НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

(повна назва інституту/факультету)

КАФЕДРА інформатики та програмної інженерії

(повна назва кафедри)

**КУРСОВА РОБОТА**

з дисципліни «Бази даних»

(назва дисципліни)

на тему:\_\_\_\_\_База даних аеропорту\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Студента (ки) \_\_2\_\_ курсу \_ІП-21\_\_\_\_\_ групи

спеціальності 121 «Інженерія програмного забезпечення»

\_Скрипець Ольги Олександрівни\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник Ліщук Катерина Ігорівна\_\_\_\_\_\_\_\_\_

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

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_ Оцінка ECTS \_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ – 2023 рік

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

Факультет Інформатики та обчислювальної техніки

(повна назва)

Кафедра Інформатики та програмної інженерії

(повна назва)

Дисципліна Бази даних

Курс \_\_2\_\_\_ Група \_\_ІП-21\_\_ Семестр \_\_\_3\_\_

**З А В Д А Н Н Я**

**НА КУРСОВУ РОБОТУ СТУДЕНТУ**

\_\_\_\_\_\_\_Скрипець Ользі Олександрівні\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові)

1. Тема роботи\_\_\_\_\_база даних аеропорту\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

керівник роботи\_\_Ліщук Катерина Ігорівна\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи\_\_\_\_\_29.12.2023\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Вихідні дані до роботи\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

­ створена база даних\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_1) Аналіз предметного середовища

2) Побудова ER-моделі

3) Побудова реляційної схеми з ER-моделі

4) Створення бази даних, у форматі обраної системи управління базою даних

5) Створення користувачів бази даних

6) Імпорт даних з використанням засобів СУБД в створену базу даних

7) Створення мовою SQL запитів

8) Оптимізація роботи запитів

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

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

6. Дата видачі завдання\_\_\_\_08.11.2023**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів виконання курсового проекту | Строк виконання етапів проекту | Примітка |
| 1 | Аналіз предметного середовища | 12.11.2023 |  |
| 2 | Побудова ER-моделі | 01.12.2023 |  |
| 3 | Побудова реляційної схеми з ER-моделі | 05.12.2023 |  |
| 4 | Створення бази даних, у форматі обраної системи управління базою даних | 10.12.2023 |  |
| 5 | Створення користувачів бази даних | 19.12.2023 |  |
| 6 | Імпорт даних з використанням засобів СУБД в створену базу даних | 21.12.2023 |  |
| 7 | Створення мовою SQL запитів | 24.12.2023 |  |
| 8 | Оптимізація роботи запитів | 25.12.2023 |  |
| 9 | Оформлення пояснювальної записки | 25.12.2023 |  |
| 10 | Захист курсової роботи | 30.12.2023 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Студент \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_**Скрипець О.О**\_\_\_**

(підпис ) (прізвище та ініціали)

**Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_**Ліщук К. І.**\_\_\_\_\_\_**

(підпис ) (прізвище та ініціали)

ЗМІСТ

[ВСТУП 5](#_Toc154761621)

[1 ОПИС ПРЕДМЕТНОГО СЕРЕДОВИЩА 6](#_Toc154761622)

[1.1 Опис об’єкту дослідження 6](#_Toc154761623)

[1.2 Опис вхідних даних 7](#_Toc154761624)

[1.3 Опис вихідних даних 7](#_Toc154761625)

[1.3 Опис основних бізнес-процесів використання даних 8](#_Toc154761626)

[2 АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ ПРОДУКТІВ 9](#_Toc154761627)

[3 ПОСТАНОВКА ЗАВДАННЯ 12](#_Toc154761628)

[4 ПОБУДОВА ER-МОДЕЛІ 14](#_Toc154761629)

[4.1 Бізнес-правила 14](#_Toc154761630)

[4.2 Сутності 15](#_Toc154761631)

[4.3 Набори атрибутів для сутностей 15](#_Toc154761632)

[4.4 Структурні зв’язки між об’єктами 16](#_Toc154761633)

[5 РЕАЛІЗАЦІЯ БАЗИ ДАНИХ 19](#_Toc154761634)

[5.1 Вибір СУБД 19](#_Toc154761635)

[5.2 Характеристика таблиць 20](#_Toc154761636)

[5.3 Методи забезпечення цілісності даних 24](#_Toc154761637)

[5.3 Схема бази даних, реалізована засобами СУБД 25](#_Toc154761638)

[5.6 Створення користувачів 26](#_Toc154761639)

[6 РОБОТА З БАЗОЮ ДАНИХ 27](#_Toc154761640)

[6.1 Тексти генераторів 27](#_Toc154761641)

[6.2 Тексти збережених процедур/функцій 32](#_Toc154761642)

[6.3 Тексти тригерів 44](#_Toc154761643)

[6.4 Тексти представлень 51](#_Toc154761644)

[6.5 SQL-запити 54](#_Toc154761645)

[6.6 Результати оптимізації 72](#_Toc154761646)

[ВИСНОВКИ 73](#_Toc154761647)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 76](#_Toc154761648)

# ВСТУП

У сучасному світі, де технології швидко розвиваються та впроваджуються в різні галузі, розробка та ефективне використання баз даних стає необхідною складовою успішної діяльності підприємств. Раніше, архіви і паперова документація ускладнювали роботу, а пошук необхідної інформації вимагав великих зусиль та часу. Завдяки цифровим базам даних цей процес став більш ефективним та швидким.

Обрана тема "База даних аеропорту" виявляється надзвичайно актуальною в контексті сучасних вимог до обслуговування пасажирів. Зазвичай, аеропорти є місцями великого скупчення людей, що породжує необхідність вдосконалення системи обробки та обліку пасажирів. Метою цієї курсової роботи є створення ефективної бази даних для аеропорту, спрямованої на впровадження самообслуговування для пасажирів.

Проект передбачає розробку системи, яка дозволить клієнтам здійснювати бронювання місць на рейси через онлайн-сервіс, надаючи можливість зручного та швидкого доступу до інформації про вільні місця. Такий підхід покликаний полегшити процес оформлення та зменшити час очікування пасажирів в аеропорту.

Вибір системи управління базами даних (СУБД) є важливим аспектом розробки. Орієнтуючись на особливості проекту та його вимоги, було вибраноMySGL**,** яка надає необхідні можливості для ефективного зберігання та обробки великого обсягу даних.

Ця курсова робота спрямована на створення інноваційного інструменту для покращення обслуговування пасажирів в аеропорту, сприяючи вдосконаленню процесів бронювання та забезпечуючи зручний та швидкий доступ до інформації для кожного пасажира.

# 1 ОПИС ПРЕДМЕТНОГО СЕРЕДОВИЩА

## 1.1 Опис об’єкту дослідження

На сучасному етапі розвитку транспортної інфраструктури літаки визнані одними з найбезпечніших та найшвидших видів транспорту. Організація їх польотів вимагає вдосконалених систем та ефективного управління, щоб забезпечити безпеку та комфорт для пасажирів.

Об'єктом мого дослідження є комплексний аеропортний процес, який включає в себе вибір літаків для кожного рейсу, а також організацію безпечної посадки та вильоту. Кожен аспект цього процесу важливий для забезпечення оптимального функціонування аеропорту та надання високоякісного обслуговування пасажирів.

Для вибору літака для конкретного рейсу враховуються різноманітні параметри, такі як кількість пасажирів, марка та інші технічні характеристики. Інформацію про кожен літак зручно узагальнити у вигляді структурованої таблиці, що дозволяє ефективно використовувати ресурси та оптимізувати вибір літаків для різних рейсів.

Однак ефективність аеропортної діяльності не обмежується лише вибором літаків. Для оптимального використання часу зайнятості смуги та уникнення конфліктів у розкладі рейсів, важливо створити детальний графік відбуття та прибуття літаків. Це вимагає комплексного планування, обліку різних факторів та врахування поточного обсягу робіт.

Окрім того, з огляду на важливість безпеки пасажирів, у кожному аеропорту перед посадкою на літак виконується обов'язкова перевірка кожної людини та її багажу. Цей аспект вимагає впровадження ефективних систем контролю та координації, щоб забезпечити високий ступінь безпеки та виключити можливі ризики.

Отже, предметне середовище нашого дослідження охоплює весь цикл аеропортної діяльності – від вибору літака до його відльоту, з фокусом на оптимізацію процесів та забезпечення безпеки та зручності для пасажирів.

## 1.2 Опис вхідних даних

Для успішного функціонування бази даних аеропорту, необхідно збирати та обробляти різноманітні дані, що входять до складу вхідної інформації. Перш за все, це включає в себе дані про наявність літаків та їх технічні характеристики, такі як вантажопідйомність, кількість пасажирів, модель та інші параметри. Ця інформація надходить в реальному часі від авіакомпаній та технічних служб аеропорту.

Крім того, для ефективного планування рейсів необхідно мати дані про графік доступності та використання польотних смуг. Це включає інформацію про час, коли конкретна смуга вільна для відправлення та прибуття літаків.

Додатковими вхідними даними є інформація про пасажирів, яка включає в себе дані для обов'язкової перевірки перед посадкою, такі як особисті дані, дані про багаж та інші важливі деталі. Ці дані можуть надходити від пасажирів при бронюванні квитків чи під час реєстрації в аеропорту.

## 1.3 Опис вихідних даних

У результаті обробки та аналізу вхідних даних система бази даних аеропорту надає ряд вихідних даних, які є основою для ефективного управління аеропортним процесом. Серед цих даних - рекомендації щодо вибору літака для конкретного рейсу, що враховують різні технічні та експлуатаційні характеристики.

Окрім того, система надає графік відбуття та прибуття літаків, враховуючи часові обмеження та доступність польотних смуг. Цей графік є основою для оптимального розподілу ресурсів та максимізації використання аеропортної інфраструктури.

Загальною метою вихідних даних є забезпечення оптимального та безпечного обслуговування пасажирів, ефективного використання ресурсів аеропорту та підвищення загального рівня ефективності аеропортної діяльності.

## 1.3 Опис основних бізнес-процесів використання даних

* Вибір літака для рейсу

Цей процес включає аналіз технічних характеристик літаків та їхню відповідність вимогам конкретного рейсу. Враховуються параметри, такі як вантажопідйомність, кількість пасажирів, наявність спеціального обладнання та інші технічні аспекти.

* Графік рейсів

Цей процес включає створення графіку відбуття та прибуття літаків з урахуванням доступності та завантаженості польотних смуг.

* Безпека пасажирів

Цей процес охоплює обов'язкову перевірку пасажирів та їхнього багажу перед посадкою на літак. Використовуються дані, отримані при реєстрації та бронюванні, для визначення безпеки та відсутності загроз для польоту.

* Керування літаками

Цей процес передбачає ефективне керування літаками. Інформація про стан літаків та останній технічний сервіс використовується для забезпечення безперебійності роботи та безпеки польотів.

# 2 АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ ПРОДУКТІВ

* [Avia.tickets.ua](https://avia.tickets.ua/)**:**

Ця платформа дозволяє користувачам здійснювати бронювання квитків на різні рейси та маршрути. Також вона дозволяє порівнювати ціни на квитки від різних авіакомпаній для вибору оптимального варіанту. А користувачі можуть отримати консультації та підтримку в режимі онлайн[1].

Переваги:

Широкий вибір авіакомпаній та рейсів.

Зручний інтерфейс та інтуїтивно зрозумілі опції бронювання.

Порівняння цін для зручного вибору користувача.

Можливість побудови складного маршруту

Недоліки:

Обмежена функціональність для аеропортів та їх управління.

Відсутність розширених можливостей для аналітики та оптимізації рейсів управління.

* [TripMyDream](https://tripmydream.ua/flights)**:**

Ця платформа дозволяє здійснювати пошук та бронювання авіаквитків. Користувачі можуть порівнювати різні пропозиції авіакомпаній для вибору оптимального варіанту. Платформа пропонує індивідуальний підбір турів та маршрутів для пасажирів[2].

Переваги:

Інноваційний підхід до підбору турів та квитків.

Гнучкі опції бронювання та зручний інтерфейс.

Індивідуальні пропозиції та рекомендації для пасажирів.

