Державний торговельно-економічний університет

Кафедра інженерії програмного забезпечення та кібербезпеки

Захищено на кафедрі інженерії

програмного забезпечення та кібербезпеки

«\_»\_\_\_2024 р.

Підпис членів комісії:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**КУРСОВА РОБОТА**

**З ДИСЦИПЛІНИ «БАЗИ ДАНИХ»**

**НА ТЕМУ:**

База даних управління продажем квитків на сайті аеропорту «Жуляни»

Студентки факультету   
інформаційних технологій

групи\_курсу 3-4

Авєріної Наталії Ігорівни

Науковий керівник

Старший викладач кафедри інженерії програмного забезпечення

Гнатченко Дмитро Дмитрович

Київ 2024

**Київський національний торговельно-економічний університет**Кафедра інженерії програмного забезпечення та кібербезпекиДисципліна «Бази даних»Напрям/cпеціальність інженерія програмного забезпечення  
Курс 3 Група 4 Семестр 6

**ЗАВДАННЯ  
на курсову роботу (проект) студента**Авєріної Наталії Ігорівни

1. Тема курсової роботи (проекту) «База даних управління продажем квитків на сайті аеропорту «Жуляни»»
2. План курсової роботи (проекту)
   1. Теоретичні аспекти розробки бази даних управління продажем квитків на авіаперевезення.
   2. Процес розробки бази даних управління продажем квитків.
   3. Процес розробки клієнтського додатку для взаємодії з базою даних.
3. Перелік графічного матеріалу
   1. 1 порівняльна таблиця.
   2. 3 моделі бази даних.
   3. 17 знімків екрану.
4. Термін подання студентом завершеної курсової роботи (проекту) на кафедру \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Термін захисту курсової роботи (проекту)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Дата видачі завдання \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Науковий керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище, ім'я, по батькові)

Завідувач кафедри \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище, ім'я, по батькові)

**Київський національний торговельно-економічний університет**

**Рецензія на курсову роботу (проект) і результат захисту**

Студентки **\_\_\_\_\_\_\_**Авєріної Наталії Ігорівни**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(прізвище, ім'я та по батькові)

\_\_\_3\_\_ курсу \_\_4\_\_\_ групи \_\_\_\_\_\_\_ФІТ\_\_\_\_\_\_\_\_\_\_\_\_\_\_факультету

Курсова робота (проект) з \_\_\_\_\_\_\_\_\_\_ «Бази даних»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(назва навчальної дисципліни)

Тема «База даних управління продажем квитків на сайті аеропорту «Жуляни»»

Реєстраційний № \_\_\_\_, дата одержання “\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_ р.

Науковий керівник \_\_\_\_\_\_\_\_\_Ст. викладач Гнатченко Д.Д.\_\_\_\_\_\_\_\_\_

(вчене звання, прізвище, ініціали)

**Зміст рецензії**

Допущено до захисту “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_ р.

Захист планується о \_\_\_\_\_\_\_\_\_\_\_\_\_ “\_\_\_\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_ р.

(час)

(місце роботи комісії)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис наукового керівника)

Курсова робота захищена “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_ р.

з оцінкою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(за шкалою КНТЕУ, національною шкалою та шкалою ЄКТС)

**Комісія:**

1. \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище, ініціали)

1. \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище, ініціали)

1. \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище, ініціали)

**ЗМІСТ**

[ВСТУП 5](#_Toc163532633)

[РОЗДІЛ 1 ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ БАЗИ ДАНИХ УПРАВЛІННЯ ПРОДАЖЕМ КВИТКІВ НА АВІАПЕРЕВЕЗЕННЯ 7](#_Toc163532634)

[1.1. Аналіз веб-сайтів з продажу квитків 7](#_Toc163532635)

[1.2. Оптимізація бізнес-процесів в авіаційній галузі через використання баз даних 8](#_Toc163532636)

[1.3. Принципи проєктування баз даних 9](#_Toc163532637)

[1.4. Проблемні питання та дискусійні аспекти баз даних з продажу квитків 9](#_Toc163532638)

[РОЗДІЛ 2 ПРОЦЕС РОЗРОБКИ БАЗИ ДАНИХ УПРАВЛІННЯ ПРОДАЖЕМ КВИТКІВ 12](#_Toc163532639)

[2.1. Планування та моделювання бази даних управління продажем квитків 12](#_Toc163532640)

[2.2 Розробка бази даних та заповнення таблиць даними 16](#_Toc163532641)

[2.3 Реалізація представлень (запитів) до бази даних 17](#_Toc163532642)

[РОЗДІЛ 3 ПРОЦЕС РОЗРОБКИ КЛІЄНТСЬКОГО ДОДАТКУ ДЛЯ ВЗАЄМОДІЇ З БАЗОЮ ДАНИХ 21](#_Toc163532643)

[3.1 Вибір технологій для створення додатку 21](#_Toc163532644)

[3.2 Розробка клієнтського додатку мовою C# на основі інтерфейсу Windows Forms 21](#_Toc163532645)

[3.3 Функціонал додатку TicketsDataManager 22](#_Toc163532646)

[ВИСНОВКИ 29](#_Toc163532647)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 30](#_Toc163532648)

[ДОДАТКИ 32](#_Toc163532649)

# **ВСТУП**

Незважаючи на тимчасові обмеження авіаперевезень, спричинені воєнним станом в Україні, потреба в розробці та впровадженні сучасних технологій управління авіаперевезеннями залишається важливою, щоб забезпечити ефективне функціонування авіаційної галузі у майбутньому. Одним із важливих аспектів є електронна система продажу квитків, яка має готуватися й адаптуватися до майбутніх потреб пасажирів.

До 24 лютого 2022 року в Україні для купівлі авіаквитків використовували переважно такі онлайн-сервіси як *avia.tickets.ua*, *booking.com*, *kiwi.com* та *aviasales.ua*, які дозволяли придбати квиток на виліт з будь-якого аеропорту країни та світу. Наразі ці сервіси не так актуальні для українців, але після відновлення авіаперевезень – будуть. Та, на жаль, деякі з них можуть не одразу повернутися до співпраці з українськими аеропортами через складність приєднання та оновлення баз даних, тому сайт аеропорту може стати єдиною надійною альтернативою онлайн купівлі.

Мета курсової роботи полягає у дослідженні, проектуванні та реалізації бази даних для оптимізації роботи веб-сайту аеропорту "Жуляни".

Завданням курсової роботи є розробка та належне проєктування бази даних, оскільки від її структури та організації залежить швидкодія та ефективність роботи веб-сайту.

Об’єктом дослідження є управління продажем авіаквитків. Предмет дослідження – управління продажем квитків на сайті аеропорту «Жуляни».

Методи дослідження, які були використані в ході написання курсової роботи:

1. Аналіз літературних джерел, який передбачав вивчення наукової літератури, статей, публікацій та документацій, що стосуються проєктування баз даних та процесів авіаперевезень.
2. Метод моделювання, що включає в себе використання CASE-технології draw.io для створення концептуальної, логічної та фізичної моделей бази даних.
3. Аналіз реальних веб-сайтів аеропортів для отримання практичного досвіду та вивчення сучасних підходів у сфері авіаперевезень.

# **РОЗДІЛ 1 ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ БАЗИ ДАНИХ УПРАВЛІННЯ ПРОДАЖЕМ КВИТКІВ НА АВІАПЕРЕВЕЗЕННЯ**

* 1. **Аналіз веб-сайтів з продажу квитків**

Сучасний ринок онлайн-продажу квитків для авіаперевезень представлений численними платформами, які конкурують між собою за увагу користувачів. Серед лідерів українського ринку є *avia.tickets.ua*, *booking.com*, *kiwi.com* та *aviasales.ua*.

Аналізуючи ці веб-сайти, можна виявити різноманітні функціональні можливості, які вони пропонують. Наприклад, *avia.tickets.ua* пропонують різноманітні опції пошуку та фільтрації, що дозволяє знаходити оптимальні пропозиції для подорожей. *booking.com*, крім авіаквитків, також спеціалізується на готельних бронюваннях та інших послугах для подорожей. Вони надають користувачам широкий вибір варіантів та можливість порівняння цін. *kiwi.com* дозволяє знаходити оптимальні маршрути та комбінувати різні перельоти для економії часу та коштів. *aviasales.ua* надають широкий вибір варіантів перельотів і забезпечують зручний інтерфейс для користувачів. Враховуючи різноманітність і функціональність цих платформ, можна зробити висновок про активність та конкуренцію на ринку онлайн-продажу авіаквитків. Кожна з цих платформ пропонує свої унікальні переваги та сервіси для задоволення потреб користувачів.

Для того, щоб забезпечити конкурентоспроможність сайту аеропорту «Жуляни», можна розглянути такі заходи:

1. Додавання нових сервісів та можливостей на сайті, таких як онлайн-бронювання готелів або автомобілів, може привернути більше користувачів.
2. Забезпечення швидкого зворотного зв’язку з клієнтами через сайт.
3. Проведення рекламних кампаній, надання спеціальних пропозицій, акцій для пасажирів, які купують квитки через сайт.
   1. **Оптимізація бізнес-процесів в авіаційній галузі через використання баз даних**

Використання баз даних для оптимізації бізнес-процесів може включати такі аспекти, як автоматизація рутинних операцій, підвищення швидкодії обробки даних, зниження витрат на обслуговування та підвищення якості обслуговування пасажирів [2]. Необхідно також враховувати аспекти масштабованості та гнучкості баз даних. З ростом обсягу даних та розширенням бізнесу авіакомпанії повинні мати можливість легко масштабувати свої бази даних, додавати нові функціональні можливості та адаптуватися до змін у вимогах ринку [1]. Додатково, використання баз даних може сприяти збільшенню ефективності управління авіакомпанією шляхом забезпечення доступу до актуальної та достовірної інформації [4]. Це дозволяє керівництву приймати обґрунтовані рішення на основі аналізу даних про виробництво, фінансові показники, попит на рейси та інші аспекти діяльності авіакомпанії. Крім того, бази даних можуть впливати на стратегію ціноутворення, допомагаючи визначити оптимальні тарифи та пропозиції для різних категорій пасажирів.

Ще однією важливою перевагою є можливість покращення обслуговування пасажирів завдяки персоналізованому підходу. Збір та аналіз даних про пасажирів дозволяє авіакомпаніям надавати індивідуальні пропозиції, забезпечуючи високий рівень задоволеності клієнтів. Наприклад, системи управління базами даних можуть автоматично враховувати попередні уподобання пасажирів щодо місця в салоні, харчування та інші послуги.

Таким чином, оптимізація бізнес-процесів управління авіаквитками через використання баз даних включає в себе широкий спектр заходів, спрямованих на підвищення ефективності, безпеки та конкурентоспроможності авіакомпаній.

* 1. **Принципи проєктування баз даних**

Проєктування баз даних для управління продажем квитків – це складний процес, який передбачає визначення структури даних, взаємозв'язків між ними та способів доступу до цих даних. Основними принципами проєктування баз даних є нормалізація, унікальність, цілісність та ефективність.

Нормалізація баз даних допомагає уникнути аномалій при зберіганні даних та забезпечує їхню консистентність. Вона дозволяє розділити дані на логічні таблиці з мінімальним дублюванням інформації, що сприяє ефективному зберіганню та обробці даних. Унікальність полягає в тому, що кожен запис у базі даних має свій унікальний ідентифікатор, який дозволяє однозначно ідентифікувати цей запис. Цілісність забезпечується за допомогою різних обмежень та правил, які гарантують правильність та цілісність даних в базі [3]. Щодо ефективності, важливо враховувати такі аспекти, як швидкодія та оптимізація запитів до бази даних. Для цього можуть використовуватися різні методи, включаючи індексацію, кешування та денормалізацію даних в деяких випадках [4]. Принципи суб'єктивності в дизайні баз даних дозволяють адаптувати структуру бази даних до реального світу та потреб користувачів. Це означає, що при проєктуванні бази даних слід брати до уваги специфіку бізнесу та вимоги користувачів, щоб забезпечити оптимальну функціональність та зручність використання.

Отже, при проєктуванні баз даних для управління продажем квитків важливо дотримуватися зазначених принципів, щоб забезпечити ефективну та надійну роботу системи.

* 1. **Проблемні питання та дискусійні аспекти баз даних з продажу квитків**

У контексті бази даних управління продажем авіаквитків, проблеми забезпечення швидкодії, надійності та гнучкості являються ключовими [7]. Порівнюючи різні технології та підходи до оптимізації баз даних у галузі авіаперевезень, можна виділити переваги та недоліки кожного з них. Реляційні бази даних, з одного боку, забезпечують структурованість та надійність, проте вимагають складних операцій з обробки великого обсягу даних [8]. Нереляційні бази даних, з іншого боку, можуть забезпечити більшу швидкодію, але при цьому можуть виникати проблеми з надійністю та консистентністю даних [9] (Таблиця 1).

Таблиця 1. Порівняльна характеристика ієрархічної, мережної та реляційної моделей баз даних.

|  |  |  |  |
| --- | --- | --- | --- |
| Характеристика | Ієрархічна модель | Мережна модель | Реляційна модель |
| **Структура даних** | Деревоподібна | Графова | Таблична |
| **Простота** | Висока | Середня | Висока |
| **Гнучкість** | Низька | Висока | Висока |
| **Зручність** | Низька | Середня | Висока |
| **Нормалізація** | Неможлива | Неможлива | Можлива |
| **Швидкодія** | Висока | Помірна | Помірна |
| **Надійність** | Висока | Середня | Висока |
| **Застосування** | Обмежене | Середнє | Широке |

В нашому випадку, реляційна база даних має більше переваг. По-перше, реляційні бази даних забезпечують структурованість даних шляхом організації їх у відповідні таблиці з визначеними відносинами між ними [10]. Це дозволяє зберігати дані про пасажирів, рейси, квитки та інші аспекти авіаперевезень у логічно впорядкованому вигляді. По-друге, реляційні бази даних забезпечують надійність даних через механізми транзакцій та контроль цілісності [5]. Це особливо важливо в авіаційній індустрії, де непередбачуваність може призвести до серйозних наслідків. По-третє, реляційні бази даних підтримують складні операції та запити, що часто використовуються в системах управління продажем авіаквитків, такі як звіти, аналітика та пошукові запити [11].

