**ЗМІСТ**

[ВСТУП 5](#_Toc72610785)

[1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 8](#_Toc72610786)

[2 РОЗРОБЛЕННЯ БАЗИ ДАНИХ ЗАСОБАМИ СКБД MYSQL 9](#_Toc72610787)

[2.1 Створення таблиць 9](#_Toc72610788)

[2.2 Створення тригерів 23](#_Toc72610789)

[2.3 Створення запитів 29](#_Toc72610790)

[2.4 Створення представлень 35](#_Toc72610791)

[2.5 Створення процедур та функцій 39](#_Toc72610792)

[2.6 Додавання користувачів та надання їм прав 45](#_Toc72610793)

[3 РОЗРОБЛЕННЯ БАЗИ ДАНИХ ЗАСОБАМИ СКБД POSTGRESQL 47](#_Toc72610794)

[3.1 Створення таблиць 47](#_Toc72610795)

[3.2 Створення тригерів 61](#_Toc72610796)

[3.3 Створення запитів 67](#_Toc72610797)

[3.4 Створення представлень 75](#_Toc72610798)

[3.5 Створення процедур та функцій 80](#_Toc72610799)

[3.6 Додавання користувачів та надання їм прав 87](#_Toc72610800)

[ВИСНОВКИ 88](#_Toc72610801)

**ВСТУП**

База даних — це певний набір даних, які пов'язані між собою спільною  ознакою або властивістю, та впорядковані, наприклад, за алфавітом. Об'єднання великої кількості даних в єдину базу дає змогу для формування  безлічі варіації групування інформації — особисті дані клієнта, історія замовлень,  каталог товарів та будь-що інше. Головною перевагою БД є швидкість внесення та  використання потрібної інформації. Також в базі даних існує певний взаємозв'язок  інформації: зміна в одному рядку може спричинити зміни в інших рядках — це  допомагає працювати з інформацією простіше і швидше

Щоб створити запит до бази даних часто використовують Structured Query  Language. SQL дає змогу додавати, редагувати та видаляти інформацію, що  міститься у таблицях. Під час програмування сайтів використовують різні системи  управління БД. До основних СКБД, відносять:

– об'єктно-реляційна система управління базами даних Oracle Database;

– вільна система управління базами даних PostgreSQL;

– система керування базами даних Microsoft SQL Сервер;

– вільна система управління базами даних MySQL.

Такі системи управління відрізняються централізованою обробкою запитів,  забезпечують надійність, доступність та безпеку БД.

СКБД — це програмні засоби, які виступають посередником між БД та її  користувачами. Завдяки сукупності мовних та програмних засобів, СКБД сприяють  створенню, ведення та спільного використання БД різними користувачами. Сучасна  програма СКБД складаються з ядра, процесору мови БД, підсистеми підтримки часу  виконання та сервісних програм, які надають додаткові можливості для  обслуговування інформаційних систем.

MySQL — це система управління реляційними базами даних, яка має більше  шести тисяч обєктів. Програма працює як сервер забезпечення  “багатокористувацького доступу” до обєктів баз даних.

Багото веб–додатків використовують MySQL в якості компонентів  програмного забезпечення. Популярність використання веб–додатків тісно повязана  з популярністю РНР і можливістю його роботи разом з MySQL.

MySQL Workbench — інструмент для візуального проектування баз даних,  що інтегрує проектування, моделювання, створення й експлуатацію БД в єдине  безкоштовне оточення для системи баз даних MySQL. Є наступником DBDesigner 4  з FabForce. Перша версія MySQL Workbench була випущена у вересні 2005 року.

Можливості програми:

− дозволяє наочно представити модель бази даних в графічному вигляді;

− наочний і функціональний механізм установки зв'язків між таблицями,  в тому числі «багато до багатьох» із створенням таблиці зв'язків;

− Reverse Engineering — відновлення структури таблиць з вже існуючої на  сервері БД (зв'язки відновлюються в InnoDB, при  використанні MyISAM зв'язки необхідно встановлювати вручну). PostgreSQL - це об'єктно-реляційна система управління базами даних  (ORDBMS) заснована на POSTGRES версії 4.2, яка була розроблена в Науковому  Комп'ютерному Департаменті Берклі Каліфорнійського Університету. СКБД надає  підтримку SQL92/SQL99 та інші сучасні можливості.

PostgreSQL належить до категорії СКБД, відомих як об'єктно-реляційні  (object-relation). Зауважимо, що тут є відмінність від тих об'єктно-орієнтованих  (object-oriented) СКБД, які в основному підтримують традиційні мови реляційних  СКБД. Однак, PostgreSQL має деякі об'єктно-орієнтовані можливості, це важливо в  світі реляційних СКБД.

Метою розроблення курсового проєкту є створення бази даних для  предметної області «Вступна кампанія у ВНЗ», закріплення знань щодо сучасних  технологій проєктування і розроблення баз даних, засвоєння основних понять  реляційних баз даних і здобуття практичних навичок створення програмних систем  з використанням системи керування базами даних MySQL, PostgreSQL.

**1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ**

Для створення бази даних було опрацьовано предметну область "Вступна  кампанія у ВНЗ". Розроблення логічної моделі методом «сутність-зв'язок» (ER методом) виконано методом низхідного проектування. Спроектовано ER-діаграму  для предметної області «Вступна кампанія у ВНЗ». Виділено наступні сутності: абітурієнти, заяви, заяви на спеціальність, вищі навчальні заклади, спеціальності,  деталі вступу, конкурс, конкурсні предмети, результати, ЗНО, сертифікати ЗНО,  пільги, наявність пільг. Для усіх таблиця даної моделі даних було створено  відповідні первинні ключі та прокладено усі потрібні зв’язки між таблицями. База даних Tickets\_booking складається з вісімнадцяти таблиць:

**2 РОЗРОБЛЕННЯ БАЗИ ДАНИХ ЗАСОБАМИ СКБД MYSQL**

MySQL — це система управління реляційними базами даних, яка має більше  шести тисяч об’єктів. Програма працює як сервер забезпечення  “багатокористувацького доступу’ до об’єктів баз даних [1].

