КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра комп'ютерної інженерії

До захисту допущено: «На правах рукопису»
Завідувач кафедри _____ Юрій Бойко
« _ » _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

«РОЗРОБКА СИСТЕМИ СКЛАДСЬКОГО ОБЛІКУ ПРОДУКЦІЇ ХАРЧОВОЇ ПРОМИСЛОВОСТІ»

Виконав:
студент 4-го курсу бакалаврату
денної форми навчання
спеціальності 123 Комп'ютерна інженерія
ОНП «_____»
Іван Малюк _____

Науковий керівник:
асистент
Юрій Юрчик _____

Рецензент:
_____
кандидат фізико-математичних наук, асистент
Іван Коломієць _____

Засвідчую, що у цій бакалаврській роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____

Робота допущена до захисту в ЕК рішенням кафедри _____
від «__»_____ 2023 р., протокол № __.

Завідувач кафедри _____,
кандидат фізико-математичних наук, доцент
Бойко Юрій Володимирович

(підпис)

Київ – 2023

ЗМІСТ

ВСТУП

Актуальність дослідження: Сучасна харчова промисловість є складним і динамічним сектором господарства, який забезпечує виробництво, переробку і постачання продуктів харчування населенню. Одним з ключових аспектів успішної роботи в галузі харчової промисловості є ефективне управління складськими процесами і продукцією. В цьому контексті система складського обліку стає невід'ємною частиною діяльності підприємств харчової промисловості. Система складського обліку визначається як комплексний підхід до ведення обліку, контролю і управління всіма складськими операціями, включаючи приймання, зберігання, переміщення та відпуск продукції. Її основна мета полягає в забезпеченні належного контролю за складськими запасами, оптимізації обсягів і розподілу продукції, забезпеченні своєчасності поставок і задоволенні потреб споживачів.

В сучасних умовах, коли ринкова конкуренція стає все більш жорсткою, ефективна система складського обліку продукції харчової промисловості набуває важливості. Вона допомагає забезпечити раціональне використання ресурсів, уникнути надмірного запасу або його дефіциту, знизити витрати на зберігання і транспортування, підвищити якість обслуговування клієнтів та загальну ефективність підприємства.

Розробка програмної системи складського обліку продукції харчової промисловості є вкрай актуальною в наш час, оскільки сучасна харчова промисловість потребує не тільки ефективного виробництва продукції, а й організації її зберігання та розподілу на складах. У зв'язку з ростом конкуренції в галузі та збільшенням обсягів виробництва продукції, необхідно мати чітку інформацію про кількість та розміщення всієї продукції на складах.

При цьому, потрібно забезпечити точний облік, контроль і керування запасами, що включає в себе відстеження надходжень і виходів товарів, контроль за термінами придатності, організацію оптимального розміщення на складі та розподіл продукції з урахуванням попиту. Недостатня ефективність

системи складського обліку може призвести до таких проблем, як зайві витрати на зберігання, втрати продукції через порушення термінів придатності, складські затори і неконтрольоване зростання запасів. Такі проблеми мають прямий вплив на фінансові показники підприємства та його конкурентоспроможність.

Розробка програмної системи складського обліку продукції харчової промисловості дозволить автоматизувати і оптимізувати процеси управління запасами та забезпечити точну інформацію про кількість, розташування та стан продукції на складах. Вона спрощує ведення обліку, дозволяє в режимі реального часу відслідковувати запаси, виконувати прогнозування потреби у товарах, покращує планування поставок та забезпечує швидкий доступ до необхідної інформації для прийняття керівних рішень.

Така система допоможе знизити ризики втрати продукції, зменшити витрати на зберігання, покращити якість обслуговування клієнтів і збільшити ефективність діяльності підприємства харчової промисловості. Вона сприятиме підвищенню конкурентоспроможності підприємств у цій галузі, їх здатності оперативно реагувати на виклики.

Мета дослідження: Метою дослідження є розробка програмної системи складського обліку продукції харчової промисловості з метою покращення ефективності управління запасами, забезпечення точного обліку і контролю за рухом продукції на складах, а також оптимізації розподілу продукції з урахуванням попиту та зменшення витрат підприємств. Виходячи з поставленої мети в даній дипломній роботі, до виконання випливають наступні **завдання:**

- провести аналіз предметної області;
- визначити специфікацію вимог до розроблюваної системи;
- розробити програмну систему.

Предмет дослідження: Предметом дослідження є система складського обліку продукції харчової промисловості, яка включає в себе процеси приймання, зберігання, переміщення та відпуску продукції на складах.

Дослідження спрямоване на вивчення проблем, що виникають у сфері складського управління продукцією харчової промисловості та розробку програмної системи, яка допоможе вирішити ці проблеми.

Об’єкт дослідження: Об'єктом дослідження є процеси управління складськими запасами продукції харчової промисловості, включаючи приймання, зберігання, переміщення та відпуск товарів. Дослідження орієнтоване на виявлення проблемних аспектів управління запасами та розробку програмної системи, яка забезпечить ефективний контроль, облік і керування складськими процесами продукції харчової промисловості.

Структура роботи: дипломна робота складається з вступу, трьох розділів, висновків, списку використаних джерел та додатків. Дипломна робота розміщена на 105 сторінках та налічує в собі 5 посилань на бібліографічні джерела.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ООП Об'єктно-Орієнтоване Програмування

CRM (англ. Customer Relationship Management) Управління взаємовідносинами з клієнтами

БД База Даних

UML (англ. Unified Modeling Language) Уніфікована мова моделювання

ORM (англ. Object-Relational Mapping) Об'єктно-реляційне відображення

SQL (англ. Structured Query Language) Мова структурованих запитів

API (англ. Application Programming Interface) Інтерфейс програмування додатків

ACID (англ. Atomicity Consistency Isolation Durability) Атомарність Узгодженість Ізольованість Довговічність

BLOB (англ. Binary Large OBject) Бінарний великий об'єкт

РРО Реєстратор Розрахункових Операцій

XML (англ. Extensible Markup Language) Розширювана мова розмітки

ФОП Фізична Особа - Підприємець

ТЕП Техніко-Економічні Показники

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз сучасного стану проблеми

Аналіз сучасного стану використання програмних систем складського обліку продукції в харчовій промисловості є важливим кроком для розуміння поточної ситуації та виявлення недоліків, які можна вирішити шляхом розробки нової програмної системи.

На сьогоднішній день, багато підприємств харчової промисловості використовують програмні системи складського обліку для автоматизації та поліпшення управління своїми запасами. Однак, існуючі системи не завжди повністю задовольняють потреби підприємств. Деякі з них можуть бути застарілими, обмеженими у функціональності або не забезпечувати достатньої точності інформації.

Аналізуючи сучасний стан використання програмних систем складського обліку в харчовій промисловості, можна виявити деякі загальні проблеми. Деякі системи можуть бути складними у використанні або вимагати значних зусиль для їх налаштування та підтримки. Також можуть відсутні функції аналізу даних, прогнозування попиту або інтеграції з іншими системами підприємства.

Розробка нової програмної системи складського обліку продукції харчової промисловості є актуальною, оскільки вона може вирішити ці проблеми і покращити ефективність управління запасами. Нова система може пропонувати зручний інтерфейс, широкі можливості аналізу даних, прогнозування попиту та інтеграцію з іншими системами, що сприятиме зниженню витрат, оптимізації процесів та підвищенню рівня обслуговування.

Крім того, розробка нової програмної системи складського обліку продукції харчової промисловості може принести наступні переваги:

1. *Точність обліку*: Нова система забезпечить точність і надійність обліку запасів продукції. Це важливо для забезпечення відповідності запасів фактичному стану та уникнення помилок, таких як надмірні або недостатні запаси.
2. *Оптимізація розподілу продукції*: Нова система дозволить виявляти тенденції попиту та прогнозувати його зміни. Це дозволить підприємствам оптимізувати розподіл продукції на складах, зменшити затримки в постачанні та забезпечити належний рівень обслуговування клієнтів.
3. *Ефективне управління запасами*: Нова система надасть можливість контролювати рух продукції на складах, виявляти швидкопсувні товари та здійснювати своєчасне поповнення запасів. Це допоможе зменшити втрати від застарілих або пошкоджених товарів, а також забезпечити ефективне використання складського простору.
4. *Зниження витрат*: Нова програмна система може забезпечити автоматизацію багатьох рутинних процесів, що зменшить витрати на ручну працю та знизить ймовірність помилок. Також можливість аналізу даних дозволить ідентифікувати області, де можна зробити економічні покращення та знизити загальні витрати підприємства.
5. *Покращення контролю*: Нова система забезпечить більш ефективний контроль за запасами, рухом товарів та виконанням складських операцій. Вона дозволить отримувати регулярні звіти та аналітику щодо стану запасів, обороту, затримок у постачанні та інших показників. Це дозволить керівництву підприємства приймати обґрунтовані рішення щодо планування виробництва, закупівель та логістики.

Програми складського обліку продукції в харчовій промисловості використовуються з метою ефективного керування запасами, планування постачання та забезпечення належної обліковості. Деякі з найпоширеніших програм складського обліку, що використовуються в галузі харчової промисловості, включають:

1. *SAP Warehouse Management (WM)*: Це одна з провідних програм для управління складом, яка надає широкий спектр функцій, включаючи отримання, зберігання та розміщення товарів на складі, виконання інвентаризації та здійснення переміщень товарів. SAP WM також забезпечує інтеграцію з іншими модулями SAP ERP, що дозволяє планувати виробництво та постачання продукції.
2. *Oracle Warehouse Management*: Ця програма надає комплексний підхід до управління складськими операціями, включаючи отримання, зберігання, переміщення та відвантаження товарів. Вона також має можливості для прогнозування попиту, оптимізації запасів та планування поставок.
3. *WMS by Manhattan Associates*: Ця програма спеціалізується на управлінні складом і надає функції, такі як приймання товарів, зберігання, підготовка до відвантаження та інвентаризація. Вона дозволяє відстежувати рух товарів на складі, забезпечує точність обліку та допомагає вирішувати проблеми, пов'язані з управлінням запасами.
4. *Infor Supply Chain Management*: Ця програма надає комплексні можливості для управління запасами, складськими операціями та логістикою в галузі харчової промисловості. Вона дозволяє відстежувати рух товарів в режимі реального часу, управляти запасами, автоматизувати процеси приймання та відвантаження товарів, а також планувати постачання та оптимізувати розподіл запасів між різними складами.

Аналіз сучасного стану використання програмних систем складського обліку в харчовій промисловості є важливим кроком для розробки програмної системи, яка буде оптимально відповідати потребам галузі. Проведений аналіз дозволяє виявити переваги та недоліки існуючих програм, визначити їхню відповідність вимогам харчової промисловості та ідентифікувати можливі проблеми, що потребують вирішення.

На підставі результатів аналізу можна розробити програмну систему складського обліку продукції харчової промисловості, яка буде забезпечувати точний облік запасів, ефективне планування постачання та оптимізацію

виробничих процесів. При цьому, програмна система повинна бути гнучкою, легко налаштовуватись під потреби конкретного підприємства і забезпечувати інтеграцію з іншими системами управління, такими як системи електронного документообігу та обліку фінансів.

Обирання мови програмування для розробки нової програмної системи складського обліку продукції харчової промисловості потребує уважного аналізу та обґрунтування. Ось декілька причин, чому мова програмування C# може бути обрана для даного проекту: Підтримка платформи .NET: C# є основною мовою програмування для платформи .NET, яка надає широкий спектр функціональності та інструментів для розробки різноманітних додатків. У випадку, якщо існують інші системи чи додатки, розроблені на платформі .NET, використання C# дозволить забезпечити легку інтеграцію та взаємодію з ними. Об'єктно-орієнтований підхід: C# є мовою, що підтримує об'єктно-орієнтоване програмування (ООП), що сприяє створенню модульного та розширюваного коду. Це може бути корисно при розробці великих та складних програмних систем, таких як система складського обліку, де важливо мати чітку структуру та можливість повторного використання коду.

На основі аналізу сучасного стану використання програмних систем складського обліку продукції в харчовій промисловості можна зробити висновок, що розробка нової програмної системи на основі мови програмування C# є обґрунтованою та доцільною. Вибір мови програмування C# для розробки такої системи обумовлений не лише її популярністю, але й низкою переваг, які надає ця мова програмування:

Підтримка платформи .NET: C# є основною мовою програмування для платформи NET. Платформа .NET має розгалужену екосистему, що сприяє швидкій розробці та забезпечує високу продуктивність. Об'єктно-орієнтована мова: C# підтримує об'єктно-орієнтовану парадигму програмування, що сприяє модульності, розширюваності та підтримці складних структур даних, що важливо для розробки програмної системи складського обліку. Широкі можливості розробки інтерфейсу користувача: C# разом з платформою .NET

надає потужні інструменти для розробки графічного інтерфейсу користувача. Це дозволяє створити зручний та інтуїтивно зрозумілий інтерфейс для користувачів програмної системи складського обліку. Багатофункціональність: C# має багатий набір функцій та бібліотек, що спрощують розробку різноманітних функціональних можливостей програмної системи.

1.2 Аналіз аналогічних програм, баз даних, систем, обґрунтування вибраного напряму та вибір методів рішення

Аналіз аналогічних програм та систем є важливою частиною процесу розробки більшості програмних систем. Цей аналіз допомагає зрозуміти наявні рішення на ринку, їхні можливості та обмеження, а також визначити потреби та вимоги для нової системи.

При проведенні аналізу варто враховувати такі аспекти:

1. Функціональні можливості: Аналогічні програми можуть включати різні функціональні можливості, такі як управління запасами, відстеження постачань, зберігання даних про продукцію, звітність та аналітику. Важливо проаналізувати, які саме функції є важливими для харчової промисловості та як вони реалізовані у наявних програмах.
2. Потужність та масштабованість: Розглядаючи аналогічні системи, слід оцінити їхню потужність та здатність масштабуватись під зростання обсягів даних та потреби підприємства. Деякі програми можуть бути придатними для невеликих підприємств, тоді як інші можуть підтримувати роботу великих компаній зі складними вимогами.
3. Інтеграція: Важливо з'ясувати, наскільки легко програмна система може інтегруватись з існуючими системами в підприємстві, такими як облік фінансів, системи електронного документообігу або CRM-системи. Інтеграція допомагає уникнути подвійного введення даних та забезпечує автоматизацію процесів.

4. Інтерфейс користувача: Важливо оцінити інтерфейс користувача аналогічних програм. Чи є вони зручними, інтуїтивно зрозумілими та легкими у використанні? При розробці нової програмної системи слід враховувати побажання користувачів та надати зручний інтерфейс для ефективної роботи з системою.
5. Супровід та підтримка: Важливо з'ясувати, який рівень супроводу та підтримки надається відповідними програмними системами. Чи є доступна технічна підтримка, оновлення та вдосконалення програмного забезпечення? Це важливо для забезпечення безперебійної роботи системи та вирішення можливих проблем.

Аналіз аналогічних програм, баз даних та систем допоможе отримати об'єктивну оцінку існуючих рішень та визначити переваги та недоліки для подальшої розробки програмної системи складського обліку продукції харчової промисловості. Цей аналіз сприятиме вдосконаленню та впровадженню ефективного та сучасного інструменту у сфері управління складськими процесами, що сприятиме покращенню продуктивності, зниженню ризиків та оптимізації виробничих процесів в харчовій промисловості.

Обґрунтування рішення створення програмної системи складського обліку продукції в харчовій промисловості базується на декількох ключових аспектах:

1. *Ефективність управління запасами*: У харчовій промисловості має велике значення точне та ефективне управління запасами продукції. Наявність програмної системи складського обліку дозволить забезпечити постійний контроль за кількістю, рухом та якістю продукції на складі. Це сприятиме запобіганню надлишковості або дефіциту запасів, зменшенню втрат та покращенню виробничої потужності.
2. *Точність та надійність обліку*: З урахуванням складності харчової промисловості, важливо мати надійний та точний облік продукції на складі. Ручний облік може призвести до помилок, затримок та невірної

інформації. Використання програмної системи складського обліку дозволить автоматизувати процеси ведення обліку, зменшити ймовірність помилок та забезпечити точність і актуальність даних.

3. *Оптимізація виробничих процесів*: Створення програмної системи складського обліку дозволить оптимізувати виробничі процеси в харчовій промисловості. Автоматизація процесів замовлення, постачання, розміщення товарів на складі та контролю за їх рухом допоможе покращити швидкість, ефективність та точність складських операцій. Це сприятиме зниженню витрат, скороченню часу на обробку замовлень та покращенню загальної продуктивності підприємства.
4. *Забезпечення високої якості продукції*: В харчовій промисловості якість продукції має критичне значення. Використання програмної системи складського обліку дозволить забезпечити контроль якості продукції на різних етапах, починаючи з прийому сировини до виробництва і закінчуючи готовою продукцією на складі. Система може містити функціонал для реєстрації параметрів якості, ведення сертифікатів якості та відстеження дати придатності продукції. Це сприятиме забезпеченню високої якості продукції, довіри споживачів та дотриманню стандартів безпеки харчових продуктів.
5. *Забезпечення відповідності нормативно-правовим вимогам*: У харчовій промисловості існують суворі нормативно-правові вимоги щодо складського обліку продукції, контролю якості та забезпечення безпеки. Створення програмної системи складського обліку дозволить врахувати ці вимоги та забезпечити відповідність усім регулятивним нормам і стандартам. Це допоможе підприємствам уникнути порушень, штрафних санкцій та забезпечити відповідність всім вимогам законодавства.

Отже, створення програмної системи складського обліку продукції в харчовій промисловості має велику актуальність і обґрунтування. Вона допоможе забезпечити ефективне управління запасами, точний облік та надійну

базу даних, оптимізувати виробничі процеси, забезпечити високу якість продукції та відповідність нормативно-правовим вимогам.

При виборі методів рішення створення програмної системи складського обліку продукції в харчовій промисловості слід враховувати наступні аспекти:

1. *Аналіз потреб і вимог*: Спочатку необхідно провести детальний аналіз потреб і вимог підприємства щодо складського обліку продукції. Це включає визначення функціональних вимог, обсягу даних, інтеграцію з існуючими системами, доступ до інформації тощо. Аналіз допоможе визначити ключові функції та можливості програмної системи.
2. *Вибір готових рішень або розробка на замовлення*: Після аналізу можна розглянути можливість використання готових програмних рішень, які вже наявні на ринку та відповідають потребам харчової промисловості. Це можуть бути спеціалізовані системи складського обліку або модулі в складських системах управління. Іншим варіантом є розробка програмної системи на замовлення, що повністю задовольнятиме унікальні потреби підприємства.
3. *Технічні вимоги та інфраструктура*: Важливо врахувати технічні вимоги та інфраструктуру, які необхідні для реалізації програмної системи. Це включає вибір платформи (операційної системи, бази даних), вимоги до обладнання, мережевої інфраструктури та безпеки даних. Забезпечення сумісності інфраструктури з обраною програмною системою є важливим кроком у реалізації проекту.
4. *Оцінка вартості та вигоди*: Потрібно оцінити вартість впровадження програмної системи, включаючи витрати на розробку програмного забезпечення, придбання обладнання, навчання персоналу та підтримку системи. Слід порівняти ці витрати з очікуваними вигодами, які можуть бути отримані від впровадження системи складського обліку. Серед можливих вигод можуть бути зниження витрат на управління запасами, оптимізація процесів, покращення контролю якості та надійності, збільшення продуктивності та задоволення потреб клієнтів.

Ця оцінка допоможе визначити економічну доцільність впровадження програмної системи.

5. *Ризики та заходи по їх зменшенню*: Треба розглянути можливі ризики, які можуть виникнути під час реалізації та експлуатації програмної системи складського обліку. Це можуть бути проблеми з інтеграцією з існуючими системами, неправильна оцінка потреб, недостатня підготовка персоналу та інші.
6. *План впровадження та підтримки*: Розробка детального плану впровадження програмної системи складського обліку, включаючи етапи, ресурси, відповідальних осіб та терміни. Вирахування плану підготовки персоналу, міграцію даних, тестування системи та поетапне впровадження. Крім того, потрібно розробити план підтримки системи після впровадження, включаючи технічну підтримку, оновлення програмного забезпечення та навчання користувачів.

РОЗДІЛ 2
ПРИНЦИПИ ПОБУДОВИ ТА СПЕЦИФІКАЦІЯ ВИМОГ ДО СИСТЕМИ СКЛАДСЬКОГО ОБЛІКУ

2.1 Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання

Кожен бізнес має свої унікальні потреби та процеси, пов'язані з товарообігом. Використання готових програмних рішень може не задовольнити всі потреби клієнта. Розробка програми, спеціально під потреби клієнта, дозволяє налаштувати та врахувати всі специфічні вимоги та процеси, що сприяє більш ефективному та точному вирішенню завдань

При написанні програми товарообігу, використання програми, розробленої спеціально під потреби клієнта, має декілька переваг

1. *Гнучкість та масштабованість*: При написанні програми, до неї ввійдуть виключно функціональні модулі та компоненти, необхідні для конкретного бізнесу та процесу товарообігу. Це сприяє полегшенню управління та розвитку програми, що дає змогу додавати нові функції, адаптувати її до змінних потреб клієнта та масштабувати її в майбутньому.
2. *Інтеграція з існуючими системами*: Багато компаній вже мають наявні системи обліку, складського управління, фінансового обліку тощо. Розробка програми, спеціально під потреби клієнта, дозволяє забезпечити зручну інтеграцію з існуючими системами, що сприяє автоматизації процесів та зменшенню зусиль для обміну даними між різними системами.

Як показують дослідження Кравчук І.[1] основні переваги щодо використання баз даних, включають:

1. *Збереження та доступ до даних*: Бази даних дозволяють ефективно зберігати, організовувати та керувати великим обсягом даних, пов'язаних з товарообігом, такими як самі товари, клієнти, замовлення, складські залишки та інше. Вони забезпечують структуроване

зберігання інформації та швидкий доступ до неї, що дозволяє ефективно опрацьовувати дані та забезпечувати швидкий доступ до необхідної інформації для прийняття рішень. Ефективність використання баз даних в бухгалтерському обліку полягає в тому, що вони забезпечують зручний механізм для організації, зберігання та доступу до великих обсягів фінансових даних. Це дозволяє бухгалтерам та менеджерам швидко отримувати актуальну інформацію, що потрібна для прийняття важливих бізнес-рішень.

2. *Забезпечення цілісності та безпеки даних*: Бази даних дозволяють встановлювати правила цілісності для забезпечення коректності та непорушення даних. Вони також забезпечують можливість налаштування прав доступу до даних, що забезпечує конфіденційність та безпеку інформації.
3. *Запити та аналітика*: Бази даних дозволяють виконувати складні запити та проводити аналітику даних, що допомагає отримувати цінну інформацію про товарообіг, тренди продажів, попит на товари та інші показники. Це дозволяє здійснювати стратегічне планування та приймати обґрунтовані рішення.
4. *Систематизація та автоматизація процесів*: Використання баз даних дозволяє систематизувати процеси товарообігу, зменшити ручну роботу та помилки, та автоматизувати багато рутинних завдань, таких як генерація звітів, облік запасів, обробка замовлень тощо.
5. *Масштабованість та резервне копіювання*: Бази даних можуть бути масштабовані для врахування зростаючих потреб бізнесу. Вони також забезпечують можливість резервного копіювання даних для забезпечення їх безпеки та відновлення в разі виникнення проблем.

Отже програмний додаток розроблено так, щоб дані товарообігу та інші відомості зберігались в базі даних.

Обрання MySQL як бази даних для програмної системи складського обліку продукції в харчовій промисловості може бути обґрунтовано кількома

причинами: Надійність та стабільність: MySQL є однією з найпопулярніших та широко використовуваних систем управління базами даних. Протягом часу вона довела свою надійність, стабільність та високу продуктивність, що робить її привабливим вибором для застосунків, в яких важлива правильність даних та транзакцій, таких як система складського обліку.

Масштабованість: MySQL добре масштабується, що означає, що вона може ефективно працювати з великим обсягом даних і високими навантаженнями. Це особливо важливо для систем складського обліку, оскільки вони мають обробляти велику кількість транзакцій та зберігати значну кількість даних про запаси, рух товарів і замовлення. Підтримка мови SQL: MySQL базується на мові запитів SQL, яка є стандартом для роботи з реляційними базами даних. Це дозволяє розробникам легко створювати складні запити, здійснювати пошук, сортування та фільтрацію даних, що є необхідними функціями для системи складського обліку.