**Висновки до Розділу 1.**

Сучасний ринок онлайн-продажу квитків для авіаперевезень представлений численними платформами, які активно конкурують між собою за увагу користувачів. Кожна з них має свої унікальні переваги, що відповідають потребам користувачів. Для забезпечення конкурентоспроможності варто розглядати різноманітні заходи, такі як додавання нових сервісів на сайті, забезпечення швидкого зворотного зв’язку та проведення рекламних кампаній.

Питання оптимізації бізнес-процесів управління авіаквитками виявляється надзвичайно важливим. Використання баз даних дозволяє автоматизувати операції, підвищує швидкість обробки даних, знижує витрати та підвищує якість обслуговування пасажирів. Масштабованість баз даних дозволяє легко адаптуватися до змін на ринку, а персоналізований підхід сприяє покращенню задоволення клієнтів.

Проєктування баз даних для управління продажем квитків є складним процесом, який вимагає врахування різних аспектів, таких як нормалізація, унікальність, цілісність та ефективність. Дотримання цих принципів допомагає уникнути аномалій, забезпечити консистентність даних, а також забезпечити швидку та ефективну роботу системи.

Дослідження проблемних питань та дискусійних аспектів показало, що існують виклики у збереженні та аналізі даних про пасажирів та рейси, а також у впровадженні інноваційних рішень у цій сфері. Було виявлено, що реляційні бази даних являються більш вигідними у порівнянні з ієрархічною та мережною моделями. Вони забезпечують структурованість, надійність, і дозволяють виконувати складні операції та запити, що є важливими в авіаційній індустрії.

# **РОЗДІЛ 2 ПРОЦЕС РОЗРОБКИ БАЗИ ДАНИХ УПРАВЛІННЯ ПРОДАЖЕМ КВИТКІВ**

## **2.1.** **Планування та моделювання бази даних управління продажем квитків**

Для розробки та впровадження бази даних необхідно здійснити докладну постановку задачі. Вона включає в себе такі пункти:

1. Аналіз потреб та вимог для сайту аеропорту «Жуляни».
2. Визначення концепції створення бази даних.
3. Визначення обсягу бази даних, включаючи типи даних.
4. Функціональні вимоги:
   1. Зберігання інформації про рейси, відправлення, аеропорти призначення, міста тощо.
   2. Облік клієнтів та пасажирів, а саме їх особисті дані, інформація про замовлення та квитки (клас, ціна, місце).
   3. Обробка даних та створення звітності про продажі
5. Технічні вимоги:
   1. Забезпечення швидкого доступу до інформації та можливість масштабування системи при збільшенні обсягу даних та користувачів.
   2. Мінімізація можливості виникнення помилок та збереження цілісності даних.
6. Технічні засоби:
   1. Використання системи управління реляційними базами даних Microsoft SQL Server для забезпечення структурованості та надійності даних.
   2. Розробка користувацього інтерфейсу у вигляді Windows Form для зручного доступу адміністраторів до системи.

Для створення бази даних управління продажем квитків було розроблено концептуальну модель, яка описує структуру та взаємозв’язки між сутностями в системі на вищому рівні абстракції.

Основною концепцією створення бази даних є зберігання даних про замовлені квитки на сайті аеропорту. Відповідно до цього, розроблена концептуальна модель (див. рис. 2.1.) має 4 сутності, які поєднані між собою відношеннями «один-до-багатьох»: 1 користувач може зробити *n* кількість замовлень, *n* кількість замовлень можуть бути оформлені на 1 рейс, 1 аеропорт може бути точкою прибуття для *n* кількості рейсів.



Рис. 2.1. Концептуальна модель бази даних управління продажем квитків

Для того, щоб відношення були нормалізованими, потрібна декомпозиція наявних відношень на дві або декілька проекцій. Наприклад, в нашій моделі є сутність «Замовлення квитка». В замовленні може міститься декілька квитків, в яких будуть різні пасажири, місця тощо. Для того, щоб атрибути були лише зі скалярними значеннями та не повторювались в таблиці, було створено окрему сутність «Квиток», яка в свою чергу містить атрибути з інформацією про пасажира (користувача), рейс, клас тощо.

Так само сутність «Аеропорт призначення» має дані про місто та країну призначення. В одній країні може бути декілька міст, які в свою чергу можуть мати декілька аеропортів. Для вилучення повторень створено «Довідник міст» та «Довідник країн».

Логічна модель (див. рис. 2.2.) визначає структуру та відносини між даними в базі даних, не звертаючи увагу на деталі їх зберігання або реалізацію в реляційних таблицях.

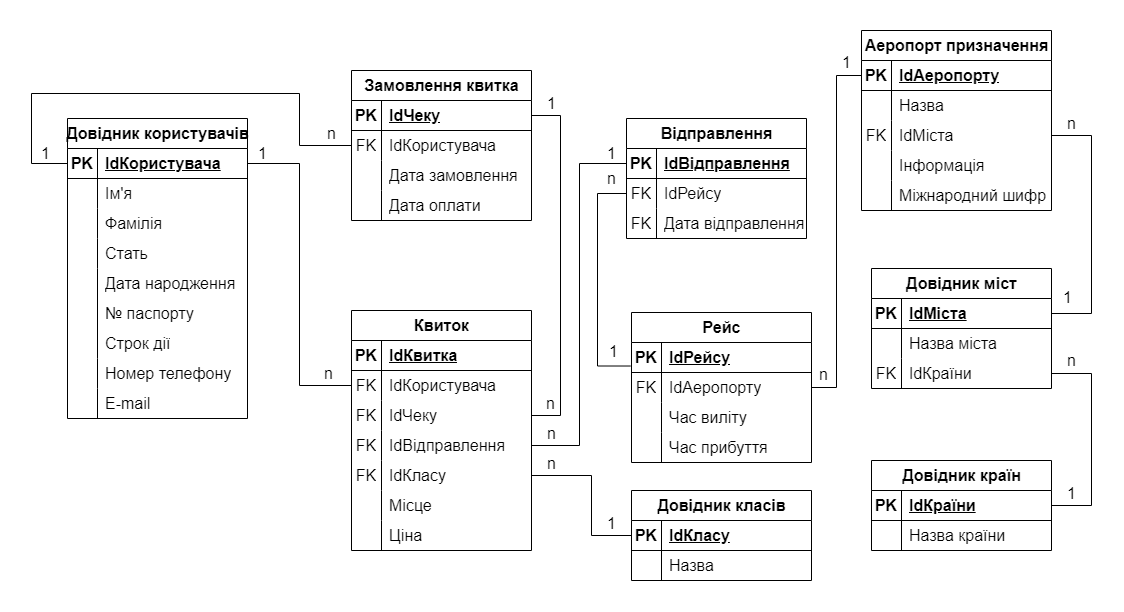


Рис. 2.2. Логічна модель бази даних управління продажем квитків

Фізична модель бази даних – це деталізоване представлення бази даних конкретною реалізацією для обраної системи керування базами даних (СКБД). В нашому випадку – для Microsoft SQL Server. Фізична модель (рисунок 2.3.) визначає тип даних кожного поля та Null-значення.

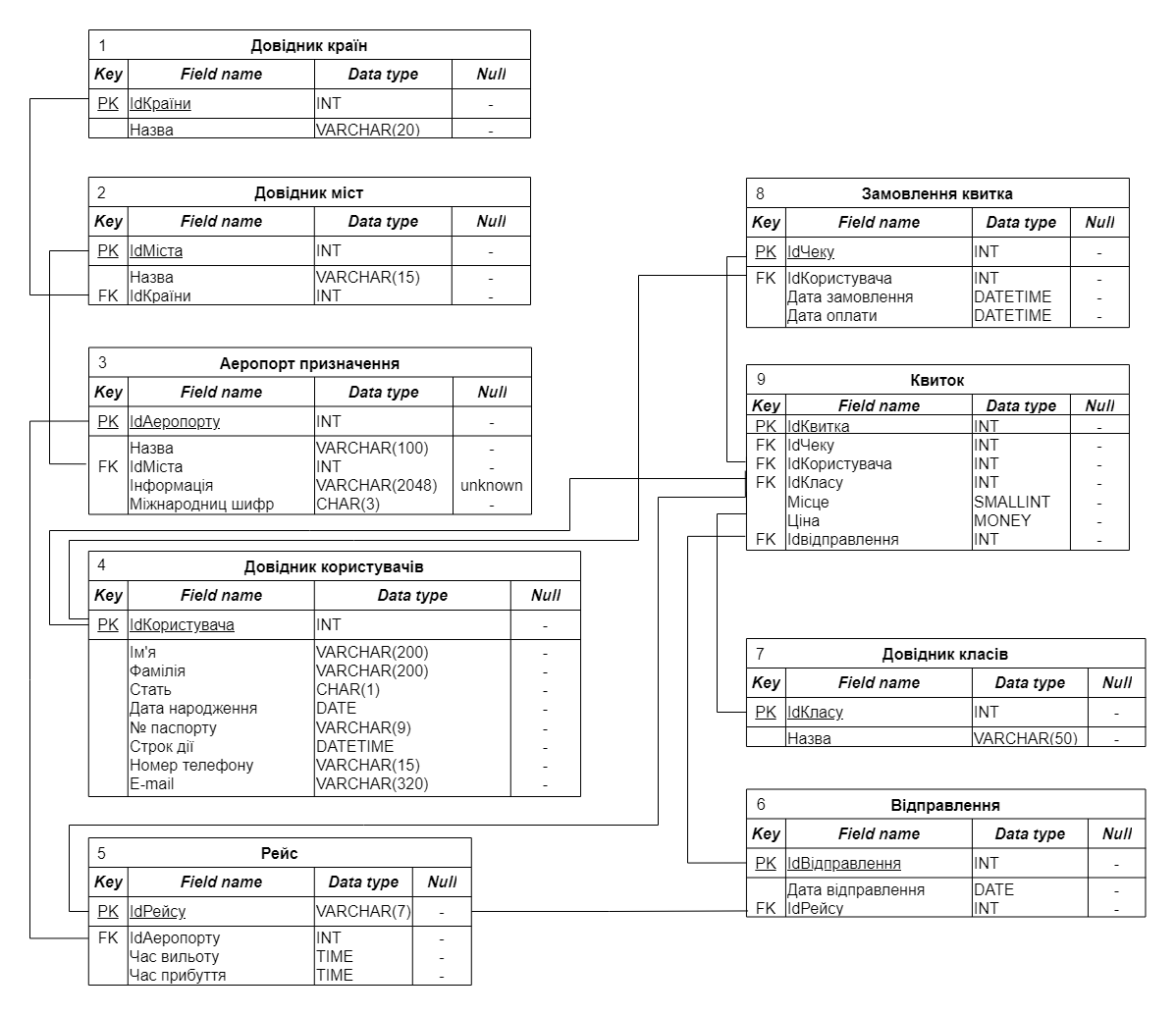


Рис. 2.3. Фізична модель бази даних управління продажем квитків

Обравши Microsoft SQL Server як СКБД, було розраховано на його стабільність та надійність. Він широко використовується в корпоративних середовищах та має велику кількість задоволених користувачів. Також Microsoft має велику спільноту та широку мережу підтримки, тож можна легко знайти відповіді на свої питання, документацію та рішення для будь-яких проблем, які могли б виникнути під час роботи з SQL Server. Microsoft SQL Server має широкий набір функцій та можливостей, що дозволяє ефективно управляти базами даних будь-якої складності. Крім того, Microsoft надає безкоштовні версії для навчання та розвитку, тож для студентів – ідеальний варіант.

## **2.2 Розробка бази даних та заповнення таблиць даними**

Створення бази даних під назвою TicketSalesDB починається командою *CREATE DATABASE*. Вказується ключове слово *ON PRIMARY*, яке означає, що наступні параметри визначають властивості основного файлу бази даних. Визначаються назва файлу, шлях до фізичного файлу, початковий розмір та крок збільшення файлу.

Для групування файлів за частотою заповнюваності таблиць було створено дві файлові групи. В одній з них будуть міститися таблиці, для яких треба багато виділеної пам’яті. Для цієї файлової групи створено два файли, щоб у випадку повного заповнення першого, дані записувались в другий. Потім, при заповненні другого файлу, розширюватись буде перший. В другій файловій групі будуть міститися таблиці, в які рідко записуються дані (скрипт див. в Додаток А).

Далі за допомогою команди *CREATE SCHEMA TicketSalesManagement* було створено схему даних, яка буде логічно об’єднувати об’єкти в межах бази даних.

Після цього було розпочато заповнення бази даних таблицями, на основі фізичної моделі. Щоб створити таблицю, існує відповідна SQL-команда – *CREATE TABLE*, після якої необхідно вказати повне ім’я таблиці з назвою відповідної схеми даних (наприклад, [SchemaName].[TableName]).

У базах даних взаємозв'язки між таблицями встановлюються за допомогою ключів, що дозволяє забезпечити зв'язність та цілісність даних (див. рис. 2.4). Первинний ключ в більшості випадків має цілочисельний тип даних INT. Він є одним з найшвидших для обробки і пошуку, займає мало місця та має достатній діапазон значень для ідентифікаторів. Первинні ключі будуть автоматично заповнюватись, так як використано властивість IDENTITY(1,1). Зв’язки між таблицями забезпечуються саме завдяки первинному та зовнішньму ключах

Всі таблиці було розміщено у відповідні, за частотою заповнюваності, файлові групи (скрипт див. в Додаток А).