**2.1 Створення таблиць**

Для створення таблиць програмним способом використовують оператор CREATE TABLE. Для цього потрібно вказати наступні дані:

* ім'я таблиці, яке вказується після ключового слова CREATE TABLE;
* імена та визначення стовпців таблиці, що відділені комами;
* в деяких СУБД також вимагається, щоби було вказано місце розташування таблиці.

Щоб добавити новий рядок в таблицю, потрібно вказати назву таблиці, перелічити назви стовпців та вказати значення для кожного стовпця за допомогою конструкції **INSERT INTO** назва\_таблиці (поле1, поле2 ...)**VALUES** (значення1, значення2 ...)*.*[4]

У відповідності до розробленої схеми даних було створено SQL таблиці бази  даних.

Створено таблицю "Type\_transportations". Структура таблиці наведена на рисунку 2.1.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Type\_transportations (

transportation\_ID INT AUTO\_INCREMENT NOT NULL,

type\_name VARCHAR(45) NOT NULL,

PRIMARY KEY (transportation\_ID));

Код заповнення таблиці:

BEGIN ;  
INSERT INTO Type\_transportations(type\_name)  
VALUES ('Повітряне'),('Колійне'),('Водне'),('Наземне');  
COMMIT ;

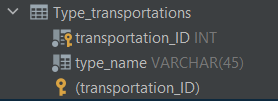


Рисунок 2.1 – Структура таблиці "Type\_transportations"

Створено таблицю " Firms". Структура таблиці наведена на рисунку 2.2.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Firms (  
 firm\_ID INT NOT NULL AUTO\_INCREMENT,  
 idTransportation INT NOT NULL,  
 name VARCHAR(45) NOT NULL,  
 email VARCHAR(45) NOT NULL,  
 tel VARCHAR(20) NOT NULL,  
 regin\_activity VARCHAR(20) NULL,  
 PRIMARY KEY (firm\_ID),  
 FOREIGN KEY (idTransportation) REFERENCES Type\_transportations (transportation\_ID)  
 ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;  
INSERT INTO Firms(idTransportation, name, email, tel, regin\_activity)  
VALUES (1,'МАУ','uia@flyuia.com','+38 (044) 581-50-50','Україна'),  
 (1,'KLM','KLM.Ukraine@klm.nl','+31 (0) 20 649 91 23','Нідерланди'),  
 (1,'Emirates Airlines','pr@emirates.com','600 555 555','ОАЕ'),  
 ...  
 (4,'Ecolines','help@ecolines.ua','+38 044 594 90 10',''),  
 (4,'EAST WEST EUROLINES','support@ewe.ua','+380988154444',''),  
 (4,'TransTempo','transtempo@ukr.net','+38 (067) 467-44-77','');  
COMMIT ;

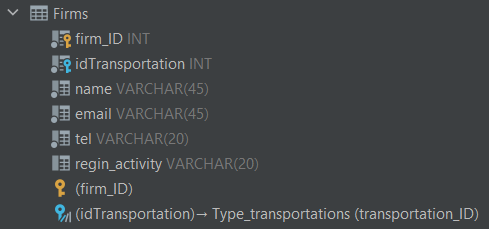


Рисунок 2.2 – Структура таблиці "Firms"

Створено таблицю " Flights". Структура таблиці наведена на рисунку 2.3.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Flights (

flight\_ID INT NOT NULL AUTO\_INCREMENT,

idFirm INT NOT NULL,

idTransport INT NOT NULL,

f\_from VARCHAR(45) NULL,

f\_to VARCHAR(45) NULL,

f\_distance FLOAT NULL,

starttime DATETIME NOT NULL,

endtime DATETIME NULL,

status ENUM('Скасований', 'Відбувся', 'Очікується') DEFAULT 'Очікується',

PRIMARY KEY (flight\_ID),

FOREIGN KEY (idFirm) REFERENCES Firms(firm\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (idTransport) REFERENCES Transport(transport\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;  
INSERT INTO Flights(idFirm, f\_from, f\_to, f\_distance, idTransport, starttime)  
VALUES (8, 'Київ', 'Дніпро', 488, 51, '2021-10-24 21:22:46'),  
 (10, 'Херсон', 'Миколаїв', 80, 57, '2021-05-08 14:01:34'),  
 (11, 'Київ', 'Луганськ', 922, 6, '2021-08-09 20:18:56'),  
 ...  
 (10, 'Кропивницький', 'Миколаїв', 182, 53, '2021-03-10 02:42:48'),  
 (3, 'Хмельницький', 'Берн', 1444, 4, '2021-03-19 11:52:21'),  
 (1, 'Київ', 'Осло', 1630, 3, '2021-01-02 18:36:57');  
COMMIT ;

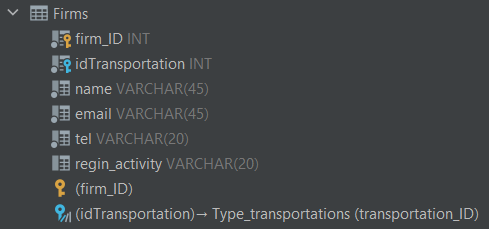


Рисунок 2.3 – Структура таблиці "Flights"