Недоліки:

Обмежена кількість джерел для порівняння цін на квитки.

Зосередженість на індивідуальному підборі турів, що може не враховувати потреби аеропорту та управління рейсами.

* [Booking.com Flights](https://www.booking.com/flights/index.uk.html?aid=2151979&label=Ukrainian-UK-124331077386-BTUdIFpHlS%2APhkn%2AovTDwAS533957472387%3Apl%3Ata%3Ap1%3Ap2%3Aac%3Aap%3Aneg%3Afi%3Atidsa-1633144487656%3Alp9061017%3Ali%3Adec%3Adm&sid=5bc97abc83a78eb0c618533ec8f9bc82)

Booking.com Flights надає користувачам можливість здійснювати пошук та бронювання авіаквитків для різних напрямків. Користувачі можуть використовувати різні фільтри та опції сортування для вибору оптимального варіанту. Платформа пропонує інтерактивну мапу маршрутів, на якій можна вибрати відповідні рейси та точки призначення. Booking.com дозволяє вибирати різні опції бронювання, включаючи гнучкі тарифи та умови скасування[3].

Переваги:

Забезпечує доступ до великої кількості авіаквитків від різних авіакомпаній.

Дозволяє користувачам вибирати рейси зручним способом, використовуючи мапу.

Надає можливість користувачам вибирати тарифи та умови скасування.

Недоліки:

Booking.com славиться своєю готельною платформою, тому аеро-функціонал може бути менш розвиненим порівняно із спеціалізованими сервісами.

Всі три програмних продукти володіють своїми перевагами та недоліками. [Avia.tickets.ua](https://avia.tickets.ua/) вражає широким вибором та зручним інтерфейсом для пасажирів, але у ньому відсутній фокус на оптимізації роботи аеропортів. З іншого боку, [TripMyDream](https://tripmydream.ua/flights) пропонує індивідуальний підбір турів, але обмежується кількістю джерел порівняння цін та не враховує потреби управління рейсами.

Booking.com Flights представляє широкий вибір авіаквитків та забезпечує користувачам гнучкі опції бронювання. Інтерактивна мапа маршрутів створює зручний інструмент для вибору рейсів. Однак, через свою загальну орієнтацію на готельний бізнес, Booking.com не має розширених функцій, специфічних для потреб авіаційного управління.

Загальний висновок полягає в тому, що для впровадження ефективної системи управління аеропортом слід розглядати програмні продукти, які враховують потреби обох сторін: пасажирів і аеропортів. Ідеальне рішення має бути комплексним, забезпечуючи зручність для клієнтів та оптимізацію операцій для команди аеропорту.

# 3 ПОСТАНОВКА ЗАВДАННЯ

У результаті аналізу предметного середовища та програмних продуктів в області авіаційного транспорту, постало завдання створення власної бази даних для ефективного управління аеропортом.

Мета роботи - розробити систему, яка сприятиме оптимізації роботи аеропорту та забезпечить зручність для пасажирів.

Задачі цієї роботи:

* Перегляд доступних рейсів.

Забезпечити можливість користувачам переглядати інформацію про доступні рейси, включаючи час відправлення, прибуття та наявність вільних місць.

* Додавання багажу.

Розробити функціонал, що дозволяє пасажирам додавати інформацію про багаж та визначати його вагу.

* Бронювання квитка.

Забезпечити користувачам можливість бронювання квитка на обраний рейс, враховуючи наявність вільних місць та інші параметри.

* Перегляд власних бронювань.

Реалізувати можливість користувачам переглядати свої попередні та поточні бронювання, включаючи деталі та статуси.

* Перегляд інформації про користувачів.

Забезпечити працівникам аеропорту можливість переглядати важливу інформацію про пасажирів, включаючи особові дані та історію польотів.

* Перегляд інформації про багаж.

Розробити функціонал для працівників, який дозволяє переглядати інформацію про багаж пасажирів, включаючи вид та вагу.

* Доступ на рейс.

Створити систему, що дозволяє працівникам аеропорту визначати можливість чи неможливість пасажира приступити до рейсу, враховуючи всі фактори безпеки та реєстрації.

# 4 ПОБУДОВА ER-МОДЕЛІ

## 4.1 Бізнес-правила

* Обмеження дат бронювання.

Дата бронювання квитка не може перевищувати дату вильоту літака. Це обмеження гарантує, що пасажири не можуть бронювати квитки на рейси, які вже відбулися.

* Максимальна кількість білетів на рейс.

Неможливо забронювати більше білетів, ніж максимальна кількість місць на літаку. Це обмеження гарантує, що не буде перевантаження та всі пасажири матимуть гарантоване місце.

* Контроль за вантажем.

Зареєструвати багаж можна лише в межах максимальної вантажопідйомності літака. Це правило запобігає перевантаженню літака та забезпечує безпеку польотів.

* Обов'язкова авторизація для всіх операцій.

Для будь-якої операції в базі даних потрібна авторизація користувача. Це обмеження гарантує безпеку даних та забезпечує, що всі операції проводяться відповідальними особами.

* Перевірка повноліття пасажирів

Для будь-якої операції в базі даних потрібно переконатися що пасажир повнолітній

## 4.2 Сутності

Для успішної реалізації бази даних аеропорту необхідно визначити ключові сутності, які відображатимуть основні об'єкти та їх взаємозв'язки в контексті авіаційної діяльності. Кожна сутність відображає конкретний аспект інформаційної системи та взаємодії між її компонентами. Ось ці сутності:

* Пасажир
* Багаж
* Літак
* Характеристика
* Маршрут
* Графік
* Місто
* Бронювання
* Працівник
* Перевірка пасажира

## 4.3 Набори атрибутів для сутностей

Для кожної сутності бази даних аеропорту визначено та наведено у таблиці 4.3.1 набір атрибутів, які деталізують та характеризують конкретний об'єкт або процес в контексті авіаційної діяльності. Атрибути визначають властивості об'єктів, їх взаємозв'язки та параметри, необхідні для ефективної роботи бази даних.

Таблиця 4.3.1 – Сутності та їхні атрибути

|  |  |
| --- | --- |
| **Сутність** | **Атрибути** |
| Passenger | PassengerID, BaggageID, Pas\_name, Pas\_Lastname, Pas\_Surname, Passport\_number, Citizenship |
| Baggage | BaggageID, weight, Type, registration\_date |
| Plane | PlaneID, passengers\_number, carrying\_capacity, CharacteristicID, RouteID |
| Characteristic | CharacteristicID, brand, date\_last\_tech\_insp, Producing\_country |
| Route | RouteID, ScheduleID, Starting \_point, End\_point, Duration |
| Schedule | ScheduleID, Schedule\_date, Departure\_time, Arrival\_time |
| City | CityID, city\_name, country |
| Booking | BookingID, PassengerID, RouteID, BaggageID, booking\_date, booking\_time, |
| Employee | EmployeeID, Emp\_name, Emp\_Lastname, Emp\_Surname, date\_of\_birth |
| Passenger\_check | Passenger\_checkID, PassengerID, BaggageID, EmployeeID, check\_result |

## 4.4 Структурні зв’язки між об’єктами

Сутність Characteristic пов’язана з Plane зв’язком один до одного бо один літак має одну характеристику.

Сутність Plane пов’язана з Route зв’язком багато до одного бо багато літаків літають по одному маршруту.

Сутність Schedule пов’язана з Route зв’язком багато до одного бо один маршрут має багато графіків.

Сутність Booking пов’язана з Route зв’язком багато до одного бо на один маршрут може бути багато бронювань.

Сутність Route пов’язана з City зв’язком багато до одного бо одне місто може бути в багатьох маршрутах.

Сутність Booking пов’язана з Baggage зв’язком один до одного бо одна бронь включає один багаж.

Сутність Booking пов’язана з Passenger зв’язком один до одного бо одна бронь включає одного пасажира.

Сутність Passenger пов’язана з Baggage зв’язком один до одного бо одна людина реєструє один багаж.

Сутність Passenger\_check пов’язана з Baggage зв’язком один до одного бо одна перевірка перевіряє один багаж.

Сутність Passenger\_check пов’язана з Passenger зв’язком один до одного бо одна перевірка перевіряє одного пасажира.

Сутність Passenger\_check пов’язана з Employee зв’язком багато до одного бо один працівник робить багато перевірок.

На рисунку 4.4.1 представлена ER-діаграма, на якій зображено сутності та їх взаємозв'язки в базі даних аеропорту. Ця діаграма служить графічним засобом моделювання бази даних, що описує ключові елементи та їх зв'язки в контексті авіаційної діяльності. Взаємодія між цими сутностями визначається зв'язками, що відображають важливі аспекти в системі, такі як бронювання пасажирів, маршрути літаків та перевірка багажу. ER-діаграма є потужним інструментом для аналізу та проектування бази даних, допомагаючи зрозуміти взаємозв'язки між різними компонентами системи.

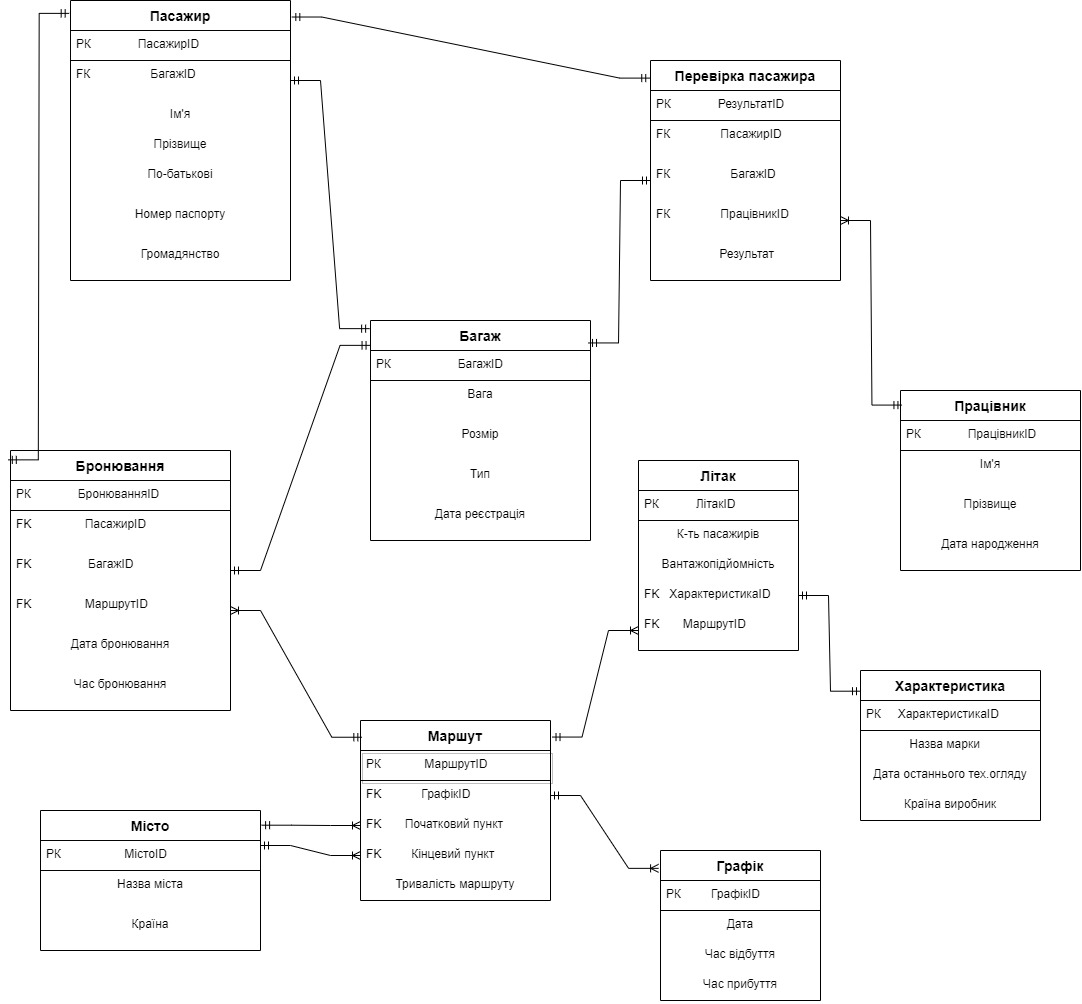


Рисунок 4.4.1 – ER-діаграма

# 5 РЕАЛІЗАЦІЯ БАЗИ ДАНИХ

## 5.1 Вибір СУБД

У процесі розробки бази даних для системи авіаперевезень, вибір був здійснений на користь MySQL Workbench з кількох ключових міркувань. Перш за все, MySQL Workbench є відкритим та безкоштовним інструментом, що робить його доступним для широкого кола користувачів. Це особливо важливо в контексті проекту, де ефективне співпрацювання розробників та можливість використовувати інструмент без значних фінансових витрат є ключовими факторами.