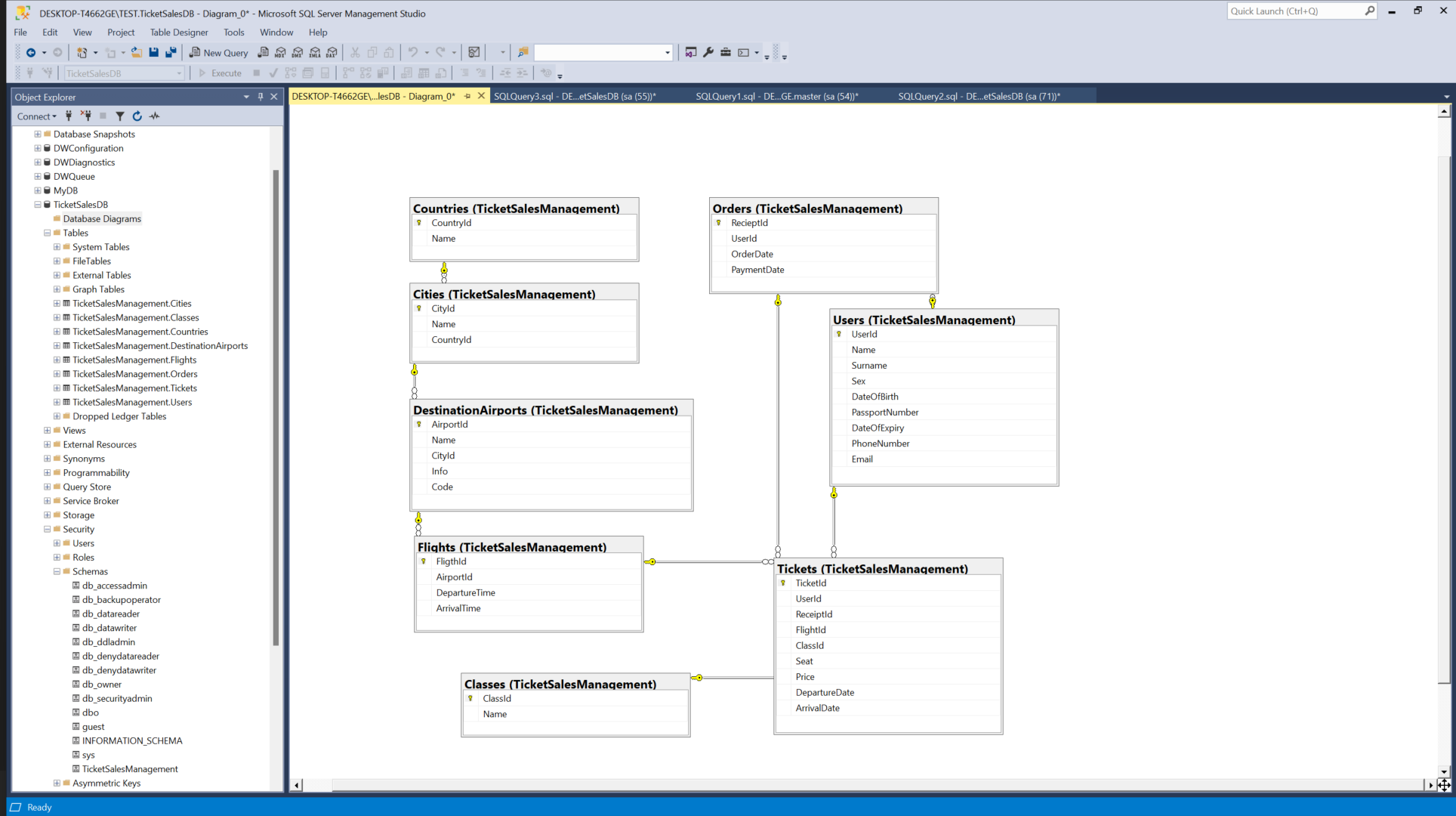


Рис. 2.4. Графічне відображення таблиць та зв’язків між ними

Для заповнення таблиць даними використовується команда INSERT INTO.

В першу чергу заповнюються таблиці, які не мають зовнішніх ключів, лише первиний (скрипт див. в Додаток Б). Це дозволяє уникнути появи посилань на неіснуючі записи. Після цього заповнюються таблиці, що мають і первинний ключ, і зовнішні (скрипт див. в Додаток Б).

## **2.3 Реалізація представлень (запитів) до бази даних**

Запити в базу даних забезпечують можливість отримання необхідної інформації для відображення у клієнтському додатку. Наприклад, для відображення списку користувачів можна використовувати запит *SELECT*. Аналогічно, для відображення статистики, такої як кількість замовлень, можна використовувати запити з агрегувальними функціями, такими як *COUNT*.

Наприклад, для того, щоб отримати суму до оплати кожного замовлення потрібно за допомогою команди JOIN об’єднати дві таблиці: «Orders» та «Tickets» за полем «RecieptId», та обчислити суму цін («Price»), використовуючи функцію агрегування *SUM****.*** За допомогою функції *COUNT*виконується підрахунок кількості квитків по кожному замовленню. Результатом виконання є таблиця, відображена на рисунку 2.5.

SELECT o.RecieptId, SUM([Price]) AS Summary, COUNT(t.TicketId) AS TicketCount

FROM [TicketSalesManagement].[Orders] o

JOIN [TicketSalesManagement].[Tickets] t

ON o.RecieptId = t.ReceiptId

GROUP BY o.RecieptId

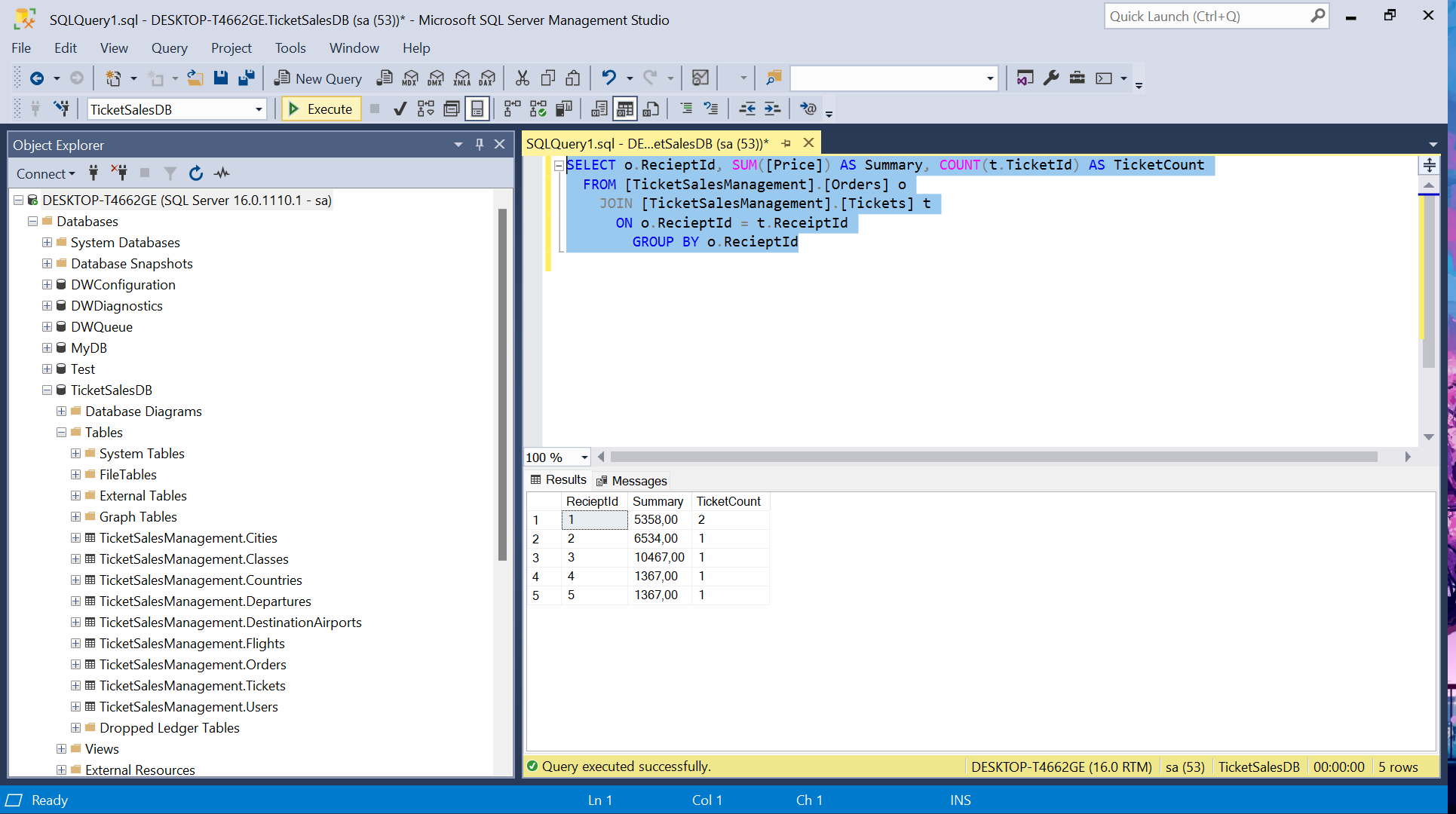


Рис. 2.5. Результат виконання запиту для обчислення суми замовлення

Для відображення в клієнтському додатку статистики продажу квитків, в базу даних потрібно зробити запит до таблиць «Flights», «Departures» та «Tickets», щоб отримати кількість квитків, придбаних на кожен з рейсів (див. рис. 2.6).

SELECT f.FlightId, COUNT(t.TicketId) AS TicketCount

FROM [TicketSalesManagement].[Flights] f

JOIN [TicketSalesManagement].[Departures] d

ON f.FlightId = d.FlightId

JOIN [TicketSalesManagement].[Tickets] t

ON t.DepartureId = d.DepartureId

GROUP BY f.FlightId;

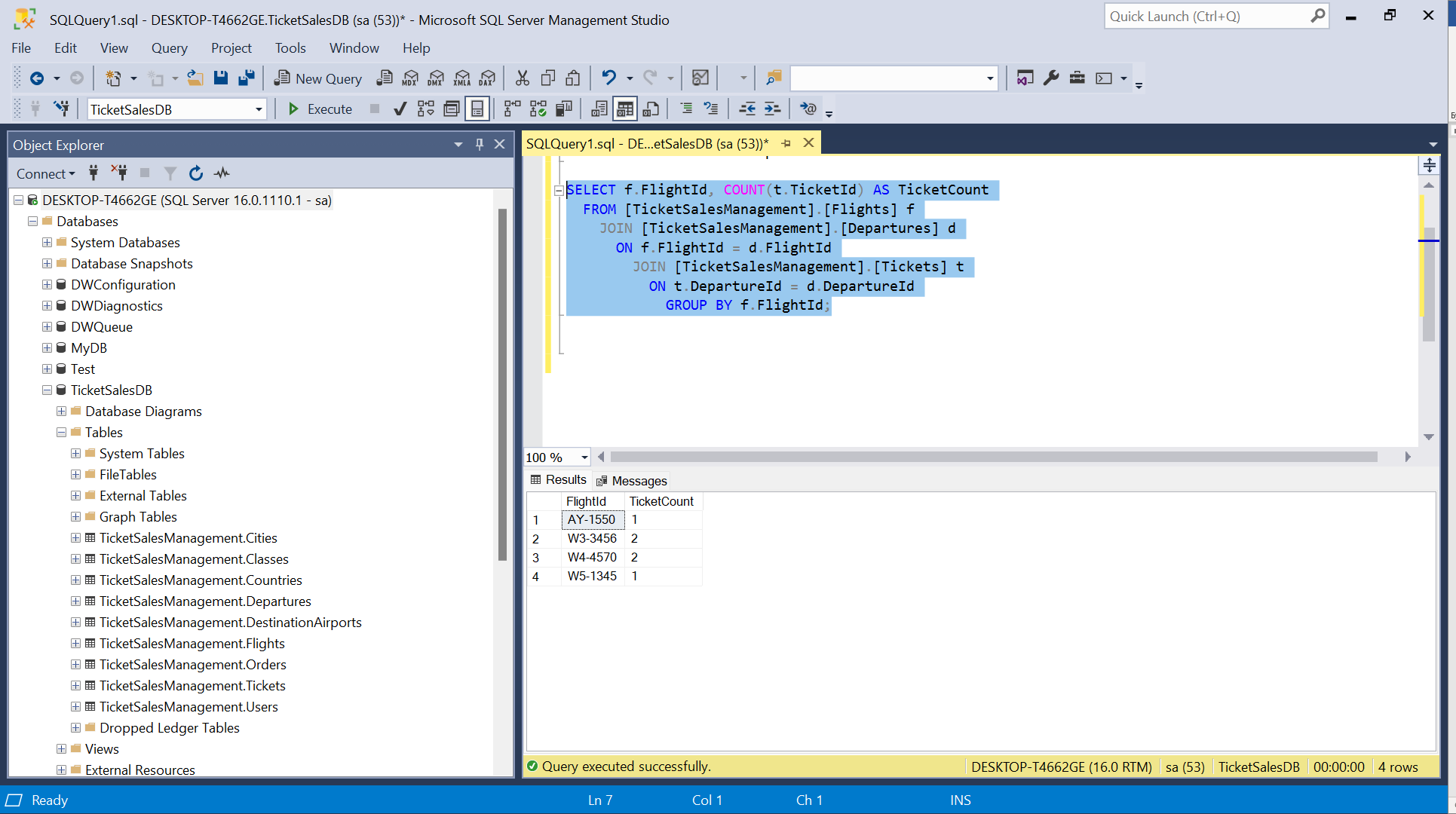


Рис. 2.6. Результат виконання запиту для обчислення суми замовлення

**Висновки до Розділу 2.**

В цьому розділі було реалізовано постановку задач, яка необхідна для розробки бази даних. Створена концептуальна модель бази даних дозволяє чітко визначити структуру та взаємозв'язки між сутностями системи. Логічна модель визначає структуру вже нормалізованої бази даних. Фізична модель доповнює логічну з урахуванням технічних особливостей.

Вибір Microsoft SQL Server як системи управління реляційними базами даних обгрунтований його стабільністю, надійністю та широким набором функцій.

