Vitalii. Ternopil National Technical University named after Ivan Puluj, Faculty of Computer Information Systems and Program Engineering, Department of Software Engineering, SPm-61 Group, Ternopil, 2021. C. - __, Fig. - __, tab. - __, slides. - __, additional - 4, bibliographer. – 6.

The purpose of the project is the development of an information system for the registration of employees' working time. This work involves the development of software based on the use of databases. To develop the information system, the MySQL database interface was used, which was implemented through ADO.NET technology in the C # programming language, the external interface of the program was developed with the Windows Presentation Foundation (WPF) subsystem. With the help of the program it is possible to observe the attendance of employees through the account of working time. You can view the current number of employees, Also, the information system provides the ability to create several offices at work, which can appoint managers who will perform the basic steps to remove and add staff. The system will reduce the company's expenses by increasing the motivation of employees and studying the time expenditures for performing one or another task.

Keywords: WORKING TIME, BUSINESS, BUSINESS MANAGEMENT, ADO.NET, BUSINESS MANAGEMENT, BUSINESS MANAGEMENT, WINDOWS PRESENTATION FOUNDATION, C #, MYSQL, DBMS.

ЗМІСТ

ВСТУП

В наші часи комп’ютерна електроніка зайняла високе місце в житі людей так як комп’ютери і любий електронний пристрій можна зустріти майже в кожному куточку світу, навіть там де людині небезпечно перебувати, в космосі, під водою, в місцях з підвищеною радіацією і в інших тому подібних місцях.

В теперішньому суспільстві інформація цініться найбільше, тому її збереження, та цільове використання є заставою для успішного ведення бізнесу, особливо для великих підприємств які через невеликі переривання в роботі чи через часткову втрату ефективності втрачатимуть прибутки які могли піти на підвищення ефективності праці шляхом заміни обладнання на більш ефективніше або стимулюванням працівників до більш високої продуктивності і відповідальності. В сфері ведення бізнесу існує проблема ведення обліку працівників та моніторингу їх робочого часу, так як більшість працівників під час роботи проводять час в соціальних мережах, перегляду фільмів, прослуховуванні музики і тим самим зменшують свою ефективність на роботі.

Для вирішення цієї проблеми було реалізовано безліч рішень для обліку робочого часу. Які здійснювали автоматичне стеження за активністю. З їх допомогою відстежується стан програм на комп'ютері та натискання клавіш при наборі тексту . Також мають місце рішення, які роблять скріншоти робочого столу і навіть здійснюють безперервний відеозапис з камер, моніторів та дисплеїв.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ МОНІТОРИНГУ РОБОЧОГО ЧАСУ

1.1 Огляд конкурентів

В сфері бізнесу мають місце проблеми що заважають збільшити прибутки підприємства і як не дивно більшість цих проблем звязані з самодисципліною. Як ми знаємо не всі люди мають достатньо твердий характер щоб виконувати свою роботу без відволікання уваги від основної задачі, аце в свою чергу знижує продуктивність працівника, в основному це вирішується через само дисцепліну, але не завжди це працює, особливо на великих підприємствах. Тому для вирішення цієї проблеми створюють доволі багато програмних комплекмів, які частково вирішують дану задачу. Для вирішення проблем було реалізовано багато програм обліку робочого часу. Серед них популярні системи автоматичного стеження. З їх допомогою здійснюється спостереження за програмам, перехоплюється обмін повідомлення. Використовуються рішення, що роблять скріншоти робочого столу і навіть проводять безперервне спостереження за працівниками через камери відеоспостереження зображень на дисплеї.

Подібні системи мають серйозні недоліки. Вони порушоють закони про приватне життя, що загрожує як самому працівнику так і роботодавцям. Роботодавці, які бажають використовувати програму відстеження дій своїх співробітників, повинні отримати письмову згоду кожного працівника, що не завжди можливо. Звичайно роботодавці часто порушують це і відповідно страждають від цього. Тому для вирішення таких складнощів існують інші менш ефективні системи.

Найкраще використовувати індивідуальні та колективні добровільні системи відліку часу (тайм-трекери). Вони повністю законні, і вони заохочують до самодисципліни та організованості.Для керівників і співробітників будь-якої

компанії час є найважливішим ресурсом. Але без належної самоорганізації та дисципліни люди, як правило, втрачають це.

Аналізуючи предметну область було виділено декілька програмних систем які ведуть облік робочого часу:

- Система обліку робочого часу Бітрікс24. У сервісі Бітрікс24 система відвідуваності інтегрована з системами органайзера, документообігу, телефонного зв'язку та управління взаємовідносинами з клієнтами (CRM). Зібрані статистичні дані автоматично передаються до керівника та бухгалтерії у вигляді звітів. Заробітна плата розраховується на основі даних бухгалтерського обліку використання робочого часу.
- MoneyPenny облік годин роботи у реальному часі. Програма для звітування про роботу команди. Програма пропонує інструменти для точного відстеження годин роботи у реальному часі, щоб постійно були в курсі, хто і над яким завданням чи проектом працює.
- Time Doctor онлайн програма обліку робочого часу. Відстежує роботу персоналу та надає розбивку про те, скільки часу витрачається на певні проекти (або клієнтів). Надається можливість знімки робочого столу (скріншотів) для моніторингу роботи співробітників.

1.2 Обґрунтування вибору напрямку дослідження

В якості інструментів розробки було вибрано:

- Базу даних MySQL 8.;
- Середовище розробки Microsoft Visual Studio 2022 Community;
- Мова програмування C#;
- Технологія WPF (Windows Presentation Foundation);
- Технологія ADO.NET;

База даних MySQL 8.

MySQL є однією з найбільш впізнаваних технологій в сучасній екосистемі великих даних. Його часто називають найпопулярнішою базою даних і наразі

користуються широким, ефективним використанням незалежно від галузі, зрозуміло, що будь-хто, хто займається корпоративними даними або загальними IT, повинен принаймні прагнути до базового знайомства з MySQL.

За допомогою MySQL навіть ті, хто вперше в реляційних системах, можуть відразу створити швидкі, потужні та безпечні системи зберігання даних. Програмний синтаксис та інтерфейси MySQL також є ідеальними шлюзами у широкий світ інших популярних мов запитів і сховищ структурованих даних.

Оскільки MySQL користується найбільшим поширенням у багатьох галузях, бізнес-користувачі від нових веб-майстрів до досвідчених менеджерів повинні прагнути зрозуміти його основні характеристики. Прийняття рішення про використання цієї технології та ефективне спілкування з нею починається з огляду базової доступності, структури, філософії та зручності MySQL.

MySQL широко сумісний. Незважаючи на те, що MySQL часто асоціюється з інтернет-додатками або веб-сервісами, MySQL був розроблений так, щоб бути сумісним з іншими технологіями та архітектурами. СУБД працює на всіх основних обчислювальних платформах, включаючи операційні системи на базі Unix, такі як безліч дистрибутивів Linux або Mac OS і Windows.

Архітектура клієнт-сервер MySQL означає, що вона може підтримувати різноманітні серверні програми, а також різні інтерфейси програмування. Завдяки архітектурній та мовній подібності дані можна безпосередньо переносити з MySQL до його форків (наприклад, MariaDB), а також до більшості інших СУБД.

Створений Oracle і Крім того, сторонні інструменти міграції дозволяють MySQL переміщувати дані до та з широкого набору загальних систем зберігання, незалежно від того, чи розроблені вони як локальні, так і хмарні. MySQL може бути розгорнуто у віртуалізованих середовищах, розподілених або централізованих, і навіть існує як портативні автономні бібліотеки для цілей навчання, тестування або невеликих додатків.

Широка сумісність MySQL з усіма цими іншими системами та програмним забезпеченням робить його особливо практичним вибором СУБД у більшості ситуацій.

Бази даних MySQL є реляційними. Основний фактор, що відрізняє реляційні бази даних від інших цифрових сховищ, полягає в тому, як дані організовані на високому рівні. Бази даних, такі як MySQL, містять записи в кількох, окремих і високо кодифікованих таблицях, на відміну від єдиного всеохоплюючого сховища або колекцій напів- чи неструктурованих документів.

Це дозволяє СУБД краще оптимізувати дії, такі як отримання даних, оновлення інформації або більш складні дії, такі як агрегації. Логічно модель визначається для всього вмісту бази даних, описуючи, наприклад, значення, дозволені в окремих стовпцях, характеристики таблиць і представлень, або як пов’язані індекси з двох таблиць.

Реляційні моделі залишаються популярними з кількох причин. Вони надають користувачам інтуїтивно зрозумілі, декларативні мови програмування — по суті, повідомляють базі даних, який результат бажаний на мові, подібній або принаймні зрозумілій, як письмова англійська, замість того, щоб ретельно кодувати кожен крок процедури, що веде до цього результату. Це переміщує велику частину роботи в СУБД і SQL, покращуючи дотримання логічних правил і економлячи цінні ресурси та робочу силу.

MySQL є відкритим вихідним кодом. Будь-яка особа або підприємство може вільно використовувати, змінювати, публікувати та розширювати базу коду MySQL Oracle з відкритим вихідним кодом. Програмне забезпечення випускається під Загальною публічною ліцензією GNU (GPL).

Для коду MySQL, який потрібно інтегрувати або включити в комерційну програму (або якщо програмне забезпечення з відкритим вихідним кодом не є пріоритетом), підприємства можуть придбати комерційно ліцензовану версію в Oracle.

Знову ж таки, ці параметри надають організаціям додаткову гнучкість, якщо вони вирішують працювати з MySQL. Загальнодоступний характер випусків із відкритим кодом збагачує документацію MySQL та культуру онлайн-підтримки, а також гарантує, що постійні або нещодавно розроблені можливості ніколи не відходять занадто далеко від поточних потреб користувачів.

MySQL простий у використанні. Хоча реляційна природа MySQL і пов’язані з цим жорсткі структури зберігання можуть здатися обмежуючими, таблична парадигма є, мабуть, найбільш інтуїтивно зрозумілою і в кінцевому підсумку забезпечує більшу зручність використання.

Фактично, MySQL робить багато поступок у підтримці якнайширшої різноманітності структур даних, від стандартних, але багатих логічних, числових, буквено-цифрових типів, типів дати та часу до більш просунутих JSON або геопросторових даних. Крім простих типів даних і великого вбудованого набору функцій, екосистема MySQL також включає в себе різноманітні інструменти, що полегшують все від керування сервером до звітності та аналізу даних.

Незалежно від загальної архітектури СУБД, користувачі завжди можуть знайти функцію MySQL, яка дозволяє їм моделювати та кодувати дані, як вони хочуть. MySQL залишається однією з найбільш простих технологій баз даних для вивчення та використання.

Microsoft Visual Studio 2022 Community

Microsoft Visual Studio — це серія продуктів Microsoft, що включає інтегроване середовище розробки програмного забезпечення та багато інших інструментів. Ці продукти дозволяють розробляти консольні та графічні додатки, включаючи підтримку технології Windows Forms, а також веб-сайти та веб-додатки з власним і розміщеним кодом для всіх платформ Microsoft Windows, Windows Mobile, Windows Phone та Windows CE, веб-служби, . NET Framework, .NET Compact Framework і Microsoft Silverlight.

Visual Studio включає один або кілька з наступних компонентів:

- Visual Basic .NET, раніше Visual Basic
- Visual C++
- Visual C#
- Visual F# (частина Visual Studio 2010);
- Налагоджувач Visual Studio

Мова програмування C#

C# (вимовляється як «сі-шарп») — це проста, сучасна, об’єктно-орієнтована та безпечна для типів мова програмування. Мова C# свої коріння в C подібних мовах, таких як C, C++, і в основному схожа на Java.

Мова програмування C# дозволить розробникам створювати різноманітні безпечні та надійні програми, такі як програми Windows, веб-додатки, програми баз даних тощо, які працюватимуть на .NET Framework.