Відкритий код та спільнота розробників: MySQL є БД з відкритим кодом, що означає, що його вихідний код доступний для всіх і може бути модифікований та пристосований до потреб конкретного проекту. Крім того, MySQL має велику спільноту розробників, яка активно підтримує та розвиває цю систему, надаючи оновлення.

2.2 Функціонально-структурна схема роботи об‘єкта проектування. (UML діаграми)

UML (Unified Modeling Language) є стандартною мовою моделювання, яка широко використовується при створенні проектів, включаючи програми товарообігу. Важливість використання UML діаграми включає наступні аспекти:

1. *Визначення вимог*: UML діаграми допомагають уточнити та визначити вимоги до програмного забезпечення. Вони дозволяють аналізувати бізнес-процеси, взаємодію з користувачами та системні вимоги, що

допомагає зрозуміти, які функції повинна виконувати програма товарообігу та як вона має бути налаштована.

2. *Візуалізація архітектури*: UML діаграми допомагають візуалізувати архітектуру програми товарообігу. Вони дозволяють структурувати компоненти системи, взаємодію між ними, їхні залежності та інтерфейси. Це сприяє кращому розумінню системи та допомагає розробникам вирішувати проблеми архітектури на ранніх етапах проекту.
3. *Проектування бази даних*: UML діаграми, такі як діаграми класів і діаграми послідовностей, можуть бути використані для проектування структури бази даних для програми товарообігу. Вони дозволяють визначити таблиці, взаємозв'язки між ними, атрибути та типи даних, що допомагає розробникам ефективно створювати та керувати базою даних.
4. *Комунікація та співпраця*: UML діаграми служать засобом комунікації між розробниками, бізнес-аналітиками та іншими учасниками проекту. Вони надають спільне розуміння системи.
5. *Виявлення потенційних проблем*: UML діаграми дозволяють виявити потенційні проблеми або неузгодженості в проекті програми на ранніх етапах розробки. Наприклад, діаграми послідовностей можуть допомогти ідентифікувати можливі проблеми з послідовністю дій або взаємодією між компонентами системи. Це дає змогу виправити проблеми ще до початку реалізації програми, що зменшує витрати часу та зусиль на подальшу розробку.
6. *Документація та підтримка:* UML діаграми служать як документація для проекту програми товарообігу. Вони допомагають зберігати та передавати знання про систему між розробниками, сприяючи підтримці та розумінню коду на протязі всього життєвого циклу проекту. Крім того, вони дозволяють швидко оновлювати або модифікувати систему, орієнтуючись на існуючі діаграми.

7. *Підтримка проектного управління*: UML діаграми можуть бути використані для планування та керування проектом програми товарообігу. Наприклад, діаграма графіку Ганта може візуалізувати послідовність та тривалість завдань, а діаграми прецедентів допомагають визначити функціональність та вимоги до системи. Це дозволяє краще організувати робочий процес, розподілити завдання та забезпечити виконання проекту вчасно та ефективно.

Спираючись на цю інформацію, можна побудувати UML діаграму застосунку та взаємодії з ним:

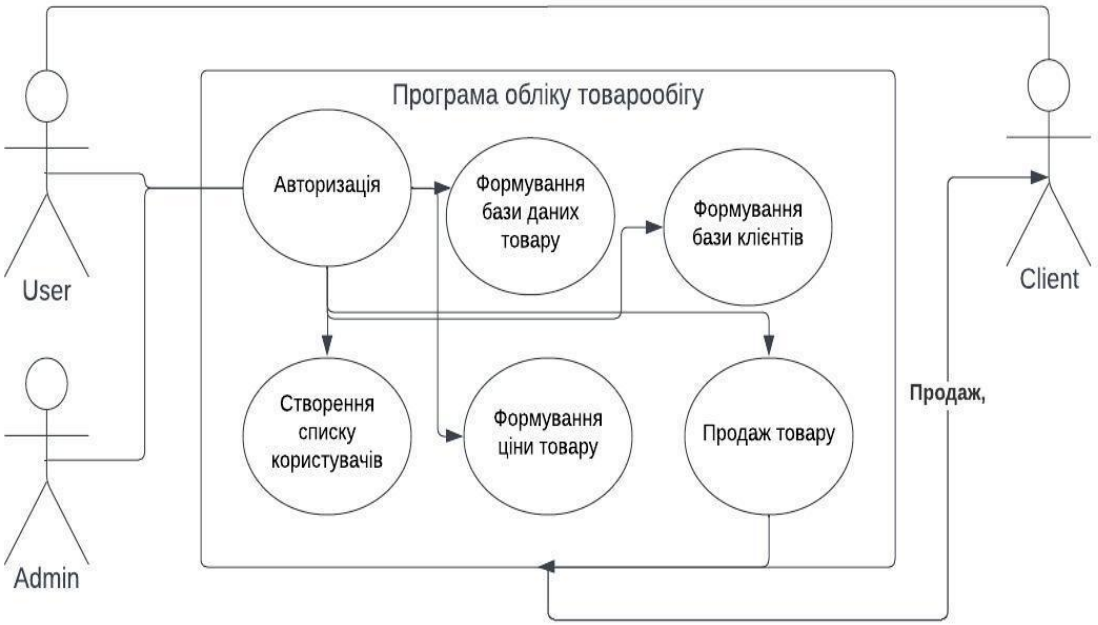

Рис. 2.1. UML діаграма системи

2.3 Стандарті методи прогнозування продажів

Прогнозування продажів є важливим процесом для багатьох компаній, оскільки воно допомагає передбачити майбутні обсяги збуту і розробити ефективні стратегії управління запасами, маркетингу та виробництва. Існує кілька стандартних методів прогнозування продажів, які можуть бути використані для цієї мети. Деякі з них включають наступне:

1. **Метод середнього значення**: Цей метод полягає в обчисленні середнього значення попередніх продажів і використанні його як прогнозуваного значення

на майбутній період. Це простий і швидкий метод, але не враховує сезонність або тренди в даних.

2. **Експоненційне згладжування**: Цей метод використовує зважене середнє значення попередніх продажів з більшою вагою недавніх даних. Він приділяє більше уваги останнім спостереженням і дозволяє виявляти тренди та зміни у споживачів.

3. **Метод часових рядів**: Цей метод використовує попередніх дані продажів для виявлення сезонності, трендів та інших регулярних патернів у часовому ряді. На основі цих виявлень можна зробити прогноз на майбутній період.

4. **Кореляційний аналіз**: дозволяє встановити залежність між двома змінними, наприклад між продажами і ціною товару, рівнем реклами або іншими факторами. Він вимірює ступінь лінійної залежності між змінними за допомогою коефіцієнта кореляції. Це допомагає виявити, які фактори можуть мати вплив на продажі і яким чином.

Кореляційний аналіз може бути використаний для виявлення залежності між продажами одного товару від іншого. Це може бути корисним для розуміння, які товари взаємодіють між собою та які фактори можуть впливати на продажі.

Наприклад, якщо у вас є дані про продажі двох товарів, наприклад, кави та печива, ви можете застосувати кореляційний аналіз, щоб встановити, чи існує статистично значима залежність між продажами цих товарів. Якщо кореляційний аналіз показує позитивну кореляцію, це означає, що збільшення продажів одного товару супроводжується збільшенням продажів іншого товару. Зворотньо, негативна кореляція вказує на протилежну залежність, коли збільшення продажів одного товару супроводжується зменшенням продажів іншого товару.

Це може бути корисним для маркетингових стратегій, управління запасами та розміщення товарів у магазинах. "Під час аналізу даних про продажі важливо не тільки виявити взаємозв'язок, але й зрозуміти, як ця інформація може бути використана для оптимізації бізнес-процесів."[2]

Наприклад, якщо ви виявите позитивну кореляцію між продажами кави та печива, ви можете розмістити їх поруч один з одним на полицях магазину або запропонувати спеціальну пропозицію на обидва товари для стимулювання спільних покупок.

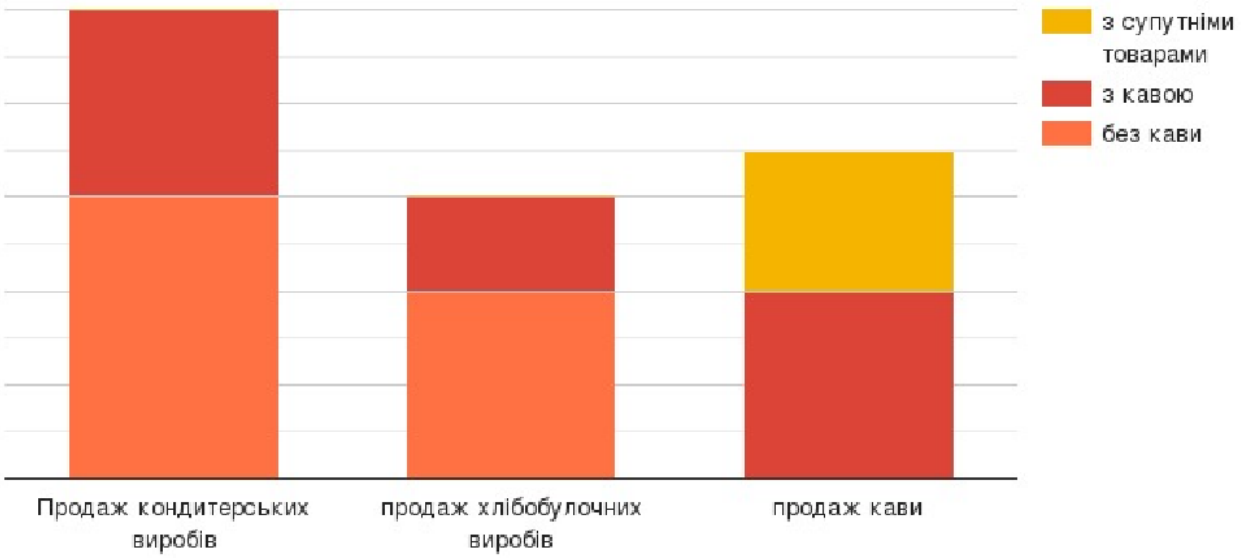

Рис. 2.2. Вплив різних товарів на продаж

На рисунку можна бачити що розміщення кавової машини збільшує продажі кондитерських виробів. Це приклад кореляційної залежності.

Але це досить тривіальний наочний приклад. Якщо асортимент товарів для аналізу достатньо великий, можуть з'являтися досить несподівані кореляційні зв'язки, як-то поява вугілля для шашлику на заправочних станціях.

Кореляція також може бути негативна. Негативна кореляція може свідчити про наявність, наприклад, товарів – конкурентів. Таким чином за умови обмеженості місця на вітрині зайві товари можуть бути виключені з неї.

2.4 Кореляційний аналіз продажів

В роботі будемо використовувати кореляційний аналіз Пірсона, також відомий як коефіцієнт кореляції Пірсона, є одним з найпоширеніших методів вимірювання ступеня лінійної залежності між двома неперервними змінними. Він використовується для виявлення та вимірювання сили та напрямку залежності між цими змінними.

Коефіцієнт кореляції Пірсона показує, наскільки близькі лінійні відносини між двома змінними. Він приймає значення в межах -1 до +1, де:

Значення +1 означає досконалу позитивну лінійну залежність, тобто коли зростання однієї змінної супроводжується зростанням іншої змінної з постійним коефіцієнтом пропорційності.

Значення -1 означає досконалу негативну лінійну залежність, тобто коли зростання однієї змінної супроводжується спаданням іншої змінної з постійним коефіцієнтом пропорційності.

Значення 0 означає відсутність лінійної залежності між змінними.

$$r_{xy} = \frac{\sum (x_i - M_x)(y_i - M_y)}{\sqrt{\sum (x_i - M_x)^2 \cdot (y_i - M_y)^2}}$$

Де r - коефіцієнт кореляції Пірсона

x_i – значення змінної x;

y_i – значення змінної y;

M_x – средне значення x;

M_y – средне значення y.

Проведемо моделювання продажів за певний період часу таким чином щоб досягти різних значень коефіцієнта кореляції

Дата	Товар 1	Товар 2
01.05.23	1	10
02.05.23	1	10
03.05.23	2	15
04.05.23	2	15
05.05.23	3	20
06.05.23	4	25
07.05.23	5	25
08.05.23	6	30
09.05.23	7	35
10.05.23	6	30
11.05.23	5	25
12.05.23	4	20
13.05.23	3	15
14.05.23	2	10
15.05.23	1	10
	Кореляція	0,976056

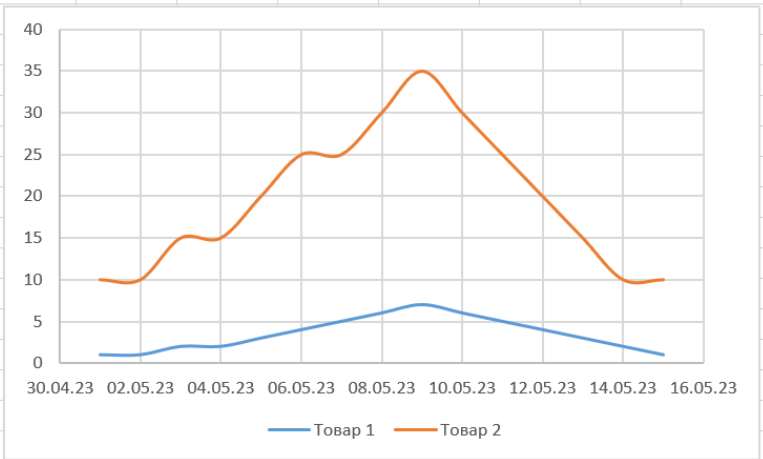

Рис. 2.3. Моделі продажу з великим коефіцієнтом кореляції

Можна чітко бачити, що чим більше продаж товару 1 тим більший продаж товару 2. Спробуємо негативну кореляцію

Дата	Товар 1	Товар 2
1.5	1	30
2.5	1	30
3.5	2	20
4.5	2	20
5.5	3	20
6.5	4	15
7.5	5	15
8.5	6	10
9.5	7	10
10.5	6	10
11.5	5	15
12.5	4	20
13.5	3	30
14.5	2	10
15.5	1	10
	Кореляція	-0,56434

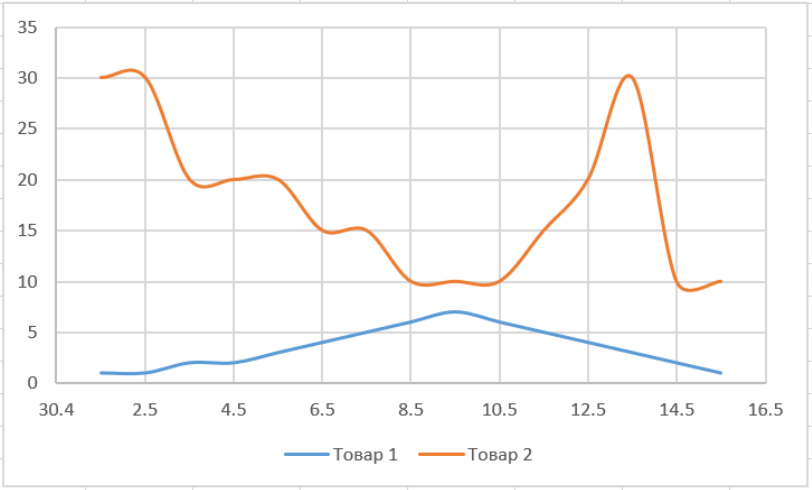

Рис. 2.4. Моделі продажу з великим коефіцієнтом зворотної кореляції

	Товар 1	Товар 2
1.5	1	10
2.5	1	20
3.5	2	15
4.5	2	15
5.5	3	20
6.5	4	10
7.5	5	25
8.5	6	30
9.5	7	10
10.5	6	30
11.5	5	10
12.5	4	5
13.5	3	15
14.5	2	30
15.5	1	10
	Кореляція	0,195832

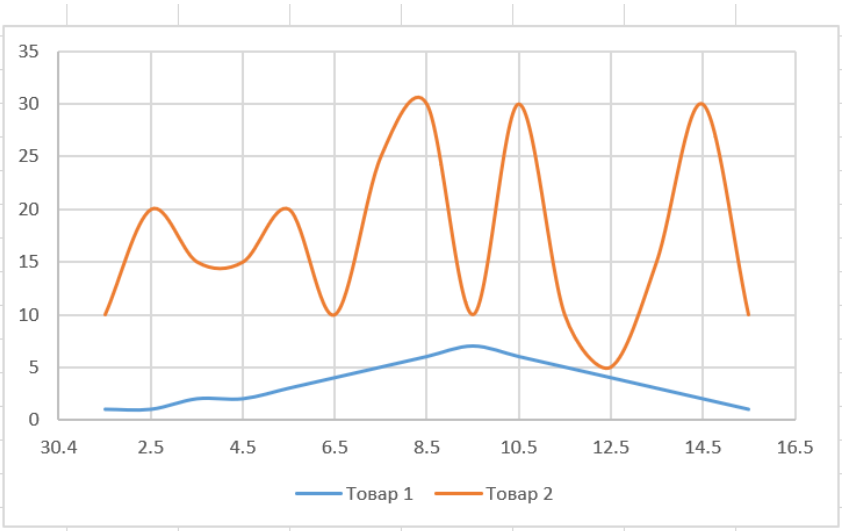

Рис. 2.5. Моделі продажу з відсутньою кореляцією

Важливо зауважити, що кореляційний аналіз Пірсона вимірює лише лінійну залежність між змінними. Інші види залежності, такі як нелінійні або неявні залежності, не будуть виявлені цим методом.

Варто відмітити що кореляція вказує лише на статистичну залежність. Як зазначає Степаненко О.: "Хоча кореляційний аналіз може бути потужним

інструментом для виявлення зв'язків між змінними, необхідно пам'ятати, що кореляція не означає причинно-наслідкового зв'язку."[2]

Так для моделі, в якій показана велика кореляція продажі можуть бути викликані сезонними продажами, або переміщенням до торгових об’єктів великої кількості людей, наприклад евакуаціями із зони бойових дій.

Там де кореляція від’ємна, приклад досить простий – коли продаються зонти – не продаються сонцезахисні окуляри. Отже, незважаючи на кореляційний взаємозв'язок, для підтвердження причинно-наслідкового зв'язку необхідне більш глибоке та комплексне статистичне дослідження, тому будемо використовувати даний тип аналізу з урахуванням здорового маркетинг-глузду.

2.5 Постановка завдання на кваліфікаційну роботу

1. Провести аналіз поточного стану систем складського обліку для підприємств харчової промисловості, визначити основні проблеми та недоліки, з якими стикаються підприємства.
2. Визначити функціональні та технічні вимоги до програмної системи складського обліку, зокрема вимоги до управління запасами, контролю якості, оптимізації виробничих процесів та забезпечення безпеки даних.
3. Розробити концепцію програмної системи складського обліку, включаючи структуру бази даних, функціональні модулі та взаємодію з іншими системами управління.
4. Розробити архітектуру програмної системи, визначити технології та інструменти, які будуть використані для реалізації системи.
5. Розробити та реалізувати програмний код програмної системи складського обліку, включаючи модулі управління запасами, контролю якості, звітності та інші необхідні функції.
6. Провести тестування розробленої програмної системи, виявити та виправити помилки та недоліки.

7. Впровадити розроблену програмну систему складського обліку на підприємстві, забезпечити інтеграцію з існуючими системами управління та підготовку персоналу.
8. Здійснити оцінку ефективності впровадженої програмної системи, порівняти результати зі звичайним методом складського обліку та оцінити отримані вигоди.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ СКЛАДСЬКОГО ОБЛІКУ

Для початку розробки перш за все потрібно визначити основну структуру програми. Після визначення основної структури програми, можна переходити до розробки алгоритмів та функціональності. Важливим етапом є також вибір мови програмування та інструментів, які будуть використовуватись під час розробки. Крім того, необхідно ретельно тестувати програму на різних етапах розробки для виявлення можливих помилок та удосконалення її роботи.

3.1 Практична реалізація об’єкта проектування. Побудова блок-схем модулів програми

Програми обліку товару як правило призначені для продажу, та зазвичай доповнюються декількома додатковими функціями. Це може бути ведення бази клієнтів, сервіси для використання касового обладнання та елементи адміністрування. До елементів адміністрування у системі також відноситься додавання співробітників та розрахунок їх заробітної плати, адже вона може нараховується як певний відсоток від продажів.

Структура додатку наведена на Рис. 3.1.

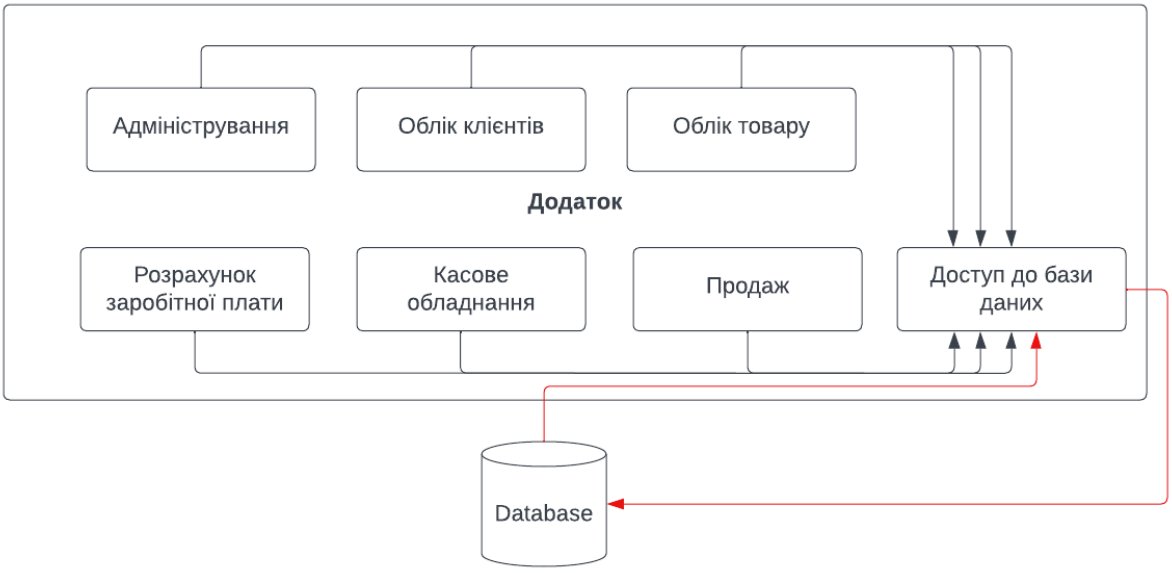

Рис. 3.1. Програмні модулі додатка, що розробляється

Додаткові функції програм обліку товарів можуть також включати можливість створення та відстеження замовлень, управління запасами товарів, аналітику продажів та складу, а також розрахунок податків та інших фінансових витрат. Результатом використання таких програм може стати більш ефективне ведення бізнесу, що сприятиме збільшенню прибутку та покращенню взаємодії з клієнтами та співробітниками. Отже у додатку є потенціал для розширення у повноцінну CRM систему у майбутньому.

3.1.1 Реалізація моделі складського обліку товару

Існують багато моделей обліку товару та його продажу. На мою думку найбільш вдала – облік за початковими та кінцевими залишками.

Вона полягає в тому, що при веденні обліку товару розглядаються його залишки на початку та в кінці певного періоду, наприклад, місяця. Початковий залишок - це кількість товару, яка була на складі на початку періоду, і яка залишилась з попереднього періоду. Кінцевий залишок - це кількість товару, яка залишається на складі в кінці періоду, після всіх операцій купівлі, продажу та інших змін.

Така модель дозволяє більш точно відслідковувати рух товару на складі, контролювати витрати на закупівлю та продаж товару, а також здійснювати прогнозування попиту на товар і планувати закупівлю відповідно до цього. Крім того, така модель є досить простою та зручною у використанні, тому вона широко застосовується в бізнесі.

Розглянемо модель на конкретному прикладі

Для її реалізації можна використати одну таблицю, але для використання SQL запитів доречно використати дві таблиці. Отже перша таблиця - це перелік товару.

Товар	Ціна	Початковий	Кількість	Кінцевий	Реалізація

		залишок	руху	залишок	
Товар 1	100	5	10	15	2000
Товар 2	200	1	1	10	2000

Таблиця 3.1. Модель обліку

Реалізація – це різниця між кінцевим залишком та початковим. До реалізації додається кількість руху. Отриманий результат множиться на ціну.