Для створення бази даних та схеми даних було використано команди CREATE DATABASE та CREATE SCHEMA відповідно. Потім за допомогою команди INSERT INTO було реалізовано заповнення таблиць даними, починаючи з тих, які не мають зовнішніх ключів, а потім ті, що мають як первинний, так і зовнішні ключі.

Також для забезпечення функціональності клієнтського додатку були розроблені та впроваджені запити, що забезпечують доступ до необхідної інформації.

# **РОЗДІЛ 3 ПРОЦЕС РОЗРОБКИ КЛІЄНТСЬКОГО ДОДАТКУ ДЛЯ ВЗАЄМОДІЇ З БАЗОЮ ДАНИХ**

## **3.1 Вибір технологій для створення додатку**

Мова програмування C# була обрана через її відмінну сумісність із середовищем .NET Framework, що забезпечує великий набір бібліотек, фреймворків та інструментів для швидкої та продуктивної розробки. C# має зрозумілий синтаксис, що спрощує розуміння коду та його підтримку у майбутньому.

.NET Framework обрано через його широкий функціонал та стабільність. Він надає доступ до різноманітних можливостей, таких як робота з базами даних, мережеві операції, робота з файлами та багато іншого.

Інтерфейс для програмування додатків Windows Forms був обраний через його простоту та зручність у створенні інтерфейсу користувача. Він має обмежені можливості дизайну порівняно з іншими інструментами, але для роботи з базою даних цього обсягу достатньо.

## **3.2 Розробка клієнтського додатку мовою C# на основі інтерфейсу Windows Forms**

Для підключення до SQL Server, виконання запитів SQL та обробки результатів запитів в першу чергу потрібно підключити простір імен System.Data.SqlClient (код див. в Додаток В). Він містить класи для взаємодії з базою даних Microsoft SQL Server з використанням платформи .NET.

Одним з таких класів є SqlConnection, який надає можливість створювати, відкривати та управляти з'єднаннями з базою даних SQL Server. Для створення екземпляру цього класу потрібно знайти базу даних в SQL Server Object Explorer, натиснути правою кнопкою миші та в контекстному меню натиснути Properties. Після цього необхідно скопіювати рядок підключення – Connection string.

Після створення об'єкту SqlConnection зазвичай потрібно відкрити з'єднання перед виконанням операцій з базою даних, а потім закрити його, коли робота з базою завершена. Для цього використовуються методи *Open()* та *Close()* відповідно.

Об’єкт класу SqlCommand використовується для виконання запитів до бази даних. Метод ExecuteNonQuery() – для виконання запитів, які не повертають результатів (INSERT, UPDATE або DELETE).

Також в розробці додатку були використані методи ExecuteReader() та ExecuteScalar(). Перший з них повертає об'єкт SqlDataReader, який дозволяє читати результати запиту по одному рядку за раз. Метод ExecuteScalar() використовується для отримання одного значення з результатів запиту, наприклад, кількості записів або значення агрегатної функції.

Важливим в розробці додатку був метод AddWithValue класу SqlCommand.Parameters, який використовується для додавання параметрів до SQL-запиту зі значеннями безпечно, що є важливим для уникнення атак на введення даних та забезпечення безпеки додатка.

Для відображення даних у табличному вигляді було використано інструмент dataGridView. За допомогою SqlDataAdapter отримується результат запиту до бази даних, а об'єкт DataTable представляє цей результат таблицею даних у пам’яті. Потім встановлюється DataSource для DataGridView і таблиця відображається в інтерфейсі додатку.

## **3.3 Функціонал додатку TicketsDataManager**

Функціями розробленого додатку є можливість виконання запитів SELECT, INSERT, UPDATE, DELETE для таблиць «Users», «Orders», «Tickets», «Departures», «Flights». Це означає, що користувачі можуть переглядати, додавати, оновлювати та видаляти дані в цих таблицях через інтерфейс додатку.

В першій вкладці кожної з таблиць є виведення результату виконання команди SELECT \* FROM (див. рис. 3.1).

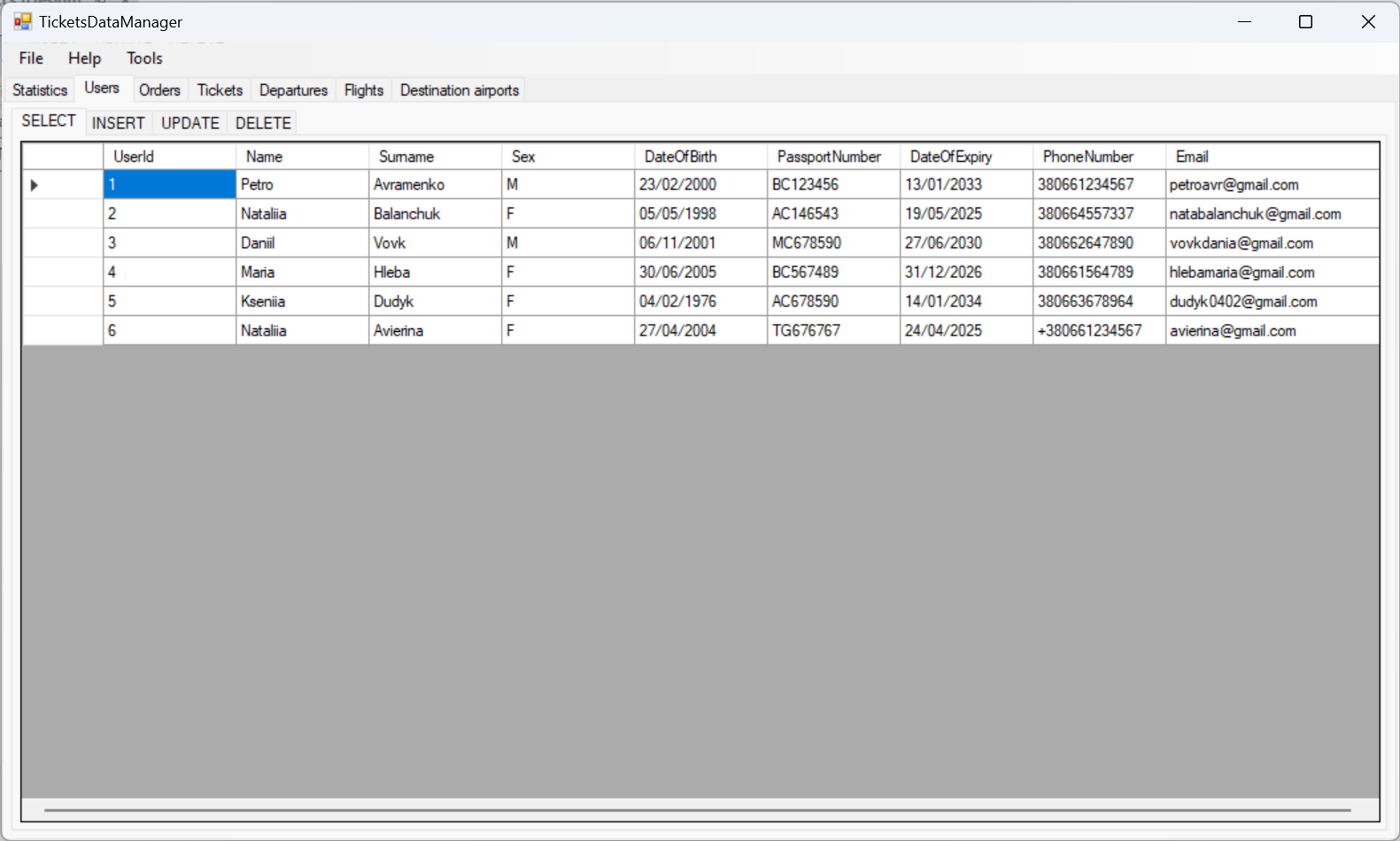


Рис. 3.1 Таблиця Users до виконання запитів

Наступні вкладки є повністю персоналізованими під стовбці таблиць. Дата та час вводяться за допомогою вікна вибору дати та часу, текстові дані в спеціальних рядках, а дані, що можна ввести з невеликого переліку, заповнюються за допомогою вікна вибору. Натискаючи на кнопку виконання дії, можна побачити повідомлення про успішний результат виконання команди (див. рис. 3.2.), або помилку виконання (рис. 3.3).

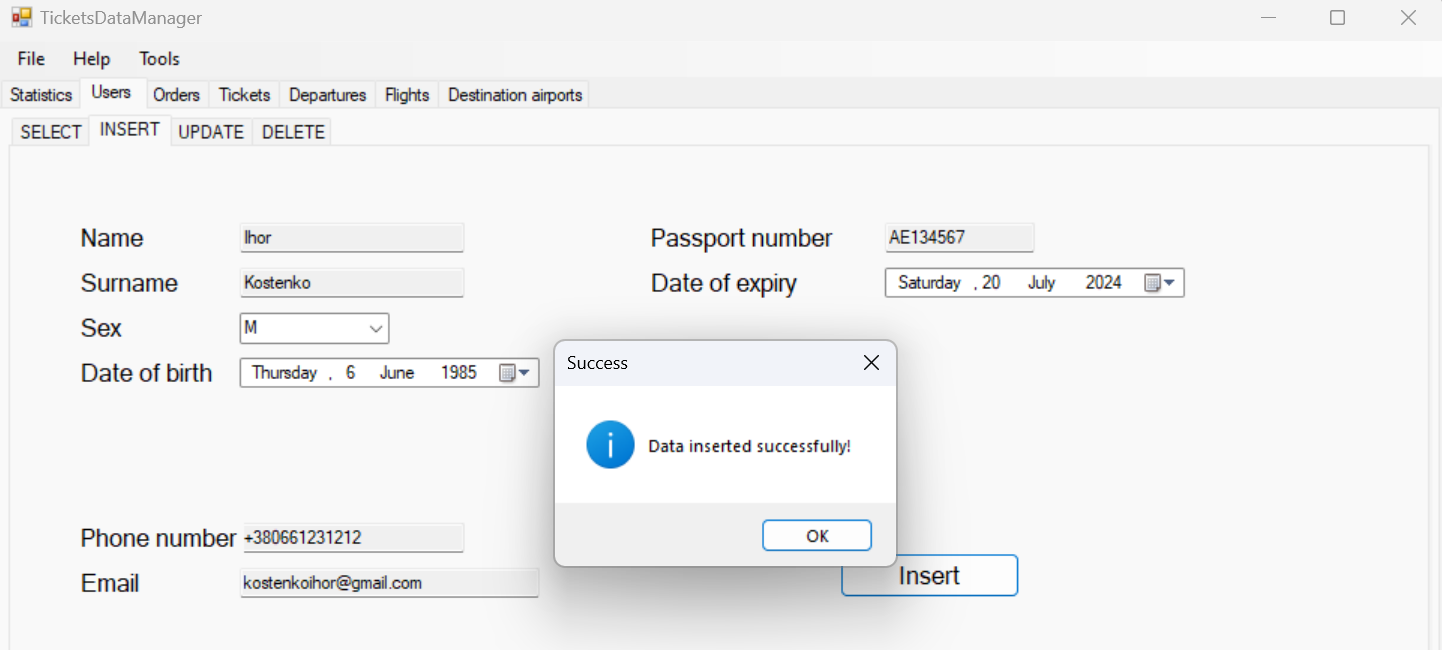


Рис. 3.2. Додавання даних на прикладі таблиці Users за допомогою інтерфейсу додатку

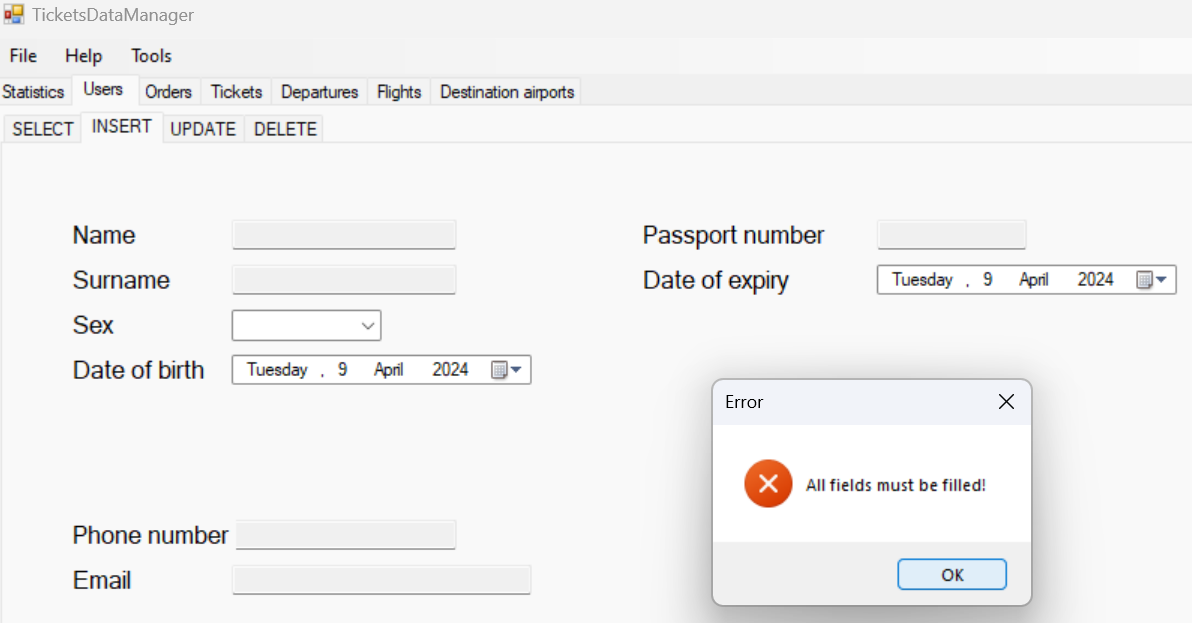


Рис. 3.3. Виведення повідомлення про помилку в зв’язку з незаповненими полями

Для оновлення даних таблиці потрібно ввести Id рядку, який буде змінюватись, вибрати з переліку потрібне поле та ввести оновлені дані (див. рис. 3.4.).