По-друге, MySQL Workbench відзначається простотою використання та інтуїтивним інтерфейсом. Це сприяє швидкому проектуванню та розробці бази даних без значного часового витрат на навчання та адаптацію.

Також важливим фактором вибору ставла сумісність із базою даних MySQL. Цей вибір логічний, оскільки база даних MySQL широко використовується в різних галузях, включаючи веб-розробку та сферу авіаперевезень.

Заслуговує уваги і функціональність MySQL Workbench, включаючи візуальне моделювання ER-діаграм, що полегшує процес проектування та створення бази даних. Цей інструмент також надає інтегровані засоби для співпраці розробників та забезпечує ефективний обмін інформацією у команді.

Отже, обрання MySQL Workbench було обґрунтовано його безкоштовністю, простотою використання, сумісністю із MySQL та наявністю необхідних функціональних можливостей для успішної розробки бази даних для системи аеропорту.

## 5.2 Характеристика таблиць

Таблиця 5.2.1 «employee» призначення для зберігання даних про працівника аеропорту який буде перевіряти пасажирів та їх багаж. Вона допомагає веденню детальної інформації про персонал, який забезпечує безпеку та контроль на території аеропорту. Структура таблиці наступна:

Таблиця 5.2.1 – Структура таблиці employee

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| employeeID | INT |  | PK | Код працівника |
| emp\_name | VARCHAR | 50 |  | Ім’я працівника |
| emp\_lastname | VARCHAR | 50 |  | Прізвище працівника |
| emp\_surname | VARCHAR | 50 |  | По-батькові працівника |
| date\_of\_birth | DATE |  |  | Дата народження |

Таблиця 5.2.2 «characteristic» призначення для зберігання даних про характеристику літака. Вона дозволяє враховувати технічні особливості кожного літака та підтримує систему управління флотом авіакомпанії. Структура таблиці наступна:

Таблиця 5.2.2 – Структура таблиці characteristic

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| characteristicID | INT |  | PK | Код характеристики |
| brand | VARCHAR | 30 |  | Назва бренду |
| date\_last\_tech\_insp | DATE |  |  | Дата останньої технічної перевірки |
| producing\_country | VARCHAR | 30 |  | Країна випуску |

Таблиця 5.2.3 «schedule» призначення для зберігання графіку польотів. Вона допомагає управляти часовими параметрами авіаперельотів та координує відправлення та прибуття літаків. Структура таблиці наступна:

Таблиця 5.2.3 – Структура таблиці schedule

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| scheduleID | INT |  | PK | Код графіку |
| schedule\_date | DATE |  |  | Дата |
| departure\_time | TIME |  |  | Час відбуття |
| arrival\_time | TIME |  |  | Час прибуття |

Таблиця 5.2.4 «city» призначення для зберігання даних про міста, в які прокладені маршрути. Вона спрощує організацію маршрутів та роботу з розкладом рейсів. Структура таблиці наступна:

Таблиця 5.2.4 – Структура таблиці city

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| cityID | INT |  | PK | Код міста |
| city\_name | VARCHAR | 50 |  | Назва міста |
| country | VARCHAR | 50 |  | Країна |

Таблиця 5.2.5 «baggage» призначення для зберігання даних про багаж, який перевозить пасажир. Вона важлива для ефективного контролю за багажем та його відслідковуванням під час авіаперевезень. Структура таблиці наступна:

Таблиця 5.2.5 – Структура таблиці baggage

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| baggageID | INT |  | PK | Код багажу |
| weight | INT |  |  | Вага багажу |
| type | VARCHAR | 50 |  | Тип |
| registration\_date | DATE |  |  | Дата реєстрації |

Таблиця 5.2.6 «route» призначення для зберігання даних маршрути польотів. Вона допомагає системі визначати оптимальні шляхи та управляти рейсами. Структура таблиці наступна:

Таблиця 5.2.6 – Структура таблиці route

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| routeID | INT |  | PK | Код маршруту |
| scheduleID | INT |  | FK | Код графіку |
| starting\_pointID | INT |  | FK | Код початкового міста |
| end\_pointID | INT |  | FK | Код кінцевого міста |
| duration | INT |  |  | Тривалість міршруту |

Таблиця 5.2.7 «plane» призначення для зберігання даних про літаки, які належать авіакопанії та літають по маршрутах. Вона спрощує управління флотом літаків та розподіл їхніх ресурсів. Структура таблиці наступна:

Таблиця 5.2.7 – Структура таблиці plane

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| planeID | INT |  | PK | Код літака |
| carrying\_capacity | INT |  |  | Вантажопід’ємність |
| max\_number\_of\_passengers | INT |  |  | К-ть пасажирів |
| characteristicID | INT |  | FK | Код характеристики |
| routeID | INT |  | FK | Код маршруту |

Таблиця 5.2.8 «passenger» призначення для зберігання даних пасажирів. Вона необхідна для ведення обліку пасажирів та їхнього багажу. Структура таблиці наступна:

Таблиця 5.2.8 – Структура таблиці passenger

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| passengerID | INT |  | PK | Код пасажира |
| baggageID | INT |  | FK | Код багажа |
| pas\_name | VARCHAR | 50 |  | Ім’я |
| pas\_lastname | VARCHAR | 50 |  | Прізвище |
| pas\_surname | VARCHAR | 50 |  | По-батькові |
| passport\_number | INT | 9 |  | Номер паспорту |
| year\_of\_birth | DATE |  |  | Дата народження |
| citizenship | VARCHAR | 50 |  | Громадянство |

Таблиця 5.2.9 «passenger\_check» призначення для зберігання даних про перевірку пасажирів, хто був допущений на рейс, а хто ні. Вона є важливою для контролю за безпекою та дотриманням правил безпеки в аеропорту. Структура таблиці наступна:

Таблиця 5.2.9 – Структура таблиці passenger\_check

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| passenger\_checkID | INT |  | PK | Код перевірки пасажира |
| passengerID | INT |  | FK | Код пасажира |
| baggageID | INT |  | FK | Код багажа |
| employeeID | INT |  | FK | Код працівника |
| check\_result | BOOLEAN |  |  | Результат перевірки |

Таблиця 5.2.10 «booking» призначення для зберігання даних про бронювання. Вона дозволяє системі відстежувати та керувати бронюванням місць на рейсах. Структура таблиці наступна:

Таблиця 5.2.10 – Структура таблиці booking

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ім’я поля** | **Тип даних** | **Розмір** | **Ключ** | **Опис** |
| bookingID | INT |  | PK | Код бронювання |
| passengerID | INT |  | FK | Код пасажира |
| baggageID | INT |  | FK | Код багажа |
| routeID | INT |  | FK | Код маршруту |
| booking\_date | DATE |  |  | Дата бронювання |
| booking\_time | TIME |  |  | Час бронювання |

## 5.3 Методи забезпечення цілісності даних

У базі даних "Airport" я використовую різні методи забезпечення цілісності даних, щоб забезпечити правильність та надійність інформації.

* PRIMARY KEY та AUTO\_INCREMENT

В кожній таблиці визначено поле, яке виступає як основний ключ (PRIMARY KEY), наприклад, `employeeID` у таблиці "employee". Це поле гарантує унікальність кожного запису. AUTO\_INCREMENT вказує базі даних автоматично збільшувати значення цього поля при вставці нових даних, забезпечуючи унікальні ідентифікатори.

* FOREIGN KEY

Використовувала зовнішні ключі (FOREIGN KEY), встановлювала зв'язки між таблицями. Наприклад, поле `scheduleID` у таблиці "route" посилається на `scheduleID` у таблиці "schedule". Це забезпечує зв'язок між маршрутами та графіками.

* CHECK Constraint

У таблиці "route" використила перевірку (CHECK), щоб гарантувати, що значення поля `duration` завжди буде більше 0. Це дозволяє уникнути введення неправильних або від'ємних значень для тривалості маршруту. У таблиці "plane" використала перевірки для полів max\_number\_of\_passengers та carrying\_capacity. Умова CHECK (max\_number\_of\_passengers > 1) гарантує, що максимальна кількість пасажирів завжди буде більше 1. Умова CHECK (carrying\_capacity > 0) гарантує, що вантажопід’ємність буде завжди позитивною.

Ці методи допомагають уникнути некоректних або неповних даних в базі, забезпечуючи її стабільність та надійність

## 5.3 Схема бази даних, реалізована засобами СУБД

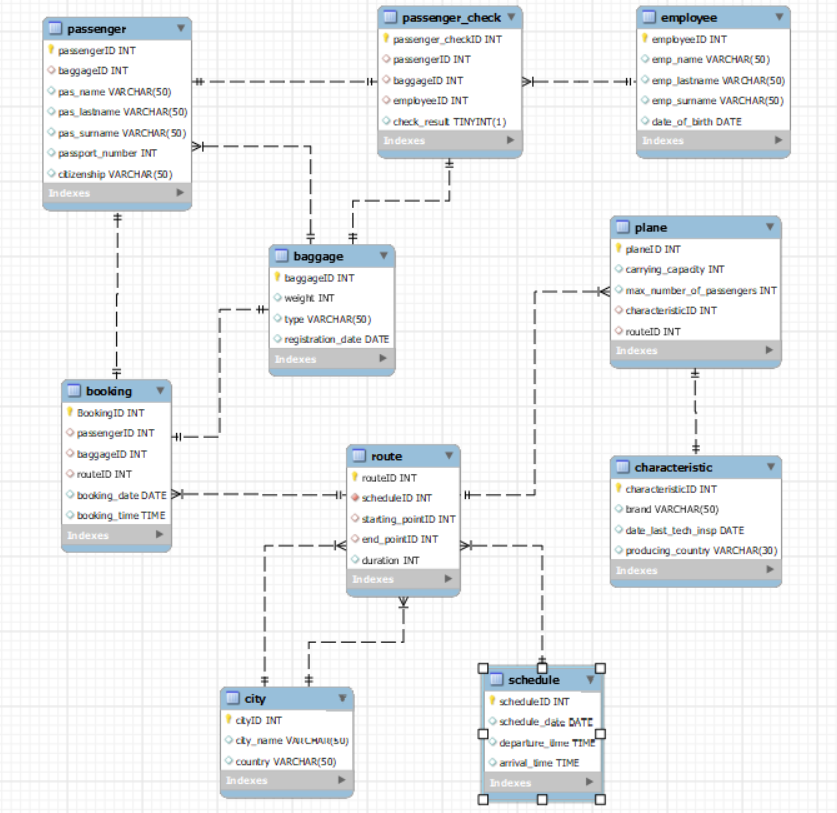
На рисунку 5.3.1 зображена схема бази даних, реалізована засобами СУБД, вона відображає структуру та зв'язки між різними сутностями в системі "Airport". Ця схема надає узагальнений вигляд на організацію інформації та взаємодії між різними частинами бази даних.

Рисунок 5.3.1 – Схема бази даних

## 5.6 Створення користувачів

Зважаючи на потреби бази даних "Airport", я створила дві ролі користувачів: AirportEmployeeRole та PassengerRole, кожна з яких має свої власні права доступу до бази даних.

CREATE ROLE AirportEmployeeRole;

CREATE ROLE PassengerRole;

Далі я визначила дві ролі: одну для співробітників аеропорту (AirportEmployeeRole) та іншу для пасажирів (PassengerRole).