Мова програмування C# створена на фреймворку .NET Framework для запуску програм C#. .NET Framework — це платформа розробки для створення програм для Windows та unix подібних систем, Тощо за допомогою мов програмування, таких як C#, F# та Visual Basic. Він складається з двох основних компонентів, таких як Common Language Runtime (CLR), механізм виконання, який обробляє запущені програми, і .NET Framework Class Library, яка надає бібліотеку перевіреного та багаторазового коду, який розробники можуть використовувати у своїх програмах.

Технологія WPF (Windows Presentation Foundation)

Технологія WPF (Windows Presentation Foundation) є частиною екосистеми платформи .NET і підсистемою для побудови графічних інтерфейсів. Якщо під час створення традиційної програми WinForms частини Windows, такі як User32 і GDI+, відповідають за відображення елементів керування та графіки, то програма WPF заснована на DirectX. Це ключова особливість візуалізації графіки в WPF: з WPF більшість роботи з відображення графіки, будь то найпростіші кнопки або складні 3D-моделі, ляже на графічний процесор, який також дозволяє використовувати апаратне прискорення графіки. Важливою особливістю є використання декларативної мови розмітки XAML на основі XML: ви можете використовувати декларативні оголошення інтерфейсу або керований код C# і VB.NET або їх комбінацію для створення насичених графічних інтерфейсів.

Переваги WPF

- Використовує традиційну мову платформи .NET - C# і VB.NET для створення логіки програми

- Можливість використовувати спеціальну мову розмітки XAML на основі xml для декларативного визначення графічних інтерфейсів, що є альтернативою програмному створенню графіки та елементів керування, а також можливість комбінувати XAML і C#/VB.NET
- Незалежність від розширення екрана: оскільки всі елементи в WPF вимірюються в одиницях, незалежних від пристрою, програми на WPF можна легко масштабувати до різних екранів з різною роздільною здатністю.
- Нові функції, які важко реалізувати в WinForms, такі як створення 3D-моделей, прив'язка даних, використання стилів, шаблонів, тем та інших елементів.
- Гарна сумісність із WinForms, тому ви можете використовувати традиційні елементи керування WinForms у програмах WPF.Багаті можливості по створенню різних додатків: це і мультимедіа, і двомірна і тривимірна графіка, і багатий набір вбудованих елементів управління, а також можливість самим створювати нові елементи, створення анімацій, прив'язка даних, стилі, шаблони, теми і багато іншого
- -Апаратне прискорення графіки - незалежно від того, чи це обробка 2D або 3D, графіки або тексту, всі компоненти програми перетворюються в зрозумілі об'єкти Direct3D, а потім візуалізуються графічним процесором, тим самим покращуючи продуктивність і роблячи графіку більш гладкою.
- -Створюйте програми для багатьох операційних систем Windows - від Windows XP до Windows 10.
- У той же час WPF також має певні обмеження. Незважаючи на те, що підтримується 3D-візуалізація, для створення програм із великою кількістю 3D-зображень, особливо ігор, краще використовувати інші інструменти – спеціальні фреймворки, такі як DirectX або Monogame або Unity.
- Також варто відзначити, що кількість програм і споживання пам’яті під час виконання на WPF в середньому дещо вищі в порівнянні з додатками на Windows Forms. Але це компенсується ширшими графічними можливостями та підвищенням продуктивності під час відображення графіки.

Технологія ADO.NET

Сьогодні обробка даних дуже важлива. Для зберігання даних використовуються різні системи керування базами даних: MS SQL Server, Oracle, MySQL тощо. Більшість програм використовують ці системи керування базами даних для зберігання даних тим чи іншим способом. Однак для зв'язку між базою даних і програмою на C# потрібен посередник. І таким посередником є технологія ADO.NET.

ADO.NET надає технологію даних на основі .NET Framework. Ця технологія надає нам набір класів, які дозволяють робити запити до бази даних, встановлювати з'єднання, отримувати відповіді від бази даних і виконувати багато інших операцій.

Слід зазначити, що систем управління базами даних може бути багато. Насправді вони можуть бути різними. MS SQL Server використовує T-SQL для створення запитів, а MySQL і Oracle використовують PL-SQL. Різні системи баз даних можуть мати різні типи даних. Можуть бути й інші моменти. Проте функції ADO.NET призначені для надання розробникам уніфікованого інтерфейсу для роботи з різними базами даних.

Основою інтерфейсу для взаємодії з базою даних в ADO.NET є обмежений діапазон об'єктів: Connection, Command, DataReader, DataSet і DataAdapter. Об'єкт Connection встановлює з'єднання з джерелом даних. Об'єкт Command дозволяє виконувати операції з використанням даних у базі даних. Об'єкт DataReader зчитує дані, отримані в результаті запиту. Об'єкт DataSet призначений для зберігання даних із бази даних і дозволяє використовувати їх незалежно від бази даних. Об'єкт DataAdapter є посередником між DataSet і джерелом даних. В основному він працює з базою даних через ці об'єкти.

Однак, щоб використовувати той самий набір об'єктів для різних джерел даних, потрібен відповідний постачальник даних. Фактично, він взаємодіє з базою даних через постачальника даних в ADO.NET. Більше того, кожне джерело даних в ADO.NET може бути своїм власним провайдером, який фактично визначає конкретну реалізацію вищезгаданих класів.

РОЗДІЛ 2. РОЗРОБКА МОДЕЛІ ТА ПРОГРАМНОГО КОМПЛЕКСУ

2.1 Проєктування інформаційної системи для моніторингу робочого часу працівників

2.1.1. Розробка моделі предметної області

Інформаційна система також передбачає апаратну частину яка в майбутньому буде розроблена на мікроконтролері Arduino та використовуючи додаткові датчики і сканери ідентифікаційного номера, буде збирати інформацію та відправляти її в базу даних.

Для автоматизації обліку працівників та їх робочого часу було розроблено модель інформаційної системи яка передбачає наступні варіанти використання:

- можливість авторизуватися у системі;
- створення облікових даних про працівника;
- перегляд усіх облікових даних;
- редагування персональних даних зі згоди працівників;
- отримання інформації про дії працівників;
- отримання інформації про розташування працівників;
- створення записів про дії працівників;
- резервне дублювання інформації про дії працівників;
- можливість взаємодіїї отримання статистичної інформації ;
- можливість перегляду камер відеоспостереження;

З даного списку видно частковий функціонал системи яка дає можливість великим підприємствам проводити облік робочого часу працівників різних відділень, що в свою чергу дозволяє збільшини доходи самого ж підприємства.

На рисунку 2.1 показанва діаграма варіантів використання інформаційної системи.

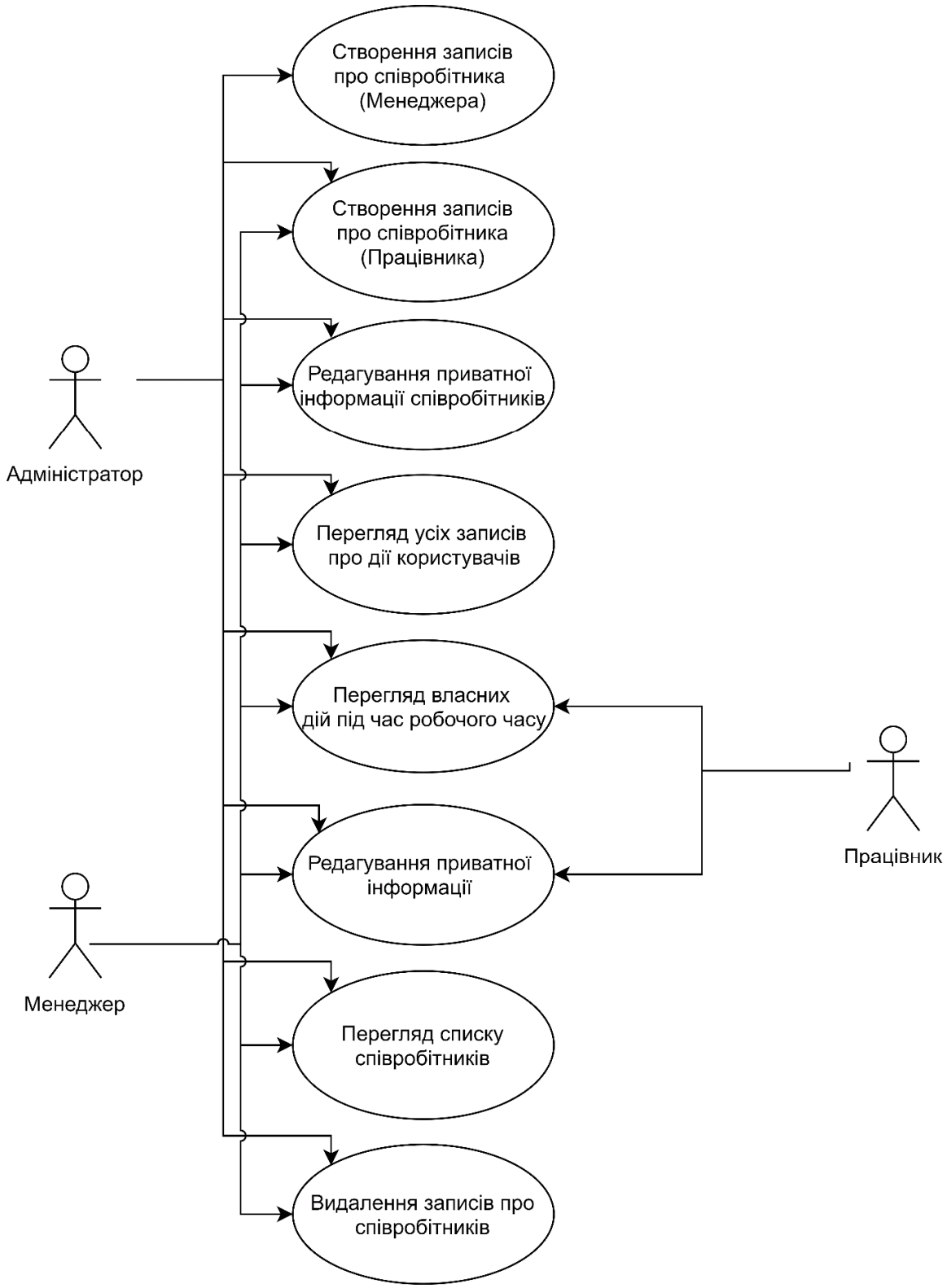

Рисунок 2.1 – Діаграма варіантів використання інформаційної системи

В інформаційній системі присутні три ролі один Адміністратор, менеджери та прості працівники які наділені певними правами та мають обмеження в доступі до інформаційної системи, розмежування доступу проводиться з врахуванням їх ролей

які будуть створені під час реєстрації конкретних працівників підприємства їх підприємством.

2.1.2. Розробка бізнес моделі

В пошуку сутностей предметної області було переглянута роль співробітника і виділено декілька сутностей, а саме:

1. Співробітники – містить ідентифікаційний номер співробітника та його ім’я, прізвищем, вік;
2. Контактна інформація – містить інформацію про електрону пошту та телефонний номер;
3. Інформація для авторизації – містить інформацію яка ідентифікує користувача в системі;
4. Місце проживання – містить інформацію про місце проживання співробітників;
5. Відділення на роботі – містить інформацію про посаду співробітника та номер відділення;
6. Дії співробітників – містять інформацію про дії які були виконанні співробітниками та час коли відбулась дія;
7. Список змін користувачів – містить інформацію про дію, час зміни інформації про користувача;;
8. Список старої інформації – містить стару інформацію про користувача;
9. Список нової інформації – містить стару інформацію про користувача;

В таблицях 2.1 – 2.9 наведено властивості сутності їх типів даних та полів.