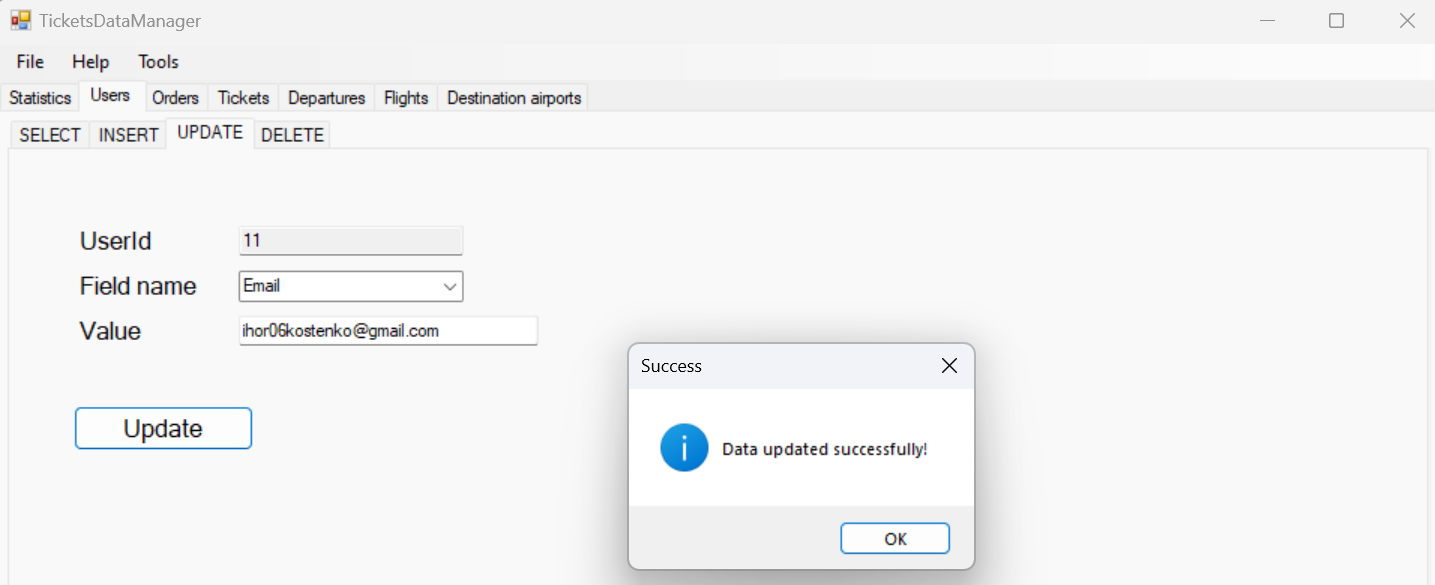


Рис. 3.4. Оновлення даних на прикладі таблиці Users за допомогою інтерфейсу додатку

Для перевірки виконання додавання та оновлення даних, можна перейти до вкладки SELECT (див. рис. 3.5.). Таблиця оновлюється кожен раз, коли користувач натискає на вкладку.

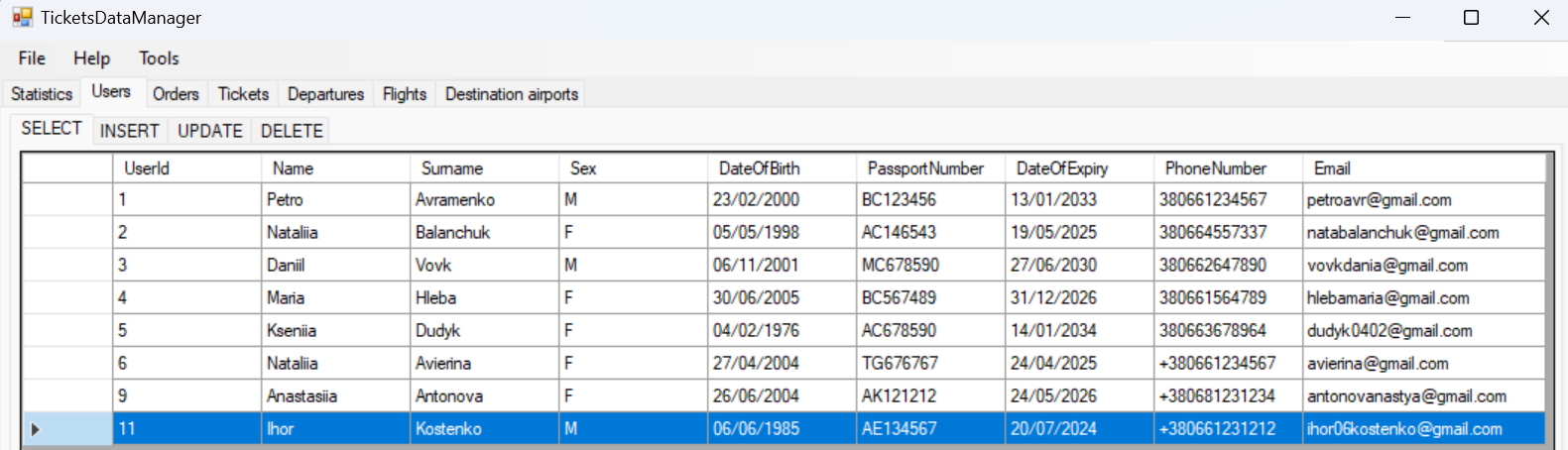


Рис. 3.5. Таблиця Users після виконання команд INSERT та UPDATE

Для видалення даних користувач має ввести лише Id потрібного рядку (див. рис. 3.6).

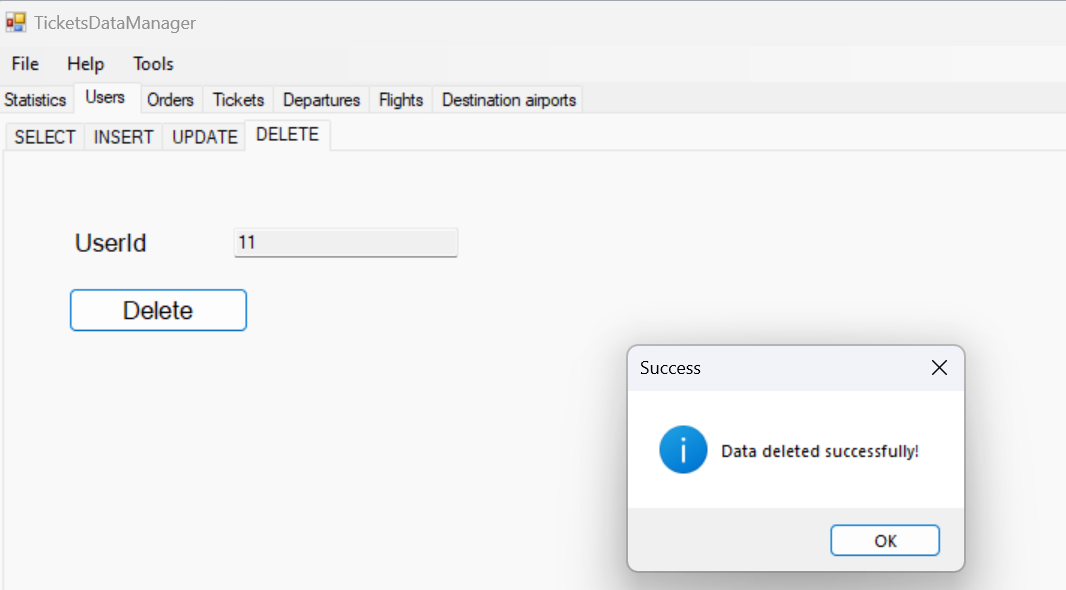


Рис. 3.6 Виконання видалення рядку даних з таблиці Users за допомогою інтерфейсу додатку

У вкладці SELECT одразу відображаються зміни. Можна побачити, що поле з ідентифікатором під номером 11 було видалено (див. рис. 3.7).

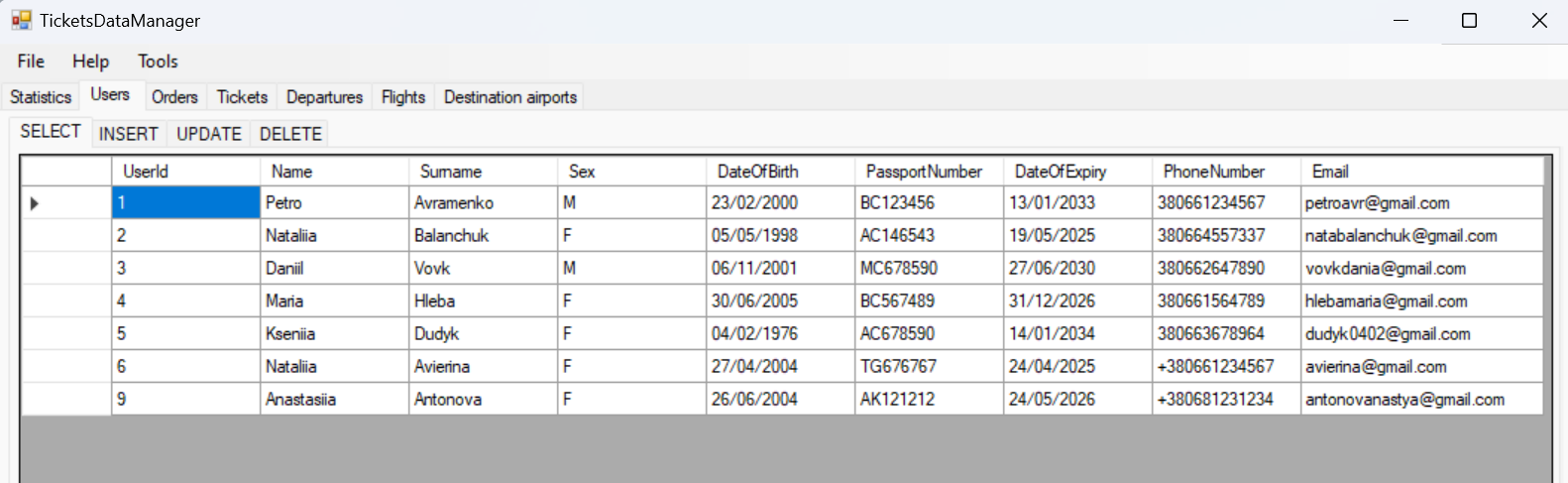


Рис. 3.7 Таблиця Users після виконання запитів

Також для зручності користувачів додаток відображає таблицю «Destination airports» та відповідні до аеропорту місто і країну (див. рис. 3.8). Це дозволяє правильно заповнювати дані інших таблиць зовнішніми ключами цієї таблиці. Проте, додаток обмежує можливість змін з метою забезпечення цілісності та стабільності даних. Таблиця містить ключову інформацію про аеропорти призначення, яка використовується в інших таблицях, і неправильні зміни можуть призвести до некоректної роботи бази даних.

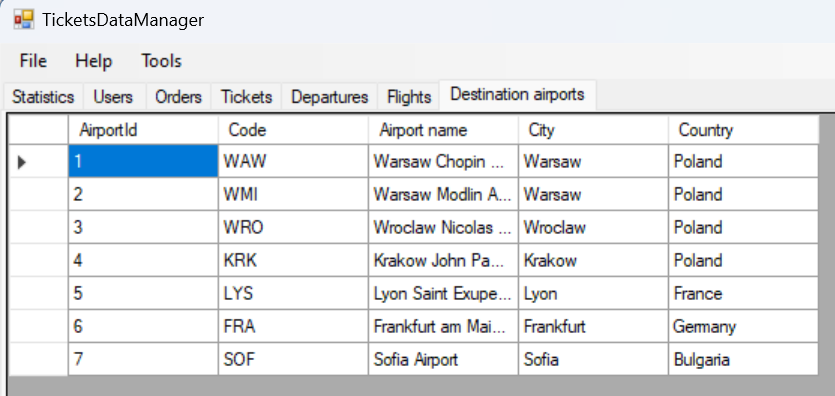


Рис. 3.8. Таблиця «Destination airports»

Зміни в таблицях Countries та Cities також недоступні через додаток. Ці дані, як правило, змінюються дуже рідко, і їх можна вносити вручну або за допомогою спеціальних інструментів управління базою даних. Такий підхід дозволяє зменшити ризик випадкових або неправильних змін у найменш змінюваних або критичних даних.

Вкладка зі статистикою наразі має дві діаграми: одна визначає кількість куплених квитків на кожен з рейсів (див. рис. 3.9.), друга – топ 3 країни за відвідуваністю (за кількість проданих квитків) (див. рис. 3.10.).