GRANT USAGE ON \*.\* TO AirportEmployeeRole;

GRANT USAGE ON \*.\* TO PassengerRole;

Обидві ролі отримали базові права на використання будь-яких об'єктів бази даних.

GRANT USAGE ON Airport.\* TO AirportEmployeeRole;

GRANT USAGE ON Airport.\* TO PassengerRole;

Потім ролі отримали права на використання бази даних "Airport".

GRANT SELECT, INSERT, UPDATE, DELETE ON Airport.\* TO AirportEmployeeRole;

GRANT CREATE, ALTER, DROP ON Airport.\* TO AirportEmployeeRole;

Співробітники аеропорту отримали додаткові права на виконання операцій SELECT, INSERT, UPDATE, DELETE для таблиць бази даних "Airport", а також права на створення, зміну та видалення таблиць.

GRANT SELECT, INSERT, UPDATE ON Airport.\* TO PassengerRole;

Пасажири отримали права на виконання операцій SELECT, INSERT, UPDATE для таблиць бази даних "Airport".

CREATE USER employee\_user IDENTIFIED BY 'employee\_user';

CREATE USER passenger\_user IDENTIFIED BY 'passenger\_user';

GRANT AirportEmployeeRole TO employee\_user;

GRANT PassengerRole TO passenger\_user;

І на завершення я створила два користувачі - employee\_user та passenger\_user - та надала їм відповідні ролі. Тепер, коли ці користувачі увійдуть в систему, їхні права будуть визначатися ролями, які їм було надано. На рисунку 5.6.1 показані створені ролі та користувачі.

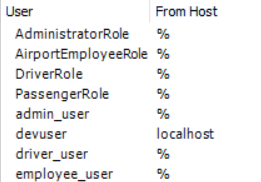


Рисунок 5.6.1 – Створені ролі та користувачі

# 6 РОБОТА З БАЗОЮ ДАНИХ

## 6.1 Тексти генераторів

Текст для створення таблиць:

CREATE TABLE employee(

employeeID INT NOT NULL PRIMARY KEY AUTO\_INCREMENT,

emp\_name VARCHAR(50) NOT NULL,

emp\_lastname VARCHAR(50) NOT NULL,

emp\_surname VARCHAR(50) NOT NULL,

date\_of\_birth DATE

);

CREATE TABLE characteristic (

characteristicID INT NOT NULL PRIMARY KEY AUTO\_INCREMENT,

brand VARCHAR(30),

date\_last\_tech\_insp DATE NOT NULL,

producing\_country VARCHAR(30)

);

CREATE TABLE schedule (

scheduleID INT NOT NULL PRIMARY KEY AUTO\_INCREMENT,

schedule\_date DATE NOT NULL,

departure\_time TIME NOT NULL,

arrival\_time TIME NOT NULL

);

CREATE TABLE city (

cityID INT NOT NULL PRIMARY KEY AUTO\_INCREMENT,

city\_name VARCHAR(50) NOT NULL,

country VARCHAR(50)

);

CREATE TABLE baggage (

baggageID INT NOT NULL PRIMARY KEY AUTO\_INCREMENT,

weight INT NOT NULL,

type VARCHAR(50),

registration\_date DATE

);

CREATE TABLE route (

routeID INT NOT NULL PRIMARY KEY AUTO\_INCREMENT,

scheduleID INT NOT NULL,

starting\_pointID INT NOT NULL,

end\_pointID INT NOT NULL,

duration INT,

FOREIGN KEY (scheduleID) REFERENCES schedule(scheduleID),

FOREIGN KEY (starting\_pointID) REFERENCES city(cityID),

FOREIGN KEY (end\_pointID) REFERENCES city(cityID),

CHECK (duration > 0)

);

CREATE TABLE plane (

planeID INT NOT NULL PRIMARY KEY AUTO\_INCREMENT,

carrying\_capacity INT NOT NULL,

max\_number\_of\_passengers INT NOT NULL,

characteristicID INT NOT NULL,

routeID INT NOT NULL,

FOREIGN KEY (characteristicID) REFERENCES characteristic(characteristicID),

FOREIGN KEY (routeID) REFERENCES route(routeID),

CHECK (max\_number\_of\_passengers > 1),

CHECK (carrying\_capacity > 0)

);

CREATE TABLE passenger (

passengerID INT NOT NULL PRIMARY KEY AUTO\_INCREMENT,

baggageID INT NOT NULL,

pas\_name VARCHAR(50) NOT NULL,

pas\_lastname VARCHAR(50) NOT NULL,

pas\_surname VARCHAR(50) NOT NULL,

passport\_number INT(9) NOT NULL,

date\_of\_birth DATE,

citizenship VARCHAR(50),

FOREIGN KEY (baggageID) REFERENCES baggage(baggageID)

);

CREATE TABLE passenger\_check (

passenger\_checkID INT NOT NULL PRIMARY KEY AUTO\_INCREMENT,

passengerID INT NOT NULL,

baggageID INT NOT NULL,

employeeID INT NOT NULL,

check\_result BOOLEAN NOT NULL,

FOREIGN KEY (passengerID) REFERENCES passenger(passengerID),

FOREIGN KEY (baggageID) REFERENCES baggage(baggageID),

FOREIGN KEY (employeeID) REFERENCES employee(employeeID)

);

CREATE TABLE booking (

bookingID INT NOT NULL PRIMARY KEY AUTO\_INCREMENT,

passengerID INT NOT NULL,

baggageID INT NOT NULL,

routeID INT NOT NULL,

booking\_date DATE,

booking\_time TIME,

FOREIGN KEY (passengerID) REFERENCES passenger(passengerID),

FOREIGN KEY (baggageID) REFERENCES baggage(baggageID),

FOREIGN KEY (routeID) REFERENCES route(routeID)

);

Текст для вставки даних:

INSERT INTO employee (emp\_name, emp\_lastname, emp\_surname, date\_of\_birth) VALUES ('Lisa', 'Ballard', 'Avery', '1989-09-07');

INSERT INTO characteristic (brand, date\_last\_tech\_insp, producing\_country) VALUES ('Briggs-Bonilla', '2023-08-15', 'Grenada');

INSERT INTO schedule (schedule\_date, departure\_time, arrival\_time) VALUES ('2023-12-26', '13:42:18', '10:00:15');

INSERT INTO city (city\_name, country) VALUES ('Kyiv', 'Ukraine');

INSERT INTO route (scheduleID, starting\_pointID, end\_pointID, duration) VALUES (3, 10, 7, 267);

INSERT INTO plane (carrying\_capacity, max\_number\_of\_passengers, characteristicID, routeID) VALUES (2367, 145, 1, 9);

INSERT INTO baggage (weight, type, registration\_date) VALUES (12, 'Бізнес', '2021-12-03');

INSERT INTO passenger (baggageID, pas\_name, pas\_lastname, pas\_surname, passport\_number, date\_of\_birth, citizenship) VALUES (1, 'Nicholas', 'Weber', 'Dorsey', 720305793, '1976-02-29', 'Pitcairn Islands');

INSERT INTO passenger\_check (passengerID, baggageID, employeeID, check\_result) VALUES (1, 1, 8, True);

INSERT INTO booking (passengerID, baggageID, routeID, booking\_date, booking\_time) VALUES (1, 1, 12, '2022-1-23', '02:15:21');

## 6.2 Тексти збережених процедур/функцій

Процедура GetRouteDetailsWithCityInfo створена для забезпечення зручного отримання деталей маршрутів разом із назвами початкового та кінцевого міста. Використовуючи INNER JOIN з таблицями route і city, процедура об'єднує дані та повертає інформацію про маршрути разом із назвами початкового та кінцевого міста.

DROP PROCEDURE IF EXISTS GetRouteDetailsWithCityInfo;

DELIMITER //

CREATE PROCEDURE GetRouteDetailsWithCityInfo()

BEGIN

SELECT r.routeID, r.duration, cs.city\_name AS starting\_city, ce.city\_name AS ending\_city

FROM route r

INNER JOIN city cs ON r.starting\_pointID = cs.cityID

INNER JOIN city ce ON r.end\_pointID = ce.cityID;

END //

DELIMITER ;

CALL GetRouteDetailsWithCityInfo();

Виклик процедури GetRouteDetailsWithCityInfo() повертає таблицю з деталями маршрутів це видно на рисунку 6.2.1 , включаючи routeID, duration, назву початкового міста (starting\_city), та назву кінцевого міста (ending\_city). Це дозволяє отримати узагальнену інформацію про маршрути разом із конкретними місцями їх початку та завершення.



Рисунок 6.2.1 – Виклик процедури GetRouteDetailsWithCityInfo

Процедура GetBaggageDetailsWithRoute створена для отримання інформації про багаж разом з даними про рейс, на якому він зареєстрований. Використовуючи INNER JOIN з таблицями baggage, booking, та route, процедура об'єднує дані та повертає інформацію про багаж та відповідний рейс.

DELIMITER //

CREATE PROCEDURE GetBaggageDetailsWithRoute()

BEGIN

SELECT b.baggageID, b.weight, b.type, b.registration\_date, r.routeID, r.duration

FROM baggage b

INNER JOIN booking bo ON b.baggageID = bo.baggageID

INNER JOIN route r ON bo.routeID = r.routeID;

END //

DELIMITER ;

call GetBaggageDetailsWithRoute();

Виклик процедури GetBaggageDetailsWithRoute() повертає таблицю з інформацією про багаж це видно на рисунку 6.2.2, включаючи baggageID, weight, type, registration\_date, routeID, та duration. Це дозволяє зручно отримати узагальнену інформацію про багаж та рейс, на якому він зареєстрований.

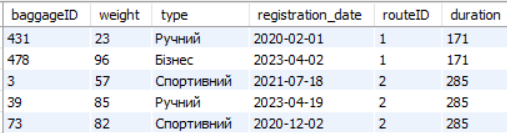


Рисунок 6.2.2 – Виклик процедури GetBaggageDetailsWithRoute

Функція GetEmployeeWithMostPassengers створена для отримання інформації про працівника, який перевірив найбільше унікальних пасажирів. Запит складається з двох частин: перша частина визначає employeeID, який має найбільше унікальних пасажирів за допомогою підзапиту та сортування; друга частина використовує отриманий employeeID для знаходження імені та кількості перевірених пасажирів.

DELIMITER //

CREATE FUNCTION GetEmployeeWithMostPassengers()

RETURNS VARCHAR(255)

DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE result VARCHAR(255);

DECLARE employee\_name VARCHAR(255);

DECLARE passenger\_count INT;

SELECT CONCAT(emp\_name, ' ', emp\_lastname) INTO employee\_name

FROM employee

WHERE employeeID = (

SELECT employeeID

FROM passenger\_check

GROUP BY employeeID

ORDER BY COUNT(DISTINCT passengerID) DESC

LIMIT 1

);

SELECT COUNT(DISTINCT passengerID) INTO passenger\_count

FROM passenger\_check

WHERE employeeID = (SELECT employeeID FROM employee WHERE CONCAT(emp\_name, ' ', emp\_lastname) = employee\_name);

SET result = CONCAT(employee\_name, ' (', passenger\_count, ' passengers)');

RETURN result;

END //

DELIMITER ;

SELECT GetEmployeeWithMostPassengers();

Виклик функції GetEmployeeWithMostPassengers() повертає рядок, який містить ім'я та прізвище працівника, кількість унікальних пасажирів, яких він перевірив, у вигляді "(Ім'я Прізвище (кількість пасажирів))" це видно на рисунку 6.2.3. Це дозволяє легко визначити працівника з найбільшою кількістю перевірених пасажирів.

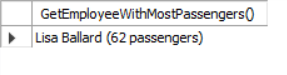


Рисунок 6.2.3 – Виклик функції GetEmployeeWithMostPassengers

Функція GetPassengerCountForRoute створена для отримання кількості пасажирів, які здійснили бронювання на певний рейс. Параметр route\_id вказує ідентифікатор рейсу, для якого необхідно отримати кількість пасажирів. Функція використовує COUNT та DISTINCT для підрахунку унікальних пасажирів, які здійснили бронювання на вказаному рейсі.