Як вираховується значення “кількості руху” показано в наступному прикладі.

Рух товару				
Товар	Ціна	Клієнт	Кількість руху	Дата
Товар 1	100	Василь	-20	01.05.2023
Товар 1	200	Тарас	30	15.04.2023

Таблиця. 3.2. Рух товару.

У випадку з Товар1 Василь є покупцем а Тарас – постачальником. Загальна кількість руху по даному товари передається в основну таблицю обліку.

Звісно, до товару в разі використання бази даних можуть бути добавлені інші таблиці, які можуть нести інформацію про властивості товару. Наприклад фото, що і буде реалізовано далі.

3.1.2 Реалізація моделі обліку постійних клієнтів

Модель обліку постійних клієнтів включає в себе зберігання та обробку інформації про клієнтів, які регулярно користуються послугами компанії або придбують її товари. Проста модель може включати в себе основну інформацію про клієнтів, таку як їхні контактні дані та історію замовлень, а також можливість створення знижок та акцій для постійних клієнтів. Успішна реалізація моделі обліку постійних клієнтів вимагає додаткових функцій та можливостей, таких як можливість аналізу даних про покупців для розробки індивідуальних пропозицій, програм лояльності та систем бонусів.

Обираємо дуже просту модель. Приклад наведено на рисунку

Клієнт	Номер телефону
Василь	067 332 223 33
Тарас	050 332 223 33

Таблиця. 3.3. Модель обліку клієнтів

Модель проста, але достатня для поточної задачі, тому не має сенсу додатково її ускладнювати. До кожного клієнта також в разі необхідності можна створити додаткові таблиці, наприклад щоб відобразити фото.

3.1.3 Вибір мови програмування

В сучасному світі можна достатньо швидко оволодіти майже будь якою мовою програмування. Але для вибору доцільно подивитись на статистику популярності мов програмування. Результати на рисунку нижче

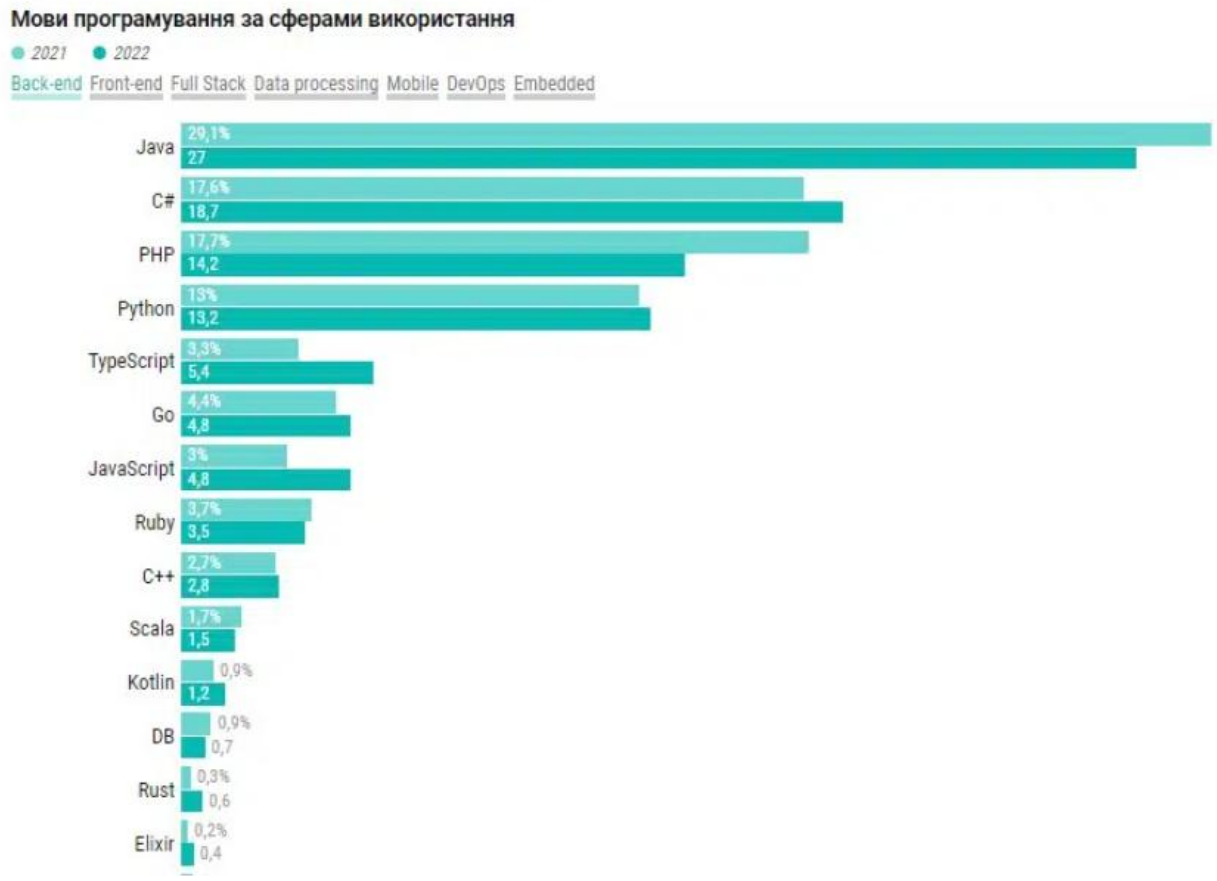

Рис. 3.2. Рейтинг мов програмування.

Дві найпопулярніші мови програмування це C# та Java.

C# - це об'єктно-орієнтована мова програмування. Він був створений у період з 1998 по 2002 рік командою інженерів Microsoft під керівництвом Андерса Хейлсберга та Скотта Вільтаумота. Мова входить у сім'ю C-подібних мов. Синтаксис наближений до Java та C++.

Спираючись на книгу О. Лавріщева[3], можна відокремити такі ключові особливості:

- статистична типізація,
- підтримується поліморфізм,
- підтримується навантаження операторів,
- доступна делегація, атрибути, події, узагальнені типи та анонімні функції.

Мова програмування C# користується високою популярністю, в основному через свою "простоту". Простота, в даному контексті, вказує на зручність використання мови для сучасних програмістів та великих команд розробників, забезпечуючи їм можливість ефективно створювати функціональні та продуктивні програмні продукти в обмежені часові рамки. Це сприяє унікальним конструкціям мови та специфічному синтаксису, що в свою чергу допомагає органічно реалізувати задумані функції.

Популярність мови – ще одна значима перевага. Багато шанувальників C# сприяють його розвитку. Також це сприятливо впливає зростання кількості вакансій, пов'язаних з розробкою мовою Microsoft. Програмісти, добре знайомі з C#, затребувані в індустрії, незважаючи на їх велику кількість, що постійно збільшується.

Зрозумілий синтаксис C# помітно спрощує розробку як таку, а й інші важливі аспекти спільної роботи, наприклад, читання чужого коду. Це спрощує процес рефакторингу та виправлення помилок при роботі над програмами у великих командах.

Також не можна не згадати низький поріг входження. C# - популярна і досить проста в опануванні технологія.

Висновки – обрано C#.

3.1.4 Вибір оптимальної моделі обміну даними між додатком та базою даних

Спираючись на книгу MacDonald M.[4], можна виділити кілька ключових моделей обміну даними між додатками та базою даних:

Модель клієнт-сервер: це найбільш поширена модель взаємодії між додатками та базами даних. У цій моделі додаток виступає в ролі клієнта, який звертається до сервера бази даних для отримання необхідної інформації. Сервер відповідає на запити додатка, виконуючи відповідні операції з даними у БД.

Модель безпосереднього доступу до бази даних: в цій моделі додаток може безпосередньо звертатися до бази даних для отримання інформації. Це означає, що додаток самостійно виконує всі необхідні операції з базою даних. Ця модель може бути корисна в деяких випадках, але не завжди рекомендується, оскільки це може призвести до проблем з безпекою та ефективністю.

Модель доступу до бази даних через сервіси: в цій моделі додаток звертається до сервісів, які надають доступ до бази даних. Ці сервіси можуть надавати різноманітні можливості, такі як кешування даних, резервне копіювання та інші.

Модель доступу до бази даних через API: в цій моделі додаток взаємодіє з базою даних через API, який надається базою даних. Цей підхід може забезпечити більш гнучкий доступ до даних, але він також може бути менш ефективним, ніж інші моделі.

Звісно, що для застосування доступу безпосередньо в програмі можуть бути використані також різні підходи. Основні з них – використання Entity Framework, або використання прямих SQL запитів.

Entity Framework, це підхід ORM (Object-Relational Mapping), який дозволяє зв'язувати об'єктно-орієнтовану модель додатка з реляційною базою даних.

Entity Framework автоматично генерує SQL-запити на основі запитів LINQ, які формулюються в коді додатка, і виконує їх на базі даних. Це дозволяє програмістам працювати з даними як з об'єкті\ами, що є більш зручним для розробки додатків, ніж пряме взаємодію з базою даних.

У підході ORM не потрібно писати складні SQL-запити, оскільки Entity Framework генерує їх автоматично. Це дозволяє зосередитися на розробці функціоналу додатка, а не на роботі з базою даних.

Entity Framework може використовуватися з різними моделями обміну даними між додатками та базою даних, включаючи клієнт-серверну, безпосередню та модель доступу до бази даних через сервіси або API.

Прямі SQL запити, це модель безпосереднього доступу до бази даних. В цій моделі додаток взаємодіє з базою даних безпосередньо, виконуючи запити та отримуючи результати безпосередньо з бази даних, без використання ORM або інших проміжних рівнів.

При використанні прямих SQL запитів можна використовувати класи з простору імен System.Data.Common, такі як DbConnection, DbCommand, DbDataReader тощо. Ці класи надають стандартизований спосіб роботи з реляційними базами даних та можуть використовуватися з будь-якою базою даних, що підтримує стандарт ADO.NET.

Рис. 3.3. Взаємодія з базою даних. Прямі SQL запити та Entity Framework.

На рисунку представлена відмінність підходів. Звісно, що обмін даними це двоспрямований процес, в такому разі відповідь з бази даних обробляється засобами ADO.NET або Entity Framework.

Для використання Entity Framework в програмі потрібно створити суворо типізовану модель класів, яка відповідає структурі в базі даних. Але надалі можна не замислюватись над SQL запитами. В іншому випадку все навпаки - доведеться писати більше коду для формування SQL запитів, але отримаємо можливість легкої зміни структури бази даних, адже не потрібно буде змінювати модель класів.

Існують також інші переваги прямих SQL запитів

1. Простота та прозорість: Використання прямих SQL запитів може бути простішим та прозорішим. Можна написати запит безпосередньо та точно контролювати його виконання.

2. Швидкодія: Прямі SQL запити можуть бути швидшими, особливо якщо використані складні запити, які вимагають великої кількості даних ORM може вносити додаткову накладну витрату на обробку запитів та перетворення даних.

3. Гнучкість: Прямі SQL запити дають більшу гнучкість та контроль над операціями з базою даних. Можна виконувати будь-які SQL запити, включаючи складні операції з об'єднаннями, підзапитами та функціями агрегування.

4. Надійність: При використанні прямих SQL запитів, можна отримати точний результат, якщо виконуєте запити в правильному форматі та з правильними параметрами. ORM може перетворити запит в інші запити, які можуть повернути неочікувані результати.

Існують також недоліки такого підходу. Насамперед це безпека. Використання прямих SQL запитів може бути менш безпечним, оскільки воно може піддати ризику SQL ін'єкцій. Це може стати проблемою, якщо не враховується правильну обробку вхідних даних та не застосовуються параметризовані запити.

Висновок – обираємо прямі SQL запити. Додаток пишеться в навчальних цілях та не буде використовувати параметри в запитах. А під параметричні запити програму в разі її комерційного використання можна легко переробити.

3.1.5 Діаграма класів додатку

Як вже описано вище – не будемо створювати модель класів для Entity Framework. Тому класів буде небагато

Клас DBMySQLUtils та DBUtils відповідають за підключення до бази даних та встановлення необхідних параметрів щодо з’єднання.

Інші класи – головна програма.

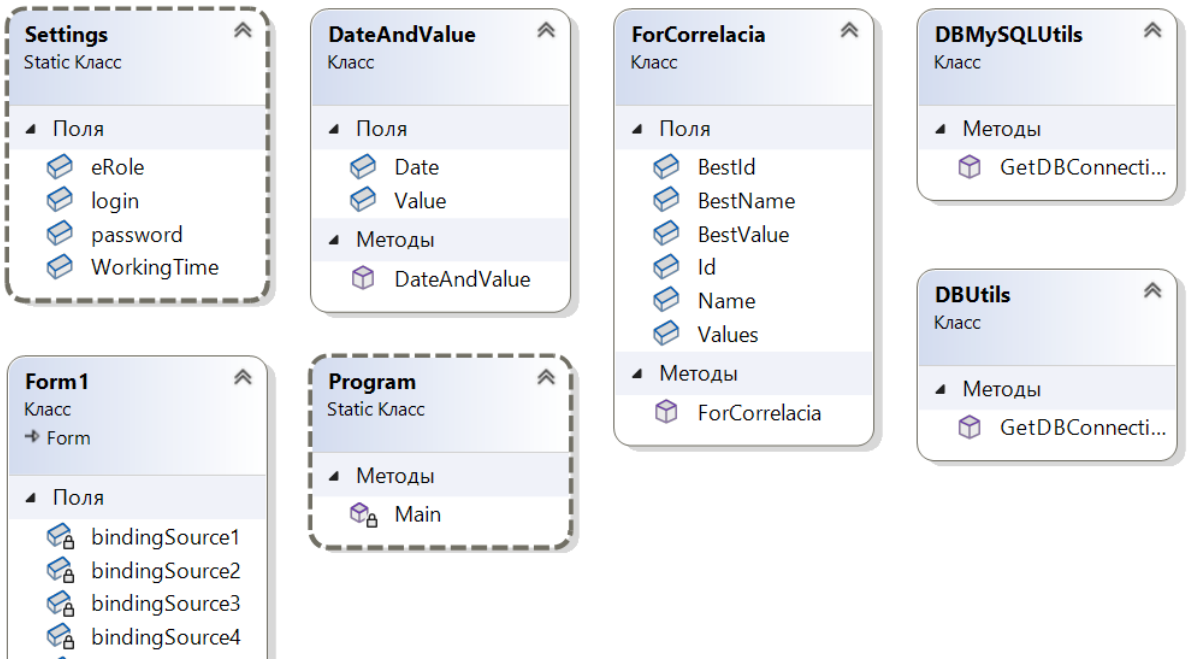

Рис. 3.4. Діаграма класів проекту.

Чим менша кількість класів та складних алгоритмів, тим менше можливостей для помилок та проблем зі сумісністю. Крім того, прості програми зазвичай мають кращу продуктивність та легше піддаюіться змінам та модифікаціям у майбутньому. Програмісти можуть швидко зрозуміти код та внести необхідні зміни без страху порушення роботи системи в цілому.

3.1.6 Створення моделі бази даних

Використання різних систем управління БД може виявитися більш або менш ефективним в залежності від конкретних задач, що постають перед ними, що обумовлено їх внутрішніми характеристиками та специфікою. Є бази даних з відкритим кодом, з можливістю масштабування та іншими перевагами. Таким чином, при виборі БД необхідно враховувати ряд факторів які можуть як сприяти ефективності реалізації задач, так і стати потенційними перешкодами.

Реляційні бази даних

Приклади: MySQL, Oracle DB, PostgreSQL. Це найпопулярніший тип БД, у яких інформація зберігається в таблицях. У рядках інформація, представлена у вигляді записів, що складаються з взаємопов'язаних атрибутів, розміщених у стовпцях, які відображають структуру даних відповідно до моделі, яка визначена схемою бази даних. Таблиці можуть взаємопов'язані.

Реляційна модель проста, але дозволяє виконати безліч різних завдань. Нею зручно користуватися, якщо потрібно зв'язати елементи даних між собою та безпечно та надійно керувати ними. Такі таблиці можна створити для зберігання та обробки телефонних номерів пацієнтів, логінів та паролів користувачів, для відстеження товарних запасів. При цьому БД забезпечує цілісність даних.

У реляційних БД є підтримка SQL, і навіть індексація, що дозволяє швидше шукати необхідні дані. Особливий плюс таких баз - нормалізація даних: вони діляться на різні таблиці, тому виключені повторювані або порожні комірки. Транзакції реляційних БД відповідають ACID - набору властивостей, що гарантує їхню надійну обробку. З мінусів баз можна відзначити відносно низьку швидкість доступу до даних, погану підтримку неструктурованих даних, складність масштабування та утворення великої кількості таблиць, через що важко зрозуміти структуру даних.

Резидентні бази даних

Приклади - Redis, Apache Ignite, Tarantool. Дані зберігаються в оперативній пам'яті. Дані обробляються швидко, тому резидентні БД є популярними там, де потрібно забезпечити максимально короткий час відгуку. Вони допомагають керувати телекомунікаційним обладнанням, проводити торги в онлайн режимі або Real-Time обслуговування. Бази in-memory підтримують і швидкий запис, і швидке читання. В основному вони працюють із записами «ключ-значення», але також можуть працювати зі стовпцями.

Щоб при несподіваному перезавантаженні не втратити дані, потрібно зробити запис із попереднім журналом на енергонезалежному пристрої. Це

можна віднести до мінусів бази in-memory — доводиться вкладатися у дорогі інфраструктурні рішення, щоб забезпечити безперебійне живлення. Також потрібно постійно копіювати інформацію на енергонезалежні носії. Ще один недолік БД - дороге масштабування, оскільки носії оперативної пам’яті дорожчі за енергонезалежні носії.

Пошукові бази даних

Приклад - Elastic. Цей тип БД необхідний отримання відомостей через фільтр. Шукати можна за будь-яким введеним значенням, у тому числі за окремими словами. Можна скористатися повнотекстовим пошуком. Пошукові бази даних добре масштабуються та зручні для зберігання журналів, об'ємних текстових значень.

Можна використовувати пошукові БД для моніторингу оптимізації цін, виявлення помилок у додатку з бронювання квитків та вирішення багатьох інших завдань. У основі можуть зберігатися мільярди документів. Пошук здійснюється швидко. Мінуси системи - погана аналітична підтримка та обмежена можливість застосування БД (можна використовувати лише для пакетних вставок).

Бази даних із широкими стовпцями

Приклади – Cassandra, Google BigTable, HBase. БД з широкими стовпцями можуть вимагати більші обсяги даних швидше, ніж звичайні реляційні. Відомості зберігаються у вигляді записів "ключ-значення" на жорсткому диску або твердому накопичувачі. Бази даних з широкими стовпцями дозволяють виконувати швидкий запис рядковим і швидким читанням по ключу.

БД добре масштабуються та підходять для організації магазинних каталогів, механізмів виявлення шахрайства. Їх зручно використовуватиме управління величезними обсягами інформації на безлічі загальних серверів у розподіленій системі. Недоліками бази даних вважається те, що вона працює у форматі "ключ-значення" і не має підтримки аналітики.

Колоночні бази даних

Приклади – Clickhouse, Vertica. У БД такого типу дані зберігаються у стовпцях, а не в рядках. Доступ до вмісту здійснюється без допомоги ключів. При використанні колоночних БД використовують пакетну вставку, щоб можна було готувати інформацію для швидкого читання стовпчиками. У стовпчастих БД є підтримка аналітики та можливість зручного масштабування.

Такі бази даних використовують там, де потрібно запитувати інформацію щодо певних стовпців, — у системах роздрібного продажу та фінансових транзакцій. Основний мінус колоночних БД полягає в їх обмеженій продуктивності при виконанні складних запитів, що охоплюють багато стовпців одночасно, а не окремо.

Документо-орієнтовані бази даних

Приклади - CouchDB, Couchbase, MongoDB. Якщо в реляційних БД для вилучення даних необхідно об'єднувати таблиці, то таких базах добре зберігається незв'язана інформація у великих обсягах. Такі БД підтримують JSON. Для будь-якого ключа можна створити складне значення і одразу включити всю структуру даних в один запис.

У документо-орієнтованих базах немає прив'язки до схеми. Вони підходять для OLTP (Online Transactional Processing) та підтримують складні типи. Такі БД зручніше використовувати для пошуку документів, або у видавничій справі. Недоліки бази даних - відсутність хорошої аналітичної підтримки та підтримки транзакцій, а також складнощі з масштабуванням.

Графові бази даних

Приклади - OrientDB, Neo4j. Дані зберігаються у вигляді графів, тобто моделей із вузлами та зв'язками. Вони досить гнучкі, з логічною структурою. Вузли служать для зберігання сутностей даних, а ребра - для зберігання взаємозв'язків між сутностями, якими можна керувати.

Графові БД застосовують для вирішення завдань у біоінформації, а також для моделювання соціальних мереж, щоб зберігати взаємопов'язану інформацію про людей. Бази даних такого типу погано піддаються масштабуванню, а другий їх недолік - необхідність використовувати спеціальну мову запитів SPARQL, яка відрізняється від SQL.

Під час розгляду можливості втілення даного проекту виникла думка, що його успішне виконання може послужити підґрунтям для подальшого використання після необхідної модернізації. Оскільки у майбутньому в розробці проекту можуть брати участь кілька розробників, було прийняте рішення використовувати технології, які є найбільш поширеними. Тому було обрано використання системи керування базами даних MySQL.

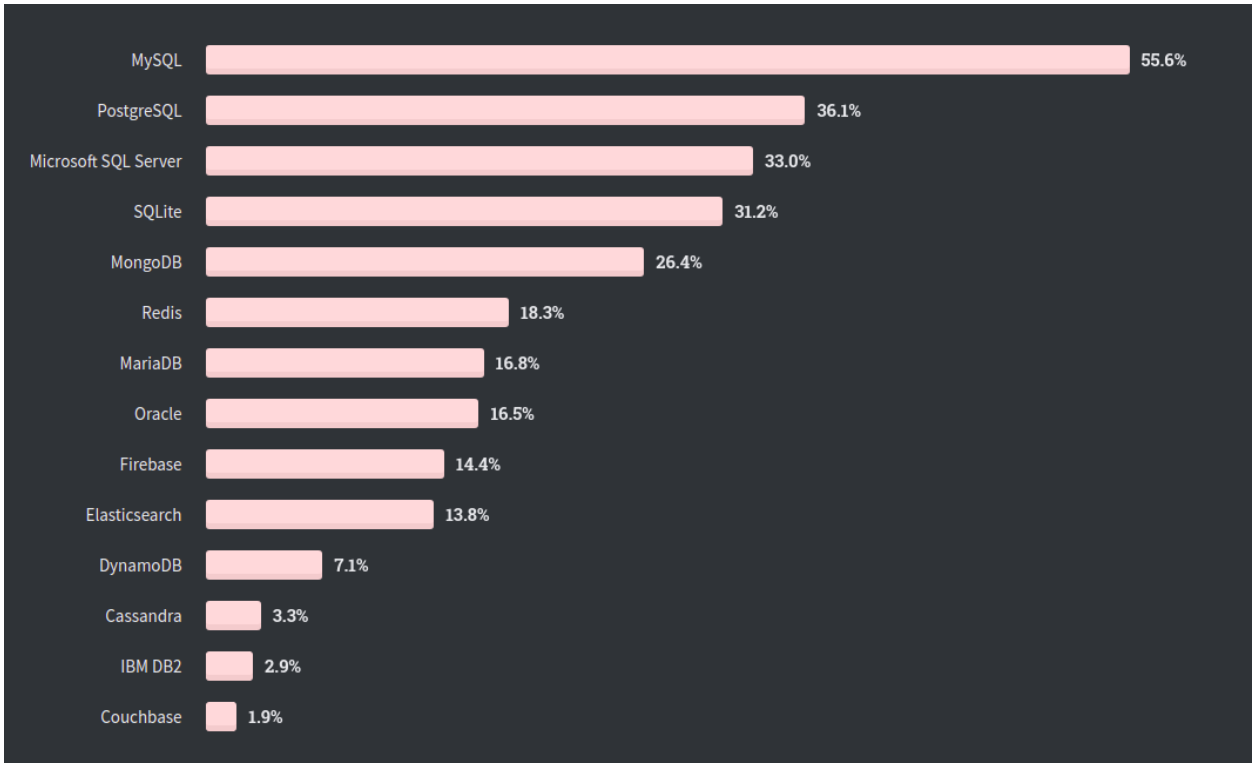

Рис. 3.5. Використання баз даних в 2022 році.