Створено таблицю "Customers". Структура таблиці наведена на рисунку 2.4.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Customers (  
 customer\_ID INT NOT NULL AUTO\_INCREMENT,  
 c\_name VARCHAR(45) NOT NULL,  
 c\_birthday DATE NOT NULL,

PRIMARY KEY (customer\_ID));

Код заповнення таблиці:

BEGIN ;  
INSERT INTO Customers(c\_name, c\_birthday)  
VALUES ('Тарасович Ратимир Вадимович', '1952-08-10'),  
 ('Ніколенко Ромашка Зорянівна', '1963-05-17'),  
 ('Кириленко Муховіст Полянович', '1987-03-19'),  
 ...  
 ( 'Мухопад Царук Адріанович', '1982-04-20'),  
 ( 'Бурбан Вернислав Артемович', '1993-03-05'),  
 ( 'Штинь Атрей Русланович', '1956-11-19'),  
 ( 'Деркач Лютобор Романович', '1966-07-25');  
COMMIT ;

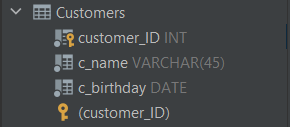


Рисунок 2.4 – Структура таблиці "Customers"

Створено таблицю "Type\_tickets". Структура таблиці наведена на рисунку 2.5.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Type\_tickets (

typeticket\_ID INT NOT NULL AUTO\_INCREMENT,

type\_afterpay ENUM('Купе дорослий', 'Купе дитячий', 'Купе студентський', 'Плацкарт дорослий',

'Плацкарт дитячий', 'Плацкарт студентський', 'Бізнес клас', 'Економ клас', 'Перший клас', 'Автобусний',

'Автобусний студентський', 'Морський', 'Морський VIP', 'Морський економ') NOT NULL,

percent\_afterpay INT NOT NULL,

PRIMARY KEY (typeticket\_ID));

Код заповнення таблиці:

BEGIN ;

INSERT INTO Type\_tickets (type\_afterpay, percent\_afterpay)

VALUES ( 'Купе дорослий', 50),

( 'Купе студентський', 30),

( 'Купе дитячий', 10),

( 'Плацкарт дорослий', 0),

( 'Плацкарт дитячий', -30),

( 'Плацкарт студентський', -50),

( 'Бізнес клас', 150),

( 'Економ клас', -10),

( 'Перший клас', 50),

( 'Автобусний', 0),

( 'Автобусний студентський', -20),

( 'Морський', 0),

( 'Морський VIP', 75),

( 'Морський економ',-15);

COMMIT ;

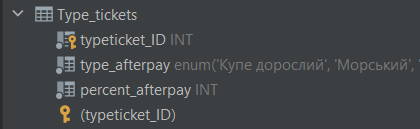


Рисунок 2.5 – Структура таблиці "Type\_tickets"

Створено таблицю "Cities". Структура таблиці наведена на рисунку 2.6.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Cities (

city\_ID INT NOT NULL AUTO\_INCREMENT,

city\_name VARCHAR(45) NOT NULL,

country VARCHAR(45) NOT NULL,

PRIMARY KEY (city\_ID));

Код заповнення таблиці:

BEGIN ;

INSERT INTO Cities (country, city\_name)

VALUES ('Молдова', 'Кишинів'),

('Греція', 'Афіни'),

('Хорватія', 'Загреб'),

...

('Україна','Хмельницький'),

('Україна','Черкаси'),

('Україна','Чернівці'),

('Україна','Чернігів');

COMMIT ;

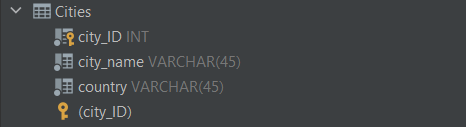


Рисунок 2.6 – Структура таблиці "Cities"

Створено таблицю "Flights\_has\_Cities". Структура таблиці наведена на рисунку 2.7.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Flights\_has\_Cities (

idFlight INT NOT NULL,

idCity1 INT NOT NULL,

idCity2 INT NULL,

primaryID INT NOT NULL AUTO\_INCREMENT,

PRIMARY KEY (primaryID),

FOREIGN KEY (idFlight) REFERENCES Flights (flight\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (idCity1) REFERENCES Cities (city\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (idCity2) REFERENCES Cities (city\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Flights\_has\_Cities(idFlight, idCity1, idCity2)

(SELECT flight\_ID, city\_ID, (SELECT city\_ID FROM Cities WHERE f\_to = city\_name) FROM Flights, Cities

WHERE f\_from = city\_name);

COMMIT ;

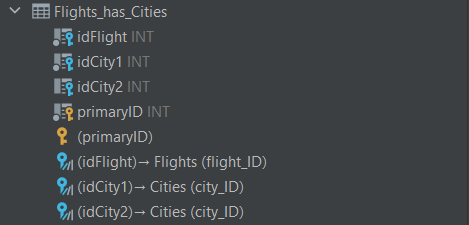


Рисунок 2.7 – Структура таблиці "Flights\_has\_Cities"

Створено таблицю "Airplane\_tickets". Структура таблиці наведена на рисунку 2.8.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Airplane\_tickets (

a\_ticket\_ID INT NOT NULL AUTO\_INCREMENT,

at\_price DECIMAL NULL,

idType\_tickets INT NOT NULL,

Flights\_has\_Cities\_primaryID INT NOT NULL,

PRIMARY KEY (a\_ticket\_ID),

FOREIGN KEY (idType\_tickets) REFERENCES Type\_tickets (typeticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Flights\_has\_Cities\_primaryID) REFERENCES Flights\_has\_Cities (primaryID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Airplane\_tickets(idType\_tickets, Flights\_has\_Cities\_primaryID)

VALUES (7, 10),

(9, 50),

(9, 13),

(8, 25),

...