DELIMITER //

CREATE FUNCTION `GetPassengerCountForRoute`(route\_id INT)

RETURNS INT

DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE passenger\_count INT;

SELECT COUNT(DISTINCT passengerID) INTO passenger\_count

FROM booking

WHERE routeID = route\_id;

RETURN passenger\_count;

END //

DELIMITER ;

Виклик функції GetPassengerCountForRoute(route\_id) повертає ціле число - кількість пасажирів, які здійснили бронювання на вказаному рейсі це видно на рисунку 6.2.4. Це дозволяє легко отримати інформацію про популярність та завантаженість конкретного рейсу.

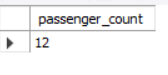


Рисунок 6.2.4 – Виклик функції GetPassengerCountForRoute

Функція GetTotalBaggageWeightForRoute створена для отримання загальної ваги багажу для певного рейсу. Параметр route\_id вказує ідентифікатор рейсу, для якого необхідно отримати загальну вагу багажу. Функція використовує INNER JOIN з таблицями baggage та booking для об'єднання інформації та SUM для підрахунку загальної ваги багажу на вказаному рейсі.

DELIMITER //

CREATE FUNCTION `GetTotalBaggageWeightForRoute`(route\_id INT)

RETURNS INT

DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE total\_weight INT;

SELECT SUM(b.weight) INTO total\_weight

FROM baggage b

JOIN booking bk ON b.baggageID = bk.baggageID

WHERE bk.routeID = route\_id;

RETURN total\_weight;

END //

DELIMITER ;

Виклик функції GetTotalBaggageWeightForRoute(route\_id) повертає ціле число - загальну вагу багажу для вказаного рейсу це видно на рисунку 6.2.5. Це дозволяє отримати інформацію про обсяг та вагу багажу, який перевозиться на конкретному рейсі.

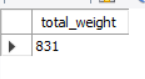


Рисунок 6.2.5 – Виклик функції GetTotalBaggageWeightForRoute

Функція GetFlightCountFromCity створена для отримання кількості рейсів, що виходять з вказаного міста. Параметр city\_name вказує назву міста, для якого необхідно отримати кількість рейсів. Функція використовує INNER JOIN з таблицями route та city для об'єднання інформації та COUNT для підрахунку кількості рейсів.

DELIMITER //

CREATE FUNCTION `GetFlightCountFromCity`(city\_name VARCHAR(50))

RETURNS INT

DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE flight\_count INT;

SELECT COUNT(\*) INTO flight\_count

FROM route r

JOIN city c ON r.starting\_pointID = c.cityID

WHERE c.city\_name = city\_name;

RETURN flight\_count;

END //

DELIMITER ;

Виклик функції GetFlightCountFromCity(city\_name) повертає ціле число - кількість рейсів, які виходять з вказаного міста це видно на рисунку 6.2.6. Це дозволяє отримати інформацію про кількість доступних рейсів з конкретного місця.



Рисунок 6.2.6 – Виклик функції GetFlightCountFromCity

Функція GetBaggageCountAfterDate створена для отримання кількості зареєстрованого багажу, реєстрація якого відбулася після вказаної дати. Параметр date вказує дату, після якої необхідно отримати кількість багажу. Функція використовує COUNT для підрахунку кількості багажу, який був зареєстрований після вказаної дати.

DELIMITER //

CREATE FUNCTION `GetBaggageCountAfterDate`(date DATE)

RETURNS INT

DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE baggage\_count INT;

SELECT COUNT(\*) INTO baggage\_count

FROM baggage

WHERE registration\_date > date;

RETURN baggage\_count;

END //

DELIMITER ;

Виклик функції GetBaggageCountAfterDate(date) повертає ціле число - кількість зареєстрованого багажу, реєстрація якого відбулася після вказаної дати це видно на рисунку 6.2.7. Це дозволяє визначити обсяг багажу, який був зареєстрований після конкретної дати.



Рисунок 6.2.7 – Виклик функції GetBaggageCountAfterDate

Функція GetPassengerCountWithBaggageType створена для отримання кількості пасажирів, які мають певний тип багажу. Параметр baggage\_type вказує тип багажу, для якого необхідно отримати кількість пасажирів. Функція використовує INNER JOIN з таблицями passenger та baggage для об'єднання інформації та COUNT для підрахунку кількості пасажирів.

DELIMITER //

CREATE FUNCTION `GetPassengerCountWithBaggageType`(baggage\_type VARCHAR(50))

RETURNS INT

DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE passenger\_count INT;

SELECT COUNT(DISTINCT p.passengerID) INTO passenger\_count

FROM passenger p

JOIN baggage b ON p.baggageID = b.baggageID

WHERE b.type = baggage\_type;

RETURN passenger\_count;

END //

DELIMITER ;

Виклик функції GetPassengerCountWithBaggageType(baggage\_type) повертає ціле число - кількість пасажирів, які мають певний тип багажу це видно на рисунку 6.2.8. Це дозволяє отримати інформацію про кількість пасажирів, які перевозять багаж певного типу.



Рисунок 6.2.8 – Виклик функції GetPassengerCountWithBaggageType

Функція GetFlightCountAtTime створена для отримання кількості рейсів, запланованих на конкретний час. Параметр time вказує час, для якого необхідно отримати кількість рейсів. Функція використовує COUNT для підрахунку кількості рейсів, які мають запланований час вказаний у параметрі.

DELIMITER //

CREATE FUNCTION `GetFlightCountAtTime`(time TIME)

RETURNS INT

DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE flight\_count INT;

SELECT COUNT(\*) INTO flight\_count

FROM schedule

WHERE departure\_time = time;

RETURN flight\_count;

END //

DELIMITER ;

Виклик функції GetFlightCountAtTime(time) повертає ціле число - кількість рейсів, запланованих на певний час це видно на рисунку 6.2.9. Це дозволяє отримати інформацію про кількість рейсів, які заплановані на конкретний момент часу.

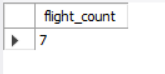


Рисунок 6.2.9 – Виклик функції GetFlightCountAtTime

Процедура UpdatePassengerBaggage призначена для оновлення інформації про багаж пасажира. Вона приймає на вхід ідентифікатор пасажира (passenger\_id), нову вагу багажу (new\_weight), новий тип багажу (new\_type) та нову дату реєстрації (new\_registration\_date). За допомогою операції UPDATE здійснюється оновлення відповідної інформації в таблиці baggage для багажу, який пов'язаний з вказаним пасажиром.

DELIMITER //

CREATE PROCEDURE `UpdatePassengerBaggage`(IN passenger\_id INT, IN new\_weight INT, IN new\_type VARCHAR(50), IN new\_registration\_date DATE)

BEGIN

UPDATE baggage b

JOIN passenger p ON b.baggageID = p.baggageID

SET b.weight = new\_weight,

b.type = new\_type,

b.registration\_date = new\_registration\_date

WHERE p.passengerID = passenger\_id;

END //

DELIMITER ;

Виклик процедури UpdatePassengerBaggage із передачею відповідних параметрів дозволяє оновити інформацію про багаж пасажира це видно на рисунку 6.2.10.



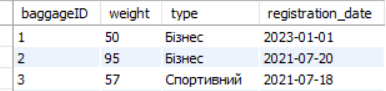


Рисунок 6.2.10 – Виклик процедури UpdatePassengerBaggage

## 6.3 Тексти тригерів

Тригер check\_date\_schedule створений для забезпечення дотримання бізнес-правила, яке обмежує дату бронювання квитка так, щоб вона не перевищувала дату вильоту літака. При спробі вставки нового запису в таблицю booking тригер автоматично перевіряє дату розкладу для відповідного маршруту та порівнює її з датою бронювання. Якщо дата бронювання пізніша за дату розкладу, тригер генерує помилку і не дозволяє вставку.

DELIMITER //

CREATE TRIGGER check\_date\_schedule

BEFORE INSERT ON booking

FOR EACH ROW

BEGIN

DECLARE schedule\_date DATE;

SELECT schedule.schedule\_date INTO schedule\_date

FROM schedule

JOIN route ON schedule.scheduleID = route.scheduleID

WHERE route.routeID = NEW.routeID;

IF NEW.booking\_date > schedule\_date THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Дата бронювання не може бути пізнішою за дату розкладу';

END IF;

END;//

DELIMITER ;

При спробі вставки нового запису в таблицю booking, тригер check\_date\_schedule перевіряє відповідність дати бронювання та дати розкладу. Якщо дата бронювання пізніша за дату розкладу, тригер генерує помилку це видно на рисунку 6.3.1 і вставка запису відхиляється, захищаючи тим самим базу даних від некоректних даних відносно бізнес-правила обмеження дат бронювання.

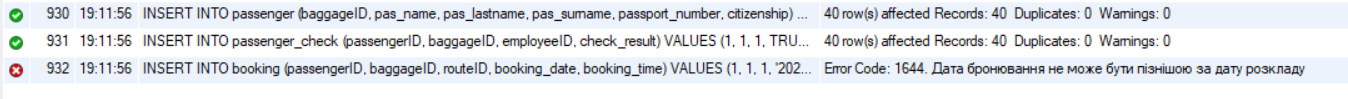


Рисунок 6.3.1 – Тригер check\_date\_schedule

Тригер check\_booking\_date створений для забезпечення дотримання бізнес-правила, яке обмежує дату бронювання квитка так, щоб вона не була ранішою за дату реєстрації багажу. При спробі вставки нового запису в таблицю booking тригер автоматично перевіряє дату реєстрації багажу для відповідного багажу та порівнює її з датою бронювання. Якщо дата бронювання раніша за дату реєстрації багажу, тригер генерує помилку і не дозволяє вставку.

DELIMITER //

CREATE TRIGGER check\_booking\_date

BEFORE INSERT ON booking

FOR EACH ROW

BEGIN

DECLARE baggage\_date DATE;

SELECT registration\_date INTO baggage\_date FROM baggage WHERE baggageID = NEW.baggageID;

IF NEW.booking\_date < baggage\_date THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Дата бронювання не може бути ранішою за дату реєстрації багажу';

END IF;

END;//

DELIMITER ;

При спробі вставки нового запису в таблицю booking, тригер check\_booking\_date перевіряє відповідність дати бронювання та дати реєстрації багажу. Якщо дата бронювання раніша за дату реєстрації багажу, тригер генерує помилку це видно на рисунку 6.3.2 і вставка запису відхиляється, захищаючи тим самим базу даних від некоректних даних відносно бізнес-правила обмеження дат бронювання.

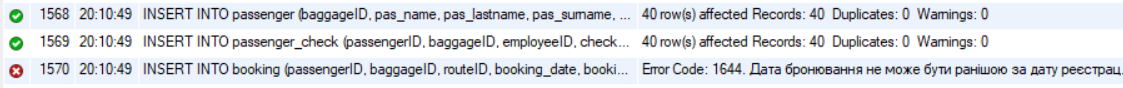


Рисунок 6.3.2 – Тригер check\_booking\_date

Тригер check\_seats створений для забезпечення дотримання бізнес-правила, яке обмежує кількість броньованих білетів для конкретного маршруту так, щоб не перевищувала максимальну кількість місць на літаку. При спробі вставки нового запису в таблицю booking тригер автоматично отримує максимальну кількість місць для відповідного маршруту з таблиці plane, а також рахує кількість вже заброньованих білетів для цього маршруту. Якщо кількість заброньованих білетів перевищує максимальну кількість місць, тригер генерує помилку і не дозволяє вставку.