Враховуючи аналіз Russell J. T. Dyer в його книзі[5], варто відзначити наступні ключові переваги MySQL:

- комфортність користування;
- високий ступінь захисту;

- легке масштабування об'ємних даних;
- швидкість

З найвідчутніших недоліків можна визначити деякі обмеження щодо функцій та проблеми з обробкою даних, від яких безпосередньо залежить оптимізація mysql запитів.

База даних MySQL може коректно працювати у наступних напрямках:

- розподілені операції;
- надійний захист інформації;
- розробка веб-сайтів та додатків, оптимізація їх роботи;
- індивідуальні рішення у роботі над нестандартними проектами.

MySql серевер можна безкоштовно завантажити з офіційного сайту. Звісно з разом з ним можна встановити MySQL Workbench.

MySQL Workbench - це кросплатформовий інструмент проектування реляційних баз даних з відкритим вихідним кодом, який додає функціональність та спрощує розробку MySQL та SQL. Він об'єднує проектування, розробку, створення, адміністрування та обслуговування SQL, а також пропонує графічний інтерфейс для структурованої роботи з базами даних.

MySQL Workbench надає можливості для управління моделями баз даних, такими як:

- Створення графічної моделі
- Зворотній інжиніринг живих баз даних у моделі (моделювання даних)
- Пряма інженерна модель у скрипт/живу базу даних і більше.

Скрипт для створення бази даних (додаток Б) містить команди для створення таблиць та їх структур. База даних містить наступні таблиці:

1. Таблиця clients містить інформацію про клієнтів, їх ідентифікатор, назву та номер телефону. Ідентифікатор є унікальним ключем для кожного клієнта, а назва є унікальним ключем для кожного запису.
2. Таблиця employees містить інформацію про співробітників, їх ідентифікатор, ПІБ, посаду, логін та пароль. Ідентифікатор є унікальним ключем для кожного співробітника.
3. Таблиця images містить інформацію про зображення товарів, їх ідентифікатор, назву товару та зображення. Ідентифікатор є унікальним ключем для кожного зображення, а назва товару є унікальним ключем для кожного запису. Таблиця містить зовнішній ключ на таблицю tovar, що вказує на зв'язок між зображенням та товаром.
4. Таблиця move містить інформацію про рух товару, їх ідентифікатор, назву товару, дату переміщення, прихід, розподіл, назву клієнта, ціну приходу та роздрібну ціну, кількість, працівника, який здійснив продаж, дату продажу та тип товару. Ідентифікатор є унікальним ключем для кожного руху товару. Таблиця містить зовнішні ключі на таблиці clients та tovar, що вказують на зв'язок між рухом товару та клієнтом/товаром.
5. Таблиця photos містить інформацію про фотографії клієнтів, їх ідентифікатор, назву клієнта та фотографію. Ідентифікатор є унікальним ключем для кожної фотографії, а назва клієнта є унікальним ключем для кожного запису.

Створення зв'язків між таблицями надає можливість реалізації різноманітних та практично значимих функцій для бази даних. Наприклад, можна створити зв'язок між таблицями "move" та "tovar", щоб відслідковувати рух товарів і знаходити відповідні товари в таблиці "tovar". Також можна використовувати зв'язки для забезпечення цілісності даних в базі. Наприклад, забороняти видаляти клієнтів, які мають рухи товарів в таблиці "move", або не дозволяти вставляти записи в таблицю "move", якщо товар з такою назвою не існує в таблиці "tovar".

Схема створеної БД наведена на Рисунку.

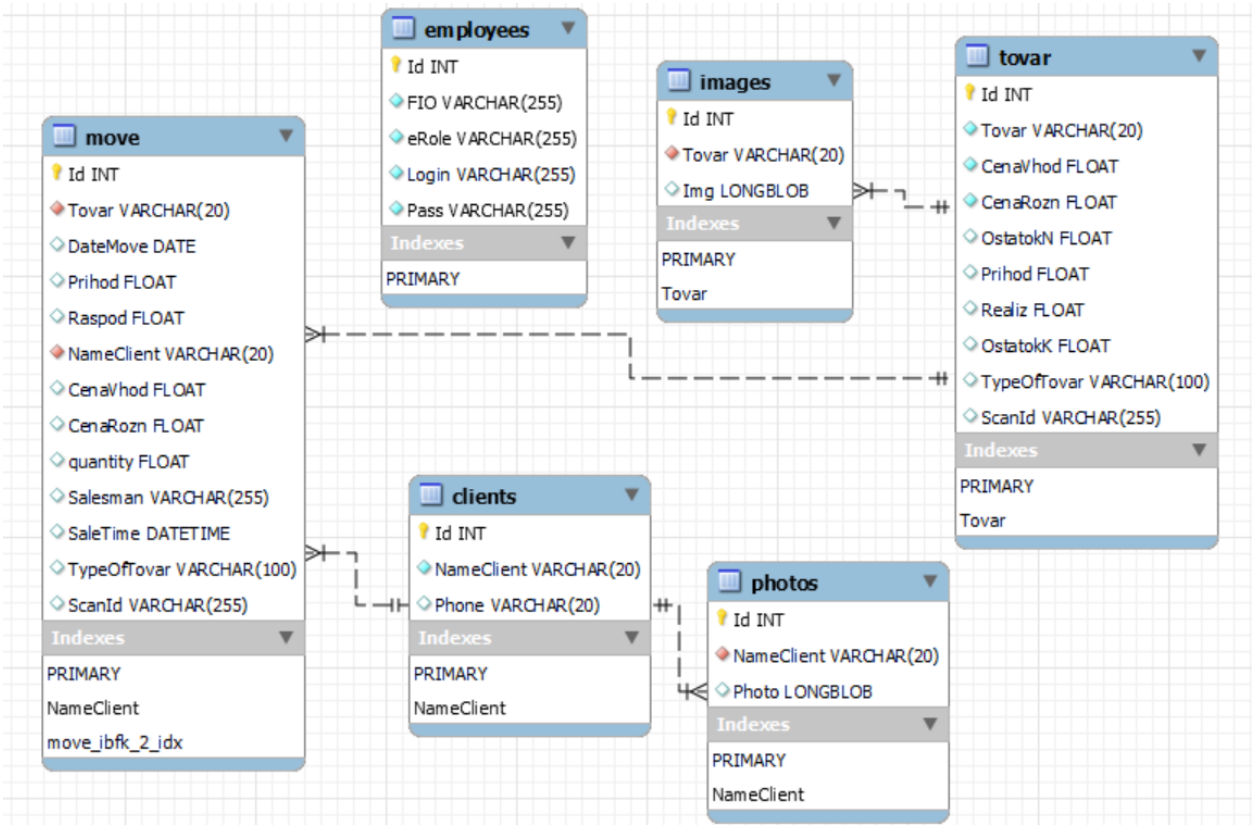

Рис. 3.6. ER діаграма бази даних

Таблиця співробітників не має зав’язків. Співробітники не мають відношення в даній моделі до обліку товару. В цій таблиці містяться імена та паролі для доступу до додатку.

Слід додати, що звязки між таблицями можливо взагалі не використовувати. Однак в такому випадку забезпечення цілісності даних покладається на програму. Можна використати транзакції.

Транзакції не можуть повністю замінити зв'язки між таблицями, але в деяких випадках вони можуть бути корисні для забезпечення консистентності даних. Транзакція - це група дій, які виконуються як одна атомарна операція, тобто виконання всіх дій у групі гарантується або жодна з них не виконується. Це означає, що якщо транзакція не може бути успішно завершена (наприклад, через помилку в операції), то всі зміни, зроблені в рамках транзакції, будуть автоматично відмінені.

Таким чином, використання транзакцій може бути корисним, коли маємо кілька запитів до бази даних, які повинні бути виконані як одна атомарна

операція. Наприклад, якщо ми хочемо додати новий запис у таблицю зав'язків між клієнтами та товаром, а також змінити залишок товару в таблиці товарів, то можна використати транзакцію, щоб забезпечити, що обидва запити виконаються як одна операція, і щоб уникнути проблем з консистентністю даних.

3.1.7 Особливості підключення та роботи з базою даних

З огляду на те, що система повинна надавати можливості роботи не тільки на локальному комп'ютері, але й в мережі – було створено окремий текстовий файл з налаштуваннями.

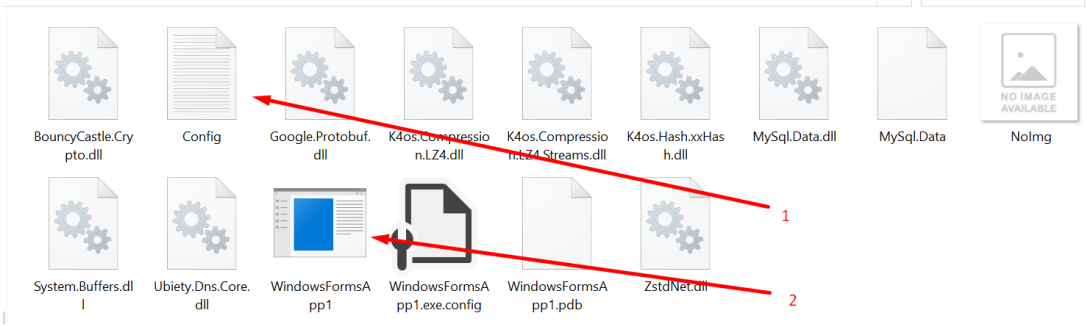

Рис. 3.7. Розташування файлу з налаштуваннями (1) відносно додатку (2)

Файл має такий зміст

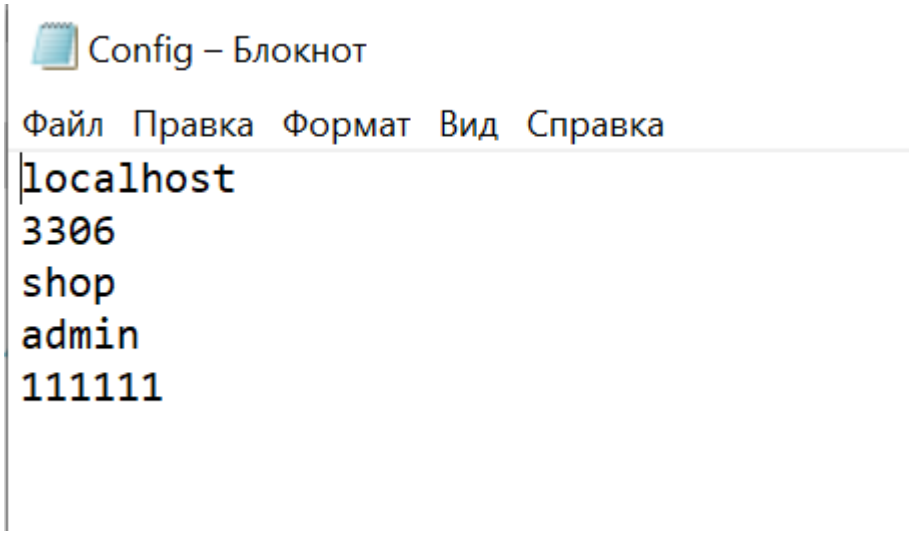

Рис. 3.8. Зміст файлу налаштувань

Зміст файлу налаштувань по рядках: IP-адреса SQL сервера, порт SQL сервера, назва бази даних, логін, пароль.

При завантажені програми спочатку проводитися зчитування налаштувань, та передача їх в класи для створення з’єднання

```
private void Form1_Load(object sender, EventArgs e)
{
    ReadConfig();
    conn = DBUtils.GetDBConnection(textBox2.Text,Convert.ToInt32(textBox3.Text), textBox4.Text, textBox5.Text, textBox6.Text);

    try
    {

        conn.Open();
        MessageBox.Show("Connection successful!");
```

Рис. 3.9. Завантаження налаштувань та передача в класи створення з’єднання

```
void ReadConfig()
{
    string path = System.IO.Path.Combine(Environment.CurrentDirectory,"Config.txt");


    using (StreamReader reader = new StreamReader(path))
    {
        string line;
        line = reader.ReadLine();
        textBox2.Text = line;
        line = reader.ReadLine();
        textBox3.Text = line;
        line = reader.ReadLine();
        textBox4.Text = line;
        line = reader.ReadLine();
        textBox5.Text = line;
        line = reader.ReadLine();
        textBox6.Text = line;

    }
}
```

Рис. 3.10. Зчитування налаштувань

```
namespace Tutorial.SqlConn
{
    class DBUtils
    {
        public static MySqlConnection GetDBConnection(string h,int p,string db,string un,string pass)
        {
            string host = h;
            int port = p;
            string database = db;
            string username = un;
            string password = pass;

            return DBMySQLUtils.GetDBConnection(host, port, database, username, password);
        }

    }
}
```

Рис. 3.11. Клас DBUtils

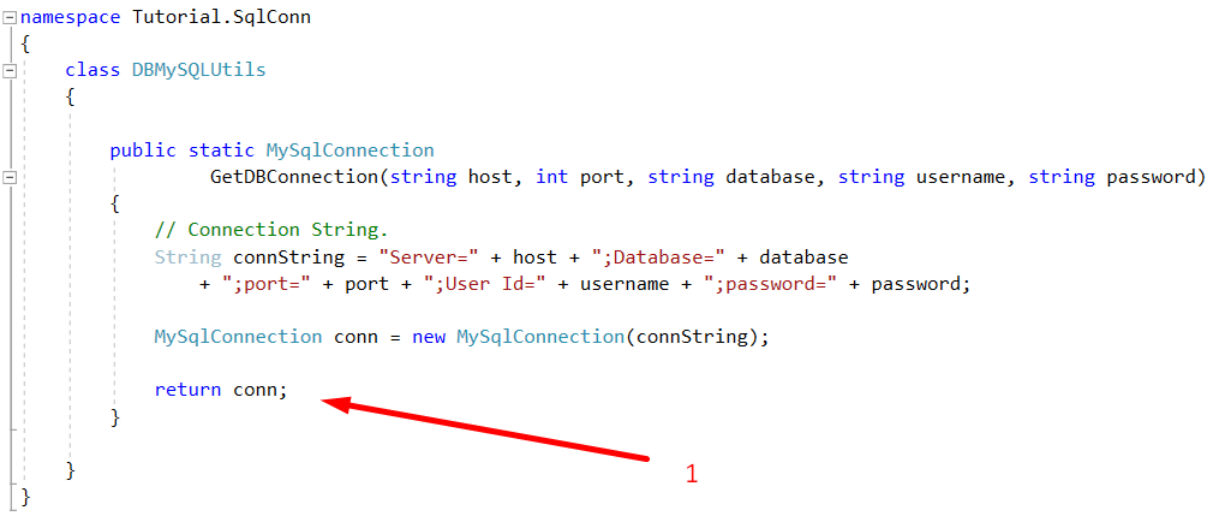

Рис. 3.12. Клас DBMySQLUtils повертає з'єднання (1)

В подальшому це з'єднання використовується для будь якої операції з даними

```
void FillEmployeesForZP()
{
    cmd = conn.CreateCommand();
    cmd.CommandText = "Select login from Employees";
    using (DbDataReader reader = cmd.ExecuteReader())
    {
        if (reader.HasRows)
        {
            comboBox1.Items.Clear();
            while (reader.Read())
            {
                comboBox1.Items.Add(reader[0].ToString());
            }
        }
    }
    cmd.Dispose();
}
```

Рис. 3.13. Приклад отримання даних за допомоги прямого SQL запиту.

Для відображення, а також редагування даних було обрано компонент dataGridView. Він представлений на рисунку.

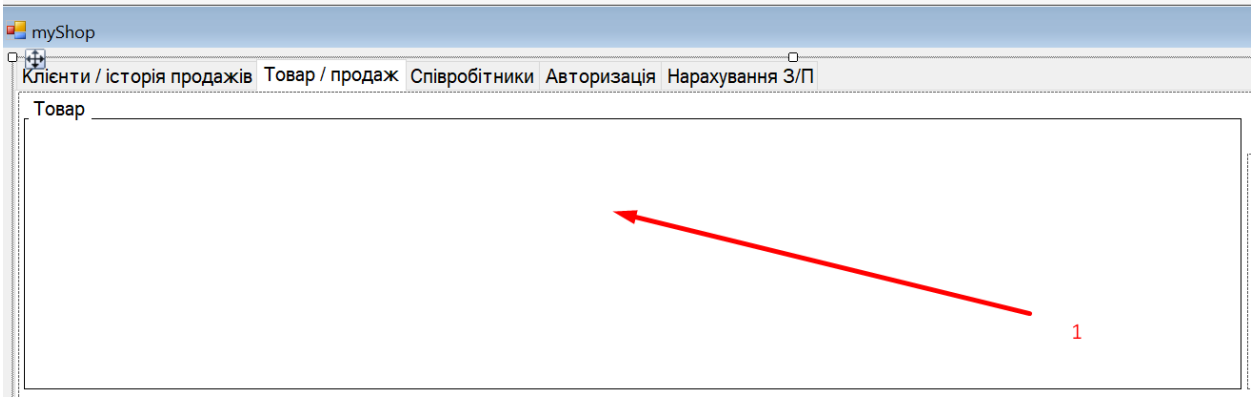

Рис. 3.14. використання елементу керування dataGridView.

Приклад коду, що заповнює даний елемент наведено нижчє.

```
void FillEmployees()
{ //процедура заполнения користувачів
    triger = 0;
    mydtadp4.SelectCommand = new MySqlCommand("select * from Employees", conn);
    cmbl = new MySqlCommandBuilder(mydtadp4);
    DataTable table = new DataTable();
    mydtadp4.Fill(table);
    bindingSource4.DataSource = table;
    dataGridView6.DataSource = bindingSource4;
    dataGridView6.Columns[0].Width = 0;
    dataGridView6.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.DisplayedCells;
    dataGridView6.GridColor = Color.Brown;

}
```

Рисунок Приклад заповнення елементу керування dataGridView.

Попершу проводиться створення об'єкту MySqlCommand, який містить SQL-запит "select * from Employees" для отримання всіх записів з таблиці "Employees" у підключенні до бази даних, яке передається як другий параметр. Потім створюється об'єкт MySqlCommandBuilder, який забезпечує автоматичне генерування команд SQL INSERT, UPDATE та DELETE, необхідних для збереження змін в даних з DataGridView в базі даних.

Далі створюється об'єкт DataTable, який буде використовуватись для зберігання даних з бази даних. Метод Fill об'єкту DataAdapter (mydtadp4) виконує запит до бази даних та заповнює DataTable (table) результатами.

Особливої уваги заслуговує в даному подході об'єкт MySqlCommandBuilder. Фактично, якщо провести необхідні налаштування користувач може змінювати данні, якими заповнено таблицю. Ці зміни поширюються на базу даних.

Таким чином даний об’єкт підтримує повноцінне редагування даних, і в програмі передбачено збереження змін (в БД)

```
private void button6_Click(object sender, EventArgs e)
{
    mydtadp.Update((DataTable)bindingSource1.DataSource);

    MessageBox.Show("SAVED");
}
```

Рис. 3.15. Збереження даних

Слід додати важливий момент – з огляду на зв’язки в БД не для всіх таблиць можна дозволяти додавання рядків або їх зміну.

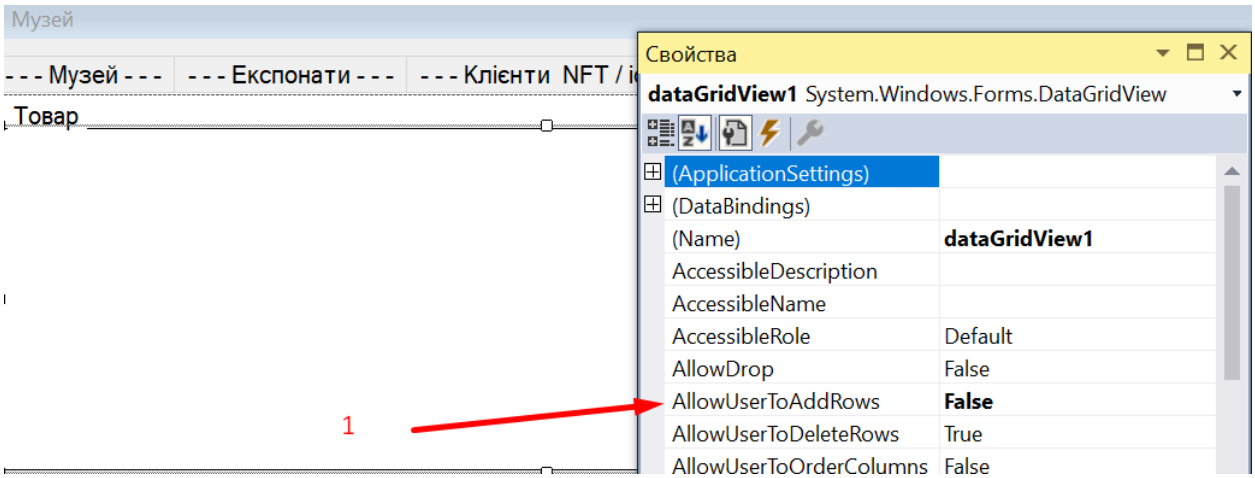

Рис. 3.16. Заборона додавання рядків в таблицю образів (вона не є батьківською)

3.1.8 Додавання та збереження фото

Існує 2 подходи зберігання зображень такими системами.

1 – Зберігання в файловій системі, а в базі даних імена до файлу

2- Зберігання в базі даних в двійковому форматі

Зазвичай використовується перший підхід. Але, є деякі ситуації, коли зберігання фото в базі даних може бути більш придатним варіантом, ніж зберігання в файловій системі. Ось деякі переваги зберігання фото в базі даних:

1. Просіша організація: Зберігання фото в базі даних дозволяє легко організувати та категоризувати фото з використанням запитів та фільтрів, що дозволяє швидко знаходити потрібні дані.
2. Безпека: Зберігання фото в базі даних дозволяє зберігати файли в захищеній інфраструктурі, що підвищує безпеку та знижує ризик втрати даних.
3. Легше керування даними: Керування даними у форматі BLOB (Binary Large OBject) в базі даних може бути легше, ніж керування файлами у файловій системі, оскільки дозволяє зберігати більше даних у форматі BLOB разом з основними даними.
4. Більша мобільність: Зберігання фото в базі даних може бути більш мобільним, оскільки збережені дані можуть бути легко переміщені на інший сервер або мігрувати на іншу базу даних.
5. Однозначність даних: Зберігання фото в базі даних дозволяє забезпечити однозначність даних, що допомагає знизити ризик зміни або втрати даних.
6. Резервне копіювання: Зберігання фото в базі даних дозволяє легко створювати резервні копії, що знижує ризик втрати даних.

Спробуємо використати другий підхід. Після звичайного відкриття діалогу для вибіру файла зображення треба файл конвертувати в строку

```
string base64;
int t = 0;
int rowCount;
string s, file;
OpenFileDialog openFileDialog1 = new OpenFileDialog();
if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
    return;
file = openFileDialog1.FileName;
base64 = Convert.ToBase64String(File.ReadAllBytes(file));
```

Рис. 3.17. Конвертація змісту графічного файлу в строку

Ми не знаємо, чи треба додавати нову строку в таблицю з зображеннями, чи ввести зміни до вже існуючої. Тому спочатку з’ясуємо це

```
cmd.CommandText = "Select * from Images where Tovar = '" + s + "'";
DbDataReader reader = cmd.ExecuteReader();
if (reader.HasRows)
{
    t = 0;
}
else
{
    t = 1;
}
cmd.Dispose();
reader.Close();
```

Рис. 3.18. Запит, чи існує строка з зображенням обраного товару

І вже на базі отриманого результату додаємо нову строку чи змінюємо стару

```
if (t==0)
    {

        cmd.CommandText = "Update Images set Img = '" + base64 + "' where Tovar = '" + s + "'";
        rowCount = cmd.ExecuteNonQuery();
    }
    else  // если этой записи нема - створюємо
    {

        cmd.CommandText = "INSERT INTO Images (Tovar,Img) VALUES ('" + s + "', '" + base64 + "')";
        rowCount = cmd.ExecuteNonQuery();
    }
```

Рис. 3.19. Створення чи оновлення запису зображення.

За такою схемою програма працює зі всіма зображеннями в усіх таблицях.

3.1.9 Транзакції в системі

Транзакція – це архів для запитів до бази. Він захищає дані завдяки принципу "все, або нічого".