Таблиця 2.1 – Employee

Назва	Тип даних	Тип поля
1	2	3
Id	Числовий	Внутрішній ключ
FirstName	Текстовий	
LastName	Текстовий	
Age	Числовий	

Таблиця 2.2 – ContactInformation

Назва	Тип даних	Тип поля
1	2	3
EmployeeId	Числовий	Зовнішній ключ
PhoneNumber	Текстовий	
Email	Текстовий	

Таблиця 2.3 – Admins

Назва	Тип даних	Тип поля
1	2	3
EmployeeId	Числовий	Зовнішній ключ
Login	Текстовий	Унікальний
Password	Текстовий	
Position	Текстовий	

Таблиця 2.4 – HomeAddress

Назва	Тип даних	Тип поля
1	2	3
EmployeeId	Числовий	Зовнішній ключ
City	Текстовий	
Street	Текстовий	
NumberHome	Числовий	

Таблиця 2.5 – WorkDepartment

Назва	Тип даних	Тип поля
1	2	3
EmployeeId	Числовий	Зовнішній ключ
DepartmentId	Числовий	
Eposition	Текстовий	

Таблиця 2.6 – DataRegister

Назва	Тип даних	Тип поля
1	2	2
Id	Числовий	Внутрішній ключ
EmployeeId	Числовий	Зовнішній ключ
Action	Текстовий	
ObjectAction	Текстовий	
DataTime	Дата/час	

Таблиця 2.7 – ReportsTableEmployee

Назва	Тип даних	Тип поля
1	2	3
Id	Числовий	Внутрішній ключ
ReportsTableEmployee	Числовий	Зовнішній ключ
ReportsTableEmployeection	Числовий	Зовнішній ключ
Action	Текстовий	
DataTime	Дата/час	

Таблиця 2.8 – ReportsTableEmployeeOld

Назва	Тип даних	Тип поля
1	2	3
Id	Числовий	Внутрішній ключ
EmployeeIdOld	Числовий	
FirstNameOld	Текстовий	
LastNameOld	Текстовий	
AgeOld	Числовий	

Таблиця 2.9 – ReportsTableEmployeeNew

Назва	Тип даних	Тип поля
1	2	3
Id	Числовий	Внутрішній ключ
EmployeeIdNew	Числовий	
FirstNameNew	Текстовий	
LastNameNew	Текстовий	
AgeNew	Числовий	

Схема баз даних — Це структура бази даних, яка є описується формальною мовою, що підтримується СКБД. Вона відноситься до організації даних по створенню плану побудови бази даних, розподіленої по таблицях. Формально кажучи, схема бази даних — це набір формул, які називаються обмеженнями цілісності. Обмеження цілісності забезпечують сумісність між усіма частинами схеми. В таблиці 2.10 наведено зв’язки між сутностями.

База даних складається з 9 таблиць, 3 тригерів та одного представлення.

Таблиця 2.10 – Зв’язки між сутностями

Тип сутності	Тип зв’язку	Тип сутності	Кардинальність
1	2	3	4
Інформація для авторизації	Має	Співробітники	1:1
Контактна інформація	Має	Співробітники	1:1
Місце проживання	Має	Співробітники	1:1
Відділення на роботі	Має	Співробітники	1:1
Дії співробітників	Має	Співробітники	1:Б
Список змін користувачів	Має	Список старої інформації	1:1
Список змін користувачів	Має	Список нової інформації	1:1

Таблиці:

- Головна таблиця “Employee” яка містить ідентифікаційний номер та ім’я з прізвищем також дана таблиця містить три тригера які реалізують різний функціонал, а саме тригер запису змін які відбулись в цій таблиці в іншу, тригер контролю цілісності який в свою чергу видаляє залежну інформацію з інших таблиць, тригер запису інформації про видалення користувача з бази даних;
- Дочірня таблиця “PhoneAndEmail” яка містить поля електронний адрес, номер телефона співробітника та “EmployeeId” яка по зовнішньому ключу зв’язана з полем Id в таблиці “Employee”;
- Дочірня таблиця “HomeAddress” яка містить поля місто проживання, вулиця проживання, номер будинку та “EmployeeId” яка по зовнішньому ключу зв’язана з полем Id в таблиці “Employee”;
- Дочірня таблиця “WorkDepartment” яка містить поля номер відділення, посаду та “EmployeeId” яка по зовнішньому ключу зв’язана з полем Id в таблиці “Employee”;
- Дочірня таблиця “Admins” яка містить поля логін, пароль, права користувача та “EmployeeId” яка по зовнішньому ключу зв’язана з полем Id в таблиці “Employee”;

- Дочірня таблиця “DateRegister” яка містить поля дії яка виконалась, назва об’єкта над яким виконалась дія, час виконання дії, ідентифікатор кожної дії та “EmployeeId” яка по зовнішньому ключу зв’язана з полем Id в таблиці “Employee”;
- Таблиця “ReportsTableEmployee” є головною таблицею для двох таблиць “ReportsTableEmployeeNew”, “ReportsTableEmployeeOld” і містить 5 полів два з яких це зовнішні ключі на таблиці зі старою та новою інформацією, дія яка виконалась та час виконання дії;
- Таблиці “ReportsTableEmployeeNew”, “ReportsTableEmployeeOld містять 5 полів ім’я, прізвище, вік, ідентифікаційний номер та ідентифікатор запису;

Представлення:

- Представлення “viewemployees” об’єднує в собі більшість таблиць для представлення інформації про співробітників в програмі;

Нормалізація бази даних

Нормалізація — це процес розділення даних на окремі зв'язані таблиці. Нормалізація усуває надмірність даних і, таким чином, дозволяє уникнути порушення цілісності даних при їх зміні, тобто уникаючи аномалій змін (update anomaly).

Як правило, нормалізація в основному використовується в висхідному підходу до розробки бази даних, тобто, коли у нас є всі атрибути, які будуть зберігатися в базі даних, Група сутностей, для яких потім створюються таблиці. Однак у низхідному підходу, коли сутності вперше ідентифіковано, а потім їх атрибути та зв'язки між ними, нормалізацію можна також використовувати, наприклад, для перевірки правильності розроблених таблиць.

У формі, яка не нормалізована, у таблиці можуть зберігатися відомості про дві або більше сутностей. Вони також можуть містити повторювані стовпці. Стовпці можуть також зберігати повторювані значення. У нормалізованої формі кожна таблиця зберігає інформацію лише про одну сутність.

Нормалізація передбачає застосування нормальних форм до структури даних. Є 7 нормальних форм. Кожна нормальна форма (крім першої) означає, що дані вже приведені до попередньої нормальної форми. Наприклад, перш ніж застосовувати третю звичайну форму до даних, необхідно застосувати другу звичайну форму. І строго кажучи, база даних вважається нормалізована, якщо вона застосовує третю нормальну форму і вище. Розробивши базу даних було побудовано схему яка зображена на рисунку 2.2

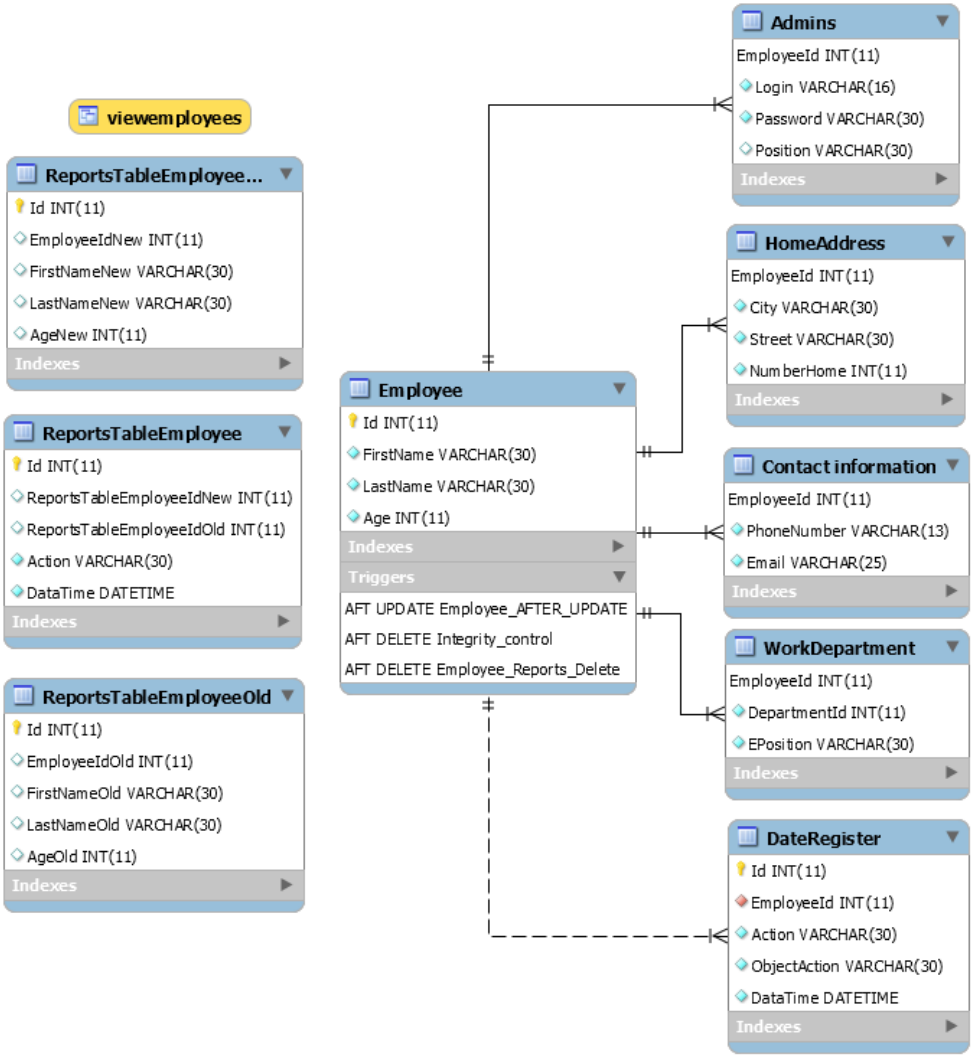

Рисунок 2.2 - Схема бази даних

Також на схемі зображено представлення яке об’єднає в собі майже всі таблиці для зручного відображення в програмі та зменшення навантаження на саму програму.

Ключовою концепцією нормалізації є функціональна залежність. Функціональна залежність описує зв'язок між зв'язками атрибутів.

Перша нормальна форма (1NF) припускає, що зберігаємі дані на перетині рядків і стовпців мають містити скалярне значення, а таблиці не повинні містити повторювані рядки.

Для досягнення першої нормальної форми в кожній таблиці було додано первинний ключ та забезпечено не повторність рядків, атомарність.

Друга нормальна форма (2NF) припускає, що кожен стовпець, який не є ключем, має залежати від первинного ключа.

Для досягнення другої нормальної форми дані що зберігаються в таблицях із композитним ключем, повністю залежать від нього.

Третя нормальна форма (3NF) припускає, що кожен стовпець, який не є ключем, повинен залежати лише від первинного ключа.

Для досягнення третьої нормальної форми була забезпечена залежність даних лише від первинного ключа.

2.1.3. Проектування архітектури

При розробці інформаційної системи в якості архітектури, була вибрана клієнт серверна архітектура, мікроконтролер Arduino для подальшого розвитку інформаційної системи. В додатку “В” наведено UML Діаграму класів на якій зображено взаємодію та асоціацію між класами.

Архітектура клієнт-сервер - це обчислювальна модель, в якій сервер розміщує, забезпечує і керує більшістю ресурсів і послуг, які споживає клієнт. Цей тип архітектури має один або кілька клієнтських комп'ютерів які мають доступ до інтернету. Архітектура клієнт-сервер також називається мережевими обчисленнями, клієнт-серверна модель, тому що всі запити проходять через мережу Інтернет.

- Клієнт: частина програмного забезпечення або програма, яка приймає вхідні дані та надсилає запит на сервери.

- Сервер: частина програмного забезпечення, яка отримує та обробляє запити від клієнтів.
- Балансувальник навантаження: відповідає за розподіл вхідного мережевого трафіку між групою бекенд-сервери для оптимізації використання ресурсів
- Протоколи мережевого рівня, такі як TCP/IP

Потік даних є односпрямованим і утворює цикл. Зазвичай він ініціюється запитом клієнта, для отримання даних і сервер обробляє запит і надсилає результат назад клієнту через протокол. Клієнти не можуть безпосередньо спілкуватися один з одним.