DELIMITER //

CREATE TRIGGER check\_seats

BEFORE INSERT ON booking

FOR EACH ROW

BEGIN

DECLARE max\_seats INT;

DECLARE booked\_seats INT;

SELECT max\_number\_of\_passengers INTO max\_seats

FROM plane

JOIN route ON plane.routeID = route.routeID

WHERE route.routeID = NEW.routeID

LIMIT 1;

SELECT COUNT(\*) INTO booked\_seats

FROM booking

WHERE routeID = NEW.routeID;

IF booked\_seats >= max\_seats THEN

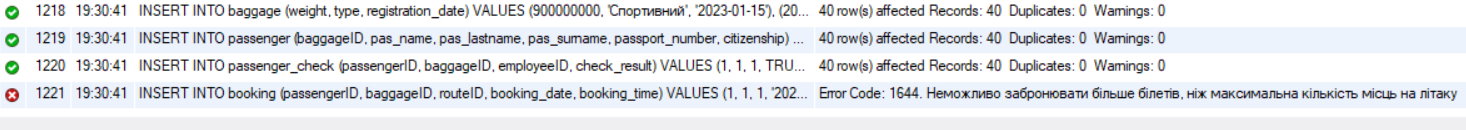
SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Неможливо забронювати більше білетів, ніж максимальна кількість місць на літаку';

END IF;

END;//

DELIMITER ;

При спробі вставки нового запису в таблицю booking, тригер check\_seats перевіряє, чи кількість броньованих білетів для конкретного маршруту не перевищує максимальну кількість місць на літаку. Якщо перевищено, тригер генерує помилку це видно на рисунку 6.3.3, і вставка запису відхиляється, захищаючи тим самим базу даних від некоректних даних відносно бізнес-правила обмеження кількості білетів на рейсі.

Рисунок 6.3.3 – Тригер check\_seats

Тригер check\_baggage\_weight створений для забезпечення дотримання бізнес-правила, яке обмежує вагу багажу для конкретного бронювання так, щоб не перевищувала максимальну вантажопідйомність літака. При спробі вставки нового запису в таблицю booking тригер автоматично отримує максимальну вантажопідйомність літака з таблиці plane, а також вагу багажу для вказаного багажу. Якщо вага багажу перевищує вантажопідйомність літака, тригер генерує помилку і не дозволяє вставку.

DELIMITER //

CREATE TRIGGER check\_baggage\_weight

BEFORE INSERT ON booking

FOR EACH ROW

BEGIN

DECLARE plane\_capacity INT;

DECLARE baggage\_weight INT;

SELECT carrying\_capacity INTO plane\_capacity

FROM plane

WHERE planeID = NEW.routeID;

SELECT weight INTO baggage\_weight

FROM baggage

WHERE baggageID = NEW.baggageID;

IF baggage\_weight > plane\_capacity THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Вага багажу перевищує вантажопідйомність літака';

END IF;

END;//

DELIMITER ;

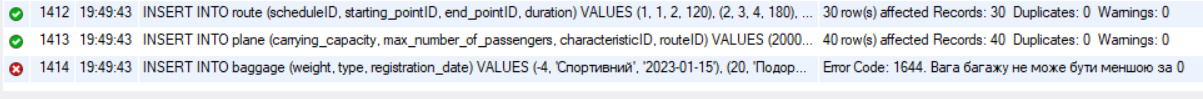
При спробі вставки нового запису в таблицю booking, тригер check\_baggage\_weight перевіряє, чи вага багажу не перевищує максимальну вантажопідйомність літака. Якщо вага багажу перевищує вантажопідйомність, тригер генерує помилку це видно на рисунку 6.3.4, і вставка запису відхиляється, захищаючи тим самим базу даних від некоректних даних відносно бізнес-правила контролю за вантажем.

Рисунок 6.3.4 – Тригер check\_baggage\_weight

Тригер check\_weight створений для валідації введеної ваги багажу перед її вставкою в таблицю baggage. При спробі вставки нового запису тригер автоматично перевіряє, чи вага багажу є додатною. Якщо вага багажу менше 0, тригер генерує помилку, і вставка запису відхиляється.

DELIMITER //

CREATE TRIGGER check\_weight

BEFORE INSERT ON baggage

FOR EACH ROW

BEGIN

IF NEW.weight < 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Вага багажу не може бути меншою за 0';

END IF;

END;//

DELIMITER ;

При спробі вставки нового запису в таблицю baggage, тригер check\_weight перевіряє, чи вага багажу не є від'ємною. Якщо вага багажу менше 0, тригер генерує помилку це видно на рисунку 6.3.5, і вставка запису відхиляється, захищаючи тим самим базу даних від некоректних даних відносно бізнес-правила валідації ваги багажу.

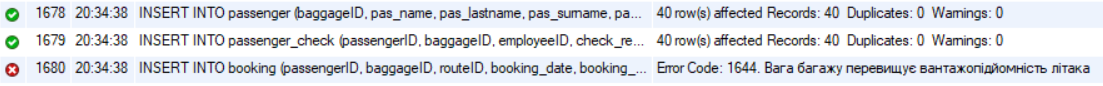


Рисунок 6.3.5 – Тригер check\_weight

Тригер check\_passenger\_age\_before\_insert реалізований для забезпечення дотримання бізнес-правила, яке обмежує додавання нового пасажира до бази даних, якщо йому менше 18 років. При спробі вставки нового запису в таблицю passenger тригер автоматично порівнює рік народження пасажира з поточним роком. Якщо різниця менше 18 років, тригер генерує помилку і вставка запису відхиляється.

DELIMITER //

CREATE TRIGGER check\_passenger\_age

BEFORE INSERT ON passenger

FOR EACH ROW

BEGIN

IF (YEAR(CURDATE()) - YEAR(NEW.date\_of\_birth)) < 18 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Пасажир має бути старше 18';

END IF;

END;//

DELIMITER ;

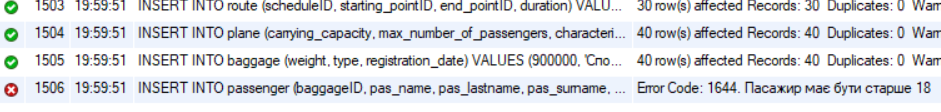
При спробі вставки нового запису в таблицю passenger, тригер check\_passenger\_age\_before\_insert перевіряє, чи пасажир є повнолітнім. Якщо пасажир молодший за 18 років, тригер генерує помилку це видно на рисунку 6.3.6, і вставка запису відхиляється, захищаючи тим самим базу даних від некоректних даних відносно бізнес-правила перевірки повноліття пасажирів.

Рисунок 6.3.6 – Тригер check\_passenger\_age

## 6.4 Тексти представлень

Це представлення створене для забезпечення зручного доступу до інформації про пасажирів, їх бронювання та відповідні маршрути та графіки.

CREATE VIEW view\_a AS

SELECT p.passengerID, p.pas\_name, p.pas\_lastname, p.pas\_surname, b.routeID, b.booking\_date, r.scheduleID, r.duration

FROM passenger p

JOIN booking b ON p.passengerID = b.passengerID

JOIN route r ON b.routeID = r.routeID;

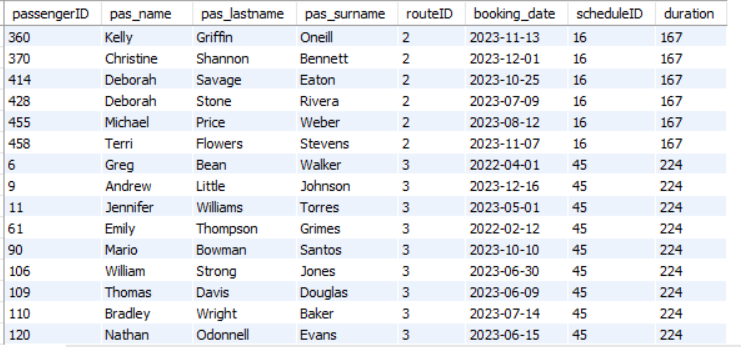
Результатом є представлення view\_a, яке тепер може використовуватися для виконання SELECT-запитів, спрощуючи отримання інформації про пасажирів та їх бронювання на маршрутах це видно на рисунку 6.4.1.

Рисунок 6.4.1 – Виклик представлення view\_a

Це представлення створене для полегшення отримання інформації про пасажирів, їх бронювання, та пов'язані графіки вильоту та прибуття.

CREATE VIEW view\_b AS

SELECT a.passengerID, a.pas\_name, a.pas\_lastname, a.pas\_surname, a.routeID, a.booking\_date, a.scheduleID, a.duration, s.departure\_time, s.arrival\_time

FROM view\_a a

JOIN schedule s ON a.scheduleID = s.scheduleID;

Результатом є нове представлення view\_b, яке може використовуватися для виконання SELECT-запитів, що включають інформацію про пасажирів, бронювання, та відповідні графіки вильоту та прибуття це видно на рисунку 6.4.2.

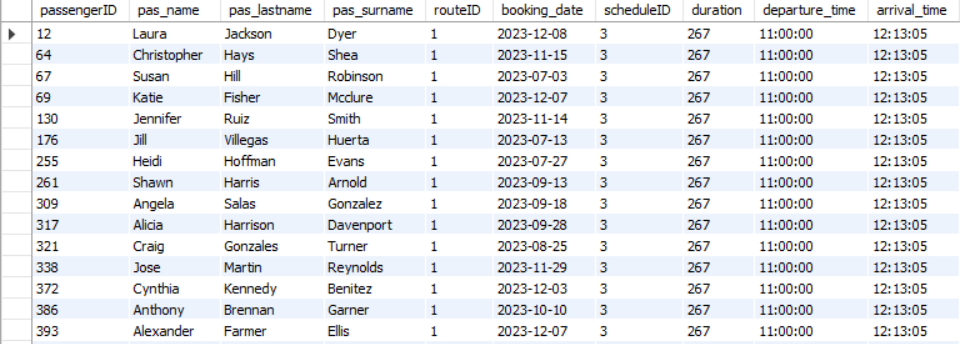


Рисунок 6.4.2 – Виклик представлення view\_b

Це представлення view\_b розширює інформацію з view\_a, додаючи дані про вагу багажу для кожного пасажира.

ALTER VIEW view\_b AS

SELECT a.passengerID, a.pas\_name, a.pas\_lastname, a.pas\_surname, a.routeID, a.booking\_date, a.scheduleID, a.duration, s.departure\_time, s.arrival\_time,

b.weight AS baggage\_weight

FROM view\_a a

JOIN schedule s ON a.scheduleID = s.scheduleID

JOIN baggage b ON a.passengerID = b.passengerID;

Результатом є змінення представлення view\_b. Це представлення може використовуватися для виконання SELECT-запитів, що включають інформацію про пасажирів, бронювання, графіки вильоту та прибуття, а також дані про їх багаж це видно на рисунку 6.4.3.

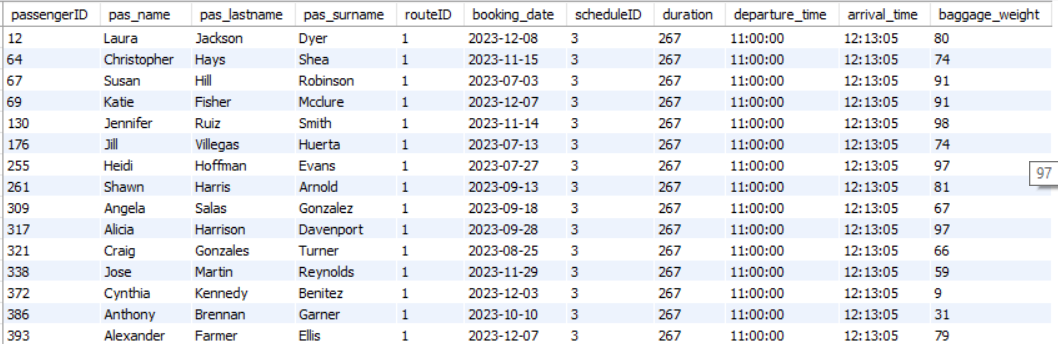


Рисунок 6.4.3 – Виклик представлення view\_b після змінення

## 6.5 SQL-запити

Запит використовує операцію JOIN для об'єднання таблиць passenger і passenger\_check за допомогою спільного поля passengerID. Результат цього об'єднання міститиме ім'я, прізвище, ім'я по батькові та результат паспортного контролю для кожного пасажира.

SELECT p.pas\_name, p.pas\_lastname, p.pas\_surname, pc.check\_result

FROM passenger p

JOIN passenger\_check pc ON p.passengerID = pc.passengerID;

Виклик запиту GetPassengersWithPassportCheck надає список пасажирів, які пройшли паспортний контроль, разом із відповідними результатами це видно на рисунку 6.5.1.