(8, 37),

(9, 7),

(8, 6);

COMMIT ;

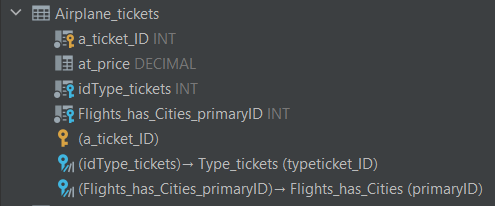


Рисунок 2.8 – Структура таблиці "Airplane\_tickets"

Створено таблицю "Routes". Структура таблиці наведена на рисунку 2.9.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Routes (

route\_ID INT NOT NULL AUTO\_INCREMENT,

Flights\_has\_Cities\_primaryID INT NOT NULL,

idCity INT NOT NULL,

PRIMARY KEY (route\_ID),

FOREIGN KEY (Flights\_has\_Cities\_primaryID) REFERENCES Flights\_has\_Cities (primaryID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (idCity) REFERENCES Cities (city\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Routes(Flights\_has\_Cities\_primaryID, idCity)

VALUES ( 8, 60),

( 8, 63),

( 8, 50),

( 29, 43),

...

( 47, 64),

( 47, 55),

( 9, 50),

( 9, 60);

COMMIT ;

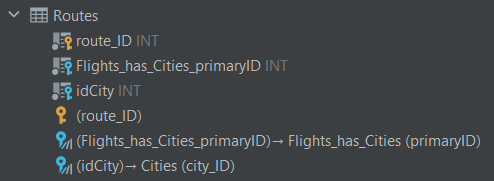


Рисунок 2.9 – Структура таблиці "Routes"

Створено таблицю "Train\_tickets". Структура таблиці наведена на рисунку 2.10.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Train\_tickets (

t\_ticket\_ID INT NOT NULL AUTO\_INCREMENT,

tt\_price DECIMAL NULL,

idType\_tickets INT NOT NULL,

idRoute INT NOT NULL,

PRIMARY KEY (t\_ticket\_ID),

FOREIGN KEY (idType\_tickets) REFERENCES Type\_tickets (typeticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (idRoute) REFERENCES Routes (route\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Train\_tickets( tt\_price, idType\_tickets, idRoute)

VALUES ( 394, 1, 31),

( 554, 1, 32),

( 221, 2, 33),

...

( 185, 1, 53),

( 394, 4, 54),

( 472, 5, 55);

COMMIT ;

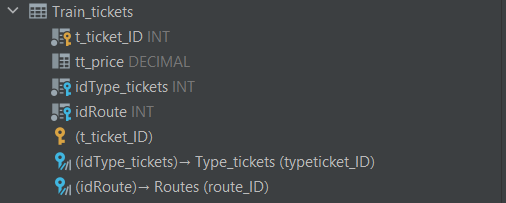


Рисунок 2.10 – Структура таблиці "Train\_tickets"

Створено таблицю "Bus\_tickets". Структура таблиці наведена на рисунку 2.11.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Bus\_tickets (

b\_ticket\_ID INT NOT NULL AUTO\_INCREMENT,

bt\_price DECIMAL NULL,

idType\_tickets INT NOT NULL,

idRoute INT NOT NULL,

PRIMARY KEY (b\_ticket\_ID),

FOREIGN KEY (idRoute) REFERENCES Routes (route\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (idType\_tickets) REFERENCES Type\_tickets (typeticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Bus\_tickets(bt\_price, idType\_tickets, idRoute)

VALUES ( 318, 10, 1),

( 212, 10, 2),

( 448, 10, 3),

...

( 638, 11, 8),

( 953, 10, 9),

( 3754, 10, 10);

COMMIT ;

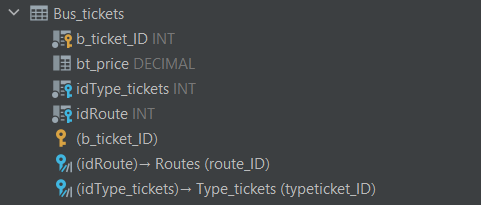


Рисунок 2.11 – Структура таблиці "Bus\_tickets"

Створено таблицю "Ship\_tickets". Структура таблиці наведена на рисунку 2.12.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Ship\_tickets (

s\_ticket\_ID INT NOT NULL AUTO\_INCREMENT,

st\_price DECIMAL NULL,

idType\_tickets INT NOT NULL,

Flights\_has\_Cities\_primaryID INT NOT NULL,

PRIMARY KEY (s\_ticket\_ID),

FOREIGN KEY (idType\_tickets) REFERENCES Type\_tickets (typeticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Flights\_has\_Cities\_primaryID) REFERENCES Flights\_has\_Cities (primaryID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Ship\_tickets(idType\_tickets, Flights\_has\_Cities\_primaryID)

VALUES ( 12, 32),

( 13, 32),

( 14, 32),

...