Типовий топологічний потік даних виглядає наступним чином:

1. Клієнт запитує дані від сервера
2. Балансувальник навантаження направляє запит на відповідний сервер
3. Сервер обробляє запит клієнта
4. Сервер запитує відповідну базу даних для отримання даних
5. База даних повертає запитані дані назад на сервер
6. Сервер обробляє дані та надсилає дані назад клієнту
7. Цей процес повторюється

Архітектура клієнт-сервер найбільш корисна для програм, які потребують розділення або абстрагування проблем між клієнтом і сервером; він призначений для систем з високою сумісність. Архітектурний стиль клієнт-сервер допомагає програмам підвищити продуктивність в масштабованості.

У системах, які потребують розділення функціональних можливостей, дизайн архітектури клієнт-сервер найбільше застосовний. Перевірка запиту та введення можуть оброблятися з боку клієнта під час завантаження балансувальник направляє запит на сервер для адекватної обробки. Сервер буде відповідає за обробку запиту клієнта та повернення результату за потрібним протоколом. Ці рівні (клієнт і сервер) виконують завдання незалежно, і вони корисні для функціональність абстрагування; наприклад, клієнту не потрібно знати, як працює сервер обробляє автентифікацію користувача або перевірку запиту.

З поділом функціональних можливостей кожен шар функціонує більше ефективно в великому масштабі. В рамках клієнт-сервера розроблені сучасні методики архітектури для вирішення проблем масштабованості, таких як балансування навантаження, сегментування та розділення. Ці методи забезпечують підвищення продуктивності для кількох запитів на стороні сервера і буде корисним для програм, які працюють із кількома користувачами одночасно.

2.2 Конструювання інформаційної системи для моніторингу робочого часу працівників

2.2.1. Реалізація ключових класів

В інформаційні системі взаємодія з базою даних здійснюється через технологію ADO.NET на Desktop application.

ADO.NET надає технологію роботи з даними, яка заснована на платформі .NET Framework. Ця технологія представляє набір класів, через які можливо відправляти запити до баз даних, встановлювати підключення, отримувати відповідь від бази даних і виробляти ряд інших операцій.

При роботі з базою даних використовувався “MySql.Data.MySqlClient” провайдер який надає бібліотеку класів для роботи з MySql базою даних та можливість працювати з нею.

На лістингу 2.1 приведено приклад взаємодії з базою даних.

Лістинг 2.1 – приклад взаємодії з базою даних

```
/*Для підключення до бази даних використовується клас від якого створюється об’єкт та через його конструктор задаються параметри для підключення до бази даних.*/
static MySqlConnection connection = new MySqlConnection("Data Source = 5.101.152.9;  user = aragorn_employee; database = aragorn_employee; password = ShEdOw85264; sslmode = none;");
//Для надсилання SQL запитів використовується метод “MySqlCommand”.
MySqlCommand command = new MySqlCommand("SELECT * FROM viewemployees", connection);
/*Для використання процедур необхідно поміняти тип команди та замість самої команди вказати назву процедури.*/
```

Продовження лістингу 2.1

```
MySqlCommand command = new MySqlCommand("CreateRecordEmployee", connection);
```

```
command.CommandType = System.Data.CommandType.StoredProcedure;
/*Для добавлення параметрів  в SQL запити використовується метод “MySqlParameter”(параметри підставляють значення в SQL запити).*/
MySqlParameter idParam = new MySqlParameter("@Id", form.eyp.crpNA.idRecord.Text);
command.Parameters.Add(idParam);
//Для відкриття та закриття з’єднання використовуються 2 метода Open() та Close().
//Для виконання SQL запиту використовується ExecuteNonQuery().
```

Для роботи з базою даних використовується створений клас “WorkDatabaseMySql” через який проводиться взаємодія з базою даних.

Метод LoginConection приймає об’єкт типа Login і виконує авторизацію користувача в програмі після чого вертає кортеж з іменем, прізвищем, ідентифікаційний номер та права користувача в програмі.

Метод LoadEmployeeDataGrid отримує дані з представлення та завантажує їх в DataTable який потім вертається з методу. На лістингу 2.2 наведено код методу LoadEmployeeDataGrid.

Лістинг 3.2 – код методу LoadEmployeeDataGrid

```
public static DataTable LoadEmployeeDataGrid()
    {
      MySqlCommand command = new MySqlCommand("SELECT * FROM viewemployees", connection);
      DataTable dt = new DataTable();//створення об’єкту DataTable
      connection.Open();
      dt.Load(command.ExecuteReader());//завантаження даних в обєкт
         connection.Close();
      return dt;
    }
```

Метод LoadMonitoringAdminDataGrid виконує вибірку усіх записів з бази даних та повертає їх в DataTable який вертається методом. На лістингу 2.3 наведено код методу LoadMonitoringAdminDataGrid.

Лістинг 2.3 – код методу LoadMonitoringAdminDataGrid

```
public static DataTable LoadMonitoringAdminDataGrid()
```

```
{
    MySqlCommand command = new MySqlCommand("SELECT Employee.Id as 'Id працівника', " +
        "Employee.FirstName as 'Ім\'я', " +
        "Employee.LastName as Прізвище, " +
        "Employee.Age as Вік, " +
        "DateRegister.Id as 'Id записі', " +
        "DateRegister.Action as Дія, " +
        "DateRegister.DataTime as 'Час дії' " +
        "FROM Employee,  DateRegister WHERE Employee.Id = EmployeeId", connection);
    DataTable dt = new DataTable();
    connection.Open();
    dt.Load(command.ExecuteReader());
    connection.Close();
    return dt;
}
```

Метод LoadMonitoringEmployeeDataGrid виконує вибірку конкретних записів з бази даних та повертає їх в DataTable який вертається методом. На лістингу 2.4 наведено код методу LoadMonitoringEmployeeDataGrid.

Лістинг 2.4 – код методу LoadMonitoringEmployeeDataGrid

```
‘public static DataTable LoadMonitoringEmployeeDataGrid(int id)
    {
    MySqlCommand command = new MySqlCommand("", connection);
    command.CommandText = "SELECT Employee.Id as 'Id праівника', " +
     @"Employee.FirstName as 'Ім\'я', " +
     "Employee.LastName as Прізвище, " +
     "Employee.Age as Вік, " +
     "DateRegister.Id as 'Id записі', " +
     "DateRegister.Action as Дія, " +
     "DateRegister.DataTime as 'Час дії' " +
     "FROM Employee,  DateRegister WHERE Employee.Id = EmployeeId AND Employee.Id = @id ";
    MySqlParameter phoneParam = new MySqlParameter("@id", id);
    command.Parameters.Add(phoneParam);
    DataTable dt = new DataTable();
    connection.Open();
    dt.Load(command.ExecuteReader());
    connection.Close();
    return dt;}
```

Метод LoadMonitoringCombobox виконує вибірку конкретних записів з бази даних, загружає отримані дані в MySqlDataReader після чого добавляє їх в список

стрічок які вертаються методом. На лістингу 2.5 наведено код методу LoadMonitoringCombobox.

Лістинг 2.5 – код методу LoadMonitoringCombobox

```
public static List<string> LoadMonitoringCombobox()
    {
      List<string> ComboBoxList = new List<string>();
MySqlCommand command = new MySqlCommand("SELECT Employee.Id as 'Id працівника', " +
    @"Employee.FirstName as 'Ім\'я', " +
        "Employee.LastName as Прізвище, " +
        "Employee.Age as Вік, " +
        "DateRegister.Id as 'Id записі', " +
        "DateRegister.Action as Дія, " +
        "DateRegister.DataTime as 'Час дії' " +
        "FROM Employee,  DateRegister WHERE Employee.Id = EmployeeId", connection);
      connection.Open();
      MySqlDataReader reader = command.ExecuteReader();
      for (int i = 0; i < 7; i++)
        ComboBoxList.Add(reader.GetName(i));
      connection.Close();
      reader.Close();
      return ComboBoxList; }
```

Метод DeleteRecordMySQL приймає стрічку в параметри і відправляє запит на видалення співробітника з бази даних. На лістингу 2.6 наведено код методу DeleteRecordMySQL.

Лістинг 2.6 – код методу DeleteRecordMySQL

```
public static void DeleteRecordMySQL(string id){
      MySqlCommand command = new MySqlCommand("DELETE FROM Employee WHERE Id = @Id;",
      connection);
      MySqlParameter idParam = new MySqlParameter("@Id", id);
      command.Parameters.Add(idParam);
      connection.Open();
      command.ExecuteNonQuery();
      connection.Close();}
```

2.2.2. Реалізація вбудованих процедур та функцій

При розробці бази даних було створено такі вбудовані процедури та функції:

Вбудовані процедури:

- Процедура “Authorization_Procedure” використовується для авторизації користувача в програмі;
- Процедура “CreateRecordEmployee” використовується для створення нової записі про співробітника;
- Процедура “UpdateRecordEmployee” використовується для редагування записів співробітників;
- Процедура “UpdateEditEmployee” використовується для редагування персональної інформації самим півробітником;
- Користувацькі функції:
- Функція “Authorization_Function” використовується в процедурі “Authorization_Procedure” для отримання ідентифікаційного номера співробітника і перевірки введеного логіну та паролю.
- Функція “ValidLogin” використовується для перевірки на подвійний логін при реєстрації співробітників;

Процедура Authorization_Procedure приймає два параметра логін та пароль користувача, викликає функцію Authorization_Function яка приймає логін та пароль та виконує пошук співробітника з заданим логіном та паролем і вразі знаходження повертає ідентифікаційний номер користувача який присвоюється в вихідний параметр “@EmployeeId”, після чого за найденим ідентифікаційним номер проводить вибірку імені, прізвища та рівень доступу до бази даних і після вибірки процедура повертає вихідні параметри в програму. На лістингу 2.7 наведено код процедури Authorization_ Procedure.

Лістинг 2.7 – код процедури Authorization_Procedure

```
CREATE DEFINER=`aragorn_employee`@`%` PROCEDURE `Authorization_Proc-edure`(
--Вхідні параметри
 IN `@Login` VARCHAR(30),
 IN `@Password` VARCHAR(30),
--Вихідні параметри
```

Продовження лістингу 2.7

```
 OUT `@EmployeeId` INT,
 OUT `@FirstName` VARCHAR(30),
```

```
 OUT `@LastName` VARCHAR(30),
  OUT `@Position` VARCHAR(30))
BEGIN
--Ініціалізація параметрів
SET `@EmployeeId` = Authorization_Function(`@Login`, `@Password`);
SET `@FirstName` = (SELECT FirstName FROM Employee WHERE Id = `@EmployeeId`);
SET `@LastName` = (SELECT LastName FROM Employee WHERE Id = `@EmployeeId`);
SET `@Position` = (SELECT Position FROM Admins WHERE EmployeeId = `@EmployeeId`);
END
```

На лістингу 2.8 наведено код Функції Authorization_Function.

Лістинг 2.8 – код функції Authorization_Function

```
CREATE DEFINER=`aragorn_employee`@`%` FUNCTION `Authorization_Fun-ction`(`@Login` VARCHAR(30), `@Password` VARCHAR(30)) RETURNS int(11)
  READS SQL DATA DETERMINISTIC
BEGIN
DECLARE Result INTEGER;
/*Виконує вибірку та порівняня  ідентифікаційної інформації після чого присвоює ідентифікаційний номер зміній*/
SET Result = (SELECT EmployeeId FROM Admins WHERE Login = `@Login` AND NOT BINARY (SELECT Admins.Password NOT LIKE BINARY `@Password`));
/*Перевіряє змінну на більш рівне нулю і в разі підтвердження функція вертає ідентифікаційний номер в інших випадках null.*/
IF (Result >=0) THEN
RETURN Result;
ELSE
RETURN NULL;
END IF;
END
```

Функція ValidLogin приймає стрічку з логіном та вертає числове значення. В процесі роботи функція перевіряє отриманий логін з наявними в базі даних і повертає відповідний результат. На лістингу 2.9 наведено код функції ValidLogin.