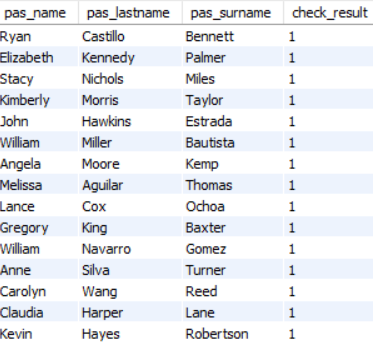


Рисунок 6.5.1 – Виклик запиту

Запит виводить інформацію про пасажирів та маршрути їх подорожей, включаючи імена пасажирів та міста початку та закінчення подорожі. Запит використовує операцію JOIN для об'єднання декількох таблиць, таких як booking, passenger, route, city, щоб отримати повну інформацію про подорожі пасажирів. Результат включає імена пасажирів (об'єднані у поле passenger\_name) та міста початку та закінчення подорожі (об'єднані у поля starting\_point та end\_point відповідно).

SELECT

CONCAT(p.pas\_name, ' ', p.pas\_lastname) AS passenger\_name,

CONCAT(c1.city\_name, ', ', c1.country) AS starting\_point,

CONCAT(c2.city\_name, ', ', c2.country) AS end\_point

FROM

booking b

JOIN passenger p ON b.passengerID = p.passengerID

JOIN route r ON b.routeID = r.routeID

JOIN city c1 ON r.starting\_pointID = c1.cityID

JOIN city c2 ON r.end\_pointID = c2.cityID;

Результат виконання запиту включає інформацію про пасажирів та деталі їх маршрутів. Кожен рядок містить ім'я пасажира (passenger\_name), місто початку подорожі (starting\_point) та місто закінчення подорожі (end\_point) це видно на рисунку 6.5.2.

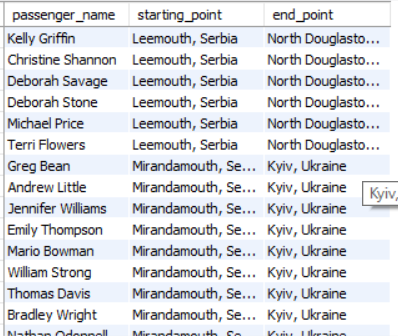


Рисунок 6.5.2 – Виклик запиту

Запит призначений для отримання інформації про пасажирів та деталі їх бронювань на рейси, які виконані з певної дати. Запит використовує об'єднання таблиць booking та passenger за їхніми відповідними ідентифікаторами пасажирів (passengerID). Умова WHERE фільтрує результати, обираючи тільки ті записи, де дата бронювання (booking\_date) є пізнішою або рівною '2023-01-01'. Результат сортується за зменшенням дати та часу бронювання.

SELECT p.pas\_name, p.pas\_lastname, b.booking\_date, b.booking\_time

FROM booking b

JOIN passenger p ON b.passengerID = p.passengerID

WHERE b.booking\_date >= '2023-01-01'

ORDER BY b.booking\_date DESC, b.booking\_time DESC;

Результатом буде створена таблиця з колонками pas\_name, pas\_lastname, booking\_date, booking\_time, яка містить інформацію про пасажирів та їхні бронювання, здійснені після 1 січня 2023 року це видно на рисунку 6.5.3. Результат впорядкований за зменшенням дати бронювання та часу бронювання.



Рисунок 6.5.3 – Виклик запиту

Запит вибирає дані про пасажирів, їхні бронювання, та інформацію про маршрути для подорожей, що стартують у місті "Париж" і мають дату бронювання до поточної дати.

SELECT p.passengerID, p.pas\_name, p.pas\_lastname, b.bookingID, b.booking\_date, r.routeID

FROM passenger p

JOIN booking b ON p.passengerID = b.passengerID

JOIN route r ON b.routeID = r.routeID

WHERE

r.starting\_pointID = (

SELECT cityID

FROM city

WHERE city\_name = 'Париж')

AND b.booking\_date < (SELECT CURDATE());

Результатом буде створена таблиця з колонками passengerID, pas\_name, pas\_lastname, bookingID, booking\_date, routeID, яка містить інформацію про пасажирів, їхні бронювання та маршрути, де місце відправлення - " Mirandamouth", а дата бронювання менше поточної дати це видно на рисунку 6.5.4.

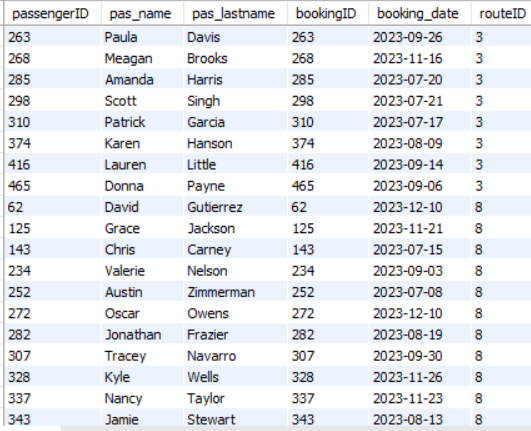


Рисунок 6.5.4 – Виклик запиту

Запит вибирає ім'я, прізвище та по батькові працівників, які мають спільні бронювання з пасажирами протягом періоду від 01 січня 2022 року до 30 квітня 2023 року.

SELECT e.emp\_name, e.emp\_lastname, e.emp\_surname

FROM employee e

WHERE EXISTS (

SELECT 1

FROM passenger p

JOIN booking b ON p.passengerID = b.passengerID

WHERE b.booking\_date >= '2022-01-01'

AND b.booking\_date <= '2023-04-30'

AND e.employeeID = b.routeID

);

Результатом буде створена таблиця з колонками emp\_name, emp\_lastname, emp\_surname, яка містить інформацію про працівників, які мали спільні бронювання з пасажирами протягом періоду від 01 січня 2022 року до 30 квітня 2023 року це видно на рисунку 6.5.5.

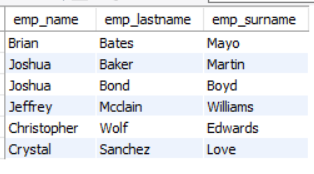


Рисунок 6.5.5 – Виклик запиту

Запит вибирає імена пасажирів, у яких є багаж із вагою більше 20 кг.

SELECT pas\_name

FROM passenger

WHERE baggageID IN (

SELECT baggageID

FROM baggage

WHERE weight > 20);

Результатом буде створена таблиця з колонкою pas\_name, яка містить імена пасажирів, у яких є багаж із вагою більше 20 кг це видно на рисунку 6.5.6.



Рисунок 6.5.6 – Виклик запиту

Запит вибирає назви міст, які виступають в якості початкових пунктів для якого-небудь маршруту.

SELECT city\_name FROM city WHERE cityID IN (

SELECT starting\_pointID

FROM route);

Результатом буде створена таблиця з колонкою city\_name, яка містить назви міст, використовуваних як початкові пункти для наявних маршрутів це видно на рисунку 6.5.7.



Рисунок 6.5.7 – Виклик запиту

Запит об'єднує назви міст, які виступають як початкові і кінцеві пункти для яких-небудь маршрутів. Запит використовує операцію об'єднання (UNION), щоб отримати унікальні назви міст з обох підзапитів. Перший підзапит визначає міста, які є початковими пунктами маршрутів, і другий - міста, які є кінцевими пунктами маршрутів.

SELECT city\_name FROM city WHERE cityID IN (SELECT starting\_pointID FROM route)

UNION

SELECT city\_name FROM city WHERE cityID IN (SELECT end\_pointID FROM route);

Результатом буде створена таблиця з колонкою city\_name, яка містить унікальні назви міст, які використовуються як початкові або кінцеві пункти для наявних маршрутів маршрутів це видно на рисунку 6.5.8.



Рисунок 6.5.8 – Виклик запиту

Запит призначений для отримання всіх записів з таблиці "route", де тривалість маршруту перевищує середню тривалість усіх маршрутів.

SELECT \*

FROM route

WHERE duration > (SELECT AVG(duration) FROM route);

Результатом буде створена таблиця з усіма стовпцями з таблиці "route", де тривалість маршруту перевищує середню тривалість усіх маршрутів це видно на рисунку 6.5.9.

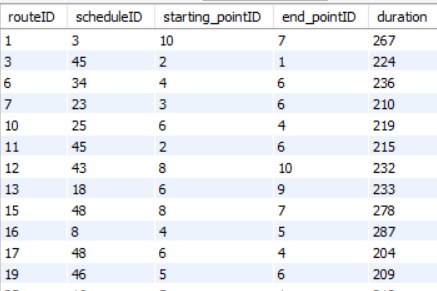


Рисунок 6.5.9 – Виклик запиту

Запит призначений для отримання інформації про пасажирів та їхній багаж, який є типу 'Подорожний' і які мають бронювання на маршрутах, які починаються у місті Київ.

SELECT p.\*, b.\*

FROM passenger p

JOIN baggage b ON p.baggageID = b.baggageID

WHERE b.type = 'Подорожний'

AND p.passengerID IN (

SELECT passengerID

FROM booking

WHERE routeID IN (

SELECT routeID

FROM route

WHERE starting\_pointID = (

SELECT cityID

FROM city

WHERE city\_name = 'Київ')));

Результатом буде створена таблиця, яка містить інформацію про пасажирів та їхній багаж, які відповідають умовам фільтрації це видно на рисунку 6.5.10.



Рисунок 6.5.10 – Виклик запиту

Запит призначений для отримання кількості унікальних пасажирів для кожного маршруту на основі інформації про бронювання.

SELECT routeID, COUNT(DISTINCT passengerID) AS passenger\_count

FROM booking

GROUP BY routeID;

Результатом буде створена таблиця, яка містить ідентифікатор маршруту та відповідну кількість унікальних пасажирів для кожного маршруту це видно на рисунку 6.5.11.

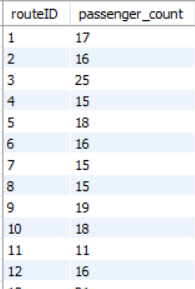


Рисунок 6.5.11 – Виклик запиту

Запит призначений для отримання максимальної кількості пасажирів для кожного маршруту на основі інформації про літаки, які обслуговують ці маршрути.

SELECT r.routeID, MAX(p.max\_number\_of\_passengers) AS max\_passengers

FROM route r

JOIN plane p ON r.routeID = p.routeID

GROUP BY r.routeID;

Результатом буде створена таблиця, яка містить ідентифікатор маршруту та відповідну максимальну кількість пасажирів для кожного маршруту це видно на рисунку 6.5.12.

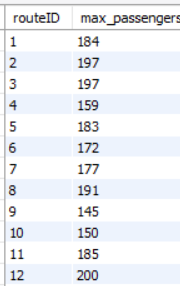


Рисунок 6.5.12 – Виклик запиту

Запит призначений для отримання списку міст, які є кінцевими точками хоча б одного рейсу.

SELECT DISTINCT c.city\_name

FROM city c

WHERE c.cityID IN (SELECT DISTINCT end\_pointID FROM route);

Результатом буде створена таблиця, яка містить список міст, які є кінцевими точками принаймні одного рейсу це видно на рисунку 6.5.13.



Рисунок 6.5.13 – Виклик запиту

Запит призначений для отримання інформації про всіх пасажирів, які здійснили бронювання на рейси, тривалість яких більше 3 годин.

SELECT p.\*

FROM passenger p

JOIN booking b ON p.passengerID = b.passengerID

JOIN route r ON b.routeID = r.routeID

WHERE r.duration > 180;

Результатом буде інформація про всіх пасажирів, які здійснили бронювання на рейси з тривалістю більше 3 годин це видно на рисунку 6.5.14.

Рисунок 6.5.14 – Виклик запиту

Запит призначений для отримання інформації про кількість перевірених багажів, що припадає на кожного працівника.