В системі проводиться зміна в двох різних таблицях – продаж образу потрібно внести до таблиці Move, одночасно з цим потрібно змінити кількість

реалізованого товару в таблиці Tovar. Ці операції заключено в транзакцію. 1- додавання нового рядку, 2- зміна кількості в іншій таблиці.

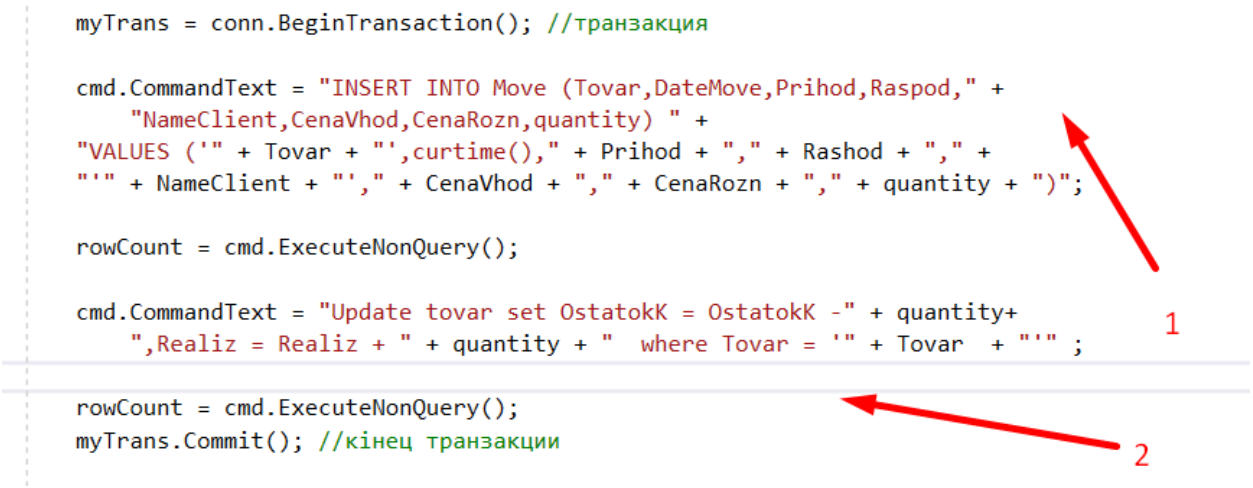

Рис. 3.20. Реалізація змін, в рамках транзакції

3.2 Формування та друк звітностей, фіскальних чеків та накладних

Додаток, що розроблено передбачає такі види звітностей

1. Звітність щодо закупівель кожного авторизованого клієнта
2. Звітність щодо продажів кожним співробітником
3. Звітність щодо робочого часу, на протязі якого співробітник перебував на робочому місці
4. Видаткова або приходна накладна, щодо руху товару.
5. Звітність для податкової в разі використання РРО.

Перші дві звітності в додатку реалізовано у вигляді візуального відображення компонентом dataGridView. Вони не передбачають збереження чи друку. Хоча такі функції звичайно можуть бути додані.

Особливої уваги заслуговує звітність робочого часу. Вона реалізована у вигляді записів у файл логування.

Звітність робочого часу, як правило, передбачає систематичне фіксування часу роботи співробітників з метою контролювання продуктивності та ефективності

їхньої роботи. В цілому, файли логування можуть бути ефективним засобом контролювання робочого часу та продуктивності співробітників, але їх використання повинно бути здійснюватися з дотриманням правових та етичних норм.

Так і зробимо. Будемо фіксувати коли співробітник зайшов до програми, а коли вийшов.

```
Settings.eRole = "";
Settings.login = "";
Settings.password = "";
WriteToLogFile("myLog.txt", Settings.eRole + " вихід з додатку " + DateTime.Now.ToString()
    + " працював часу " + ConvertSecondsToTime(Settings.WorkingTime));
timer1.Stop();
this.Text = "myShop";
```

Рис. 3.21. Фрагмент звітності щодо робочого часу з використанням найпростішого логування.

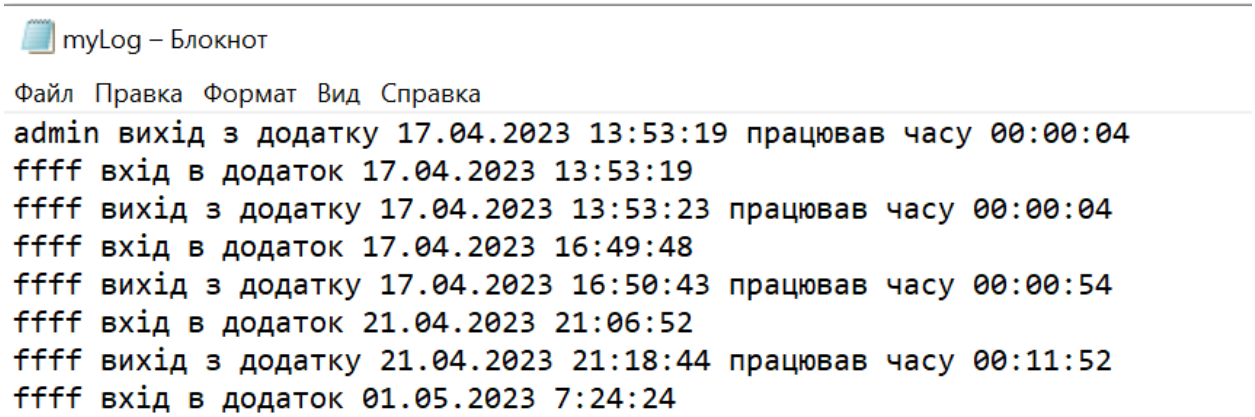

Рис. 3.22 Зміст файлу логування.

3.2.1 Друк накладних та фіскальних чеків

Формування накладних та фіскальних чеків програмою обліку продажів є важливим етапом ведення бізнесу, оскільки ці документи є обов'язковими для звітності перед державними органами. За допомогою програмного забезпечення можна автоматизувати процес формування документів, що значно економить час та зменшує кількість помилок.

PDF формат є одним з найбільш популярних форматів для зберігання та передачі документів через Інтернет, оскільки він зберігає форматування та структуру документа без залежності від операційної системи та програмного

забезпечення. Таким чином, використання PDF формату для зберігання та передачі накладних та фіскальних чеків дозволяє забезпечити їхню безпеку та цілісність.

Для формування PDF використана бібліотека iTextSharp. Фрагмент коду наведено нижчє

```
public void Nakladnaya()
{
    double Sum = 0;
    string tempImagePath = "1.jpg";
    string pdfPath = "Накладная.pdf";

    iTextSharp.text.Document document = new iTextSharp.text.Document();
    PdfWriter writer = PdfWriter.GetInstance(document, new FileStream(pdfPath, FileMode.Create));
    document.Open();

    var baseFont = BaseFont.CreateFont(@"c:\windows\fonts\arial.ttf", BaseFont.IDENTITY_H, BaseFont.EMBEDDED);

    var font = new iTextSharp.text.Font(baseFont, 16, iTextSharp.text.Font.BOLD);
    var paragraph = new Paragraph("Видаткова накладна !", font);
    paragraph.Alignment = Element.ALIGN_CENTER;
    paragraph.SpacingAfter = 10f;
    document.Add(paragraph);

    font = new iTextSharp.text.Font(baseFont, 14, iTextSharp.text.Font.NORMAL);
    paragraph = new Paragraph(DateTime.Now.ToString(), font);
    paragraph.Alignment = Element.ALIGN_CENTER;
    paragraph.SpacingAfter = 10f;
    document.Add(paragraph);

    string NameClient ="Клієнт: "+dataGridView2.CurrentRow.Cells[1].Value.ToString();
    font = new iTextSharp.text.Font(baseFont, 14, iTextSharp.text.Font.NORMAL);
    paragraph = new Paragraph(NameClient, font);
    paragraph.Alignment = Element.ALIGN_LEFT;
    paragraph.SpacingAfter = 20f;
    document.Add(paragraph);
```

Рис. 3.23. Формування документу для друку

Даний код генерує PDF-документ "Накладна", який містить інформацію про товари та послуги, які були продані клієнту. За допомогою бібліотеки iTextSharp створюється документ, додаються різноманітні елементи, такі як заголовки, таблиці, зображення та інші, що форматуються з використанням шрифтів. Крім цього, у коді виконуються розрахунки загальної суми продажів, яку також додають до звітного документа. У кінці коду документ зберігається в PDF форматі та відкривається на комп'ютері користувача, після чого виводиться повідомлення про те, що звітний документ було створено.

3.3.2 Звітність для податкової в разі використання РРО

Звітність для податкової установлює обов'язковість передачі звітів про продажі за кожен день використання розрахункового касового апарату. В звіті передається інформація про загальну суму продажу за день. Ці дані передаються в форматі XML, що дозволяє забезпечити їхню безпеку та точність обробки. Звітність є важливим елементом у веденні бухгалтерського обліку і дозволяє контролювати фінансові операції компанії.

Код, який використано наведено нижчє

```
double sum = 0;
DateTime today = DateTime.Now;
DateTime startOfDay = new DateTime(today.Year, today.Month, today.Day, 0, 0, 0);
DateTime endOfDay = startOfDay.AddDays(1);

cmd = conn.CreateCommand();
cmd.CommandText = "SELECT SUM(quantity * CenaRozn) FROM move WHERE quantity < 0 AND SaleTime >= @StartOfDay AND SaleTime < @EndOfDay";
cmd.Parameters.AddWithValue("@StartOfDay", startOfDay);
cmd.Parameters.AddWithValue("@EndOfDay", endOfDay);
object result = cmd.ExecuteScalar();
if (result != null && result != DBNull.Value)
{
    sum = Convert.ToDouble(result);
}
cmd.Dispose();


XmlDocument xmlDocument = new XmlDocument();
XmlElement rootElement = xmlDocument.CreateElement("Report");
XmlElement sumElement = xmlDocument.CreateElement("Sum");
sumElement.InnerText = sum.ToString();
XmlElement dateElement = xmlDocument.CreateElement("Date");
dateElement.InnerText = today.ToString();
XmlElement fopIdElement = xmlDocument.CreateElement("FOPID");
fopIdElement.InnerText = "123123";
rootElement.AppendChild(sumElement);
rootElement.AppendChild(dateElement);
rootElement.AppendChild(fopIdElement);
xmlDocument.AppendChild(rootElement);
xmlDocument.Save("report.xml");
```

Рис. 3.23. Фрагмент коду для звітності

Код знаходить загальну суму продажів за поточний день за допомогою запиту до бази даних, використовуючи розрахунково-касовий апарат. Результат запиту зберігається в змінну "sum". Потім створюється XML-документ зі структурою звіту, включаючи суму продажів, дату, та ідентифікаційний номер ФОП. Останнім кроком, звіт зберігається у файл "report.xml". Код також виводить повідомлення про успішне створення звіту.

3.3 Авторизація користувачів додатку

Авторизація користувачів та розподіл їхніх прав доступу є ключовим етапом у забезпеченні безпеки програмного забезпечення. Захист інформації про

складський облік та продажі від несанкціонованого доступу є особливо важливим завданням для підприємств.

Введемо для співробітників декілька ролей. Ділити персонал на різні ролі з різними правами доступу дозволяє забезпечити обмеження доступу до чутливих даних, що дозволяє зменшити ризик витоку інформації та зберегти конфіденційність даних.

Фрагмент коду авторизації наведено нижче

```
//перевірка паролю
string l = textBox7.Text;
string p = textBox8.Text;
label16.Text = "Не авторизовано";
NoAcsessTab();
cmd = conn.CreateCommand();
cmd.CommandText = "Select eRole from Employees where Login = '" + l + "' AND Pass ='"+ p +"'";
using (DbDataReader reader = cmd.ExecuteReader())
{
    if (reader.HasRows)
    {
        reader.Read();
       if (Settings.eRole!="" && Settings.eRole != null)
        {
            WriteToLogFile("myLog.txt", Settings.eRole + " вихід з додатку " + DateTime.Now.ToString() +
                " працював часу "
                + ConvertSecondsToTime(Settings.WorkingTime));
        }
        Settings.eRole = reader[0].ToString();
        Settings.login = l;
        Settings.password = p;
        Settings.WorkingTime = 0;
        label16.Text = "Авторизовано, ви " + Settings.eRole;
        AcsessTab();

        WriteToLogFile("myLog.txt", Settings.eRole  + " вхід в додаток "+ DateTime.Now.ToString());
        timer1.Start();
    }
    else
    {
        Settings.eRole = "";
        Settings.login = "";
        Settings.password = "";
        WriteToLogFile("myLog.txt", Settings.eRole + " вихід з додатку " + DateTime.Now.ToString()
            + " працював часу " + ConvertSecondsToTime(Settings.WorkingTime));
        timer1.Stop();
        this.Text = "myShop";
    }
}
```

Рис. 3.24. Фрагмент коду авторизації

Цей код є функцією перевірки логіну та пароля користувача, збереженого в базі даних програми. Після введення логіну та паролю користувача та натискання відповідної кнопки, виконується запит до бази даних на перевірку наявності користувача з введеним логіном та паролем. Якщо користувач існує в базі

даних, відбувається присвоєння його ролі (визначеної у базі даних) у змінну Settings.eRole, логін та пароль також зберігаються в змінних Settings.login та Settings.password відповідно. Також у змінну Settings.WorkingTime присвоюється значення 0. Якщо користувач не знайдений, змінні Settings.eRole, Settings.login та Settings.password будуть встановлені на порожні рядки.

Якщо користувач успішно авторизований, з'являється повідомлення "Авторизовано, ви [роль користувача]", а також виконуються функції AcsessTab() та timer1.Start(). Функція AcsessTab() відповідає за доступ користувача до різних вкладок програми, в залежності від його ролі. timer1.Start() запускає лічильник робочого часу.

Адміністратору надається право корегувати список співробітників, видаляти та створювати їх. В системі також може бути декілька адміністраторів.

Виходячи з того, що адміністратор може побажати очистити систему – він може видалити сам себе. То для повторного входу в систему потрібен поринанні один користувач. Це передбачено в програмі

```
void isEmployeesAdmin()
{
    cmd = conn.CreateCommand();

     cmd.CommandText =
        "INSERT INTO Employees (FIO, eRole, Login, Pass)  SELECT 'admin', 'admin', 'admin'," +
        " 'admin' FROM dual WHERE NOT EXISTS( SELECT * FROM Employees WHERE eRole = 'admin')";
     cmd.ExecuteNonQuery();

    cmd.Dispose();

}
```

Рис. 3.25 Запобігання видалення всіх користувачів

Даний SQL-запит створює нового користувача з правами адміністратора, якщо жоден користувач з правами адміністратора не існує в базі даних.

Запит складається з двох частин: перша частина визначає значення полів нового користувача, а друга частина перевіряє, чи існує користувач з правами адміністратора. Якщо такого користувача не існує, тоді до таблиці Employees

додається новий запис зі значеннями, які були визначені у першій частині запиту.

Команда FROM dual використовується тут тому, що Oracle потребує таблицю для виконання запиту, навіть якщо він не повертає жодного запису. Тому dual - це таблиця з одним стовпцем і одним рядком, яка використовується для таких випадків.

Можна цілком сказати, що даний запит демонструє потужність прямих SQL-запитів.

3.4 Побудова статистичних моделей продажів, кореляційний аналіз

Дані про продажі знаходяться в базі даних, вони там перебувають після певних операцій руху товару. Отже процес побудови статистичних моделей може включати наступні кроки:

Збір даних: Виконується запит до бази даних для отримання необхідних даних про продажі, таких як кількість проданих товарів, ціни, дати тощо. Ці дані можуть бути отримані через SQL-запити.

Передобробка даних: Отримані дані піддаються передобробці, включаючи очищення від відсутніх значень (якщо такі є), видалення дублікатів, нормалізацію або шкалювання, якщо це потрібно.

Аналіз та моделювання: Застосовуються різні статистичні методи і моделі для аналізу та прогнозування продажів. Це можуть бути методи, такі як лінійна регресія, часові ряди, машинне навчання або інші алгоритми, в залежності від специфіки даних та поставленої задачі.

В нашому випадку будемо аналізувати кореляційну залежність продажів одного товару від іншого.

Використання моделей: Після оцінки та валідації моделей, їх можна використовувати для прогнозування майбутніх продажів, вирішення бізнес-задач, планування запасів, оптимізації

Отже додаймо наступні класи

```
public class ForCorrelacia
{
    public string Name;
    public string Id;
    public List<DateAndValue> Values;
    public double BestValue;
    public string BestId;
    public string BestName;
    public ForCorrelacia(string name, string id, DateTime min,DateTime max)
    {
        Name = name;
        Id = id;
        BestValue = 0;
        BestId = "";
        BestName = "";
        Values = new List<DateAndValue>();
        DateTime temp = min;
        while (temp <= max)
        {
            Values.Add(new DateAndValue(temp,0));
            temp = temp.AddDays(1);
        }

    }
}
```

Рисунок Клас контейнер для вмісту залежності продажу певного товару

```
public class DateAndValue
{
    public DateTime Date;
    public double Value;
    public DateAndValue(DateTime date, double value)
    {
        Date = date;
        Value = value;
    }
}
```

Рис. 3.26. Тип даних для оцінки кореляційної залежності

Фрагмент коду для заповнення спуску всіх товарів може виглядати наступним чином.

Отже

1. Створення порожнього списку Correl типу List<ForCorrelacia>. Цей список буде використовуватись для збереження об'єктів типу ForCorrelacia.
2. Очищення елементів checkedListBox1. Це вимагається, щоб позбутися попередньої інформації, якщо вона була вже відображена.
3. Отримання значень dateMin та dateMax з dateTimePicker3 та dateTimePicker4 відповідно. Ці значення використовуються для визначення меж дати, які будуть враховуватись при виконанні запиту до бази даних.

4. Створення команди cmd для виконання запиту до бази даних. Запит вибирає унікальні значення Tovar та ScanId з таблиці move, де SaleTime знаходиться між dateMin та dateMax.
5. Виконання запиту до бази даних через ExecuteReader(). Отримані дані перебираються по одному запису за допомогою циклу while (reader.Read()).
6. Для кожного запису створюється новий об'єкт типу ForCorrelacia, використовуючи значення першого і другого стовпців (reader[0] і reader[1]) та передані значення dateMin та dateMax. Цей об'єкт додається до списку Correl. Це і є товар, що аналізується.

```
Correl = new List<ForCorrelacia>();
checkedListBox1.Items.Clear();
DateTime dateMin, dateMax;
dateMin = dateTimePicker3.Value.Date;
dateMax = dateTimePicker4.Value.Date;
cmd = conn.CreateCommand();
cmd.CommandText = "SELECT Distinct Tovar, ScanId FROM move WHERE SaleTime BETWEEN @dateMin AND @dateMax";
cmd.Parameters.AddWithValue("@dateMin", dateMin);
cmd.Parameters.AddWithValue("@dateMax", dateMax);

using (DbDataReader reader = cmd.ExecuteReader())
{
    if (reader.HasRows)
    {
        while (reader.Read())
        {
            Correl.Add(new ForCorrelacia(reader[0].ToString(),
               reader[1].ToString(),
                 dateMin, dateMax));
            checkedListBox1.Items.Add(reader[0].ToString()+";scanId:"+ reader[1].ToString());
        }

    }
}
cmd.Dispose();
```

Рис. 3.27. Підготовка даних до аналізу. Створення списку товарів.

Тепер потрібно заповнити його даними, за кожен день продажу

1. Для кожного об'єкта ForCorrelacia в списку Correl виконується цикл foreach.
2. Створюється нова команда для виконання запиту до бази даних. Запит вибирає суму Raspod з таблиці move, де значення Raspod більше 0, ScanId співпадає з correl.Id і SaleTime знаходиться між temp та temp.AddDays(1).
3. В циклі заповнюються дані щодо продажу за кожен день

```
foreach (ForCorrelacia correl in Correl)
{
    DateTime temp = dateMin;

    while (temp <= dateMax)
    {
        cmd = conn.CreateCommand();
        cmd.CommandText = "SELECT Sum(Raspod) FROM move WHERE Raspod>0 and ScanId = "
            + correl.Id
            + " and (SaleTime BETWEEN @dateMin AND @dateMax)";
        cmd.Parameters.AddWithValue("@dateMin", temp);
        cmd.Parameters.AddWithValue("@dateMax", temp.AddDays(1));

        using (DbDataReader reader = cmd.ExecuteReader())
        {
            if (reader.HasRows)
            {
                reader.Read();

                    DateAndValue TargetCorrel = correl.Values.Find(p => p.Date >= temp
                    && p.Date<temp.AddDays(1));
                    if (TargetCorrel != null)
                    {
                       if (double.TryParse(reader[0].ToString(),out TargetCorrel.Value))
                       {

                       }
                    }

            }
        }
        cmd.Dispose();
        temp=temp.AddDays(1);
    }
}
```

Рис. 3.28. Підготовка моделі даних, заповнення списку продажів

Модель створено. Після цього вже не складає труднощів проводити її аналіз.

Наприклад побудуємо залежність продажів від часу

```
ForCorrelacia correl = Correl.Find(p=>p.Id== scanId);
foreach (DateAndValue dt in correl.Values)
{
    newSeries.Points.AddXY(dt.Date.ToString("dd.MM"), dt.Value);
    x++;
}

chart1.Series.Add(newSeries);
```

Рис. 3.29. Фрагмент коду для побудови залежності продажів від часу

В такому випадку розрахунок кореляції виглядає наступним чином

```
public Tuple<ForCorrelacia, ForCorrelacia> FindMaxCorrelation(List<ForCorrelacia> correlList)
{
    double maxCorrelation = 0;
    ForCorrelacia maxCorrelationClass1 = null;
    ForCorrelacia maxCorrelationClass2 = null;

    for (int i = 0; i < correlList.Count - 1; i++)
    {
        for (int j = i + 1; j < correlList.Count; j++)
        {
            double correlation = CalculateCorrelation(correlList[i].Values, correlList[j].Values)
            if (correlation > maxCorrelation)
            {
                maxCorrelation = correlation;
                maxCorrelationClass1 = correlList[i];
                maxCorrelationClass2 = correlList[j];
                maxCorrelationClass1.BestValue = correlation;
                maxCorrelationClass2.BestValue = correlation;
                maxCorrelationClass1.BestId = maxCorrelationClass2.Id;
                maxCorrelationClass2.BestId = maxCorrelationClass1.Id;
            }
        }
    }

    return new Tuple<ForCorrelacia, ForCorrelacia>(maxCorrelationClass1, maxCorrelationClass2);
}
```

Рис. 3.20. Перебір списку для знаходження товарів, з максимальним значенням кореляції функції продажу від часу

Безпосередньо кореляція розрахована наступним чином

```
double[] firstValues = new double[values1.Count];
double[] secondValues = new double[values1.Count];
int i = 0;
foreach (DateAndValue value in values1)
{
    firstValues[i] = value.Value;
    i++;
}
i = 0;
foreach (DateAndValue value in values2)
{
    secondValues[i] = value.Value;
    i++;
}
double meanFirst = firstValues.Sum() / firstValues.Length;
double meanSecond = secondValues.Sum() / secondValues.Length;
double sumProduct = 0;
double sumSquareFirst = 0;
double sumSquareSecond = 0;
for (i = 0; i < firstValues.Length; i++)
{
    double deviationFirst = firstValues[i] - meanFirst;
    double deviationSecond = secondValues[i] - meanSecond;

    sumProduct += deviationFirst * deviationSecond;
    sumSquareFirst += deviationFirst * deviationFirst;
    sumSquareSecond += deviationSecond * deviationSecond;
}

double correlation = sumProduct / Math.Sqrt(sumSquareFirst * sumSquareSecond);

return correlation;
```

Рис. 3.31. Розрахунок кореляції послідовностей

3.5 Спеціальні розрахунки, тестування системи

Спеціальні розрахунки та тестування системи є важливою складовою розробки будь-якої програми. Вони дозволяють перевірити коректність роботи програми та виявити та усунути можливі помилки. Це особливо важливо в системах товарообігу, де точність обробки даних має значний вплив на результати бізнесу.

3.5.1 Проведення повного циклу товарообігу, як метод тестування

Проведення повного циклу товарообігу є важливим методом тестування системи управління товарообігом. Цей метод дозволяє перевірити правильність роботи всіх модулів системи, від постачання товарів до їх реалізації та звітності. Результати цього тестування можуть допомогти виявити можливі проблеми та недоліки системи та внести необхідні виправлення для покращення її ефективності та продуктивності.