Лістинг 2.9 – код функції ValidLogin

```
CREATE DEFINER=`aragorn_employee`@`%` FUNCTION `ValidLogin`(`@Login` VARCHAR(30)) RETURNS int(11)
   READS SQL DATA
```

```
   DETERMINISTIC
BEGIN
--Перевірка та порівняння логіну вразі знаходження
IF (SELECT (SELECT Login FROM Admins WHERE Login = `@Login`)  LIKE BINARY `@Login`)THEN
RETURN 1;
ELSE
RETURN 0;
END IF;
END
```

Процедура CreateRecordEmployee приймає 13 вхідних параметрів та повертає 1 вихідний параметр. Процедура виконує добавлення записі про співробітника в базу даних (реєструє), в процесі реєстрації проходить перевірка на повторну реєстрацію за однаковим ідентифікаційним номером і вразі підтвердження процедура вертає false, в іншому випадку процедура виконує операції INSERT та вертає true. На лістингу 2.10 наведено код функції CreateRecordEmployee.

Лістинг 2.10 – код функції CreateRecordEmployee

```
CREATE DEFINER=`aragorn_employee`@`%` PROCEDURE `CreateRecordEmployee`(
IN `@Id` INT,  IN `@FirstName` VARCHAR(30), IN `@LastName`VARCHAR(30), IN `@Age` INT,
IN `@Login` VARCHAR(30),  IN `@Password` VARCHAR(30), IN `@Email` VARCHAR(13),
IN `@City` VARCHAR(30),  IN `@Street` VARCHAR(30),  IN `@NumberHome` INT, IN `@Phone` varchar(13),
IN `@Department` INT,  IN `@EPosition` VARCHAR(30), OUT `@Result` BOOL)
BEGIN
--Перевірка на подвійне використання ідентифікаційного номера
if (SELECT Id FROM Employee WHERE Id = `@Id`)
THEN
SET `@Id` = -1,
`@Result` = FALSE;
ELSE
SET `@Result` = TRUE;
END IF;
IF (`@Result` = TRUE) THEN
START TRANSACTION;--початок транзакції та самої реєстрації
INSERT INTO Employee(Id, FirstName, LastName, Age) VALUES (`@Id`,`@FirstName`, `@LastName`, `@Age`);
INSERT INTO Admins(EmployeeId, Login, Password) VALUES (`@Id`, `@Login`, `@Password`);
```

Продовження лістингу 2.10

```
INSERT INTO Email(EmployeeId, Email) VALUES (`@Id`, `@Email`);
INSERT INTO HomeAddress(EmployeeId, City, Street, NumberHome) VALUES (`@Id`, `@City`, `@Street`, `@NumberHome`);
INSERT INTO Phone(EmployeeId, PhoneNumber) VALUES (`@Id`, `@Phone`);
INSERT INTO WorkDepartment(EmployeeId, DepartmentId, EPosition) VALUES (`@Id`, `@Department`, `@EPosition`);
COMMIT;
END IF;
END
```

Процедур UpdateEditEmployee, UpdateRecordEmployee дуже схожі за своєю дією але використовуються в різних випадках, наприклад, якщо адміністратору необхідно оновити інформацію про співробітника то буде використовуватися процедура UpdateRecordEmployee, інша процедура для змінення своєї персональної інформації, тобто використовується при зміні приватної інформації. На лістингу 2.11 наведено код процедур UpdateEditEmployee, UpdateRecordEmployee.

Лістинг 2.11 – код процедур UpdateEditEmployee, UpdateRecordEmployee

```
--Код процедури UpdateRecordEmployee:
CREATE DEFINER=`aragorn_employee`@`%` PROCEDURE `UpdateRecordEmployee`(
IN `@Id` INT, IN `@FirstName` VARCHAR(30), IN `@LastName` VARCHAR(30), IN `@Age` INT,
IN `@PhoneNumber` VARCHAR(13), IN `@Email` VARCHAR(30), IN `@City` VARCHAR(30),
IN `@Street` VARCHAR(30), IN `@NumberHome` VARCHAR(30), IN `@DepartmentId` INT,
IN `@EPosition` VARCHAR(30))
BEGIN
Start transaction;
UPDATE Employee
SET FirstName = `@FirstName`, LastName = `@LastName`, Age = `@Age` WHERE Id = `@Id`;
UPDATE Phone SET PhoneNumber = `@PhoneNumber` WHERE EmployeeId = `@Id`;
UPDATE Email SET Email = `@Email` WHERE EmployeeId = `@Id`;
UPDATE HomeAddress SET City = `@City`, Street = `@Street`, NumberHome = `@NumberHome` WHERE EmployeeId = `@Id`;
UPDATE WorkDepartment SET DepartmentId = `@DepartmentId`, Eposition = `@EPosition` WHERE EmployeeId = `@Id`;
commit;
END
--Код процедури UpdateEditEmployee:
```

Продовження лістингу 2.11

```
CREATE DEFINER=`aragorn_employee`@`%` PROCEDURE `UpdateEditEmployee`(
IN `@Id` INT, IN `@FirstName` VARCHAR(30), IN `@LastName` VARCHAR(30), IN `@Age` INT,
IN `@PhoneNumber` VARCHAR(13), IN `@Email` VARCHAR(30), IN `@City` VARCHAR(30),
IN `@Street` VARCHAR(30), IN `@NumberHome` VARCHAR(30), IN `@Password` VARCHAR(30),
IN `@NewPassword` VARCHAR(30), OUT `@Result` INT)
BEGIN
IF (SELECT (SELECT Password FROM Admins WHERE EmployeeId = `@Id`) LIKE BINARY `@Password`)
THEN
Start transaction;
IF (`@NewPassword` != '')THEN
UPDATE Admins SET Password = `@NewPassword` WHERE EmployeeId = `@Id`;
END IF;
SET `@Result` = 1;
UPDATE Employee SET FirstName = `@FirstName`, LastName = `@LastName`, Age = `@Age` WHERE Id =
`@Id`;
UPDATE Phone SET PhoneNumber = `@PhoneNumber` WHERE EmployeeId = `@Id`;
UPDATE Email SET Email = `@Email` WHERE EmployeeId = `@Id`;
UPDATE HomeAddress SET City = `@City`, Street = `@Street`, NumberHome = `@NumberHome` WHERE
EmployeeId = `@Id`;
commit;
ELSE SET `@Result` = 0;
END IF;
END
```

З перегляду на вище вказані процедури та функції можна зробити висновок що інформаційна система володіє достатньо великим функціоналом.

2.2.3. Розробка GUI

Розробка GUI проводилась з використанням мови розмітки XAML яку використовує WPF для розміщення елементів графіки.

Після встановлення інформаційної системи необхідно ввійти в систему використовуючи власний логін та пароль.

Після входу в систему користувачеві надаються відповідно права за якими він може виконувати ті чи інші дії. Перша що може робити тільки адміністратор це створення нової записі про співробітника і надання йому певних правил (Менеджер/співробітник). Друга можливість це моніторинг дій користувачів, тут

доступ до цієї можливості надається безпосередньо адміністратору та менеджеру, за винятком того що, співробітники можуть спостерігати лише за своїми діями. Програма володіє ще більшим функціоналом який буде розглянутий дальше.

Для входу в систему необхідно запустити програму кликнувши два рази по ярлику програми, після того запуститься вікно в якому необхідно вказати свій логін та пароль. На рисунку 2.3 зображено зовнішній вигляд вікна авторизації.

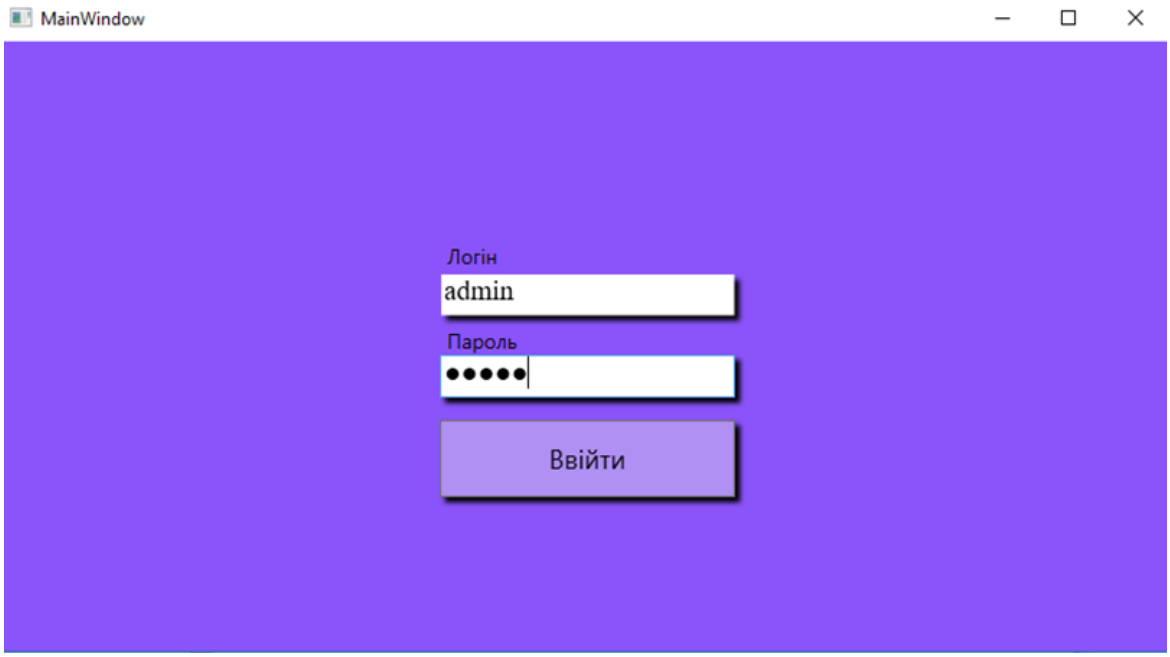

Рисунок 2.3 – Зовнішній вигляд вікна авторизації

На даному рисунку зображено зовнішній вигляд вікна авторизації на якому здійснюється вхід в систему в ролі адміністратора.

Після входу в програму, користувача автоматично перекине на вкладку співробітники якщо в користувача права адміністратора або менеджера, працівника перекине на вкладку моніторинг.

В вкладці “Співробітники” можна створити запис про співробітника, видалити запис про співробітника, редагувати запис про співробітника. Також можна змінити власну персональну інформацію за допомогою кнопки “Редагувати”, яка розташована на лівій, верхній стороні екрану, поруч з цією вкладкою є вкладка “вийти з аккаунту”, яка дозволяє вийти з облікового запису та увійти в систему під

іншим користувачем. Вкладка “Моніторинг” розташована в лівій стороні, по середині та дозволяє переглядати облік робочого часу працівників.

На рисунку 2.4 зображено вкладка “Співробітники”.

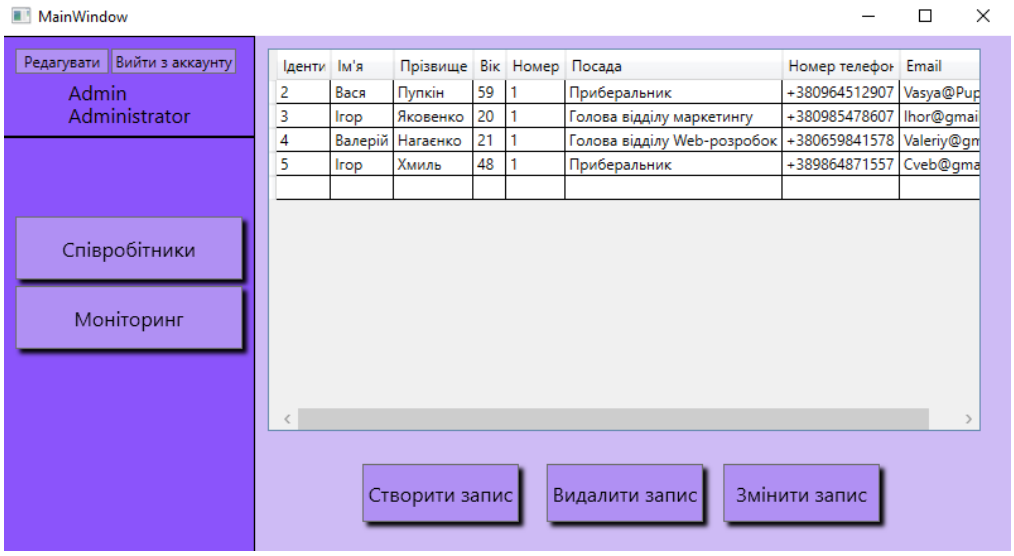

Рисунок 2.4 – Вкладка “Співробітники”

На рисунку 2.5 зображено вкладка “Моніторинг”.