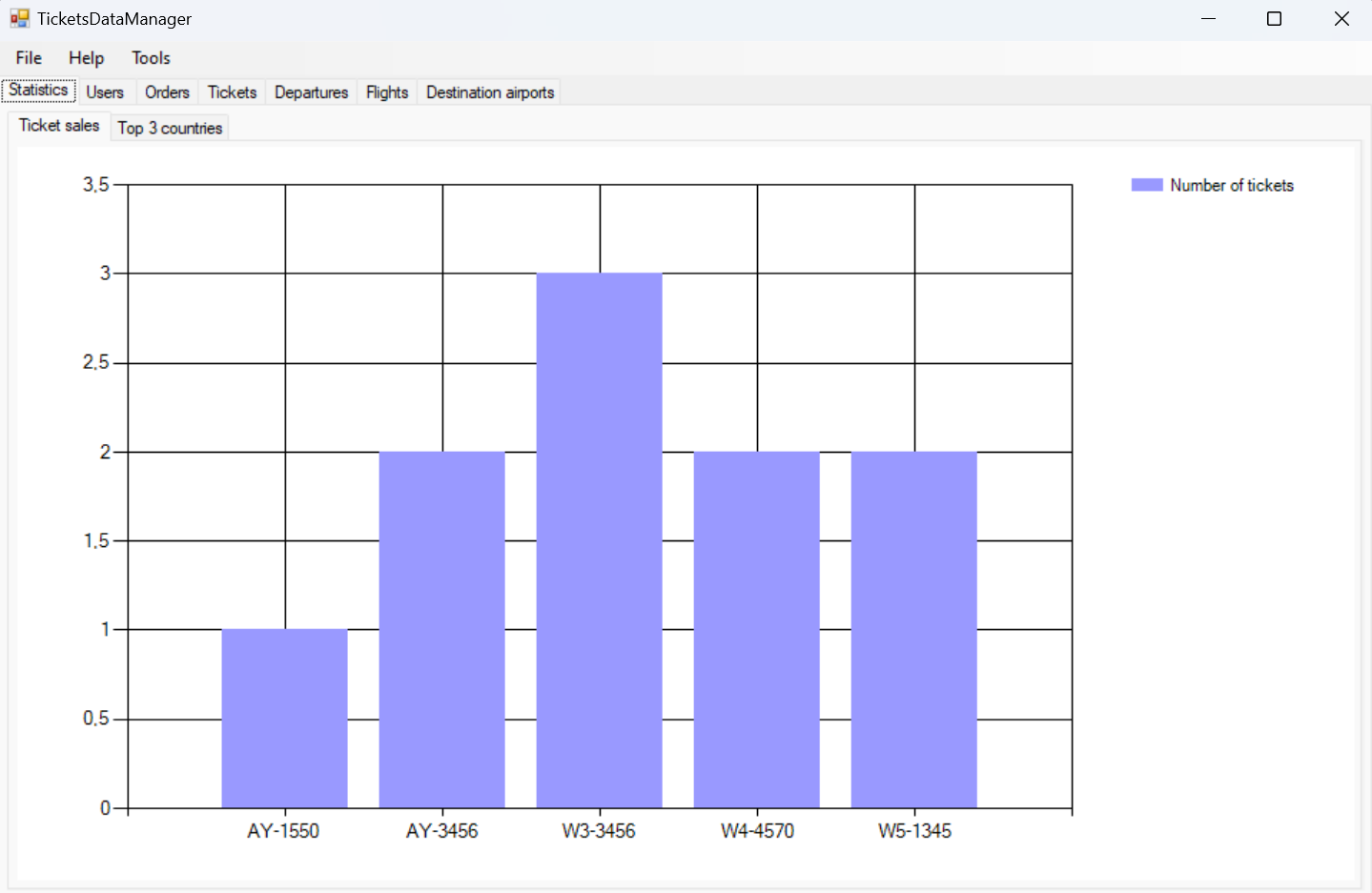


Рис. 3.9. Відображення в клієнтському додатку діаграми проданих квитків за рейсами



Рис. 3.10. Відображення в клієнтському додатку діаграми топ 3 країни за кількістю проданих квитків

**Висновки до Розділу 3.**

Для розробки додатку обрано мову програмування C# та платформу .NET Framework. Використано інтерфейс програмування Windows Forms для зручного створення інтерфейсу користувача. Для підключення до бази даних використовувалися класи SqlConnection та SqlCommand простору імен System.Data.SqlClient та DataTable простору імен System.Data. Додаток має функціонал для виконання запитів SELECT, INSERT, UPDATE, DELETE для певних таблиць, але обмежує доступ до змін у таблицях, що рідко змінюються, для забезпечення цілісності та стабільності даних. Також додаток відображає невелику статистику продажу квитків на сайті, роблячи відповідний запит в базу даних.

# **ВИСНОВКИ**

У курсовій роботі було досліджено процес управління продажем авіаквитків. Проаналізовано ринок онлайн продажу та висвітлено переваги і недоліки розроблюваної системи, порівняно з конкурентами. Розглянуто методи оптимізації бізнес-процесів за допомогою баз даних та принципи проєктування. Було визначено проблемні питання для кращого розуміння завдання роботи над системою.

Також було здійснено постановку завдання для розробки бази даних. Для кращого розуміння структури, взаємозв’язків та технічних особливостей, було використано метод моделювання бази даних.

Обравши за СКБД Microsoft SQL Server за допомогою мови запитів було створено та заповнено даними базу даних TicketSalesDB. Для виведення аналізу даних в клієнтський додаток було розроблено багатотабличні запити.

Клієнтський додаток створений мовою C# в середовищі .NET Framework на основі інтерфейсу для програмування додатків Windows forms. Програма має функції перегляду, додавання, оновлення та видалення даних з таблиць, а також перегляд аналізу даних у вигляді стовбчастих діаграм.

Загалом, результати роботи вказують на успішність розробленої системи у вирішенні завдань управління продажем авіаквитків і підтверджують актуальність та значимість проведених досліджень.

# **СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

***Монографії:***

1. Коваленко, О. (2018). Основи баз даних та СУБД. Київ: Видавництво "Європейський вік".

***Підручники***

1. Синельник, Д., Голландець, С., Татарчук, І. (2004). Бази даних: Підручник. Київ: Техніка.
2. Elmasri, R., & Navathe, S. B. Fundamentals of Database Systems. 7th ed. Addison-Wesley, 2016.
3. 5Ramakrishnan, R., & Gehrke, J. Database Management Systems. 3rd ed. McGraw-Hill, 2003.
4. C. J. Date, "An Introduction to Database Systems", Addison-Wesley, 2003.

***Книги колективу авторів***

1. Рейнольдс, Г., Каслер, Д. (2012). Управління базами даних. Київ: Видавничий будинок "ІнЖеК".

***Статті***

1. Smith, J. (2022). Challenges in database management for airline ticket sales. Journal of Airline Technology, 10(2), 45-58.
2. Brown, L. (2019). Relational database management systems: Advantages and limitations. Aviation Computing Review, 16(1), 30-42.
3. Garcia, M. (2018). Non-relational databases in aviation: A comparative study. Journal of Aviation Technology and Engineering, 7(2), 15-28.

***Електронні ресурси***

1. "Introduction to Relational Databases", Microsoft Docs, <https://docs.microsoft.com/en-us/sql/relational-databases/sql-server-data-tier-applications/introduction-to-relational-databases>
2. "Relational Database Management Systems", Study.com, <https://study.com/academy/lesson/relational-database-management-systems-rdbms.html>

# **ДОДАТКИ**

Додаток А

**Скрипт створення бази даних з файловими групами**

CREATE DATABASE TicketSalesDB

ON PRIMARY

(NAME = 'TicketSalesDb\_Primary',

FILENAME = 'C:\MY FILES\KNUTE\3-2\Data\TicketSalesDb\_prm.mdf',

SIZE = 10MB,

FILEGROWTH = 1MB),

FILEGROUP FrequentlyAccessedGroup

(NAME = 'TicketSales\_FreqAccessed\_1',

FILENAME = 'C:\MY FILES\KNUTE\3-2\Data\FreqAccessed\_1.ndf',

SIZE = 10MB,

FILEGROWTH = 1MB),

(NAME = 'TicketSales\_FreqAccessed\_2',

FILENAME = 'C:\MY FILES\KNUTE\3-2\Data\FreqAccessed\_2.ndf',

SIZE = 10MB,

FILEGROWTH = 1MB),

FILEGROUP InfrequentlyAccessedGroup

(NAME = 'TicketSales\_InfreqAccessed\_1',

FILENAME = 'C:\MY FILES\KNUTE\3-2\Data\InfreqAccessed\_1.ndf',

SIZE = 3MB,

FILEGROWTH = 100KB)

GO

**Скрипт створення таблиць бази даних TicketSalesDB**

USE TicketSalesDB;

CREATE TABLE TicketSalesManagement.Countries

(CountryId INT PRIMARY KEY IDENTITY(1,1),

Name VARCHAR(20))

ON InfrequentlyAccessedGroup;

CREATE TABLE TicketSalesManagement.Cities

(CityId INT PRIMARY KEY IDENTITY(1,1),

Name VARCHAR(15),

CountryId INT FOREIGN KEY REFERENCES TicketSalesManagement.Countries)

ON InfrequentlyAccessedGroup;

CREATE TABLE TicketSalesManagement.DestinationAirports

(AirportId INT PRIMARY KEY IDENTITY(1,1),

Name VARCHAR(20),

CityId INT FOREIGN KEY REFERENCES TicketSalesManagement.Cities,

Info VARCHAR(2048) NULL,

Code CHAR(3))

ON InfrequentlyAccessedGroup;

CREATE TABLE TicketSalesManagement.Users

(UserId INT PRIMARY KEY IDENTITY(1,1),

Name VARCHAR(200),

Surname VARCHAR(200),

Sex CHAR(1),

DateOfBirth DATE,

PassportNumber VARCHAR(9),

DateOfExpiry DATE,

PhoneNumber VARCHAR(15),

Email VARCHAR(320))

ON FrequentlyAccessedGroup;

CREATE TABLE TicketSalesManagement.Flights

(FlightId VARCHAR(7) PRIMARY KEY,

AirportId INT FOREIGN KEY REFERENCES TicketSalesManagement.DestinationAirports,

DepartureTime TIME,

ArrivalTime TIME)

ON InfrequentlyAccessedGroup;

CREATE TABLE TicketSalesManagement.Classes

(ClassId INT PRIMARY KEY IDENTITY(1,1),

Name VARCHAR(50))

ON InfrequentlyAccessedGroup;

CREATE TABLE TicketSalesManagement.Orders

(RecieptId INT PRIMARY KEY IDENTITY(1,1),

UserId INT FOREIGN KEY REFERENCES TicketSalesManagement.Users,

OrderDate DATETIME,

PaymentDate DATETIME)

ON FrequentlyAccessedGroup;

CREATE TABLE TicketSalesManagement.Tickets

(TicketId INT PRIMARY KEY IDENTITY(1,1),

UserId INT FOREIGN KEY REFERENCES TicketSalesManagement.Users,

ReceiptId INT FOREIGN KEY REFERENCES TicketSalesManagement.Orders,

FlightId VARCHAR(7) FOREIGN KEY REFERENCES TicketSalesManagement.Flights,

ClassId INT FOREIGN KEY REFERENCES TicketSalesManagement.Classes,

Seat SMALLINT,

Price MONEY,

DepartureDate DATE,

ArrivalDate DATE)

ON FrequentlyAccessedGroup;

CREATE TABLE TicketSalesManagement.Departures

(DepartureId INT PRIMARY KEY IDENTITY(1,1),

DepartureDate DATE,

FlightId VARCHAR(7) FOREIGN KEY REFERENCES [TicketSalesManagement].[Flights])

ON FrequentlyAccessedGroup;

GO

Додаток Б

**Скрипт заповнення даними таблиць, що мають лише первинний ключ**

USE TicketSalesDB;

INSERT INTO TicketSalesManagement.Users (Name, Surname, Sex, DateOfBirth, PassportNumber, DateOfExpiry, PhoneNumber, Email)

VALUES

('Petro', 'Avramenko', 'M', '2000-02-23', 'BC123456', '2033-01-13', '380661234567', 'petroavr@gmail.com'),

('Nataliia', 'Balanchuk', 'F', '1998-05-05', 'AC146543', '2025-05-19', '380664557337', 'natabalanchuk@gmail.com'),

('Daniil', 'Vovk', 'M', '2001-11-06', 'MC678590', '2030-06-27', '380662647890', 'vovkdania@gmail.com'),

('Maria', 'Hleba', 'F', '2005-06-30', 'BC567489', '2026-12-31', '380661564789', 'hlebamaria@gmail.com'),

('Kseniia', 'Dudyk', 'F', '1976-02-04', 'AC678590', '2034-01-14', '380663678964', 'dudyk0402@gmail.com');

INSERT INTO TicketSalesManagement.Classes (Name)

VALUES

('Economy'),

('Business'),

('First');

INSERT INTO TicketSalesManagement.Countries (Name)

VALUES

('Poland'),

('France'),

('Germany'),

('Bulgaria'),

('Hungary'),

('Austria');

**Скрипт заповнення даними таблиць, що мають і первинний, і зовнішні ключі**

USE TicketSalesDB;

INSERT INTO TicketSalesManagement.Cities (Name, CountryId)

VALUES

('Warsaw', 1),

('Wroclaw', 1),

('Gdansk', 1),

('Krakow', 1),

('Poznan', 1),

('Paris', 2),

('Lyon', 2),

('Nice', 2),

('Toulouse', 2),

('Berlin', 3),

('Cologne', 3),

('Frankfurt', 3),

('Hamburg', 3),

('Munich', 3),

('Sofia', 4),

('Burgas', 4),

('Varna', 4),

('Plodiv', 4),

('Budapest', 5),

('Debrecen', 5),

('Vienna', 6),

('Graz', 6),

('Linz', 6),

('Salzburg', 6);

INSERT INTO TicketSalesManagement.DestinationAirports (Name, CityId, Info, Code)

VALUES

('Warsaw Chopin Airport', 1, 'Warsaw Frederic Chopin Airport is the country’s both largest and busiest airport. Moreover, it handles approximately 40-50% of Poland’s air passenger traffic. The best way to reach Warsaw city centre is by 20-minute train ride. Another possibility are bus lines 175, 188 and N32 (during night).', 'WAW'),

('Warsaw Modlin Airport', 1, NULL, 'WMI'),

('Wroclaw Nicolas Copernicus Airport', 2, 'Wroclaw Nicolas Copernicus Airport is an international airport serving the city of Wroclaw in the South West of Poland. The airport is situated 10km south-west of the city centre. It has two passenger terminals and one cargo terminal. Due to its vicinity to the city, it is not a problem to reach it by bus or by car. Bus line 106 goes daily to/from the Central Railway/Bus Station every 20 minutes. The entire journey takes approximately 40 minutes, depending on the traffic and wheather conditions. Night bus line 206, operates on the same route through the Airport.', 'WRO'),