Перше вікно системи

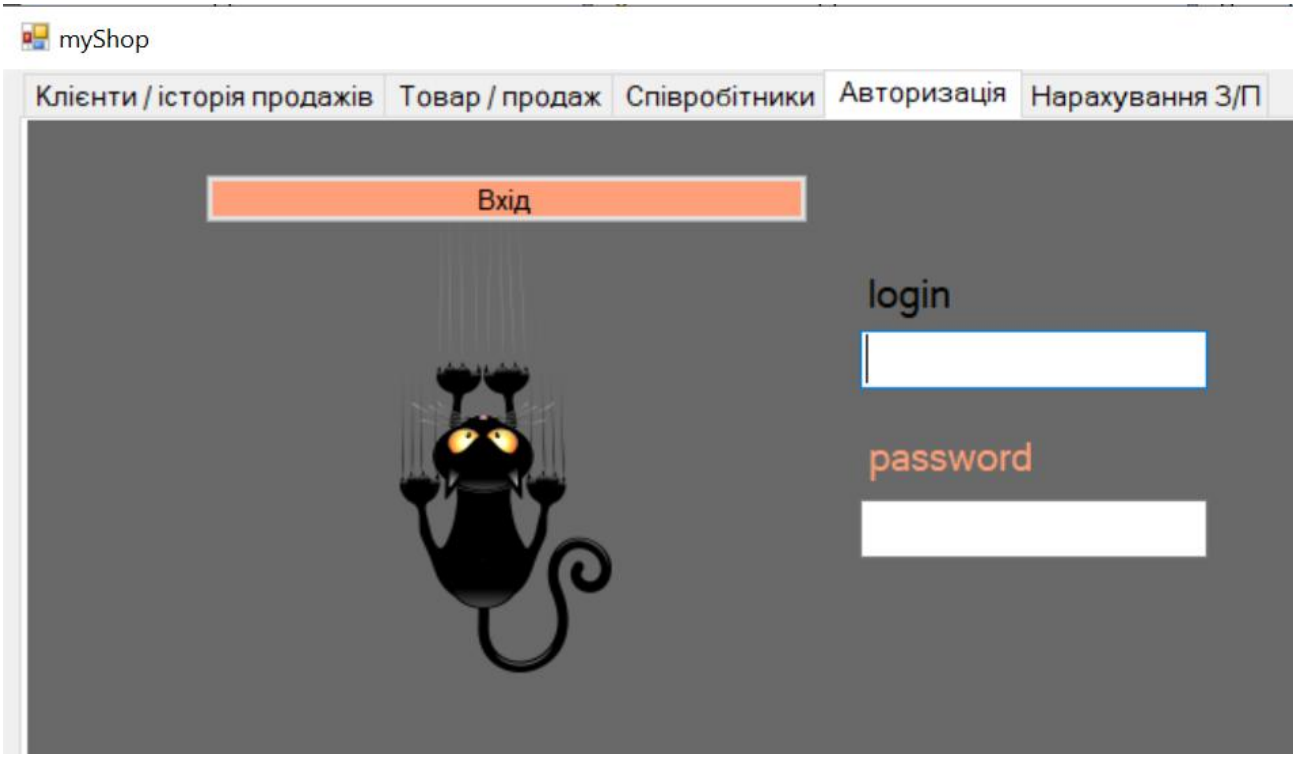

Рис. 3.32. Вхід до системи

При вході до системи як адміністратор можна змінювати налаштування програми, та створювати облікові записи співробітників.

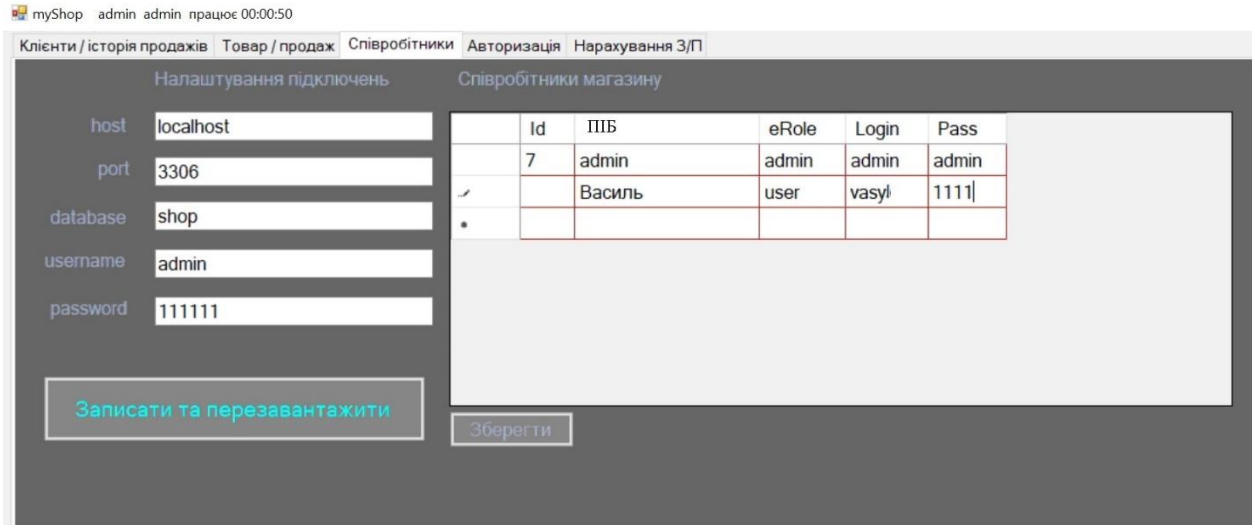

Рис. 3.33. Створення облікових записів.

Створимо декілька клієнтів

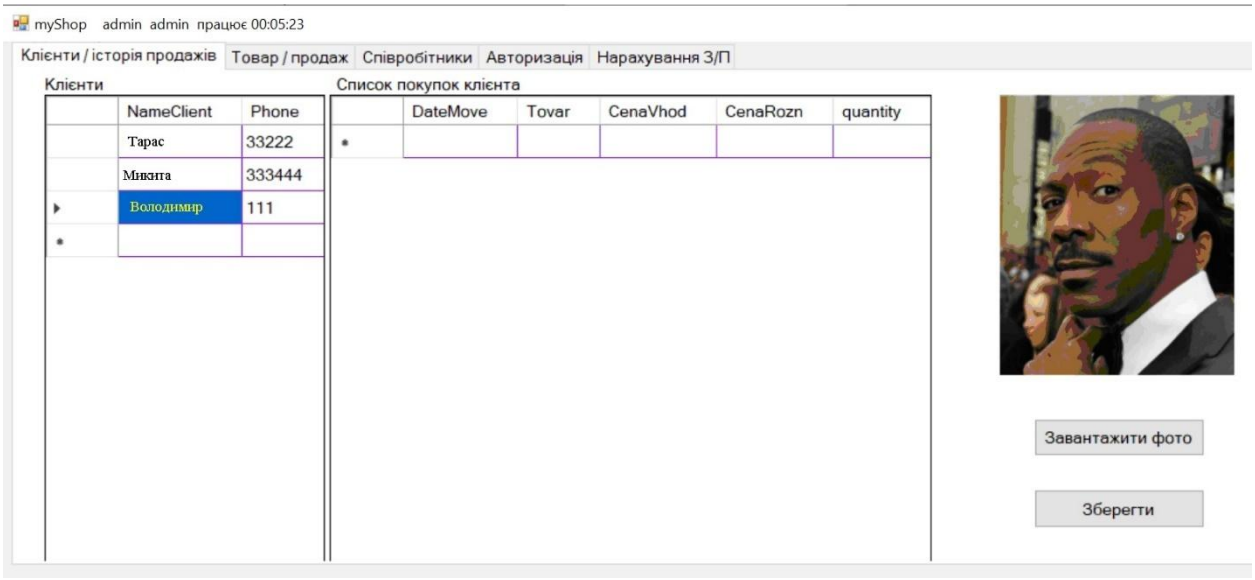

Рис. 3.34. Створення клієнтів

Створення кількох позицій товару, також загрузка до них фото

Результат на рисунку

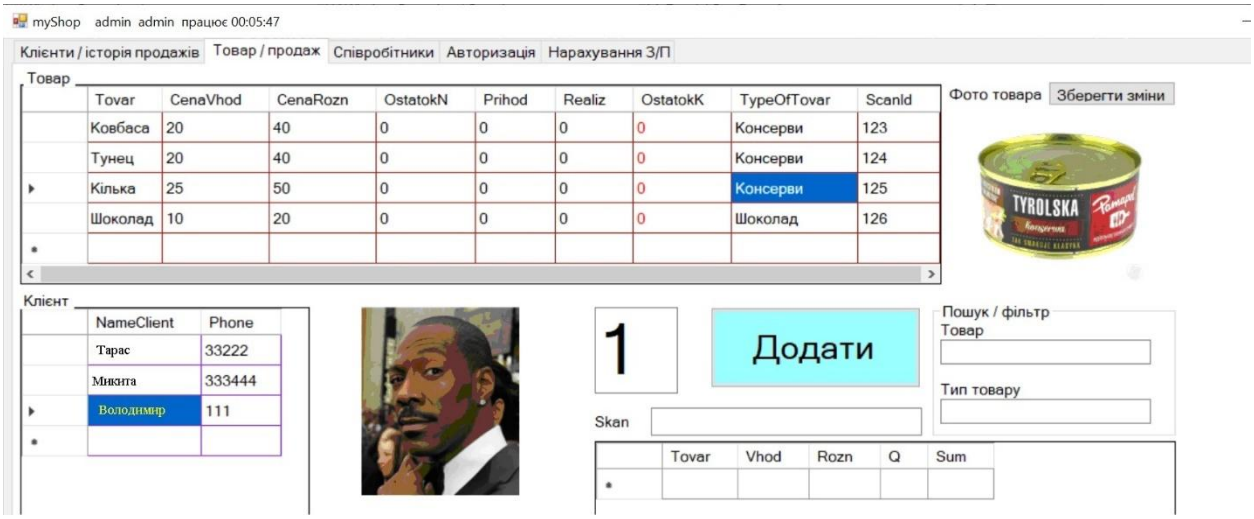

Рис. 3.35. Створені товари

Для використання фільтрів достатньо почати вводити перші букви назви товару.

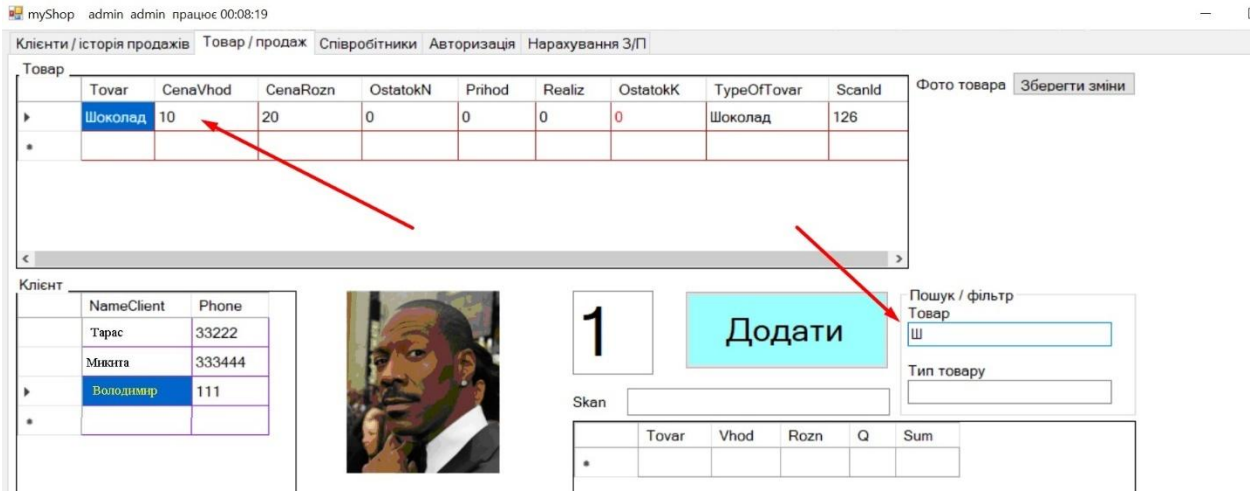

Рис. 3.36. Тест фільтрів

Наповнювання сиску товарів продемонстровано на рисунку.

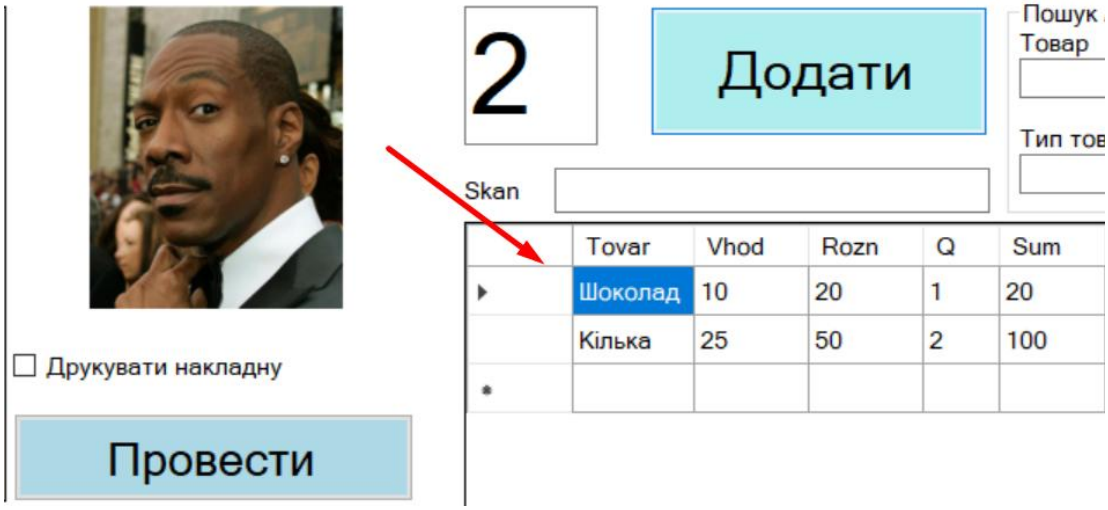

Рис. 3.37. Попередня накладна – список покупок

При натисканні кнопки «провести» операція продажу завершується.

Формується видаткова накладна:

Видаткова накладна !

05.05.2023 19:39:48

Клієнт: Вова

Товар	Кількість	Ціна	Сума
Шоколад	1	20	20
Кілька	2	50	100

Разом: 120

Рис. 3.38. Видаткова накладна

В програмі ведеться звітність щодо покупок клієнта. Отже, данні після продажу оновлено.

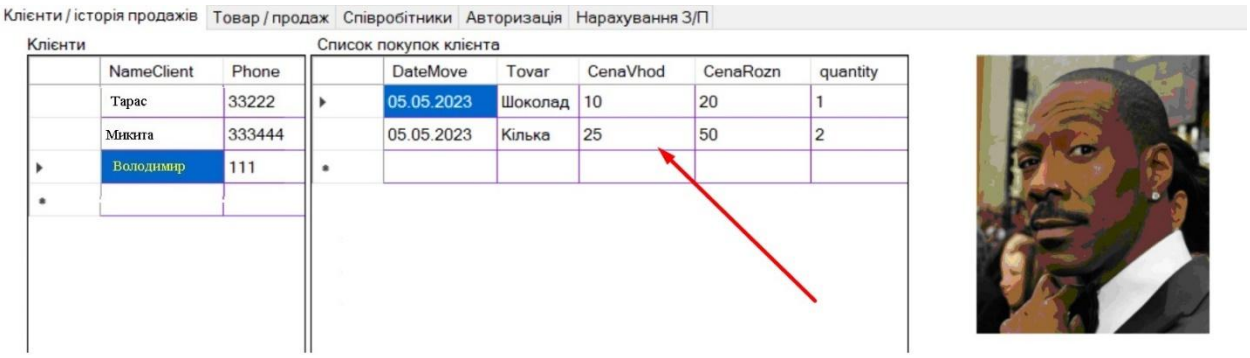

Рис. 3.39 Картка клієнта

В програмі є можливість сформувати звітність щодо продажу конкретним співробітником. Цей показник зазвичай враховується при аналізі економічної ефективності.

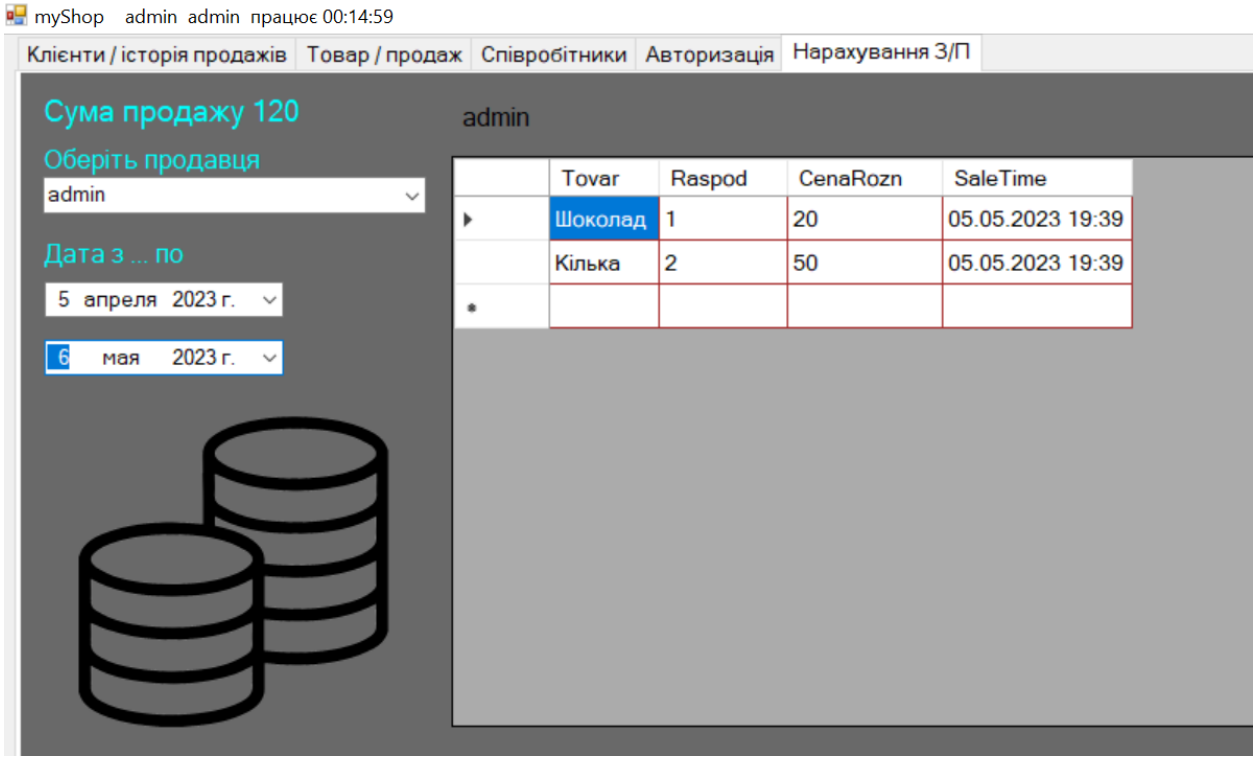

Рис. 3.40. Звітність щодо продажів співробітниками.

Формування звітнсті для Державної податкової служби:

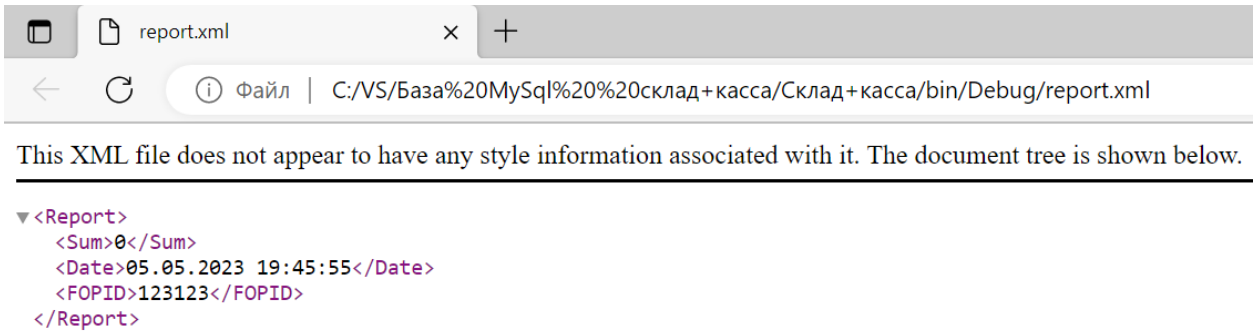

Рис. 3.41. Звітність щодо продажів за день

Звітність формується в форматі XML та готова до відправлення.

Аналіз продажів

База даних поки що не налічує багато записів, але вже можна спробувати її аналізувати

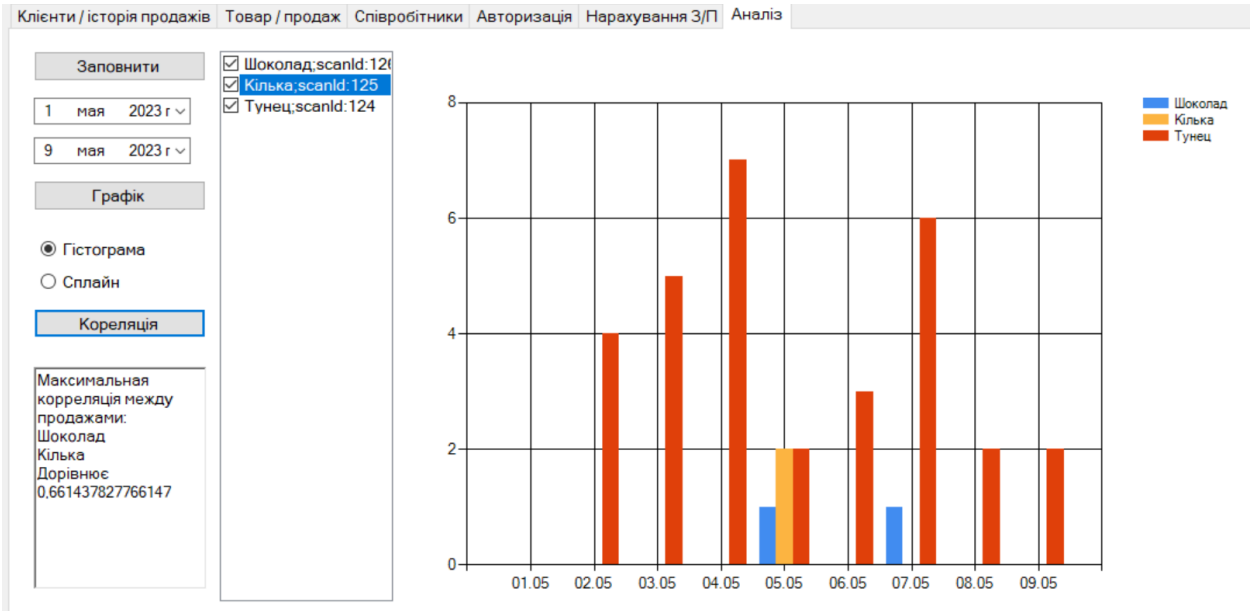

Рис. 3.42. Робота вкладки «Аналіз»

Можна бачити що «Тунець» активно продається на початку травня, а ось продаж “Кільки” пов'язаний з продажами “Шоколаду”.

3.5.2 Інструкція користувача з інсталяції та використання програмного продукту

Станом на даний момент програма має незначне обчислювальне навантаження. Програма працює на операційній системі Windows 10 64 та використовує базу MySql. Отже, можливо сформувати вимоги до комп'ютера:

Операційна система Windows 10 64-бітна версія

Мінімум 2 ГБ оперативної пам'яті

Мінімум 100 МБ вільного місця на жорсткому диску

Доступ до Інтернету

MySQL сервер

Інструкція з інсталяції та використання програмного продукту:

Крок 1: Завантажити виконуваний файл програми з відповідного джерела.

Крок 2: Запустити виконуваний файл програми (натиснути двічі на файлі з розширенням .exe).

Крок 3: Якщо система спитає про підтвердження встановлення, підтвердити його.

Крок 4: Обрати папку, в якій буде встановлено програму.

Крок 5: Ввести дані для підключення до бази даних MySQL. Це можна зробити в налаштуваннях програми.