SELECT e.\*, COUNT(pc.baggageID) AS checked\_baggage\_count

FROM employee e

LEFT JOIN passenger\_check pc ON e.employeeID = pc.employeeID

GROUP BY e.employeeID;

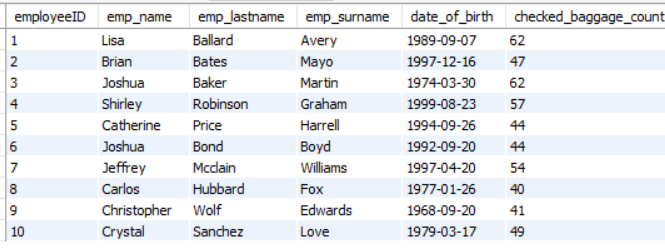
Результатом буде таблиця, яка містить інформацію про кожного працівника разом з кількістю перевірених багажів, яка припадає на кожного працівника це видно на рисунку 6.5.15.

Рисунок 6.5.15 – Виклик запиту

Запит призначений для отримання інформації про літаки, які призначені для маршрутів, що починаються в країні " Ghana ".

SELECT pl.\*

FROM plane pl

JOIN route r ON pl.routeID = r.routeID

JOIN city c ON r.starting\_pointID = c.cityID

WHERE c.country = Ghana;

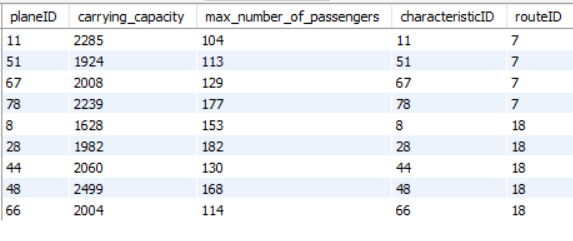
Результатом буде таблиця, яка містить інформацію про літаки, які призначені для маршрутів, що починаються в країні " Ghana " це видно на рисунку 6.5.16.

Рисунок 6.5.16 – Виклик запиту

Цей запит призначений для отримання інформації про кількість рейсів, які мають певне місто як початкову точку.

SELECT c.city\_name, COUNT(r.routeID) AS flight\_count

FROM city c

LEFT JOIN route r ON c.cityID = r.starting\_pointID

GROUP BY c.cityID;

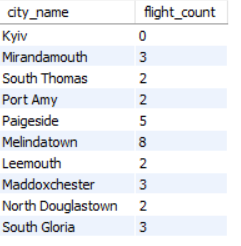
Результатом буде таблиця, що містить ім'я міста та кількість рейсів, які мають його початковою точкою це видно на рисунку 6.5.17.

Рисунок 6.5.17 – Виклик запиту

Цей запит призначений для отримання кількості міжнародних рейсів на основі даних про громадянство пасажирів та маршрути. Запит використовує умову EXISTS для перевірки існування хоча б одного пасажира на рейсі, громадянство якого відрізняється від країни початкової точки маршруту. Це дозволяє визначити, що рейс є міжнародним. Підрахунок кількості міжнародних рейсів виконується за допомогою функції COUNT(DISTINCT r.routeID).

SELECT COUNT(DISTINCT r.routeID) AS international\_flights\_count

FROM route r

WHERE EXISTS (

SELECT 1

FROM booking b

JOIN passenger p ON b.passengerID = p.passengerID

JOIN city c ON p.citizenship = c.country

WHERE r.routeID = b.routeID AND r.starting\_pointID != c.cityID

);

Результатом буде таблиця, що містить ціле число, що представляє кількість міжнародних рейсів на основі вищезазначених умов це видно на рисунку 6.5.18.

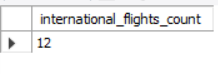


Рисунок 6.5.18 – Виклик запиту

Цей запит призначений для вибору всіх працівників, які не мають жодного запису в таблиці passenger\_check, тобто тих, хто не здійснював пасажирський контроль.

SELECT e.\*

FROM employee e

LEFT JOIN passenger\_check pc ON e.employeeID = pc.employeeID

WHERE pc.passenger\_checkID IS NULL;

Результатом буде таблиця, що містить всі записи з таблиці employee, які не мають відповідних записів в таблиці passenger\_check це видно на рисунку 6.5.19.

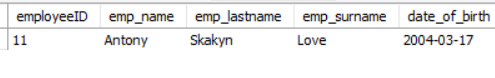


Рисунок 6.5.19 – Виклик запиту

Цей запит призначений для отримання максимальної ваги багажу серед усіх записів у таблиці baggage, які мають відповідний запис у таблиці booking.

SELECT MAX(b.weight) AS max\_baggage\_weight

FROM baggage b

JOIN booking bk ON b.baggageID = bk.baggageID;

Результатом буде один рядок результату, який містить максимальну вагу багажу серед пасажирів, які здійснили бронювання це видно на рисунку 6.5.20.

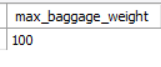


Рисунок 6.5.20 – Виклик запиту

Цей запит призначений для отримання ідентифікатора маршруту (routeID), для якого існує найбільше кількість бронювань.

SELECT r.routeID, COUNT(bk.bookingID) AS booking\_count

FROM route r

LEFT JOIN booking bk ON r.routeID = bk.routeID

GROUP BY r.routeID

ORDER BY booking\_count DESC

LIMIT 1;

Результатом буде рядок, який містить ідентифікатор (routeID) та кількість бронювань (booking\_count) для маршруту з найбільшою популярністю це видно на рисунку 6.5.21.

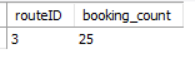


Рисунок 6.5.21 – Виклик запиту

Цей запит призначений для отримання ідентифікатора маршруту (routeID) та тривалості (duration), де тривалість менше або дорівнює середній тривалості всіх маршрутів.

SELECT r.routeID, r.duration

FROM route r

WHERE r.duration <= (SELECT AVG(duration) FROM route);

Результатом буде рядок результату, який містить ідентифікатор маршруту та його тривалість для маршрутів з тривалістю менше або дорівнює середній тривалості це видно на рисунку 6.5.22.

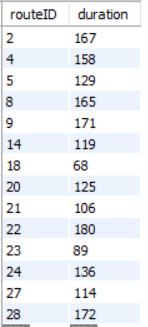


Рисунок 6.5.22 – Виклик запиту

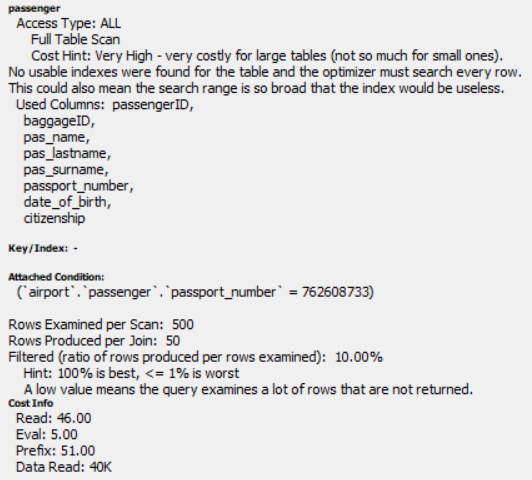
## 6.6 Результати оптимізації

Було створено некластерний індекс idx\_passenger\_passport\_number на колонці passport\_number в таблиці passenger. Це оптимізувало пошук за номером паспорта в таблиці passenger, оскільки індекси значно прискорюють пошук даних в базі даних.

CREATE INDEX idx\_passenger\_passport\_number

ON passenger (passport\_number);

Індекси в базі даних працюють подібно до індексів у книзі. Вони дають базі даних “посилання” на рядки таблиці, які містять певне значення, що дозволяє базі даних швидко знаходити ці рядки без необхідності переглядати кожен рядок таблиці. Нижче на рисунках 6.6.1 та 6.6.2 наведені схеми та детальна інформація виконання запиту знаходження пасажира за номером його паспорту до та після оптимізації



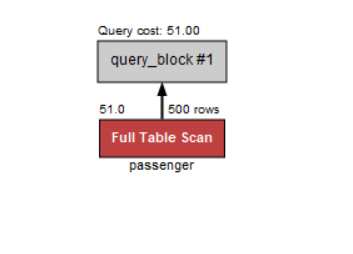
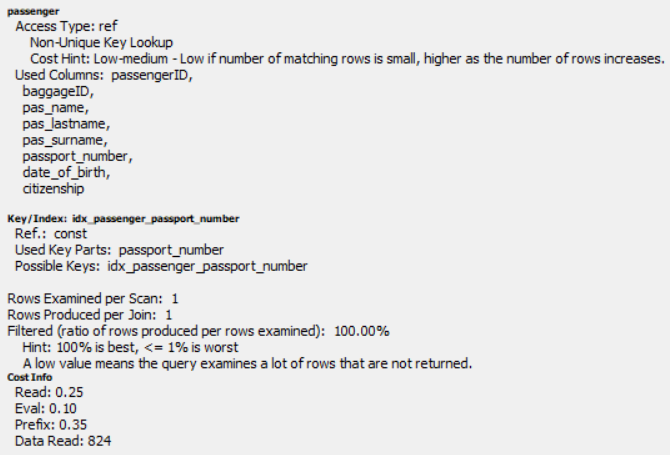


Рисунок 6.6.1 – Інформація виконання запиту без індекса



# 

Рисунок 6.6.2 – Інформація виконання запиту з індексом

Для прикладу використовувався простий SQL запит

Select \* FROM passenger

WHERE passport\_number = '762608733';

Аналізуючи, бачимо, що використання індексів може значно покращити ефективність запитів до бази даних.

У першому сценарії, коли індекси не використовуються, запит до бази даних потребує перевірки 500 рядків для отримання потрібної інформації.

У другому сценарії, коли використовується індекс, кількість рядків, які потрібно перевірити, зменшується до всього 1. Це означає, що запит виконується набагато швидше, а ефективність збільшується аж в 500 разів.

Отже, можна зробити висновок, що використання індексів є критично важливим для оптимізації роботи з базами даних. Вони дозволяють значно прискорити пошук даних, що робить роботу з базою даних більш ефективною та продуктивною. Звичайно, створення і підтримка індексів також вимагає ресурсів, тому важливо використовувати їх розумно, враховуючи специфіку конкретної бази даних та запитів, які до неї надходять.

# ВИСНОВКИ

У ході виконання курсової роботи було проведено аналіз предметного середовища для розробки бази даних для аеропорту. Описано об'єкт дослідження, вхідні та вихідні дані, а також основні бізнес-процеси використання даних в аеропорту.

Було проведено аналіз існуючих програмних продуктів в сфері авіаційної індустрії для визначення та узагальнення кращих практик і можливостей, які можуть бути використані для розробки бази даних.

Сформульовано завдання та визначено вимоги до розробки бази даних, що враховують особливості роботи аеропорту та його персоналу.

Було побудовано ER-модель, в якій визначено бізнес-правила, сутності, набори атрибутів та структурні зв'язки між об'єктами.

Проведено реалізацію бази даних, обрано систему управління базами даних, описано характеристики таблиць, методи забезпечення цілісності даних та побудовано схему бази даних.

Розглянуто процес роботи з базою даних, включаючи тексти генераторів, збережених процедур/функцій, тригерів, представлень та SQL-запитів.

Завершальним етапом було проведено оптимізацію бази даних за допомогою створення індексів та оптимізованих SQL-запитів, що призвело до покращення продуктивності системи.

Отже, курсова робота дозволила створити та оптимізувати базу даних для аеропорту, що відповідає вимогам бізнесу та сприятиме ефективній роботі аеропортового персоналу.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. <https://avia.tickets.ua/search/result?session_id=eca94400195cd2255a424ab27360a8d3>
2. <https://tripmydream.ua/flights>
3. <https://www.booking.com/flights/index.uk.html?aid=2151979&label=Ukrainian-UK-124331077386-BTUdIFpHlS%2APhkn%2AovTDwAS533957472387%3Apl%3Ata%3Ap1%3Ap2%3Aac%3Aap%3Aneg%3Afi%3Atidsa-1633144487656%3Alp9061017%3Ali%3Adec%3Adm&sid=5bc97abc83a78eb0c618533ec8f9bc82>
4. <https://itvdn.com/ua/blog/article/m-sql>
5. <https://dou.ua/lenta/articles/sql-query-mysql-server/>
6. <https://dou.ua/forums/topic/45982/>
7. <https://iteducenter.medium.com/%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE-%D0%BF%D1%80%D0%BE-%D1%81%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%B5-%D1%96%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%B8-%D0%B2-mysql-5f3d61f36f71>