( 13, 32),

( 13, 32),

( 13, 32);

COMMIT ;

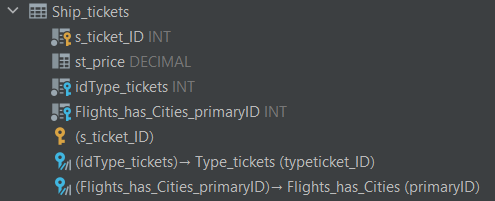


Рисунок 2.12 – Структура таблиці "Ship\_tickets"

Створено таблицю "Train\_tickets\_has\_Customers". Структура таблиці наведена на рисунку 2.13.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Train\_tickets\_has\_Customers (

Train\_tickets\_t\_ticket\_ID INT NOT NULL,

Customers\_customer\_ID INT NOT NULL,

t\_booking\_time DATETIME NOT NULL,

t\_numwagon INT NOT NULL,

t\_seat INT NOT NULL,

comments VARCHAR(45) NULL,

PRIMARY KEY (Train\_tickets\_t\_ticket\_ID, Customers\_customer\_ID),

FOREIGN KEY (Train\_tickets\_t\_ticket\_ID) REFERENCES Train\_tickets (t\_ticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Customers\_customer\_ID) REFERENCES Customers (customer\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Train\_tickets\_has\_Customers(train\_tickets\_t\_ticket\_id, customers\_customer\_id, t\_booking\_time, t\_numwagon, t\_seat)

VALUES ( 1, 2, '2020-03-15 09:59:44', 1, 21),

( 2, 4, '2020-09-01 05:27:15', 6, 13),

( 3, 6, '2020-11-11 16:03:38', 4, 17),

...

( 23, 46, '2020-05-06 04:59:54', 7, 42),

( 24, 48, '2020-12-04 22:03:56', 1, 45),

( 25, 50, '2020-05-01 14:58:58', 2, 21);

COMMIT ;

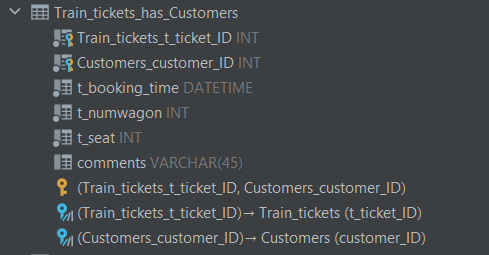


Рисунок 2.13 – Структура таблиці "Train\_tickets\_has\_Customers"

Створено таблицю "Airplane\_tickets\_has\_Customers". Структура таблиці наведена на рисунку 2.14.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Airplane\_tickets\_has\_Customers (

Airplane\_tickets\_a\_ticket\_ID INT NOT NULL,

Customers\_customer\_ID INT NOT NULL,

a\_booking\_time DATETIME NOT NULL,

at\_class VARCHAR(15) NULL,

a\_seat INT NOT NULL,

comments VARCHAR(45) NULL,

PRIMARY KEY (Airplane\_tickets\_a\_ticket\_ID, Customers\_customer\_ID),

FOREIGN KEY (Airplane\_tickets\_a\_ticket\_ID) REFERENCES Airplane\_tickets (a\_ticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Customers\_customer\_ID) REFERENCES Customers (customer\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

COMMIT ;

INSERT INTO Airplane\_tickets\_has\_Customers

(Airplane\_tickets\_a\_ticket\_ID, Customers\_customer\_ID, a\_booking\_time, a\_seat, comments)

VALUES ( 1, 1, '2020-10-09 20:21:36', 1, NULL),

( 2, 2, '2020-01-14 09:55:01', 2, NULL),

( 3, 3, '2020-06-23 14:27:41', 3, NULL),

...

( 23, 23, '2020-04-16 16:54:07', 23, NULL),

( 24, 24, '2020-06-25 04:22:31', 24, NULL),

( 25, 25, '2020-10-28 08:31:29', 25, NULL);

COMMIT ;

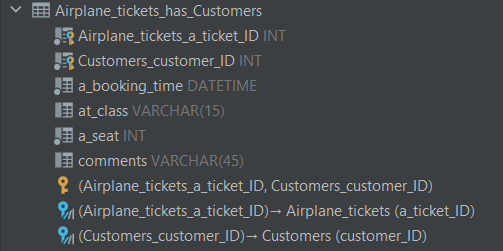


Рисунок 2.14 – Структура таблиці "Airplane\_tickets\_has\_Customers"

Створено таблицю "Bus\_tickets\_has\_Customers". Структура таблиці наведена на рисунку 2.15.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Bus\_tickets\_has\_Customers (

Bus\_tickets\_b\_ticket\_ID INT NOT NULL,

Customers\_customer\_ID INT NOT NULL,

b\_booking\_time DATETIME NOT NULL,

b\_seat INT NOT NULL,

comments VARCHAR(45) NULL,

PRIMARY KEY (Bus\_tickets\_b\_ticket\_ID, Customers\_customer\_ID),

FOREIGN KEY (Bus\_tickets\_b\_ticket\_ID) REFERENCES Bus\_tickets (b\_ticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Customers\_customer\_ID) REFERENCES Customers (customer\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Bus\_tickets\_has\_Customers(bus\_tickets\_b\_ticket\_id, customers\_customer\_id, b\_booking\_time, b\_seat)

VALUES ( 1, 26, '2020-05-08 02:11:23', 1),

( 2, 27, '2020-10-06 21:24:28', 4),

( 3, 28, '2020-05-14 23:59:25', 7),

...