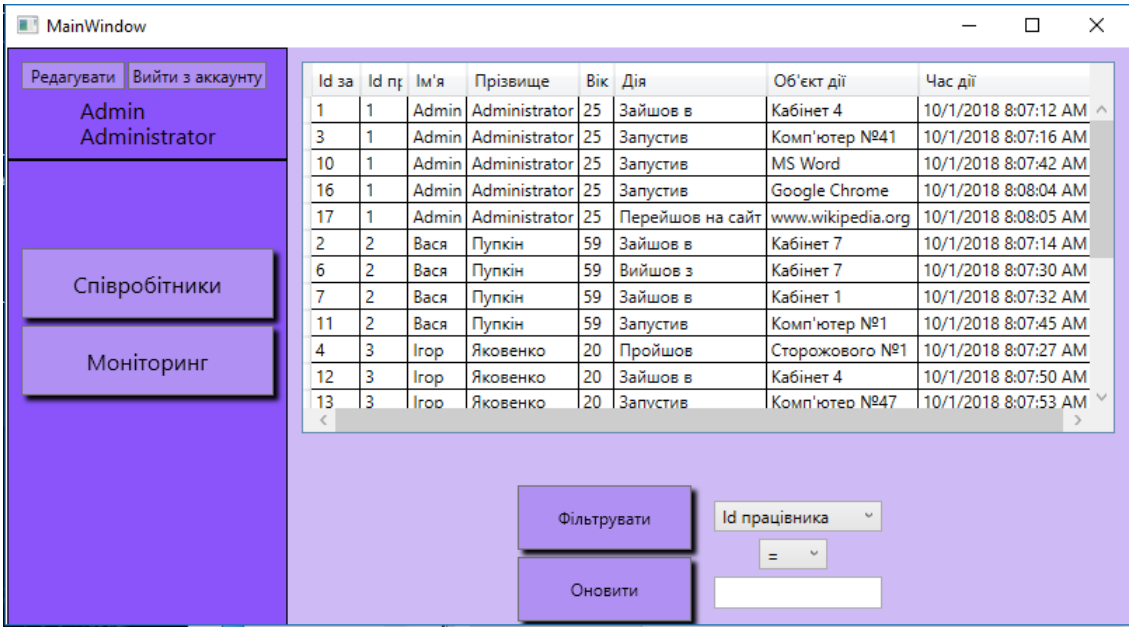

Рисунок 2.5 – Вкладка “Співробітники”

Для видалення та редагування інформації про користувачів використовуються кнопки які відкривають свої вікна. На рисунку 2.6 зображено процес реєстрації співробітника. На рисунку 2.7 зображено вікно видалення записі про користувача.

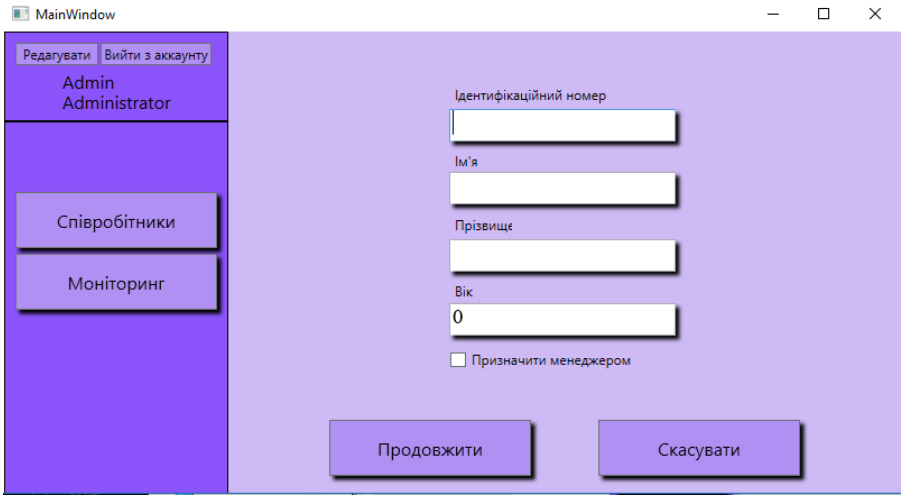

Рисунок 2.6 – Процес реєстрації співробітника

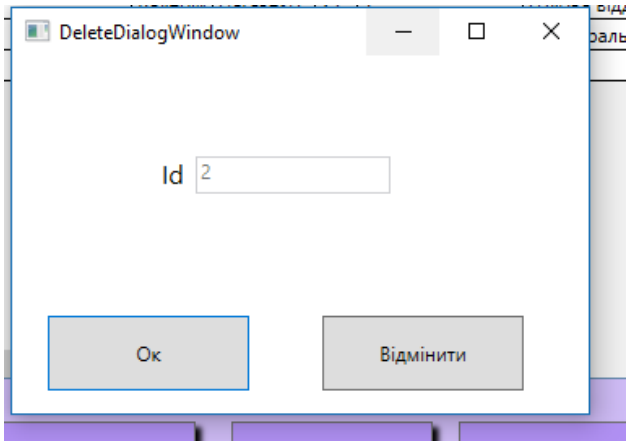

Рисунок 2.7 – Вікно видалення записі про користувача

Для видалення та зміни інформації про користувача необхідно спершу виділити запис про нього а потім на кнопку дії.

В вікні редагування інформації про співробітника можна змінити більшість інформації крім ідентифікаційного номера співробітника. Після завершення зміни

необхідно натиснути кнопку “Зберегти”, після цього вікно закриється і інформація збережеться.

На рисунку 2.8 зображено вікно редагування інформації про співробітника.

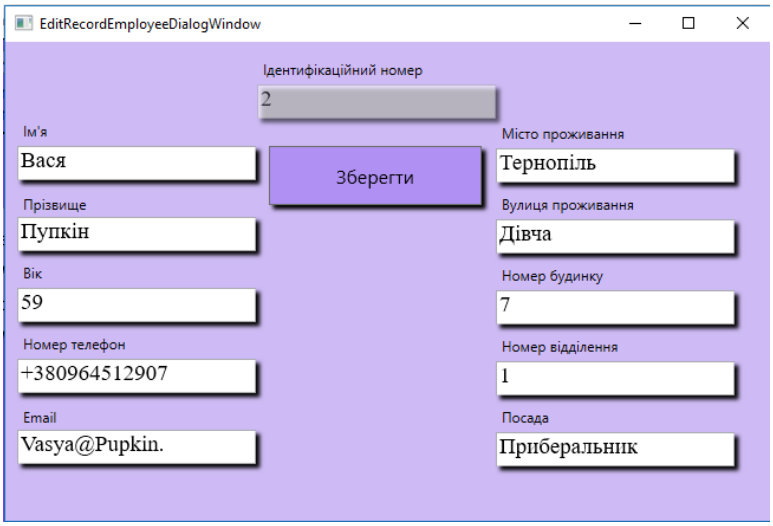

Рисунок 2.8 – Вікно редагування інформації про співробітника

Для зміни персональної інформації є вікно зміни персональної інформації, для відкриття цього вікна необхідно натиснути на кнопку “Редагувати”.

В вікні зміни персональної інформації, дещо менша кількість параметрів яку можна змінити, але при тому тут дається можливість змінити власний пароль.

На рисунку 2.9 зображено вікно зміни персональної інформації.

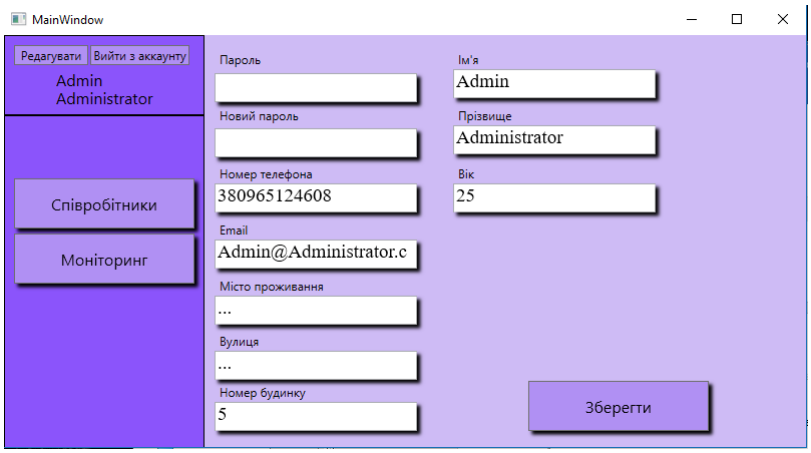

Рисунок 2.9 – Вікно зміни персональної інформації

Для підтвердження зміни персональної інформації необхідно ввести пароль, при тому новий пароль можна не вводити і зміни паролю не відбудеться, в іншому випадку зміна паролю відбудеться на вказаний в полі “Новий пароль”.

2.2.4. Тестування програмного забезпечення та оцінка якості

Основна валідація даних в інформаційній системі проводиться в програмі за допомогою створеного класу Validate який наслідується від інтерфейсу IDataErrorInfo. В даному класі реалізована логіка валідації даних які вводяться в полях.

У той час як нове програмне забезпечення розробляється, іноді може знадобитися протестувати частини програмного забезпечення. Ці тести мають бути записані, щоб задокументувати, що розробка пройшла, як було заплановано.

Програмні продукти вимагають перевірки. Для програмного продукту, який розглядається як інкапсульований функціонал. Підрозділ, мета перевірки полягає в тому, щоб встановити докази того, що його вимоги виконано і що він адекватно працює у своєму реальному або очікуваному оточенні.

Комп’ютерні системи вимагають перевірки в середовищі, в якому вони використовуються. Остаточна перевірка може поєднувати окремі завдання перевірки, що проводяться на всіх програмних продуктах, утворюючи цілісну комп’ютерну систему.

Програма може бути оснащена підпрограмами або функціями, які можуть використовується для тестування або перевірки критичних послідовностей і керування даними. Під час розробки програми слід пам’ятати про вимоги до тестування та оцінювання. Не відкидаючи остаточного тесту всіх куточків програми, добре організована структура може сама по собі забезпечити адекватний тест основних питань підтвердження:

Дані зазвичай зчитуються з вимірювального пристрою, відображаються графічно, а потім зберігаються у файлі даних. Для перевірки потоку даних можна

використовувати засоби, які можуть зчитувати збережені дані для перегляду. Якщо перевірені дані утворюють штучний розпізнаваний шаблон, тестується і сам графічний дисплей.

Найпростіший спосіб перевірки обчислень – це довести, що задані вхідні значення дають очікуване результати. Іноді може бути зручно створити спеціальні додаткові програми тестування для допомоги перевірка складних розрахунків. Такі тестові програми також мають бути підтверджено.

Умова, за якої працює програма, зазвичай контролюється кількома більши або менш заданими параметрами. Зробивши ці параметри доступними та доступними через засоби інтерфейсу користувача, можна перевірити цілісність налаштувань програми.

Валідація програмного забезпечення – встановлення за допомогою об’єктивних доказів того, що всі вимоги до програмного забезпечення були реалізовані правильно і повністю та простежуються до системних вимог.

Метою процесу валідація є забезпечення відповідності конкретним вимогам програмного продукту.

Процес верифікації може виконуватися підрядниками або іншими особами (наприклад, замовниками), які здійснюють дії відповідно до плану для впровадження та виконання процесу, який відображає елементи та цілі аудиту. При цьому методи, засоби та процедури виконання завдань процесу використовуються для встановлення відповідності вимог тестування та правильності характеристик використання програмного продукту проекту вимогам.