Крок 6: Після підключення до бази даних програма повністю готова до роботи.

Крок 7: Для завершення роботи з програмою її можна закрити використовуючи відповідну опцію меню.

3.5.3 Інструкція по створенню бази даних додатку

Програма може працювати як локально так і в мережі. Програма використовує MySQL.

Отже по-перше треба встановити MySQL. Завантажування краще проводити з офіційного сайту.

Після встановлення та створення необхідних налаштувань треба створити базу даних. Програма використовує базу з ім’ям “shop“. Після створення БД потрібно створити необхідні таблиці.

Краще всього скористатись скріптом, який наведено в попередньому розділі. Але можна використовувати сервіси імпорту даних, які представляє програмне забезпечення MySQL.

3.5.4 Налаштування зовнішнього вигляду звітної документації

Програма на даному етапі створення використовує друк до PDF файлів. Використовується бібліотека iTextSharp. Отже всі налаштування друку встановлюються при розробці додатку.

Бібліотека iTextSharp надає розширені можливості для налаштування зовнішнього вигляду звітної документації. Основні можливості, щодо форматування, включають:

Шрифти: Можна налаштувати різні шрифти для тексту заголовків, параграфів, списків і т. д. iTextSharp підтримує використання різних типів шрифтів, розмірів і стилів.

Кольори: Для підкреслення важливих елементів або для створення розмітки можна використовувати різні кольори. iTextSharp дозволяє встановлювати кольори фону, тексту, рамок і т. д.

Вирівнювання: Можна встановлювати вирівнювання тексту по горизонталі і вертикалі, щоб досягти бажаного розташування елементів на сторінці.

Відступи: За допомогою iTextSharp можна встановити відступи зверху, знизу, зліва і справа для контенту сторінки. Це дозволяє створювати зручний макет для звітів.

Розмір сторінки: Бібліотека дозволяє встановлювати розмір сторінки документа, такий як A4, Letter і т. д. Також можна налаштувати орієнтацію сторінки (портретну або альбомну).

Зображення: iTextSharp дозволяє додавати зображення до звітів, наприклад, логотипи компаній або графіки. Можна налаштовувати розмір, розташування і вирізання зображень.

Таблиці: Є можливість створювати таблиці з різним форматуванням, включаючи розмір стовпців, вирівнювання, стиль рамок і т. д.

Бібліотека iTextSharp не надає вбудованого інтерфейсу користувача. iTextSharp є бібліотекою для роботи з PDF-документами у програмному коді. Вона дозволяє програмістам створювати і налаштовувати PDF-файли безпосередньо з коду. Однак слід додати що існують інші види звітностей де ця можливість присутня. Наприклад FastReport. Це графічний інструмент для створення звітів і документації. Він надає готовий інтерфейс користувача, який дозволяє дизайнерам та користувачам створювати та налаштовувати звіти без необхідності писати код.

Однак програма на даному етапі розробки використовує перший варіант.

3.6 Техніко-економічні показники програмного забезпечення

Техніко-економічні показники (ТЕП) програмного забезпечення включають технічні показники, які оцінюють продуктивність і функціональні можливості програми, а також економічні показники, які оцінюють витрати та користь, пов'язані з розробкою та використанням програми. Основні ТЕП для програми обліку товарообігу можуть включати:

Технічні показники:

Продуктивність: Швидкість обробки даних, завантаження та збереження інформації, оптимізація запитів до бази даних.

Масштабованість: Здатність програми працювати з великим обсягом даних і розширюватися залежно від потреб бізнесу.

Надійність: Стійкість програми до помилок, відновлення після відмов, захист даних від втрати.

Безпека: Захист бази даних та інформації, автентифікація та авторизація користувачів, захист від несанкціонованого доступу.

Зручність використання: Інтуїтивно зрозумілий інтерфейс користувача, зручність навігації та роботи з програмою.

Економічні показники:

Вартість розробки: Оцінка витрат на розробку програми, включаючи зарплати розробників, придбання необхідного обладнання та програмних засобів.

Вартість впровадження: Оцінка витрат на впровадження програми, включаючи налаштування бази даних, навчання персоналу, імпорт даних з існуючих систем.

Витрати на супровід: Оцінка витрат на підтримку та розвиток програми, включаючи виправлення помилок, впровадження нових функцій, оновлення програмного забезпеченя.

Користь: Оцінка економічної користі від використання програми, такі як зниження трудовитрат, підвищення ефективності обліку та звітності, зниження помилок та втрати товарів, поліпшення контролю над товарообігом і складським управлінням.

Повернення інвестицій (ROI): Оцінка того, скільки часу знадобиться для отримання повернення витрат на розробку і впровадження програми через збільшення ефективності та зниження витрат.

Термін служби: Період, протягом якого програма буде використовуватися без значних модифікацій або замін.

Сумарні витрати володіння (TCO): Оцінка загальних витрат на розробку, впровадження, супровід та оновлення програми протягом всього періоду її використання.

Віддача від інвестицій: Оцінка фінансових вигод, отриманих в результаті використання програми, враховуючи всі витрати і користь.

Ці техніко-економічні показники допомагають оцінити ефективність та цілеспрямованість розробки програми і визначити її вартість і вигоди для бізнесу.

ВИСНОВКИ

Було розроблено програмне забезпечення для управління товарообігом, яке включає функції обліку продажів, закупівель та звітності. Дослідження виконано з метою розробки ефективної та надійної системи для автоматизації бізнес-процесів та поліпшення управління товарообігом.

Під час реалізації програмного забезпечення були досягнуті такі основні цілі:

- Розроблено функціональне програмне забезпечення для управління товарообігом, яке включає необхідні функції обліку продажів, закупівель та звітності.
- Впроваджено систему бази даних для зберігання та організації інформації про товари, клієнтів, продажі та співробітників.
- Використано рольову модель доступу для обмеження прав користувачів та забезпечення безпеки даних.
- Забезпечено можливість ведення документації про закупівлі та продажі, включаючи формування накладних та фіскальних чеків.
- Реалізовано аналітичні звіти та статистичні моделі для аналізу продажів та кореляційного аналізу між товарами.

Після проведеного дослідження та реалізації програмного забезпечення можна зробити наступні висновки:є

- Розроблене програмне забезпечення дозволяє ефективно вести облік продажів, закупівель та звітності, спрощуючи бізнес-процеси та полегшуючи управління товарообігом.
- Впровадження системи бази даних забезпечує надійне зберігання та доступ до даних, а також покращує швидкість операцій та ефективність роботи з інформацією.
- Використання рольової моделі доступу забезпечує контроль над правами користувачів та забезпечує конфіденційність даних.

- Наявність аналітичних звітів та статистичних моделей дозволяє аналізувати продажі та виявляти залежності між товарами, що допомагає в прийнятті управлінських рішень.

Дослідження в області кореляції товарів має велике значення для бізнесу, оскільки воно допомагає розуміти взаємозв'язки між різними товарами та їх вплив на продажі. Аналіз кореляційних залежностей між товарами дозволяє виявити зв'язки, які можуть бути використані для розробки ефективних стратегій маркетингу, планування асортименту та управління запасами. Застосування статистичних моделей дозволяє прогнозувати зміни попиту на товари та ефективно керувати запасами, уникати недостачі або надлишків товарів на полицях.

Дослідження має деякі обмеження:

- Розроблене програмне забезпечення було протестоване лише на обмеженій кількості тестових даних та сценаріїв. Для підтвердження його ефективності та надійності потрібно провести додаткові тестування з реальними даними та реальними умовами використання.
- Наявність обмеженого обсягу даних для аналізу може призвести до недостатньо точних результатів статистичних моделей. Рекомендується збільшити обсяг даних для отримання більш точних та надійних прогнозів.

Загалом, розроблене програмне забезпечення для управління товарообігом є ефективним інструментом для автоматизації бізнес-процесів та поліпшення управління продажами та закупівлями. Проведені дослідження та отримані результати підтверджують досягнення поставлених цілей та вказують на можливості подальшого розширення та вдосконалення системи.

ПОСИЛАННЯ

1. Кравчук І. Бухгалтерський облік з використанням баз даних у середовищі C#: Практичний посібник. Київ: Видавництво "Підручники і посібники", 2022.
2. Степаненко О. Розробка програмного забезпечення для бухгалтерського обліку в середовищі C#: Теорія та практика. Київ: Видавництво "БІОС", 2020.
3. Лавріщев О. Мова програмування C#. Київ: Видавництво "Підручники і посібники", 2019.
4. MacDonald M. Pro C# 7 and .NET Core 2.0: With MySQL and MariaDB Data Access. Berkeley, CA: Apress, 2018.
5. Russell J. T. Dyer. Learning MySQL and MariaDB: Heading in the Right Direction with MySQL and MariaDB 1st Edition. Sebastopol, CA: O'Reilly Media, 2015.

ДОДАТКИ

Додаток А

Класи підключення до БД

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MySql.Data.MySqlClient;

namespace Tutorial.SqlConn
{
   class DBUtils
   {
      public static MySqlConnection GetDBConnection(string h,int p,string db,string un,string pass)
      {
         string host = h;
         int port = p;
         string database = db;
         string username = un;
         string password = pass;

         return DBMySQLUtils.GetDBConnection(host, port, database, username, password);
      }

   }
         }



namespace Tutorial.SqlConn
{
   class DBMySQLUtils
   {

      public static MySqlConnection
            GetDBConnection(string host, int port, string database, string username, string password)
      {
         String connString = "Server=" + host + ";Database=" + database
            + ";port=" + port + ";User Id=" + username + ";password=" + password;

         MySqlConnection conn = new MySqlConnection(connString);

         return conn;
      }

   }
         }
```

Додаток Б

Створення БД

```
CREATE Database Shop;
Use Shop;
CREATE TABLE `clients` (
  `Id` int NOT NULL AUTO_INCREMENT,
  `NameClient` varchar(20) NOT NULL,
  `Phone` varchar(20) DEFAULT NULL,
  PRIMARY KEY (`Id`),
  UNIQUE KEY `NameClient` (`NameClient`);

CREATE TABLE `employees` (
  `Id` int NOT NULL AUTO_INCREMENT,
  `FIO` varchar(255) NOT NULL,
  `eRole` varchar(255) NOT NULL,
  `Login` varchar(255) NOT NULL,
  `Pass` varchar(255) NOT NULL,
  PRIMARY KEY (`Id`);

CREATE TABLE `images` (
  `Id` int NOT NULL AUTO_INCREMENT,
  `Tovar` varchar(20) NOT NULL,
  `Img` longblob,
```

```
  PRIMARY KEY (`Id`),
  UNIQUE KEY `Tovar` (`Tovar`),
  CONSTRAINT `images_ibfk_1` FOREIGN KEY (`Tovar`) REFERENCES
`tovar` (`Tovar`) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE `move` (
  `Id` int NOT NULL AUTO_INCREMENT,
  `Tovar` varchar(20) NOT NULL,
  `DateMove` date DEFAULT NULL,
  `Prihod` float DEFAULT NULL,
  `Raspod` float DEFAULT NULL,
  `NameClient` varchar(20) NOT NULL,
  `CenaVhod` float DEFAULT NULL,
  `CenaRozn` float DEFAULT NULL,
  `quantity` float DEFAULT NULL,
  `Salesman` varchar(255) DEFAULT NULL,
  `SaleTime` datetime DEFAULT NULL,
  `TypeOfTovar` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`Id`),
  KEY `NameClient` (`NameClient`),
  KEY `move_ibfk_2_idx` (`Tovar`),
```

```
    CONSTRAINT `move_ibfk_1` FOREIGN KEY (`NameClient`) REFERENCES
`clients` (`NameClient`),
    CONSTRAINT `move_ibfk_2` FOREIGN KEY (`Tovar`) REFERENCES
`tovar` (`Tovar`) ON DELETE CASCADE ON UPDATE CASCADE
    );

    CREATE TABLE `photos` (
      `Id` int NOT NULL AUTO_INCREMENT,
      `NameClient` varchar(20) NOT NULL,
      `Photo` longblob,
      PRIMARY KEY (`Id`),
      UNIQUE KEY `NameClient` (`NameClient`),
      CONSTRAINT `photos_ibfk_1` FOREIGN KEY (`NameClient`)
REFERENCES `clients` (`NameClient`) ON DELETE CASCADE ON UPDATE
CASCADE
    );

    CREATE TABLE `tovar` (
      `Id` int NOT NULL AUTO_INCREMENT,
      `Tovar` varchar(20) NOT NULL,
      `CenaVhod` float NOT NULL,
      `CenaRozn` float NOT NULL,
      `OstatokN` float DEFAULT NULL,
      `Prihod` float DEFAULT NULL,
```

```
  `Realiz` float DEFAULT NULL,
  `OstatokK` float DEFAULT NULL,
  `TypeOfTovar` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`Id`),
  UNIQUE KEY `Tovar` (`Tovar`)
);
```

Додаток В

Основна програма

```
using MySql.Data.MySqlClient;
using System;
using System.Data;
using System.Drawing;
using System.Windows.Forms;
using Tutorial.SqlConn;
using System.Data.Common;
using System.IO;
using iTextSharp.text.pdf;
using iTextSharp.text;
using System.Xml;

namespace WindowsFormsApp1
{
   public partial class Form1 : Form
   {
      MySqlConnection conn;
      int triger = 0;
      MySqlDataAdapter mydtadp = new MySqlDataAdapter();
      BindingSource bindingSource1 = new BindingSource();
      MySqlDataAdapter mydtadp2 = new MySqlDataAdapter();
      BindingSource bindingSource2 = new BindingSource();
      MySqlDataAdapter mydtadp3 = new MySqlDataAdapter();
      BindingSource bindingSource3 = new BindingSource();
      MySqlDataAdapter mydtadp4 = new MySqlDataAdapter();
      BindingSource bindingSource4 = new BindingSource();
      MySqlCommandBuilder cmbl, cmb2, cmb3;

      MySqlCommand cmd;
      MySqlTransaction myTrans;

      public Form1()
      {
         InitializeComponent();
         tabControl1.Selecting += tabControl1_Selecting;

         this.Width = 1500;
      }

      private void tabControl1_Selecting(object sender, TabControlCancelEventArgs e)
      {
         if (!e.TabPage.Enabled)
         {
            e.Cancel = true;
         }
      }


      private void Form1_Load(object sender, EventArgs e)
      {
         ReadConfig();
         conn = DBUtils.GetDBConnection(textBox2.Text,Convert.ToInt32(textBox3.Text), textBox4.Text, textBox5.Text,
textBox6.Text);
         NoAcsessTab();
         try
         {

            conn.Open();
            MessageBox.Show("Connection successful!");
            ZapolnitClient();
```

```
    }
    catch
    {
        MessageBox.Show("Error Connection");

    }
    isEmployeesAdmin();
    tabControl1.SelectedIndex = 3;


}

void WriteConfig()
{
    string path = System.IO.Path.Combine(Environment.CurrentDirectory, "Config.txt");
    using (StreamWriter writer = new StreamWriter(path, false))
    {
        writer.WriteLine(textBox2.Text);
        writer.WriteLine(textBox3.Text);
        writer.WriteLine(textBox4.Text);
        writer.WriteLine(textBox5.Text);
        writer.WriteLine(textBox6.Text);
    }
}


void ReadConfig()
{
    string path = System.IO.Path.Combine(Environment.CurrentDirectory,"Config.txt");

    using (StreamReader reader = new StreamReader(path))
    {
        string line;
        line = reader.ReadLine();
        textBox2.Text = line;
        line = reader.ReadLine();
        textBox3.Text = line;
        line = reader.ReadLine();
        textBox4.Text = line;
        line = reader.ReadLine();
        textBox5.Text = line;
        line = reader.ReadLine();
        textBox6.Text = line;

    }
}



private void tabControl1_SelectedIndexChanged(object sender, EventArgs e)
{
    triger = 0;

    isRole();

    if (tabControl1.SelectedIndex == 1)
    {
        ZapolnitTovar(textBox9.Text,textBox10.Text,"",false);
        ZapolnitClient();
    }

    if (tabControl1.SelectedIndex == 2)
    {
        FillEmployees();
    }
```

```
        if (tabControl1.SelectedIndex == 4)
        {
            dateTimePicker2.Value = DateTime.Now;
            dateTimePicker1.Value = DateTime.Now.AddDays(-30);
            FillEmployeesForZP();
        }

        triger = 1;
    }

    void FillEmployeesForZP()
    {
        cmd = conn.CreateCommand();
        cmd.CommandText = "Select login from Employees";
        using (DbDataReader reader = cmd.ExecuteReader())
        {
            if (reader.HasRows)
            {
                comboBox1.Items.Clear();
                while (reader.Read())
                {
                    comboBox1.Items.Add(reader[0].ToString());
                }
            }
        }
        cmd.Dispose();
    }

    void FillProdagi()
    {
        cmd = conn.CreateCommand();
        cmd.CommandText = "Select FIO from Employees where login = '"+comboBox1.Text +"'";
        using (DbDataReader reader = cmd.ExecuteReader())
        {
            if (reader.HasRows)
            {
                reader.Read();

                    label17.Text = reader[0].ToString();

            }
        }

        cmd.Dispose();

        DateTime dateMin, dateMax;

        dateMin = dateTimePicker1.Value.Date;
        dateMax = dateTimePicker2.Value.Date;

        mydtadp4.SelectCommand = new MySqlCommand("SELECT Tovar, Raspod, CenaRozn, SaleTime FROM move
WHERE Salesman = '" + comboBox1.Text + "' AND SaleTime BETWEEN @dateMin AND @dateMax", conn);

        mydtadp4.SelectCommand.Parameters.AddWithValue("@dateMin", dateMin);
        mydtadp4.SelectCommand.Parameters.AddWithValue("@dateMax", dateMax);

        cmbl = new MySqlCommandBuilder(mydtadp4);
        DataTable table = new DataTable();
        mydtadp4.Fill(table);
        bindingSource4.DataSource = table;
        dataGridView7.DataSource = bindingSource4;
        dataGridView7.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.DisplayedCells;
        dataGridView7.GridColor = Color.Brown;

        double sum = 0;
        for (int i= 0;i< dataGridView7.Rows.Count;i++)
```

```
        {
            object cellValue1 = dataGridView7.Rows[i].Cells[1].Value;
            object cellValue2 = dataGridView7.Rows[i].Cells[2].Value;
            sum += Convert.ToDouble(cellValue1) * Convert.ToDouble(cellValue2);
        }
        label18.Text = "Сума продажу "+ sum;

    }

    void NoAcsessTab()
    {
        tabControl1.TabStop = false;

        int ind = 0;
        foreach (TabPage page in tabControl1.TabPages)
        {
            if (ind!=3) page.Enabled = false;
            ind++;
        }

    }

    void AcsessTab()
    {
        int j = 0;
        foreach (TabPage page in tabControl1.TabPages)
        {
            if (j!=2) page.Enabled = true;
            else
            {
                if (Settings.eRole=="admin") page.Enabled = true;
            }
            j++;
        }

                }

void ZapolnitTovar(string tov,string typeTov, string toScan,bool Scan)
    {
        triger = 0;

        string myQuery = "";

        if (tov !="" && tov !=null)
        {
            myQuery += " where Tovar Like '"+ tov + "%'";
        }

        if (typeTov != "" && typeTov != null)
        {
            if (myQuery == "")
            {
                myQuery += " where TypeOfTovar Like '" + typeTov + "%'";
            }
            else
            {
                myQuery += " and TypeOfTovar Like '" + typeTov + "%'";
            }
        }

        if (Scan)
        {
            myQuery = "select * from Tovar where ScanId = '"+ toScan + "'";
        }
        else
        {
```

```
        myQuery = "select * from Tovar" + myQuery;
    }


    mydtadp.SelectCommand = new MySqlCommand(myQuery, conn);
    cmbl = new MySqlCommandBuilder(mydtadp);
    DataTable table = new DataTable();
    mydtadp.Fill(table);
    bindingSource1.DataSource = table;
    dataGridView1.DataSource = bindingSource1;
    dataGridView1.Columns["Id"].ReadOnly = true;
    dataGridView1.Columns["Id"].Visible = false;
    dataGridView1.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.DisplayedCells;
    dataGridView1.GridColor = Color.Brown;
    dataGridView1.Columns[7].DefaultCellStyle.ForeColor = Color.Red;

}


void FillEmployees()
{
    triger = 0;
    mydtadp4.SelectCommand = new MySqlCommand("select * from Employees", conn);
    cmbl = new MySqlCommandBuilder(mydtadp4);
    DataTable table = new DataTable();
    mydtadp4.Fill(table);
    bindingSource4.DataSource = table;
    dataGridView6.DataSource = bindingSource4;
    dataGridView6.Columns[0].Width = 0;
    dataGridView6.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.DisplayedCells;
    dataGridView6.GridColor = Color.Brown;

}

void ZapolnitClient()
{
    triger = 0;
    mydtadp2.SelectCommand = new MySqlCommand("select * from Clients", conn);
    cmb2 = new MySqlCommandBuilder(mydtadp2);
    DataTable table2 = new DataTable();
    mydtadp2.Fill(table2);
    bindingSource2.DataSource = table2;
    dataGridView2.DataSource = bindingSource2;
    dataGridView2.Columns["Id"].ReadOnly = true;
    dataGridView2.Columns["Id"].Visible = false;
    dataGridView2.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.DisplayedCells;
    dataGridView2.GridColor = Color.BlueViolet;

    dataGridView3.DataSource = bindingSource2;
    dataGridView3.Columns["Id"].ReadOnly = true;
    dataGridView3.Columns["Id"].Visible = false;
    dataGridView3.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.DisplayedCells;
    dataGridView3.GridColor = Color.BlueViolet;

}

private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
    conn.Close();
    conn.Dispose();
}

private void button6_Click(object sender, EventArgs e)
{
    mydtadp.Update((DataTable)bindingSource1.DataSource);

    MessageBox.Show("SAVED");
```

```
    }

    private void dataGridView1_SelectionChanged(object sender, EventArgs e)
    {
        pictureBox1.Image = null;
        if (triger == 1 && dataGridView1.CurrentRow != null)
        {
            ZapolnitFotoTovara();
        }

    }

    void ZapolnitFotoTovara()
    {
        string s = "";
        string base64FromDataBase;
        s = dataGridView1.CurrentRow.Cells[1].Value.ToString();
        cmd = conn.CreateCommand();
        cmd.CommandText = "Select * from Images where Tovar = '" + s + "'";
        using (DbDataReader reader = cmd.ExecuteReader())
        {
            if (reader.HasRows)
            {
                while (reader.Read())
                {
                    try
                    {
                        base64FromDataBase = reader.GetString(2);
                        var image = System.Drawing.Image.FromStream(new
MemoryStream(Convert.FromBase64String(base64FromDataBase)));
                        pictureBox1.Image = image;
                    }
                    catch
                    {

                    }
                }
            }
        }
        cmd.Dispose();
    }

    void ZapolnitFotoClienta()
    {
        string s = "";
        string base64FromDataBase;
        s = dataGridView3.CurrentRow.Cells[1].Value.ToString();
        cmd = conn.CreateCommand();
        cmd.CommandText = "Select * from photos where NameClient = '" + s + "'";
        using (DbDataReader reader = cmd.ExecuteReader())
        {
            if (reader.HasRows)
            {
                while (reader.Read())
                {
                    base64FromDataBase = reader.GetString(2);
                    var image = System.Drawing.Image.FromStream(new
MemoryStream(Convert.FromBase64String(base64FromDataBase)));
                    pictureBox2.Image = image;
                }
            }
        }
        cmd.Dispose();
    }

    private void button1_Click(object sender, EventArgs e)
    {
```

```
    mydtadp2.Update((DataTable)bindingSource2.DataSource);
    MessageBox.Show("SAVED");
}

private void button2_Click(object sender, EventArgs e)
{
    string base64;
    int t = 0;
    int rowCount;
    string s, file;
    OpenFileDialog openFileDialog1 = new OpenFileDialog();
    if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
        return;
    file = openFileDialog1.FileName;
    base64 = Convert.ToBase64String(File.ReadAllBytes(file));

    cmd = conn.CreateCommand();

    s = dataGridView3.CurrentRow.Cells[1].Value.ToString();
    cmd.CommandText = "Select * from photos where NameClient = '" + s + "'";
    DbDataReader reader = cmd.ExecuteReader();
    if (reader.HasRows)
    {
        t = 0;
    }
    else
    {
        t = 1;
    }
    cmd.Dispose();
    reader.Close();

    if (t == 0)
    {

        cmd.CommandText = "Update photos set Photo = '" + base64 + "' where NameClient = '" + s + "'";
        rowCount = cmd.ExecuteNonQuery();
    }
    else
    {

        cmd.CommandText = "INSERT INTO photos (NameClient,Photo) VALUES ('" + s + "', '" + base64 + "')";
        rowCount = cmd.ExecuteNonQuery();
    }

    cmd.Dispose();
}

private void dataGridView3_SelectionChanged(object sender, EventArgs e)
{
    pictureBox2.Image = null;
    pictureBox3.Image = null;
    if (dataGridView3.CurrentRow != null)
    {
        ZapolnitFotoClienta();
        ZapolnenieProdagClienta(dataGridView3.CurrentRow.Cells[1].Value.ToString());
        pictureBox3.Image = pictureBox2.Image;
    }