( 33, 29, '2020-08-12 08:28:02', 19),

( 34, 26, '2020-10-04 00:22:49', 1),

( 35, 29, '2020-01-25 05:00:16', 4);

COMMIT ;

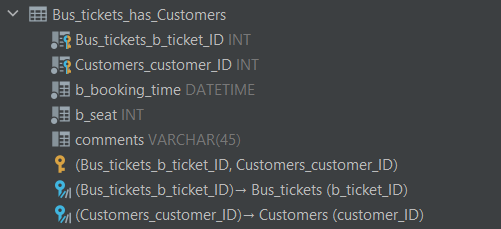


Рисунок 2.15 – Структура таблиці "Bus\_tickets\_has\_Customers"

Створено таблицю "Ship\_tickets\_has\_Customers". Структура таблиці наведена на рисунку 2.16.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Ship\_tickets\_has\_Customers (

Ship\_tickets\_s\_ticket\_ID INT NOT NULL,

Customers\_customer\_ID INT NOT NULL,

s\_booking\_time DATETIME NOT NULL,

s\_class VARCHAR(15) NULL,

s\_seat INT NOT NULL,

comments VARCHAR(45) NULL,

PRIMARY KEY (Ship\_tickets\_s\_ticket\_ID, Customers\_customer\_ID),

FOREIGN KEY (Ship\_tickets\_s\_ticket\_ID) REFERENCES Ship\_tickets (s\_ticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Customers\_customer\_ID) REFERENCES Customers (customer\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Ship\_tickets\_has\_Customers(ship\_tickets\_s\_ticket\_id, customers\_customer\_id, s\_booking\_time, s\_seat, comments)

VALUES ( 1, 26, '2020-03-24 05:43:44', 23, NULL),

( 2, 27, '2020-02-25 13:56:49', 97, NULL),

( 3, 28, '2020-09-11 10:54:59', 45, NULL),

( 4, 29, '2020-11-19 02:10:09', 12, NULL),

( 5, 30, '2020-02-14 01:36:50', 88, NULL);

COMMIT ;

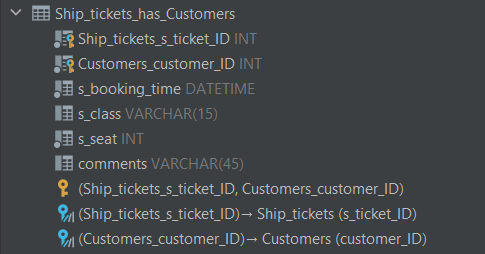


Рисунок 2.16 – Структура таблиці "Ship\_tickets\_has\_Customers"

Створено таблицю "Transport\_has\_Type\_tickets". Структура таблиці наведена на рисунку 2.17.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Transport\_has\_Type\_tickets (

Transport\_idTransport INT NOT NULL,

Type\_ticketa\_idType\_tickets INT NOT NULL,

count\_seats\_type INT NOT NULL,

PRIMARY KEY (Transport\_idTransport, Type\_ticketa\_idType\_tickets),

FOREIGN KEY (Transport\_idTransport) REFERENCES Transport (transport\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Type\_ticketa\_idType\_tickets) REFERENCES Type\_tickets (typeticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Transport\_has\_Type\_tickets(transport\_idtransport, type\_ticketa\_idtype\_tickets, count\_seats\_type)

VALUES (1, 7, 50),

(1, 8, 75),

...

(66,13, 50),

(66,14, 50);

COMMIT ;

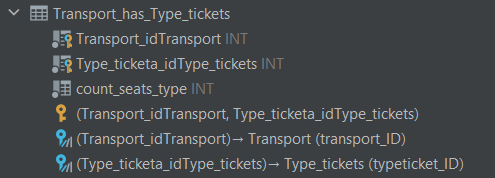


Рисунок 2.17 – Структура таблиці "Transport\_has\_Type\_tickets"

Створено таблицю "Transport". Структура таблиці наведена на рисунку 2.18.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Transport (

transport\_ID INT NOT NULL AUTO\_INCREMENT,

name VARCHAR(45) NOT NULL,

number VARCHAR(10) NOT NULL,

countplace INT NOT NULL,

PRIMARY KEY (transport\_ID));

Код заповнення таблиці:

BEGIN ;

INSERT INTO Transport(name, number, countplace)

VALUES ( 'Boeing', '777', 250),

( 'Airbus', 'A340', 170),

( 'Airbus', 'A330', 125),

...

( 'Celebrity Xpedition', '', 5530),

( 'Forse le Carib', '', 570),

( 'Porco Fugo', '', 240);

COMMIT ;

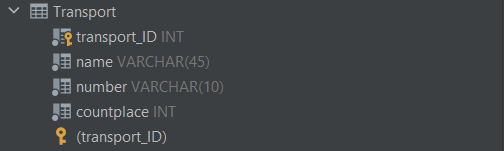


Рисунок 2.18 – Структура таблиці "Transport"

У ER-діаграмі бази даних "Резервування квитків" відображено всі таблиці, їх поля та зв’язки між таблицями і подано в додатку А. Вигляд таблиць подано у додатку Б.

**2.2 Створення тригерів**

Тригер - це збережена процедура, яка не викликається безпосередньо, а виконується при настанні певної події (вставка, видалення, оновлення рядка).[5]

У результаті аналізу завдання на курсове проєктування було створено наступні тригери.