('Krakow John Paul II International Airport', 4, 'Saint John Paul II Airport Krakow–Balice is an international airport in the village of Balice, 11km from Krakow. Shuttle buses link the Balice Airport with the Krakow Glowny central station, located close to the Old Town historic district in the middle of the city. The buses run every 30 minutes and the journey takes less than 20 minutes. Another means of transportation is the train. The earliest train to the Krakow Airport leaves from the central station at 4:05 am and the latest one at 11:35 pm. The first and the last departure from the airport station are 5:16 am and 0:15 am respectively.', 'KRK'),

('Lyon Saint Exupery Airport', 7, 'Lyon-Saint Exupery Airport is located 20 km south east from Lyon’s city center and is the main international airport that serves Lyon. This airport in France has two passenger terminals which are interconnected on the landside by a central building that itself has a foot-bridge to the nearby Gare de Lyon Saint-Exupery high-speed railway station and the Rhonexpress terminus. Taking a taxi or a train will get you to the city center.', 'LYS'),

('Frankfurt am Main Airport', 12, 'Frankfurt Airport is the busiest airport by passenger traffic in Germany as well as the 6th busiest in Europe.', 'FRA'),

('Sofia Airport', 15, 'Sofia Airport is the main international airport of Bulgaria, located 10 km east of the centre of the capital Sofia.', 'SOF');

INSERT INTO TicketSalesManagement.Flights (FlightId, AirportId, DepartureTime, ArrivalTime)

VALUES

('W5-1050', 1, '10:30:00', '12:20:00'),

('W4-4570', 2, '13:10:00', '14:04:00'),

('AY-1550', 5, '22:54:00', '01:03:00'),

('W3-3456', 6, '05:50:00', '07:10:00'),

('W5-1345', 2, '16:30:00', '17:40:00');

INSERT INTO TicketSalesManagement.Orders (UserId, OrderDate, PaymentDate)

VALUES

(1, '2015-04-01 10:21:56', '2015-04-01 10:25:43'),

(3, '2015-04-01 13:14:34', '2015-04-01 13:20:33'),

(4, '2015-04-02 16:41:26', '2015-04-02 16:43:45'),

(5, '2015-04-03 19:17:24', '2015-04-03 19:20:05'),

(1, '2015-04-06 01:13:15', '2015-04-06 01:14:07');

INSERT INTO TicketSalesManagement.Departures(DepartureDate, FlightId) VALUES

('2024-04-20','W3-3456'),

('2024-04-20', 'W5-1345'),

('2024-04-20', 'AY-1550'),

('2024-04-20', 'W4-4570'),

('2024-04-21', 'W3-3456'),

('2024-04-21', 'W5-1345'),

('2024-04-21', 'W4-4570'),

('2024-04-22', 'W5-1345'),

('2024-04-22', 'AY-1550'),

('2024-04-22', 'W4-4570');

INSERT INTO TicketSalesManagement.Tickets (UserId, ReceiptId, ClassId, Seat, Price, DepartureId)

VALUES

(1, 1, 1, 56, 2678, 1),

(2, 1, 1, 57, 2680, 1),

(3, 2, 2, 13, 6534, 2),

(4, 3, 3, 5, 10467, 3),

(5, 4, 1, 5, 1367, 4),

(2, 5, 1, 10, 1367, 7)

Додаток В

Програмний код додатку TicketSalesManager

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Data;

using System.Data.SqlClient;

using System.Runtime.Remoting.Contexts;

using System.Xml;

using System.Collections;

using System.Drawing.Text;

using System.Windows.Forms.DataVisualization.Charting;

namespace TicketsDataManager

{

public partial class Form1 : Form

{

SqlConnection sqlConnection;

public Form1()

{

InitializeComponent();

}

private async void Form1\_Load(object sender, EventArgs e)

{

string connectionString =

@"Data Source = DESKTOP-T4662GE; Initial Catalog = TicketSalesDB; User ID = sa; Password = qwaszx1209; Connect Timeout = 30; Encrypt = False; TrustServerCertificate = False; ApplicationIntent = ReadWrite; MultiSubnetFailover = False";

sqlConnection = new SqlConnection(connectionString);

await sqlConnection.OpenAsync();

try

{

PopulateComboBox("FlightId", "TicketSalesManagement.Flights", comboBox2);

PopulateComboBoxWithFieldNames("TicketSalesManagement.Users", comboBox4);

PopulateComboBoxWithFieldNames("TicketSalesManagement.Orders", comboBox7);

PopulateComboBoxWithFieldNames("TicketSalesManagement.Tickets", comboBox6);

string query = @"

SELECT f.FlightId, COUNT(t.TicketId) AS TicketCount

FROM [TicketSalesManagement].[Flights] f

JOIN [TicketSalesManagement].[Departures] d ON f.FlightId = d.FlightId

JOIN [TicketSalesManagement].[Tickets] t ON t.DepartureId = d.DepartureId

GROUP BY f.FlightId";

FillChart(query, "FlightId", "TicketCount", "Number of tickets", chart1);

}

catch (Exception ex)

{

MessageBox.Show("Error: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

finally

{

sqlConnection.Close();

}

}

private void exitToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (sqlConnection != null && sqlConnection.State != ConnectionState.Closed)

{

sqlConnection.Close();

}

}

private void Form1\_FormClosed(object sender, FormClosedEventArgs e)

{

if (sqlConnection != null && sqlConnection.State != ConnectionState.Closed)

{

sqlConnection.Close();

}

}

private void refreshToolStripMenuItem\_Click(object sender, EventArgs e)

{

sqlConnection.Open();

try

{

PopulateComboBox("FlightId", "TicketSalesManagement.Flights", comboBox2);

PopulateComboBoxWithFieldNames("TicketSalesManagement.Users", comboBox4);

PopulateComboBoxWithFieldNames("TicketSalesManagement.Orders", comboBox7);

string query = @"

SELECT f.FlightId, COUNT(t.TicketId) AS TicketCount

FROM [TicketSalesManagement].[Flights] f

JOIN [TicketSalesManagement].[Departures] d ON f.FlightId = d.FlightId

JOIN [TicketSalesManagement].[Tickets] t ON t.DepartureId = d.DepartureId

GROUP BY f.FlightId";

FillChart(query, "FlightId", "TicketCount", "Number of tickets", chart1);

}

catch (Exception ex)

{

MessageBox.Show("Error: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

finally

{

sqlConnection.Close();

}

}

private DataTable SelectAllFromCommand(string tableName)

{

string query = @"SELECT \* FROM " + tableName;

SqlDataAdapter adapter = new SqlDataAdapter(query, sqlConnection);

DataTable dataTable = new DataTable();

adapter.Fill(dataTable);

return dataTable;

}

private void SelectUsers(object sender, EventArgs e)

{

try

{

DataTable dataTable = SelectAllFromCommand("TicketSalesManagement.Users");

dataGridView1.DataSource = dataTable;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message.ToString(), ex.Source.ToString(), MessageBoxButtons.OK, MessageBoxIcon.Error);

}

finally { sqlConnection.Close(); }

}

private void SelectOrders(object sender, EventArgs e)

{

try

{

DataTable dataTable = SelectAllFromCommand("TicketSalesManagement.Orders");

dataGridView2.DataSource = dataTable;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message.ToString(), ex.Source.ToString(), MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void SelectTickets(object sender, EventArgs e)

{

try

{

DataTable dataTable = SelectAllFromCommand("TicketSalesManagement.Tickets");

dataGridView3.DataSource = dataTable;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message.ToString(), ex.Source.ToString(), MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void SelectDepartures(object sender, EventArgs e)

{

try

{

DataTable dataTable = SelectAllFromCommand("TicketSalesManagement.Departures");

dataGridView5.DataSource = dataTable;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message.ToString(), ex.Source.ToString(), MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void SelectFlights(object sender, EventArgs e)

{

try

{

DataTable dataTable = SelectAllFromCommand("TicketSalesManagement.Flights");

dataGridView4.DataSource = dataTable;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message.ToString(), ex.Source.ToString(), MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void SelectDestinationAirports(object sender, EventArgs e)

{

try

{

string query = @"SELECT da.AirportId, da.Code, da.Name AS ""Airport name"", ci.Name AS ""City"", co.Name AS ""Country""

FROM TicketSalesManagement.DestinationAirports da

JOIN TicketSalesManagement.Cities ci

ON da.CityId = ci.CityId

JOIN TicketSalesManagement.Countries co

ON ci.CountryId = co.CountryId";

SqlDataAdapter adapter = new SqlDataAdapter(query, sqlConnection);

DataTable dataTable = new DataTable();

adapter.Fill(dataTable);

dataGridView6.DataSource = dataTable;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message.ToString(), ex.Source.ToString(), MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private async void button1\_Click(object sender, EventArgs e) // If UserId is digit

{

await sqlConnection.OpenAsync();

DateTime today = DateTime.Today;

if (!string.IsNullOrEmpty(textBox1.Text) && !string.IsNullOrWhiteSpace(textBox1.Text) &&

!string.IsNullOrEmpty(textBox6.Text) && !string.IsNullOrWhiteSpace(textBox6.Text) &&

!string.IsNullOrEmpty(textBox5.Text) && !string.IsNullOrWhiteSpace(textBox5.Text) &&

!string.IsNullOrEmpty(textBox4.Text) && !string.IsNullOrWhiteSpace(textBox4.Text) &&

!string.IsNullOrEmpty(textBox2.Text) && !string.IsNullOrWhiteSpace(textBox2.Text) &&

comboBox1.SelectedIndex != -1 &&

dateTimePicker1.Value != DateTime.MinValue &&

dateTimePicker2.Value != DateTime.MinValue)

{

if (dateTimePicker1.Value < today)

{

if (dateTimePicker2.Value > today)

{

try

{

SqlCommand command = new SqlCommand("INSERT INTO TicketSalesManagement.Users (Name, Surname, Sex, DateOfBirth, PassportNumber, DateOfExpiry, PhoneNumber, Email) VALUES(@Name, @Surname, @Sex, @DateOfBirth, @PassportNumber, @DateOfExpiry, @PhoneNumber, @Email)", sqlConnection);

command.Parameters.AddWithValue("Name", textBox1.Text);

command.Parameters.AddWithValue("Surname", textBox6.Text);

command.Parameters.AddWithValue("PassportNumber", textBox5.Text);

command.Parameters.AddWithValue("PhoneNumber", textBox4.Text);

command.Parameters.AddWithValue("Email", textBox2.Text);

command.Parameters.AddWithValue("DateOfBirth", dateTimePicker1.Value);

command.Parameters.AddWithValue("DateOfExpiry", dateTimePicker2.Value);

command.Parameters.AddWithValue("Sex", comboBox1.SelectedItem.ToString());

await command.ExecuteNonQueryAsync();

MessageBox.Show("Data inserted successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show("Error inserting data: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

finally

{

sqlConnection.Close();

}

}

else

{

MessageBox.Show("The date of expiry cannot be less than today!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

else

{

MessageBox.Show("The birth date cannot be greater than today!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

else

{

MessageBox.Show("All fields must be filled!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private async void button2\_Click(object sender, EventArgs e) // If UserId is digit

{

await sqlConnection.OpenAsync();

DateTime now = DateTime.Now;

if (!string.IsNullOrEmpty(textBox10.Text) && !string.IsNullOrWhiteSpace(textBox10.Text))

{

try

{

if (dateTimePicker3.Value <= now && dateTimePicker4.Value <= now)

{

if (!IfExist("UserId", "TicketSalesManagement.Users", textBox10.Text))

{

MessageBox.Show("Entered UserId does not exist!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

SqlCommand command = new SqlCommand("INSERT INTO TicketSalesManagement.Orders (UserId, OrderDate, PaymentDate) VALUES(@UserId, @OrderDate, @PaymentDate)", sqlConnection);

command.Parameters.AddWithValue("UserId", textBox10.Text);

command.Parameters.AddWithValue("OrderDate", dateTimePicker3.Value);

command.Parameters.AddWithValue("PaymentDate", dateTimePicker4.Value);

await command.ExecuteNonQueryAsync();

MessageBox.Show("Data deleted successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

else

{

MessageBox.Show("Order date and payment date cannot be greater than today!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

catch (Exception ex)

{

MessageBox.Show("Error inserting data: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

finally

{

sqlConnection.Close();

}

}

else

{

MessageBox.Show("All fields must be filled!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private async void button3\_Click(object sender, EventArgs e) // If UserId, ReceiptId, DepartureId is digit

{

await sqlConnection.OpenAsync();

if (!string.IsNullOrEmpty(textBox15.Text) && !string.IsNullOrWhiteSpace(textBox15.Text) &&

!string.IsNullOrEmpty(textBox11.Text) && !string.IsNullOrWhiteSpace(textBox11.Text) &&

!string.IsNullOrEmpty(textBox12.Text) && !string.IsNullOrWhiteSpace(textBox12.Text) &&

!string.IsNullOrEmpty(textBox13.Text) && !string.IsNullOrWhiteSpace(textBox13.Text) &&

!string.IsNullOrEmpty(textBox14.Text) && !string.IsNullOrWhiteSpace(textBox14.Text) &&

comboBox3.SelectedIndex != -1)

{

if (!IfExist("UserId", "TicketSalesManagement.Users", textBox15.Text))

{

MessageBox.Show("Entered UserId does not exist!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else if (!IfExist("RecieptId", "TicketSalesManagement.Orders", textBox11.Text)) // In the Orders table, the field is mistakenly named "ReceiptId", but in others "ReceiptId".