Верифікація та валідація полягає у перевірці специфікації та правильності програми відповідно до вимог та формального опису програми.

Перевірка програмного коду допомагає зробити висновки про коректність створеної програмної системи в процесі проектування та після завершення розробки. Перевірка дозволяє визначити доцільність виконання вимог шляхом перегляду, перевірки та оцінки результатів проектування в процесі життєвого циклу, щоб підтвердити, що вимоги реалізовані правильно та відповідають умовам

та обмеженням системи. Верифікація та валідація забезпечують перевірку повноти, узгодженості та чіткості специфікації та коректності функцій системи.

Верифікації і валідації піддаються:

- компоненти системи, їх інтерфейси (програмне забезпечення, технології та інформація) та взаємодія об'єктів (протоколи, повідомлення) у розподіленому середовищі;
- опис доступу до бази даних, методів запобігання несанкціонованому доступу до різних даних користувача;
- системні файли; -Випробування, процедури тестування та набори вхідних даних. Виконуйте інші операції з іншими процесами життєвого циклу: - Використовувати методи та процедури для огляду прогресу розробки для перевірки та контролю проектних рішень;
- доступ до системи CASE, яка включає процедури перевірки вимог до продукції;
- перегляньте та перевірте, чи відповідають проміжні результати вимогам, щоб підтвердити, що програмна система має правильну

При тестуванні валідації полів для введення даних було провірено два варіанта, з правильним введенням даних та некоректним введенням даних. На рисунку 2.10 зображено відвалідовані поля.

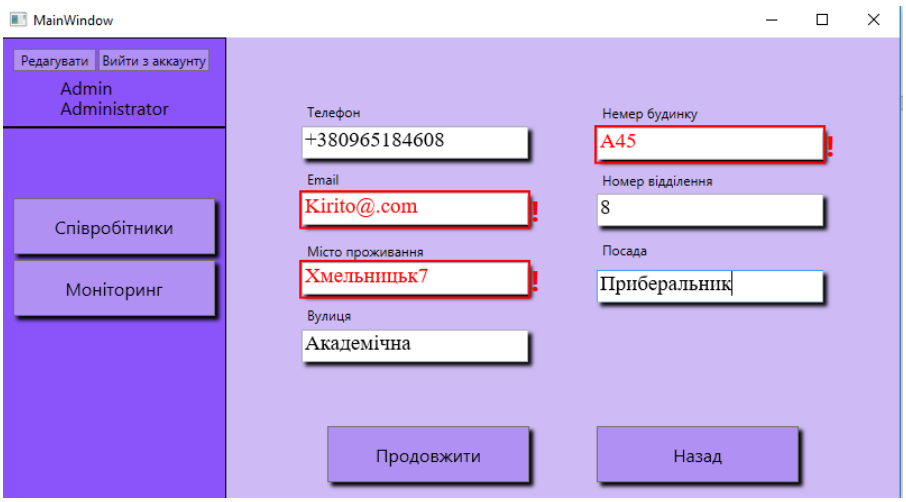

Рисунок 2.10 Відвалідовані поля

При введені некоректної інформації рамки поля і текст стають червоними та при наведені на поля висвічуються підказки з прикладами правильного вводу.

Навантажувальне тестування використовується для перевірки бази даних. Тестування навантаження — це лише одна з форм тестування продуктивності. Зазвичай він використовується для оцінки поведінки програми (додатка) із заданим очікуваним навантаженням. Наприклад, цим навантаженням може бути кількість користувачів, які будуть використовувати програму одночасно. Цей тип тестування дозволяє отримати час відповіді для всіх основних бізнес-операцій.

Для навантажувального тестування було використано спеціальну утиліту “mysqlslap”.

Mysqlslap – діагностична програма, розроблена для навантажувального тестування серверів MySQL та звітування про час виконання кожного етапу. Вона емулює запити багатьох користувачів до однієї бази даних.

Для запуску тестування необхідно запустити консольний рядок і в ньому перейти в папку до програми та використовуючи спеціальну команду з ключами відправити запит.

Зразок команди: mysqlslap –u “Ім’я користувача” -p --host=”адреса бази даних” --port=”порт бази даних” --create-schema=”назва бази даних” --query="запит який буде виконуватися в тестувані" concurrency=”кількість користувачів” --iterations=”повторів для одного користувача. Слід зауважити що чим більша кількість спроб та користувачів буде проводити тестування то більш точнішим буде тест, але при тому він може затягнутися надовго.

Протестував базу даних на одночасну вибірку інформації з бази даних, однією тисячу користувачів, які відправляють по десять запитів. Після відправлення запиту з ключами на тестування програма запросила пароль, після чого почалось тестування і було виведено результати.

На рисунку 2.11 зображено результати тестування бази даних на вибірку.

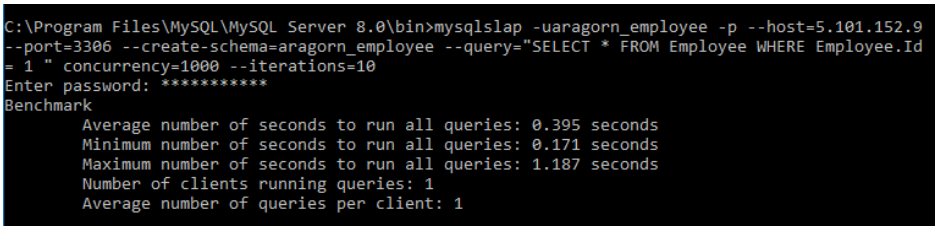

Рисунок 2.11 – Результати тестування бази даних на вибірку

Результат тестування виявився цілком успішним і швидким, можна сказати що база даних пройшла тестування на вибірку.

Протестував базу даних на вибірку представлення з бази даних однією тисячу користувачів, які відправляють по 15 запитів. На рисунку 2.12 зображено результати тестування бази даних на вибірку представлення.

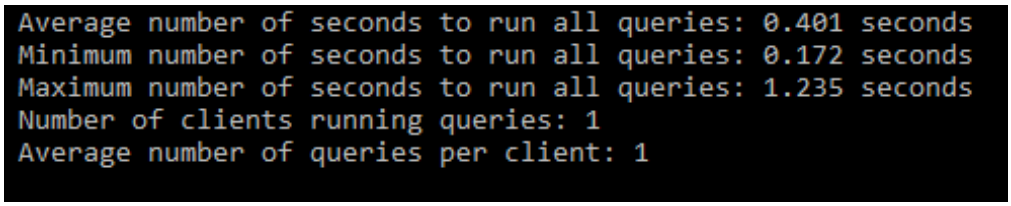

Рисунок 2.12 – Результати тестування бази даних на вибірку представлення

Результати тестування теж виявилися успішними і швидкими, але порівняючи результати між попередніми тестуваннями можна сказати що вибірка представлення виконується дещо повільніше чим проста вибірка.

Тестуючи базу даних на відправлення запиту авторизації однією тисячу користувачів які відправляють по 5 запитів.

На рисунку 2.13 зображено результати тестування бази даних на відправлення запитів авторизації.

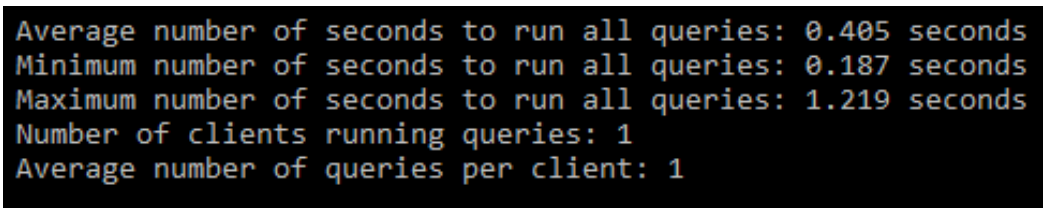

Рисунок 2.13 – Результати тестування бази даних на відправлення запитів авторизації

Після тестування бази даних можна сказати що сервер на якій розташована база даних справляється зі своєю роботою а сама база даних була спроектована правильно і без явних дефектів.

РОЗДІЛ 3. ОХОРОНА ПРАЦІ ТА БЕЗПЕКИ В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

3.1 Охорона праці

3.2 Безпека в надзвичайних ситуаціях

ВИСНОВОК

В процесі виконання атестаційна роботи магістра було розроблено інформаційну систему обліку ефективності робочого часу працівників.

В першому розділі було проведено грунтовний аналіз сфери автоматизації, де чітко вказав перелік акторів та їх сценарії використання системи, було вказано конкретні технічні засоби на базі яких було проведено розробку програмного продукту.

В другому розділі було показано, які сутності є корисними для інформаційної системи та було представлено їх основні характеристики, побудував схему бази даних відносно виявлених сутностей та їх атрибутів, логічно пов'язуючи сутності між собою та представлено варіанти використання. Була проведена нормалізація бази даних з усіма вимогами 1-ї, 2-ї та 3-ї нормальних форм. В результаті було побудовано ER-діаграми, архітектуру програмного забезпечення з використанням UML-діаграм, також було описано взаємодію програми з базою даних і конкретну реалізацію класів та методів. Також була описана частина функціоналу яка розташована на стороні бази даних. Також розглянуто забезпечення якості бази даних, та програмної системи, провів навантажувальне тестування бази даних та валідацію даних.

Під час виконання роботи, було виявлено позитивні та негативні сторони програмного забезпечення, та було виконано завдання яке було описано в постанові задач. Проаналізувавши базу даних можна сказати що вона є стабільною та продуктивною. Інформаційна система забезпечує облік робочого часу працівників, облік працівників.

ПЕРЕЛІК ПОСИЛАНЬ

1. Руководство по ADO.NET и работе с базами данных [Електронний ресурс] – URL: https://metanit.com/sharp/adonet/ – 31.10.2015 р.

2. Руководство по MySQL [Електронний ресурс] – URL: https://metanit.com/sql /tutorial/ 06.12.2018. – 06.12.2018.

3. Язык программирования C# и .NET [Електронний ресурс] – URL: https://metanit.com/sharp/general.php – 06.12.2018 р.

4. Руководство по WPF MySQL [Електронний ресурс] – URL: https://metanit.com/sharp/wpf/ Дата доступу: 06.12.2018. – Заголовок з екрану.

5. Платформа ASP.NET https://metanit.com/sharp/mvc.php Дата доступу: 06.12.2018. – Заголовок з екрану.