Тригер, який записує видалені дані з таблиці “Transpot” в таблицю “Del\_transport” (рисунок 2.19):

delimiter //

CREATE TRIGGER saveDelTransport BEFORE DELETE ON Transport

FOR EACH ROW

BEGIN

INSERT INTO Del\_transport(transport\_ID, name, number, countplace, time)

VALUES(OLD.transport\_ID, OLD.name, OLD.number, OLD.countplace, NOW());

END;//

delimiter ;

Код створення допоміжної таблиці:

CREATE TABLE Del\_transport LIKE Transport;

ALTER TABLE Del\_transport ADD COLUMN time DATETIME NULL;

Виконано тестове видалення для перевірки результату:

DELETE FROM Transport WHERE transport\_ID < 5;

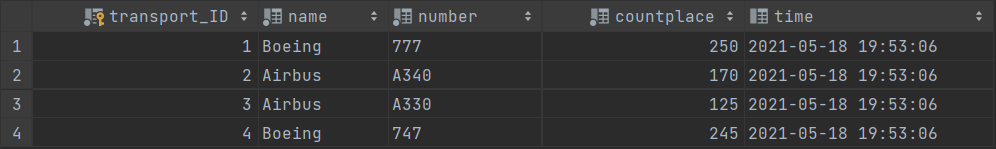


Рисунок 2.19 – Результат виконання тригера

Тригер, який викликає збережену процедуру, що обчислює приблизний час прибуття рейсу (рисунок 2.20):

delimiter //

CREATE TRIGGER setEndTimeFlight BEFORE INSERT ON Flights

FOR EACH ROW

BEGIN

CASE

WHEN NEW.idFirm <= 7

THEN CALL calc\_end\_time(NEW.endtime, NEW.f\_distance,

NEW.starttime, 800);

WHEN NEW.idFirm > 7 AND NEW.idFirm <= 11

THEN CALL calc\_end\_time(NEW.endtime, NEW.f\_distance,

NEW.starttime, 50);

WHEN NEW.idFirm > 11 AND NEW.idFirm <= 15

THEN CALL calc\_end\_time(NEW.endtime, NEW.f\_distance,

NEW.starttime, 22);

WHEN NEW.idFirm > 15

THEN CALL calc\_end\_time(NEW.endtime, NEW.f\_distance,

NEW.starttime, 65);

END CASE;

END;//

delimiter ;

Введено тестові дані для перевірки результату:

INSERT INTO Flights(idFirm, f\_from, f\_to, f\_distance, idTransport, starttime)

VALUES (8, 'Київ', 'Дніпро', 488, 51, '2021-10-24 21:22:46'),

(10, 'Херсон', 'Миколаїв', 80, 57, '2021-05-08 14:01:34');

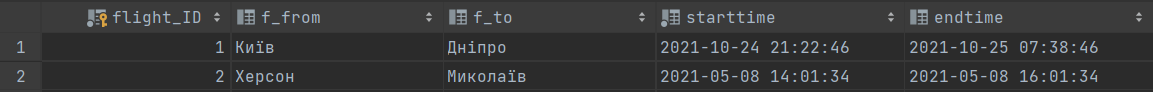


Рисунок 2.20 – Результат виконання тригера

Тригер, який виводить користувацьке повідомлення про помилку, якщо введено від’ємне значення відстані (рисунок 2.21):

delimiter //

CREATE TRIGGER checkDistance BEFORE INSERT ON Flights

FOR EACH ROW

BEGIN

CASE

WHEN NEW.f\_distance <= 0

THEN SIGNAL sqlstate '45001' SET message\_text = 'Invalid distance!';

WHEN NEW.f\_from REGEXP '[0-9]' OR NEW.f\_to REGEXP '[0-9]'

THEN SIGNAL sqlstate '45001' SET message\_text = 'This string can`t include digits!';

ELSE BEGIN END;

END CASE;

END;//

delimiter ;

Введено тестові дані для перевірки результату:

INSERT INTO Flights(idFirm, f\_from, f\_to, f\_distance, idTransport, starttime)

VALUES (8, 'Київ', 'Дніпро', -488, 51, '2021-10-24 21:22:46');

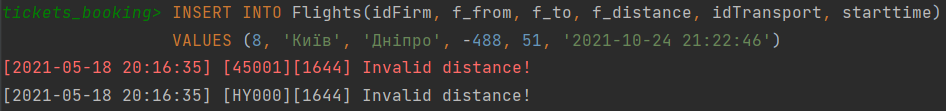


Рисунок 2.21 – Результат виконання тригера

Тригер, який встановлює статус рейсу відносно поточної дати та часу (рисунок 2.22):

delimiter //

CREATE TRIGGER checkStatus BEFORE UPDATE ON Flights

FOR EACH ROW

BEGIN

CASE

WHEN CURRENT\_TIMESTAMP < NEW.endtime AND OLD.status = 'Відбувся'

THEN SET NEW.status = 'Очікується';

WHEN NEW.endtime < CURRENT\_TIMESTAMP AND OLD.status = 'Очікується'

THEN SET NEW.status = 'Відбувся';

END CASE;

END;//

delimiter ;

Введено тестові дані для перевірки результату:

UPDATE Flights SET endtime = '2021-01-08 22:00:00' WHERE flight\_ID = 6;



Рисунок 2.22 – Результат виконання тригера

Тригер, який викликає процедуру, що встановлює вартість авіаквитка (рисунок 2.23):

delimiter //

CREATE TRIGGER setPriceAirTicket BEFORE INSERT ON Airplane\_tickets

FOR EACH ROW

BEGIN

CALL calc\_price\_at( NEW.at\_price, NEW.Flights\_has\_Cities\_primaryID, NEW.idType\_tickets);

END;//

delimiter ;

Введено тестові дані для перевірки результату:

INSERT INTO Airplane\_tickets(idType\_tickets, Flights\_has\_Cities\_primaryID)

VALUES (7, 10),

(9, 50),

(9, 13);