}

void LoadFotoDlaTovara()
{
    string base64;
    int t = 0;
    int rowCount;
    string s, file;
```

```
            OpenFileDialog openFileDialog1 = new OpenFileDialog();
            if (openFileDialog1.ShowDialog() == DialogResult.Cancel)
                return;
            file = openFileDialog1.FileName;
            base64 = Convert.ToBase64String(File.ReadAllBytes(file));

            cmd = conn.CreateCommand();

            s = dataGridView1.CurrentRow.Cells[1].Value.ToString();
            cmd.CommandText = "Select * from Images where Tovar = '" + s + "'";
            DbDataReader reader = cmd.ExecuteReader();
            if (reader.HasRows)
            {
                t = 0;
            }
            else
            {
                t = 1;
            }
            cmd.Dispose();
            reader.Close();

            if (t == 0)
            {

                cmd.CommandText = "Update Images set Img = '" + base64 + "' where Tovar = '" + s + "'";
                rowCount = cmd.ExecuteNonQuery();
            }
            else
            {

                cmd.CommandText = "INSERT INTO Images (Tovar,Img) VALUES ('" + s + "', '" + base64 + "')";
                rowCount = cmd.ExecuteNonQuery();
            }

            cmd.Dispose();
                    }

        private void button3_Click(object sender, EventArgs e)
        {
            AddToNakladnaya();
        }

        private void dataGridView2_SelectionChanged(object sender, EventArgs e)
        {
            if (dataGridView2.CurrentRow != null)
            {
                ClearNakladnaya();
            }
        }

        private void button4_Click(object sender, EventArgs e)
        {
            string Tovar,  NameClient;
            double Prihod, Rashod, CenaVhod, CenaRozn, quantity;
            int rowCount;
            cmd = conn.CreateCommand();
            if (dataGridView2.CurrentRow != null)
            {
                for (int i = 0; i < dataGridView4.RowCount - 1; i++)
                {

                    Tovar = dataGridView4.Rows[i].Cells[0].Value.ToString();
                    Prihod = 0;
                    Rashod = Convert.ToDouble(dataGridView4.Rows[i].Cells[3].Value.ToString());
                    NameClient = dataGridView2.CurrentRow.Cells[1].Value.ToString();
```

```
                CenaVhod = Convert.ToDouble(dataGridView4.Rows[i].Cells[1].Value.ToString());
                CenaRozn = Convert.ToDouble(dataGridView4.Rows[i].Cells[2].Value.ToString());
                quantity = Convert.ToDouble(dataGridView4.Rows[i].Cells[3].Value.ToString());

                myTrans = conn.BeginTransaction();

                cmd.CommandText = "INSERT INTO Move
(Tovar,DateMove,Prihod,Raspod,NameClient,CenaVhod,CenaRozn,quantity,Salesman,SaleTime) " +
                "VALUES ('" + Tovar + "',curtime()," + Prihod + "," + Rashod + ",'" + NameClient + "'," + CenaVhod + "," +
CenaRozn + "," + quantity + ", + '"+ Settings.login + "',CURRENT_TIMESTAMP())";
                rowCount = cmd.ExecuteNonQuery();
                cmd.CommandText = "Update tovar set OstatokK = OstatokK -" + quantity+ ",Realiz = Realiz + " + quantity + "
where Tovar = '" + Tovar  + "'" ;
                rowCount = cmd.ExecuteNonQuery();
                myTrans.Commit();

            }
            MessageBox.Show("Успішно продано");
            dataGridView1.Refresh();

            if (checkBox1.Checked) Nakladnaya();

            ClearNakladnaya();
        }

        cmd.Dispose();
    }

    private void button7_Click(object sender, EventArgs e)
    {
        WriteConfig();
        ReadConfig();
        conn = DBUtils.GetDBConnection(textBox2.Text, Convert.ToInt32(textBox3.Text), textBox4.Text, textBox5.Text,
textBox6.Text);

        try
        {

            conn.Open();
            MessageBox.Show("Connection successful!");
            ZapolnitClient();



        }
        catch
        {
            MessageBox.Show("Error Connection");
        }

    }

    private void button8_Click(object sender, EventArgs e)
    {

        mydtadp4.Update((DataTable)bindingSource4.DataSource);
        MessageBox.Show("SAVED");
    }



    void isRole()
    {
        if (Settings.eRole == "" || Settings.eRole == null)
        {
            tabControl1.SelectedIndex = 3;
        }
```

```
        if (Settings.eRole!="admin" && tabControl1.SelectedIndex==2)
        {
            tabControl1.SelectedIndex = 3;
        }
}

void isEmployeesAdmin()
{
    cmd = conn.CreateCommand();

     cmd.CommandText =
        "INSERT INTO Employees (FIO, eRole, Login, Pass)  SELECT 'admin', 'admin', 'admin'," +
        " 'admin' FROM dual WHERE NOT EXISTS( SELECT * FROM Employees WHERE eRole = 'admin')";
     cmd.ExecuteNonQuery();

    cmd.Dispose();

}


private void button9_Click(object sender, EventArgs e)
{
    string l = textBox7.Text;
    string p = textBox8.Text;
    label16.Text = "Не авторизовано";
    NoAcsessTab();
    cmd = conn.CreateCommand();
    cmd.CommandText = "Select eRole from Employees where Login = '" + l + "' AND Pass ='"+ p +"'";
    using (DbDataReader reader = cmd.ExecuteReader())
    {
        if (reader.HasRows)
        {
            reader.Read();
           if (Settings.eRole!="" && Settings.eRole != null)
            {
                WriteToLogFile("myLog.txt", Settings.eRole + " вихід з додатку " + DateTime.Now.ToString() +
                    " працював часу "
                    + ConvertSecondsToTime(Settings.WorkingTime));
            }
            Settings.eRole = reader[0].ToString();
            Settings.login = l;
            Settings.password = p;
            Settings.WorkingTime = 0;
            label16.Text = "Авторизовано, ви " + Settings.eRole;
            AcsessTab();

            WriteToLogFile("myLog.txt", Settings.eRole  + " вхід в додаток "+ DateTime.Now.ToString());
            timer1.Start();
        }
        else
        {
            Settings.eRole = "";
            Settings.login = "";
            Settings.password = "";
            WriteToLogFile("myLog.txt", Settings.eRole + " вихід з додатку " + DateTime.Now.ToString()
                + " працював часу " + ConvertSecondsToTime(Settings.WorkingTime));
            timer1.Stop();
            this.Text = "myShop";
        }
    }

    cmd.Dispose();
    if (Settings.eRole != "" && Settings.eRole != null)
        tabControl1.SelectedIndex = 1;

}
```

```
public void WriteToLogFile(string filePath, string line)
{
    if (!File.Exists(filePath))
    {
        using (StreamWriter writer = new StreamWriter(filePath))
        {
            writer.WriteLine(line);
        }
    }
    else
    {
        using (StreamWriter writer = new StreamWriter(filePath, true))
        {
            writer.WriteLine(line);
        }
    }
}

string getWorkingNameRoleAndTime()
{
    string s = "myShop    ";
    s += Settings.login + "  ";
    s += Settings.eRole + "  ";
    s += "працює ";
    s += ConvertSecondsToTime(Settings.WorkingTime);
    return s;

}

public string ConvertSecondsToTime(int seconds)
{
    TimeSpan time = TimeSpan.FromSeconds(seconds);
    return string.Format("{0:D2}:{1:D2}:{2:D2}",
        (int)time.TotalHours,
        time.Minutes,
        time.Seconds);
}

private void pictureBox1_DoubleClick(object sender, EventArgs e)
{
    if (dataGridView1.CurrentRow!=null)
    {

        LoadFotoDlaTovara();

    }
}



private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (triger == 1)
    {
        FillProdagi();
    }

}

private void textBox9_TextChanged(object sender, EventArgs e)
{
    triger = 0;
    ZapolnitTovar(textBox9.Text, textBox10.Text, "", false);
    triger = 1;
}

private void textBox10_TextChanged(object sender, EventArgs e)
```

```
        {
            triger = 0;
            ZapolnitTovar(textBox9.Text, textBox10.Text, "", false);
            triger = 1;
        }



        private void Form1_ResizeEnd(object sender, EventArgs e)
        {
            this.Width = 1500;
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            Settings.WorkingTime++;
            this.Text =   getWorkingNameRoleAndTime();
                    }

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            WriteToLogFile("myLog.txt", Settings.eRole + " вихід з додатку " + DateTime.Now.ToString() + " працював часу
"+ ConvertSecondsToTime(Settings.WorkingTime));
        }

        void ClearNakladnaya()
        {
            while (dataGridView4.ColumnCount != 0)
            {
                dataGridView4.Columns.RemoveAt(0);
            }
            dataGridView4.Columns.Add("Tovar", "Tovar");
            dataGridView4.Columns.Add("Vhod", "Vhod");
            dataGridView4.Columns.Add("Rozn", "Rozn");
            dataGridView4.Columns.Add("Q", "Q");
            dataGridView4.Columns.Add("Sum", "Sum");
            dataGridView4.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.DisplayedCells;
        }

        private void Form1_Resize(object sender, EventArgs e)
        {
            tabControl1.Width = this.Width;
            tabControl1.Height = this.Height-50;
            tabControl1.Top = 0;
        }

        private void dateTimePicker1_ValueChanged(object sender, EventArgs e)
        {
            FillProdagi();
        }

        private void dateTimePicker2_ValueChanged(object sender, EventArgs e)
        {
            FillProdagi();
        }

        private void textBox11_Enter(object sender, EventArgs e)
        {


        }

        private void textBox11_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Enter)
            {
```

```
            ScanSale();
        }
    }

    void ScanSale()
    {
        ZapolnitTovar("", "", textBox11.Text, true);
        if (dataGridView1.RowCount > 1)
        {
            dataGridView1.CurrentCell = dataGridView1.Rows[0].Cells[3];
            if (dataGridView1.RowCount == 2)
            {
                AddToNakladnaya();
                textBox11.Text = "";
            }
            else
            {
                button3.Select();
            }

        }
    }

    void ZapolnenieProdagClienta(string s)
    {
        mydtadp3.SelectCommand = new MySqlCommand("select DateMove,Tovar,CenaVhod,CenaRozn,quantity from
Move where NameClient = '" + s +"'", conn);
        cmb3 = new MySqlCommandBuilder(mydtadp3);
        DataTable table3 = new DataTable();
        mydtadp3.Fill(table3);
        bindingSource3.DataSource = table3;
        dataGridView5.DataSource = bindingSource3;
        dataGridView5.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.DisplayedCells;
        dataGridView5.GridColor = Color.BlueViolet;
    }

    private void button5_Click(object sender, EventArgs e)
    {
        double sum = 0;
        DateTime today = DateTime.Now;
        DateTime startOfDay = new DateTime(today.Year, today.Month, today.Day, 0, 0, 0);
        DateTime endOfDay = startOfDay.AddDays(1);

        cmd = conn.CreateCommand();
        cmd.CommandText = "SELECT SUM(quantity * CenaRozn) FROM move WHERE quantity < 0 AND SaleTime >=
@StartOfDay AND SaleTime < @EndOfDay";
        cmd.Parameters.AddWithValue("@StartOfDay", startOfDay);
        cmd.Parameters.AddWithValue("@EndOfDay", endOfDay);
        object result = cmd.ExecuteScalar();
        if (result != null && result != DBNull.Value)
        {
            sum = Convert.ToDouble(result);
        }
        cmd.Dispose();


        XmlDocument xmlDocument = new XmlDocument();
        XmlElement rootElement = xmlDocument.CreateElement("Report");
        XmlElement sumElement = xmlDocument.CreateElement("Sum");
        sumElement.InnerText = sum.ToString();
        XmlElement dateElement = xmlDocument.CreateElement("Date");
        dateElement.InnerText = today.ToString();
        XmlElement fopIdElement = xmlDocument.CreateElement("FOPID");
        fopIdElement.InnerText = "123123";
        rootElement.AppendChild(sumElement);
        rootElement.AppendChild(dateElement);
        rootElement.AppendChild(fopIdElement);
```

```
            xmlDocument.AppendChild(rootElement);
            xmlDocument.Save("report.xml");

            MessageBox.Show("Звіт за день сформовано");
        }

        void AddToNakladnaya()
        {
            if (dataGridView1.CurrentRow != null)
            {

                double value1,value2;
                if (Double.TryParse(textBox1.Text, out value1)
                    && Double.TryParse(dataGridView1.CurrentRow.Cells[3].Value.ToString(), out value2))
                {
                    dataGridView4.Rows.Add(" ");
                    dataGridView4.Rows[dataGridView4.RowCount - 2].Cells[0].Value
                        = dataGridView1.CurrentRow.Cells[1].Value.ToString();

                    dataGridView4.Rows[dataGridView4.RowCount - 2].Cells[1].Value
                        = dataGridView1.CurrentRow.Cells[2].Value.ToString();

                    dataGridView4.Rows[dataGridView4.RowCount - 2].Cells[2].Value
                        = dataGridView1.CurrentRow.Cells[3].Value.ToString();

                    dataGridView4.Rows[dataGridView4.RowCount - 2].Cells[3].Value
                        = textBox1.Text;

                    dataGridView4.Rows[dataGridView4.RowCount - 2].Cells[4].Value
                        = value1 * value2;

                }
            }
        }

        public void Nakladnaya()
        {
            double Sum = 0;
            string tempImagePath = "1.jpg";
            string pdfPath = "Накладная.pdf";

            iTextSharp.text.Document document = new iTextSharp.text.Document();
            PdfWriter writer = PdfWriter.GetInstance(document, new FileStream(pdfPath, FileMode.Create));
            document.Open();

            var baseFont = BaseFont.CreateFont(@"c:\windows\fonts\arial.ttf", BaseFont.IDENTITY_H,
BaseFont.EMBEDDED);

            var font = new iTextSharp.text.Font(baseFont, 16, iTextSharp.text.Font.BOLD);
            var paragraph = new Paragraph("Видаткова накладна !", font);
            paragraph.Alignment = Element.ALIGN_CENTER;
            paragraph.SpacingAfter = 10f;
            document.Add(paragraph);

            font = new iTextSharp.text.Font(baseFont, 14, iTextSharp.text.Font.NORMAL);
            paragraph = new Paragraph(DateTime.Now.ToString(), font);
            paragraph.Alignment = Element.ALIGN_CENTER;
            paragraph.SpacingAfter = 10f;
            document.Add(paragraph);

            string NameClient ="Клієнт: "+dataGridView2.CurrentRow.Cells[1].Value.ToString();
            font = new iTextSharp.text.Font(baseFont, 14, iTextSharp.text.Font.NORMAL);
            paragraph = new Paragraph(NameClient, font);
            paragraph.Alignment = Element.ALIGN_LEFT;
            paragraph.SpacingAfter = 20f;
            document.Add(paragraph);
```

```
            font = new iTextSharp.text.Font(baseFont, 12, iTextSharp.text.Font.NORMAL);
            PdfPTable table = new PdfPTable(4);
            table.WidthPercentage = 100;
            float[] columnWidths = {0.55f, 0.15f, 0.15f, 0.15f };
            table.SetWidths(columnWidths);

            table.AddCell(new PdfPCell(new Phrase("Товар", font)) { HorizontalAlignment = Element.ALIGN_CENTER });
            table.AddCell(new PdfPCell(new Phrase("Кількість", font)) { HorizontalAlignment = Element.ALIGN_CENTER });
            table.AddCell(new PdfPCell(new Phrase("Ціна", font)) { HorizontalAlignment = Element.ALIGN_CENTER });
            table.AddCell(new PdfPCell(new Phrase("Сума", font)) { HorizontalAlignment = Element.ALIGN_CENTER });


            for (int i = 0; i < dataGridView4.RowCount - 1; i++)
            {
                string Tovar = dataGridView4.Rows[i].Cells[0].Value.ToString();
                double CenaRozn = Convert.ToDouble(dataGridView4.Rows[i].Cells[2].Value.ToString());
                double quantity = Convert.ToDouble(dataGridView4.Rows[i].Cells[3].Value.ToString());

                table.AddCell(new PdfPCell(new Phrase(Tovar, font)) { HorizontalAlignment = Element.ALIGN_LEFT });
                table.AddCell(quantity.ToString());
                table.AddCell(CenaRozn.ToString());
                table.AddCell((quantity * CenaRozn).ToString());
                Sum += quantity * CenaRozn;

            }

            document.Add(table);

            string SummString = "Разом: "+ Sum;
            font = new iTextSharp.text.Font(baseFont, 14, iTextSharp.text.Font.NORMAL);
            paragraph = new Paragraph(SummString, font);
            paragraph.Alignment = Element.ALIGN_LEFT;
            paragraph.SpacingBefore = 20f;
            document.Add(paragraph);

            iTextSharp.text.Image image = iTextSharp.text.Image.GetInstance(tempImagePath);
            image.ScaleAbsolute(4f * 72f / 2.54f, 4f * 72f / 2.54f);
            image.Alignment = iTextSharp.text.Image.ALIGN_RIGHT;
            document.Add(image);

            document.Close();
            writer.Close();
            System.Diagnostics.Process.Start(pdfPath);
            MessageBox.Show("Отчет сформирован");
        }

    }
private void button11_Click(object sender, EventArgs e)
    {
        int x=0;
        chart1.Series.Clear();
        foreach (var selectedIndex in checkedListBox1.CheckedIndices)
        {
            int selectedRowIndex = (int)selectedIndex;
            string[] names = checkedListBox1.Items[selectedRowIndex].
                ToString().Split(new string[] { ";scanId:" }, StringSplitOptions.None);

            string selectedName = names[0];
            string scanId = names[1];

            System.Windows.Forms.DataVisualization.Charting.Series newSeries =
                new System.Windows.Forms.DataVisualization.Charting.Series(selectedName);
            if (radioButton1.Checked)
            {
                newSeries.ChartType = SeriesChartType.Spline;
                Color transparentColor = Color.FromArgb(128, newSeries.Color);
                newSeries.Color = transparentColor;
```

```
            }
            if (radioButton2.Checked)
            {
                newSeries.ChartType = SeriesChartType.Column;
            }
            newSeries.BorderWidth = 2;
            ForCorrelacia correl = Correl.Find(p=>p.Id== scanId);
            foreach (DateAndValue dt in correl.Values)
            {
                newSeries.Points.AddXY(dt.Date.ToString("dd.MM"), dt.Value);
                x++;
            }

            chart1.Series.Add(newSeries);
        }

    }

    private void button10_Click(object sender, EventArgs e)
    {
        FillProdagiCorel();

    }


    void FillProdagiCorel()
    {
        Correl = new List<ForCorrelacia>();
        checkedListBox1.Items.Clear();
        DateTime dateMin, dateMax;
        dateMin = dateTimePicker3.Value.Date;
        dateMax = dateTimePicker4.Value.Date;
        cmd = conn.CreateCommand();
        cmd.CommandText = "SELECT Distinct Tovar, ScanId FROM move WHERE SaleTime BETWEEN @dateMin
AND @dateMax";
        cmd.Parameters.AddWithValue("@dateMin", dateMin);
        cmd.Parameters.AddWithValue("@dateMax", dateMax);

        using (DbDataReader reader = cmd.ExecuteReader())
        {
            if (reader.HasRows)
            {
                while (reader.Read())
                {
                    Correl.Add(new ForCorrelacia(reader[0].ToString(),
                      reader[1].ToString(),
                        dateMin, dateMax));
                    checkedListBox1.Items.Add(reader[0].ToString()+";scanId:"+ reader[1].ToString());
                }

            }
        }
        cmd.Dispose();

        foreach (ForCorrelacia correl in Correl)
        {
            DateTime temp = dateMin;

            while (temp <= dateMax)
            {
                cmd = conn.CreateCommand();
                cmd.CommandText = "SELECT Sum(Raspod) FROM move WHERE Raspod>0 and ScanId = "
                    + correl.Id
                    + " and (SaleTime BETWEEN @dateMin AND @dateMax)";
                cmd.Parameters.AddWithValue("@dateMin", temp);
                cmd.Parameters.AddWithValue("@dateMax", temp.AddDays(1));
```

```
            using (DbDataReader reader = cmd.ExecuteReader())
            {
                if (reader.HasRows)
                {
                    reader.Read();

                        DateAndValue TargetCorrel = correl.Values.Find(p => p.Date >= temp
                        && p.Date<temp.AddDays(1));
                        if (TargetCorrel != null)
                        {
                          if (double.TryParse(reader[0].ToString(),out TargetCorrel.Value))
                          {

                          }
                        }

                }
            }
            cmd.Dispose();
            temp=temp.AddDays(1);
        }
    }
}

private void button12_Click(object sender, EventArgs e)
{
    Tuple<ForCorrelacia, ForCorrelacia> maxCorrelationClasses = FindMaxCorrelation(Correl);
    richTextBox1.Text = "";
    if (maxCorrelationClasses != null)
    {
        ForCorrelacia class1 = maxCorrelationClasses.Item1;
        ForCorrelacia class2 = maxCorrelationClasses.Item2;

        richTextBox1.Text += "Максимальная корреляція между продажами:\n";
        richTextBox1.Text += class1.Name + "\n";
        richTextBox1.Text += class2.Name + "\n";
        richTextBox1.Text += "Дорівнює " + class1.BestValue;

    }
    else
    {
        richTextBox1.Text += "Помилка\n";
    }
}

public Tuple<ForCorrelacia, ForCorrelacia> FindMaxCorrelation(List<ForCorrelacia> correlList)
{
    double maxCorrelation = 0;
    ForCorrelacia maxCorrelationClass1 = null;
    ForCorrelacia maxCorrelationClass2 = null;

    for (int i = 0; i < correlList.Count - 1; i++)
    {
        for (int j = i + 1; j < correlList.Count; j++)
        {
            double correlation = CalculateCorrelation(correlList[i].Values, correlList[j].Values);
            if (correlation > maxCorrelation)
            {
                maxCorrelation = correlation;
                maxCorrelationClass1 = correlList[i];
                maxCorrelationClass2 = correlList[j];
                maxCorrelationClass1.BestValue = correlation;
                maxCorrelationClass2.BestValue = correlation;
                maxCorrelationClass1.BestId = maxCorrelationClass2.Id;
                maxCorrelationClass2.BestId = maxCorrelationClass1.Id;
            }
        }
```

```
        }

        return new Tuple<ForCorrelacia, ForCorrelacia>(maxCorrelationClass1, maxCorrelationClass2);
    }

    public double CalculateCorrelation(List<DateAndValue> values1, List<DateAndValue> values2)
    {
        double[] firstValues = new double[values1.Count];
        double[] secondValues = new double[values1.Count];
        int i = 0;
        foreach (DateAndValue value in values1)
        {
            firstValues[i] = value.Value;
            i++;
        }
        i = 0;
        foreach (DateAndValue value in values2)
        {
            secondValues[i] = value.Value;
            i++;
        }
        double meanFirst = firstValues.Sum() / firstValues.Length;
        double meanSecond = secondValues.Sum() / secondValues.Length;
        double sumProduct = 0;
        double sumSquareFirst = 0;
        double sumSquareSecond = 0;
        for (i = 0; i < firstValues.Length; i++)
        {
            double deviationFirst = firstValues[i] - meanFirst;
            double deviationSecond = secondValues[i] - meanSecond;

            sumProduct += deviationFirst * deviationSecond;
            sumSquareFirst += deviationFirst * deviationFirst;
            sumSquareSecond += deviationSecond * deviationSecond;
        }

        double correlation = sumProduct / Math.Sqrt(sumSquareFirst * sumSquareSecond);

        return correlation;
    }
}
        }
```