6. mysqlslap — Load Emulation Client. MySQL [Електронний ресурс] – URL: https://dev.mysql.com/doc/ refman/8.0/en/mysqlslap.html Дата доступу: 06.12.2018 р

ДОДАТКИ

Додаток Б – SQL скрипти для створення таблиць бази даних

```
CREATE TABLE Employee
(
        Id INT PRIMARY KEY NOT NULL,
        FirstName varchar(30) NOT NULL,
   LastName varchar(30) NOT NULL,
   Age int NOT NULL  CHECK(Age >0 AND Age < 100)
);

CREATE TABLE Phone
(       EmployeeId INT NOT NULL PRIMARY KEY,
        PhoneNumber varchar(13) NOT NULL CHECK(PhoneNumber !=''),
   FOREIGN KEY (EmployeeId)  REFERENCES Employee(Id)  ON DELETE CASCADE
);

CREATE TABLE HomeAddress
(
        EmployeeId INT NOT NULL PRIMARY KEY,
        City varchar(30) NOT NULL,
        Street varchar(30) NOT NULL,
   NumberHome int NOT NULL,
   FOREIGN KEY (EmployeeId)  REFERENCES Employee(Id)  ON DELETE CASCADE
);

CREATE TABLE Email
(
        EmployeeId INT UNIQUE NOT NULL PRIMARY KEY,
        Email varchar(30) UNIQUE NOT NULL CHECK(Email !=''),
        FOREIGN KEY (EmployeeId)  REFERENCES Employee(Id)  ON DELETE CASCADE
);
```

```
CREATE TABLE Admins
(
        EmployeeId INT PRIMARY key UNIQUE,
        Login varchar(16) NOT NULL UNIQUE ,
        Password varchar(30) NOT NULL,
   Position VARCHAR(30) DEFAULT 'Працівник',
   FOREIGN KEY (EmployeeId)  REFERENCES Employee(Id)  ON DELETE CASCADE
);

CREATE TABLE DateRegister
(
        Id int auto_increment PRIMARY KEY NOT NULL,
        EmployeeId INT NOT NULL,
   Action VARCHAR(30) NOT NULL,
   ObjectAction VARCHAR(30) NOT NULL,
        DataTime DATETIME NOT NULL,
        FOREIGN KEY (EmployeeId)  REFERENCES Employee(Id)  ON DELETE CASCADE
);
CREATE TABLE WorkDepartment
(
        EmployeeId INT PRIMARY KEY,
   DepartmentId int NOT NULL,
   EPosition  varchar(30) NOT NULL,
        FOREIGN KEY (EmployeeId)  REFERENCES Employee(Id)  ON DELETE CASCADE
);

CREATE TABLE ReportsTableEmployeeOld
(
        Id INT auto_increment PRIMARY KEY,
        EmployeeIdOld Int,
   FirstNameOld VARCHAR(30),
   LastNameOld VARCHAR(30),
```

```
    AgeOld INT
);

CREATE TABLE ReportsTableEmployeeNew
(
        Id INT auto_increment PRIMARY KEY,
        EmployeeIdNew Int,
    FirstNameNew VARCHAR(30),
    LastNameNew VARCHAR(30),
    AgeNew INT
);

CREATE TABLE ReportsTableEmployee
(
        Id INT auto_increment PRIMARY KEY,
    ReportsTableEmployeeIdNew INT references ReportsTableEmployeeNew(Id),
        ReportsTableEmployeeIdOld INT references ReportsTableEmployeeOld(Id),
        Action VARCHAR(30) NOT NULL,
    DataTime DATETIME NOT NULL
);
        /*insert into Employee (Id, FirstName, LastName, Age) Values (1,'Admin','Administrator', 28);*/
    insert into Admins (EmployeeId, Login, Password,Position) Values (1,'admin','admin','Адміністратор') ;
    INSERT INTO Email(EmployeeId, Email) VALUES (1, 'Admin@Administrator.com');
        INSERT INTO HomeAddress(EmployeeId, City, Street, NumberHome) VALUES (1, '...', '...', 5);
        INSERT INTO Phone(EmployeeId, PhoneNumber) VALUES (1, +380965124608);
    SELECT * FROM Admins, Employee;
CREATE DEFINER=`root`@`localhost` FUNCTION `ValidLogin`(`@Login` VARCHAR(30)) RETURNS int(11)
    READS SQL DATA
    DETERMINISTIC
BEGIN

IF (SELECT (SELECT Login FROM Admins WHERE Login = `@Login`)  LIKE BINARY `@Login`)THEN
RETURN 1;
```

```
ELSE
RETURN 0;
END IF;
END
CREATE DEFINER=`aragorn_employee`@`%` FUNCTION `Authorization_Function`(`@Login` VARCHAR(30), `@Password` VARCHAR(30)) RETURNS int(11)
    READS SQL DATA
    DETERMINISTIC
BEGIN
DECLARE Result INTEGER;
SET Result = (SELECT EmployeeId FROM Admins WHERE Login = `@Login` AND NOT BINARY (SELECT Admins.Password NOT LIKE BINARY `@Password`));
IF (Result >=0) THEN
RETURN Result;
ELSE
RETURN NULL;
END IF;
END
CREATE
    ALGORITHM = UNDEFINED
    SQL SECURITY DEFINER
VIEW `viewemployees` AS
    SELECT
        `Employee`.`Id` AS `Ідентифікаціфйний номер`,
        `Employee`.`FirstName` AS `Ім'я`,
        `Employee`.`LastName` AS `Прізвище`,
        `Employee`.`Age` AS `Вік`,
        `WorkDepartment`.`DepartmentId` AS `Номер відділення`,
        `WorkDepartment`.`EPosition` AS `Посада`,
        `Phone`.`PhoneNumber` AS `Номер телефону`,
        `Email`.`Email` AS `Email`
    FROM
        (((`Employee`
        JOIN `WorkDepartment` ON ((`Eorkdepartment`.`EmployeeId` = `Employee`.`Id`)))
```

```
    JOIN `Phone` ON ((`Phone`.`EmployeeId` = `Employee`.`Id`)))
    JOIN `Email` ON ((`Email`.`EmployeeId` = `Employee`.`Id`)))
CREATE DEFINER=`root`@`localhost` PROCEDURE `CreateRecordEmployee`(
IN `@Id` INT,
IN `@FirstName` VARCHAR(30),
IN `@LastName`VARCHAR(30),
IN `@Age` INT,
IN `@Login` VARCHAR(30),
IN `@Password` VARCHAR(30),
IN `@Email` VARCHAR(13),
IN `@City` VARCHAR(30),
IN `@Street` VARCHAR(30),
IN `@NumberHome` INT,
IN `@Phone` varchar(13),
IN `@Department` INT,
IN `@EPosition` VARCHAR(30),
IN `@AuthPosition` BOOL,
OUT `@Result` BOOL)
BEGIN

if (SELECT Id FROM Employee WHERE Id = `@Id`)
THEN
SET `@Id` = -1,
        `@Result` = FALSE;
ELSE
SET `@Result` = TRUE;
END IF;


IF (`@Result` = TRUE) THEN
START TRANSACTION;
INSERT INTO Employee(Id, FirstName, LastName, Age) VALUES (`@Id`,`@FirstName`, `@LastName`, `@Age`);
IF (`@AuthPosition` = FALSE) THEN
```

```
INSERT INTO Admins(EmployeeId, Login, Password) VALUES (`@Id`, `@Login`, `@Password`);

ELSE INSERT INTO Admins(EmployeeId, Login, Position, Password ) VALUES (`@Id`, `@Login`, 'Менеджер',
`@Password`);

END IF;

INSERT INTO Email(EmployeeId, Email) VALUES (`@Id`, `@Email`);

INSERT INTO HomeAddress(EmployeeId, City, Street, NumberHome) VALUES (`@Id`, `@City`, `@Street`,
`@NumberHome`);

INSERT INTO Phone(EmployeeId, PhoneNumber) VALUES (`@Id`, `@Phone`);

INSERT INTO WorkDepartment(EmployeeId, DepartmentId, EPosition) VALUES (`@Id`, `@Department`, `@EPosition`);

COMMIT;

END IF;

END

CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateEditEmployee`(IN `@Id` INT,

IN `@FirstName` VARCHAR(30),

IN `@LastName` VARCHAR(30),

IN `@Age` INT,

IN `@PhoneNumber` VARCHAR(13),

IN `@Email` VARCHAR(30),

IN `@City` VARCHAR(30),

IN `@Street` VARCHAR(30),

IN `@NumberHome` VARCHAR(30),

IN `@Password` VARCHAR(30),

IN `@NewPassword` VARCHAR(30),

OUT `@Result` INT)

BEGIN


IF (SELECT (SELECT Password FROM Admins WHERE EmployeeId = `@Id`) LIKE BINARY `@Password`) THEN

Start transaction;

IF (`@NewPassword` != '')THEN

UPDATE Admins

SET  Password = `@NewPassword` WHERE  EmployeeId = `@Id`;

END IF;

SET `@Result` = 1;

UPDATE Employee
```

```
SET FirstName = `@FirstName`, LastName = `@LastName`, Age = `@Age` WHERE  Id = `@Id`;
UPDATE Phone
SET PhoneNumber = `@PhoneNumber` WHERE  EmployeeId = `@Id`;
UPDATE Email
SET Email = `@Email` WHERE  EmployeeId = `@Id`;
UPDATE HomeAddress
SET City = `@City`, Street = `@Street`, NumberHome = `@NumberHome` WHERE  EmployeeId = `@Id`;
commit;
ELSE SET `@Result` = 0;
END IF;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateRecordEmployee`(
IN `@Id` INT,
IN `@FirstName` VARCHAR(30),
IN `@LastName` VARCHAR(30),
IN `@Age` INT,
IN `@PhoneNumber` VARCHAR(13),
IN `@Email` VARCHAR(30),
IN `@City` VARCHAR(30),
IN `@Street` VARCHAR(30),
IN `@NumberHome` VARCHAR(30),
IN `@DepartmentId` INT,
IN `@EPosition` VARCHAR(30))
BEGIN
Start transaction;
UPDATE Employee
SET FirstName = `@FirstName`, LastName = `@LastName`, Age = `@Age` WHERE  Id = `@Id`;
UPDATE Phone
SET PhoneNumber = `@PhoneNumber` WHERE  EmployeeId = `@Id`;
UPDATE Email
SET Email = `@Email` WHERE  EmployeeId = `@Id`;
UPDATE HomeAddress
```

```
SET City = `@City`, Street = `@Street`, NumberHome = `@NumberHome` WHERE  EmployeeId = `@Id`;
UPDATE WorkDepartment
SET DepartmentId = `@DepartmentId`, Eposition = `@EPosition` WHERE  EmployeeId = `@Id`;
commit;
END

CREATE DEFINER=`root`@`localhost` PROCEDURE `Authorization_Procedure`(
 IN `@Login` VARCHAR(30),
 IN `@Password` VARCHAR(30),
 OUT `@EmployeeId` INT,
 OUT `@FirstName` VARCHAR(30),
 OUT `@LastName` VARCHAR(30),
  OUT `@Position` VARCHAR(30))
BEGIN
/*DECLARE `@EmployeeId` INT;*/
SET `@EmployeeId` = Authorization_Function(`@Login`, `@Password`);
SET `@FirstName` = (SELECT FirstName FROM Employee WHERE Id = `@EmployeeId`);
SET `@LastName` = (SELECT LastName FROM Employee WHERE Id = `@EmployeeId`);
SET `@Position` = (SELECT Position FROM Admins WHERE EmployeeId = `@EmployeeId`);
END
CREATE DEFINER=`root`@`localhost` TRIGGER `Integrity_control` AFTER DELETE ON `employee` FOR EACH ROW
BEGIN
DELETE FROM Admins WHERE EmployeeId = OLD.Id;
DELETE FROM Email WHERE EmployeeId = OLD.Id;
DELETE FROM Phone WHERE EmployeeId = OLD.Id;
DELETE FROM HomeAddress WHERE EmployeeId = OLD.Id;
DELETE FROM WorkDepartment WHERE EmployeeId = OLD.Id;
END
CREATE DEFINER=`aragorn_employee`@`%` TRIGGER `Employee_AFTER_UPDATE` AFTER UPDATE ON `Employee`
FOR EACH ROW BEGIN
DECLARE `@Id` INT;
INSERT INTO `ReportsTableEmployeeOld` (EmployeeIdOld, FirstNameOld, LastNameOld, AgeOld)
VALUES (Old.Id ,Old.FirstName, Old.LastName, Old.Age);
SET `@Id` = LAST_INSERT_ID();
```

```
INSERT INTO `ReportsTableEmployeeNew` (EmployeeIdNew, FirstNameNew, LastNameNew, AgeNew)

VALUES (New.Id ,New.FirstName, New.LastName, New.Age);

INSERT INTO `ReportsTableEmployee` (ReportsTableEmployeeIdOld, ReportsTableEmployeeIdNew, Action, DataTime)

VALUES (`@Id`, LAST_INSERT_ID(), 'Видалення',CURRENT_TIMESTAMP());

END

CREATE DEFINER=`root`@`localhost` TRIGGER `Employee_Reports_Delete` AFTER DELETE ON `employee` FOR EACH ROW BEGIN

DECLARE `@Id` INT;

INSERT INTO `reportstableemployeeold` (EmployeeIdOld, FirstNameOld, LastNameOld, AgeOld)

VALUES (Old.Id ,Old.FirstName, Old.LastName, Old.Age);

SET `@Id` = LAST_INSERT_ID();

INSERT INTO `reportstableemployee` (ReportsTableEmployeeIdOld , Action, DataTime)

VALUES (`@Id`, 'Видалення',CURRENT_TIMESTAMP());

END
```

Додаток В – UML діаграма класів