{

MessageBox.Show("Entered ReceiptId does not exist!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else if (!IfExist("DepartureId", "TicketSalesManagement.Departures", textBox12.Text))

{

MessageBox.Show("Entered DepartureId does not exist!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

try

{

SqlCommand command = new SqlCommand("INSERT INTO TicketSalesManagement.Tickets (UserId, ReceiptId, DepartureId, ClassId, Seat, Price) VALUES(@UserId, @ReceiptId, @DepartureId, @ClassId, @Seat, @Price)", sqlConnection);

command.Parameters.AddWithValue("UserId", textBox15.Text);

command.Parameters.AddWithValue("ReceiptId", textBox11.Text);

command.Parameters.AddWithValue("DepartureId", textBox12.Text);

command.Parameters.AddWithValue("ClassId", comboBox3.SelectedIndex + 1);

command.Parameters.AddWithValue("Seat", textBox13.Text);

command.Parameters.AddWithValue("Price", textBox14.Text);

await command.ExecuteNonQueryAsync();

MessageBox.Show("Data inserted successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show("Error inserting data: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

finally

{

sqlConnection.Close();

}

}

}

else

{

MessageBox.Show("All fields must be filled!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private async void button5\_Click(object sender, EventArgs e)

{

await sqlConnection.OpenAsync();

if (comboBox2.SelectedIndex != -1 && IfExist("FlightId", "TicketSalesManagement.Flights", comboBox2.SelectedItem.ToString()))

{

try

{

SqlCommand command = new SqlCommand("INSERT INTO TicketSalesManagement.Departures (DepartureDate , FlightId) VALUES(@DepartureDate, @FlightId)", sqlConnection);

command.Parameters.AddWithValue("DepartureDate", dateTimePicker10.Value);

command.Parameters.AddWithValue("FlightId", comboBox2.SelectedItem.ToString());

await command.ExecuteNonQueryAsync();

MessageBox.Show("Data inserted successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show("Error inserting data: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

finally

{

sqlConnection.Close();

}

}

else

{

MessageBox.Show("DepartureId does not exist or is not entered!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private async void button4\_Click(object sender, EventArgs e) // If AirportId is digit

{

await sqlConnection.OpenAsync();

if (!string.IsNullOrEmpty(textBox20.Text) && !string.IsNullOrWhiteSpace(textBox20.Text))

{

if(!IfExist("FlightId", "TicketSalesManagement.Flights", textBox3.Text))

{

if(IfExist("AirportId", "TicketSalesManagement.DestinationAirports", textBox20.Text))

{

try

{

SqlCommand command = new SqlCommand("INSERT INTO TicketSalesManagement.Flights (FlightId, AirportId , DepartureTime, ArrivalTime) VALUES(@FlightId, @AirportId , @DepartureTime, @ArrivalTime)", sqlConnection);

command.Parameters.AddWithValue("FlightId", textBox3.Text);

command.Parameters.AddWithValue("AirportId", textBox20.Text);

command.Parameters.AddWithValue("DepartureTime", dateTimePicker7.Value);

command.Parameters.AddWithValue("ArrivalTime", dateTimePicker8.Value);

await command.ExecuteNonQueryAsync();

MessageBox.Show("Data inserted successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show("Error inserting data: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

finally

{

sqlConnection.Close();

}

}

else

{

MessageBox.Show("Entered AirportId does not exist!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

else

{

MessageBox.Show("Entered FlightId already exist!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

else

{

MessageBox.Show("All fields must be filled!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void UpdateFieldValue(string tableName, string fieldName, string newValue, string idField, int idValue)

{

if(!string.IsNullOrEmpty(newValue) && !string.IsNullOrWhiteSpace(newValue) &&

!string.IsNullOrEmpty(idValue.ToString()))

{

try

{

SqlCommand command = new SqlCommand($"UPDATE {tableName} SET {fieldName} = @newValue WHERE {idField} = @idValue", sqlConnection);

command.Parameters.AddWithValue("@newValue", newValue);

command.Parameters.AddWithValue("@idValue", idValue);

command.ExecuteNonQuery();

MessageBox.Show("Data updated successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show("Error: " + ex.Message);

}

}

else

{

MessageBox.Show("All fields must be filled!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void UpdateFieldValue(string tableName, string fieldName, DateTime newValue, string idField, int idValue)

{

try

{

SqlCommand command = new SqlCommand($"UPDATE {tableName} SET {fieldName} = @newValue WHERE {idField} = @idValue", sqlConnection);

command.Parameters.AddWithValue("@newValue", newValue);

command.Parameters.AddWithValue("@idValue", idValue);

command.ExecuteNonQuery();

MessageBox.Show("Data updated successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show("Error: " + ex.Message);

}

finally

{

sqlConnection.Close();

}

}

private bool IfExist (string fieldName, string tableName, string value)

{

try

{

string query = $"SELECT COUNT(\*) FROM {tableName} WHERE {fieldName} = '{value}'";

SqlCommand whereCommand = new SqlCommand(query, sqlConnection);

int count = (int)whereCommand.ExecuteScalar();

return count != 0;

}

catch (Exception ex)

{

MessageBox.Show("Error: " + ex.Message);

return false;

}

}

private void PopulateComboBox(string fieldName, string tableName, ComboBox comboBox)

{

try

{

string query = $"SELECT {fieldName} FROM {tableName}";

SqlCommand command = new SqlCommand(query, sqlConnection);

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

string value = reader.GetString(0);

comboBox.Items.Add(value);

}

}

catch (Exception ex)

{

MessageBox.Show("Error: " + ex.Message);

}

finally

{

sqlConnection.Close();

}

}

private void PopulateComboBoxWithFieldNames(string tableName, ComboBox comboBox)

{

sqlConnection.Open();

try

{

SqlDataAdapter adapter = new SqlDataAdapter($"SELECT TOP 1 \* FROM {tableName}", sqlConnection);

DataTable table = new DataTable();

adapter.FillSchema(table, SchemaType.Source);

for (int i = 1; i < table.Columns.Count; i++)

{

comboBox.Items.Add(table.Columns[i].ColumnName);

}

}

catch (Exception ex)

{

MessageBox.Show("Error: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

finally

{

sqlConnection.Close();

}

}

private async void button6\_Click(object sender, EventArgs e)

{

sqlConnection.Open();

if (!string.IsNullOrEmpty(textBox7.Text) && !string.IsNullOrWhiteSpace(textBox7.Text) &&

comboBox4.SelectedIndex != -1)

{

string tableName = "TicketSalesManagement.Users";

string idField = "UserId";

string idValue = textBox7.Text;

if (IfExist(idField, tableName, idValue.ToString()))

{

try

{

switch (comboBox4.SelectedItem)

{

case "Name":

{

UpdateFieldValue(tableName, "Name", textBox8.Text, idField, Convert.ToInt32(idValue));

break;

}

case "Surname":

{

UpdateFieldValue(tableName, "Surname", textBox8.Text, idField, Convert.ToInt32(idValue));

break;

}

case "Sex":

{

UpdateFieldValue(tableName, "Sex", comboBox5.SelectedItem.ToString(), idField, Convert.ToInt32(idValue));

break;

}

case "DateOfBirth":

{

UpdateFieldValue(tableName, "DateOfBirth", dateTimePicker5.Value, idField, Convert.ToInt32(idValue));

break;

}

case "PassportNumber":

{

UpdateFieldValue(tableName, "PassportNumber", textBox8.Text, idField, Convert.ToInt32(idValue));

break;

}

case "DateOfExpiry":

{

UpdateFieldValue(tableName, "DateOfExpiry", dateTimePicker5.Value, idField, Convert.ToInt32(idValue));

break;

}

case "PhoneNumber":

{

UpdateFieldValue(tableName, "PhoneNumber", textBox8.Text, idField, Convert.ToInt32(idValue));

break;

}

case "Email":

{

UpdateFieldValue(tableName, "Email", textBox8.Text, idField, Convert.ToInt32(idValue));

break;

}

}

}

catch (Exception ex)

{

MessageBox.Show("Error inserting data: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

finally

{

sqlConnection.Close();

}

}

else

{

MessageBox.Show("Entered UserId does not exist!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

else

{

MessageBox.Show("All fields must be filled!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void comboBox4\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox4.SelectedItem.ToString() == "Name"

|| comboBox4.SelectedItem.ToString() == "Surname"

|| comboBox4.SelectedItem.ToString() == "PassportNumber"

|| comboBox4.SelectedItem.ToString() == "PhoneNumber"

|| comboBox4.SelectedItem.ToString() == "Email")

{

comboBox5.Visible = false;

dateTimePicker5.Visible = false;

textBox8.Clear();

textBox8.Visible = true;

}

else if(comboBox4.SelectedItem.ToString() == "Sex")

{

textBox8.Visible = false;

dateTimePicker5.Visible = false;

comboBox5.SelectedIndex = -1;

comboBox5.Items.Add("M");

comboBox5.Items.Add("F");

comboBox5.Visible = true;

}

else if(comboBox4.SelectedItem.ToString() == "DateOfExpiry"

|| comboBox4.SelectedItem.ToString() == "DateOfBirth")

{

textBox8.Visible = false;

comboBox5.Visible = false;

dateTimePicker5.Visible = true;

}

else

{

textBox8.Visible = false;

comboBox5.Visible = false;

dateTimePicker5.Visible = false;

}

}

private async void button7\_Click(object sender, EventArgs e)

{

await sqlConnection.OpenAsync();

if (!string.IsNullOrEmpty(textBox7.Text) && !string.IsNullOrWhiteSpace(textBox7.Text) &&

comboBox4.SelectedIndex != -1)

{

string tableName = "TicketSalesManagement.Users";

string idField = "RecieptId";

int idValue = Convert.ToInt32(textBox7.Text);

switch (comboBox4.SelectedItem)

{

case "UserId":

{

UpdateFieldValue(tableName, "UserId", textBox8.Text, idField, idValue);

break;

}

case "OrderDate":

{

UpdateFieldValue(tableName, "OrderDate", dateTimePicker6.Value, idField, idValue);

break;

}

case "PaymentDate":

{

UpdateFieldValue(tableName, "PaymentDate", dateTimePicker6.Value, idField, idValue);

break;

}

}

}

}

private void comboBox7\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBox7.SelectedItem.ToString() == "UserId")

{

dateTimePicker6.Visible = false;

textBox9.Clear();

textBox9.Visible = true;

}

else if (comboBox7.SelectedItem.ToString() == "OrderDate"

|| comboBox7.SelectedItem.ToString() == "PaymentDate")

{

textBox9.Visible = false;

dateTimePicker6.Visible = true;

}

}

private async void button8\_Click(object sender, EventArgs e)

{

await sqlConnection.OpenAsync();

if (!string.IsNullOrEmpty(textBox18.Text))

{

try

{

if(IfExist("UserId", "TicketSalesManagement.Users", textBox18.Text))

{

SqlCommand command = new SqlCommand($"DELETE FROM TicketSalesManagement.Users WHERE UserId = @UserId", sqlConnection);

command.Parameters.AddWithValue("@UserId", textBox18.Text);

await command.ExecuteNonQueryAsync();

MessageBox.Show("Data deleted successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show("Entered UserId does not exist!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

catch (Exception ex)

{

MessageBox.Show("Error: " + ex.Message);

}

finally

{

sqlConnection.Close();

}

}

}

private async void FillChart(string query, string xValue, string yValue, string legendText, Chart chart)

{

await sqlConnection.OpenAsync();

using (SqlCommand command = new SqlCommand(query, sqlConnection))

{

DataTable dataTable = new DataTable();

using (SqlDataAdapter adapter = new SqlDataAdapter(command))

{

adapter.Fill(dataTable);

}

sqlConnection.Close();

chart.DataSource = dataTable;

chart.Series["Series1"].XValueMember = xValue;

chart.Series["Series1"].YValueMembers = yValue;

chart.Series["Series1"].ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Column;

chart.Series["Series1"].LegendText = legendText;

}

}

private void tabControl7\_Click(object sender, EventArgs e)

{

string query = @"

SELECT f.FlightId, COUNT(t.TicketId) AS TicketCount

FROM [TicketSalesManagement].[Flights] f

JOIN [TicketSalesManagement].[Departures] d ON f.FlightId = d.FlightId

JOIN [TicketSalesManagement].[Tickets] t ON t.DepartureId = d.DepartureId

GROUP BY f.FlightId";

FillChart(query, "FlightId", "TicketCount", "Number of tickets", chart1);

}

private void chart2\_Click(object sender, EventArgs e)

{

string query = @"

SELECT TOP (3) co.Name AS Country, COUNT(TicketId) AS TicketCount

FROM TicketSalesManagement.Tickets t

JOIN TicketSalesManagement.Departures d

ON t.DepartureId = d.DepartureId

JOIN TicketSalesManagement.Flights f

ON f.FlightId = d.FlightId

JOIN TicketSalesManagement.DestinationAirports da

ON f.AirportId = da.AirportId

JOIN TicketSalesManagement.Cities ci

ON da.CityId = ci.CityId

JOIN TicketSalesManagement.Countries co

ON ci.CountryId = co.CountryId

GROUP BY co.Name

ORDER BY TicketCount DESC;";

FillChart(query, "Country", "TicketCount", "Number of sold tickets", chart2);

}

private async void button13\_Click(object sender, EventArgs e)

{

await sqlConnection.OpenAsync();

if (!string.IsNullOrEmpty(textBox24.Text) && !string.IsNullOrWhiteSpace(textBox24.Text) &&

comboBox6.SelectedIndex != -1)

{

string tableName = "TicketSalesManagement.Tickets";

string idField = "TicketId";

int idValue = Convert.ToInt32(textBox24.Text);

switch (comboBox6.SelectedItem)

{

case "UserId":

{

UpdateFieldValue(tableName, "UserId", textBox23.Text, idField, idValue);

break;

}

case "ReceiptId":

{

UpdateFieldValue(tableName, "ReceiptId", textBox23.Text, idField, idValue);

break;

}

case "ClassId":

{

PopulateComboBoxWithFieldNames("TicketSalesManagement.Classes", comboBox10);

UpdateFieldValue(tableName, "ClassId", comboBox10.SelectedItem.ToString(), idField, Convert.ToInt32(idValue));

break;

}

case "Seat":

{

UpdateFieldValue(tableName, "Seat", textBox23.Text, idField, idValue);

break;

}

case "Price":

{

UpdateFieldValue(tableName, "Price", textBox23.Text, idField, idValue);

break;

}

case "DepartureId":

{

UpdateFieldValue(tableName, "DepartureId", textBox23.Text, idField, idValue);

break;

}

}

}

}

}

}