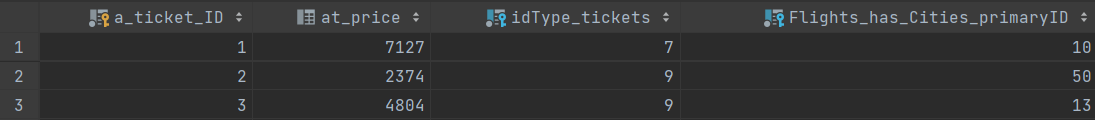


Рисунок 2.23 – Результат виконання тригера

Тригер, який визначає в якому класі знаходиться місце пасажира за його квитком (рисунок 2.24):

delimiter //

CREATE TRIGGER setClassAirHasCus BEFORE INSERT ON Airplane\_tickets\_has\_Customers

FOR EACH ROW

BEGIN

CASE

WHEN (SELECT idType\_tickets FROM Airplane\_tickets WHERE NEW.Airplane\_tickets\_a\_ticket\_ID = a\_ticket\_ID) = 7

THEN SET NEW.at\_class = 'Бізнес';

WHEN (SELECT idType\_tickets FROM Airplane\_tickets WHERE NEW.Airplane\_tickets\_a\_ticket\_ID = a\_ticket\_ID) = 8

THEN SET NEW.at\_class = 'Економ';

WHEN (SELECT idType\_tickets FROM Airplane\_tickets WHERE NEW.Airplane\_tickets\_a\_ticket\_ID = a\_ticket\_ID) = 9

THEN SET NEW.at\_class = 'Перший';

END CASE;

END;//

delimiter ;

Введено тестові дані для перевірки результату:

INSERT INTO Airplane\_tickets\_has\_Customers(Airplane\_tickets\_a\_ticket\_ID, Customers\_customer\_ID, a\_booking\_time, a\_seat, comments)

VALUES ( 1, 1, '2020-10-09 20:21:36', 1, NULL),

( 2, 2, '2020-01-14 09:55:01', 2, NULL);

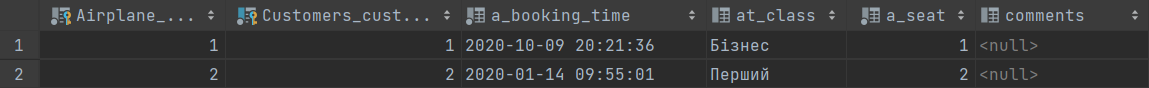


Рисунок 2.24 – Результат виконання тригера

Тригер, який викликає процедуру, що встановлює вартість квитка на водні види транспорту (рисунок 2.25):

delimiter //

CREATE TRIGGER setPriceShipTicket BEFORE INSERT ON Ship\_tickets

FOR EACH ROW

BEGIN

CALL calc\_price\_st( NEW.st\_price, NEW.Flights\_has\_Cities\_primaryID, NEW.idType\_tickets);

END;//

delimiter ;

Введено тестові дані для перевірки результату:

INSERT INTO Ship\_tickets(idType\_tickets, Flights\_has\_Cities\_primaryID)

VALUES ( 12, 32),

( 13, 32);

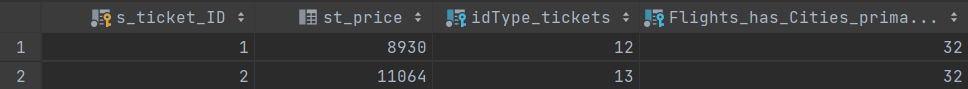


Рисунок 2.25 – Результат виконання тригера

Тригер, який визначає в якому класі знаходиться місце пасажира за його квитком (рисунок 2.26):

delimiter //

CREATE TRIGGER setClassShipHasCus BEFORE INSERT ON Ship\_tickets\_has\_Customers

FOR EACH ROW

BEGIN

CASE

WHEN (SELECT idType\_tickets

FROM Ship\_tickets

WHERE NEW.Ship\_tickets\_s\_ticket\_ID = Ship\_tickets.s\_ticket\_ID) = 12

THEN SET NEW.s\_class = 'Стандарт';

WHEN (SELECT idType\_tickets

FROM Ship\_tickets

WHERE NEW.Ship\_tickets\_s\_ticket\_ID = Ship\_tickets.s\_ticket\_ID) = 13

THEN SET NEW.s\_class = 'Преміум';

WHEN (SELECT idType\_tickets

FROM Ship\_tickets

WHERE NEW.Ship\_tickets\_s\_ticket\_ID = Ship\_tickets.s\_ticket\_ID) = 14

THEN SET NEW.s\_class = 'Економ';

END CASE;

END;//

delimiter ;

Введено тестові дані для перевірки результату:

INSERT INTO Ship\_tickets\_has\_Customers(ship\_tickets\_s\_ticket\_id, customers\_customer\_id, s\_booking\_time, s\_seat, comments)

VALUES ( 1, 26, '2020-03-24 05:43:44', 23, NULL),

( 2, 27, '2020-02-25 13:56:49', 97, NULL);

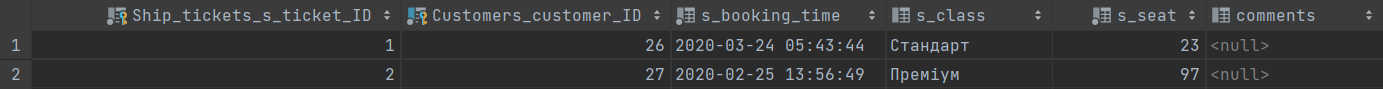


Рисунок 2.26 – Результат виконання тригера

Тригер, який викликає процедуру, що встановлює вартість автобусного квитка (рисунок 2.27):

delimiter //

CREATE TRIGGER setPriceBusTicket BEFORE INSERT ON Bus\_tickets

FOR EACH ROW

BEGIN

CALL calc\_price\_bt(NEW.bt\_price,NEW.idType\_tickets);

END;//

delimiter ;

Введено тестові дані для перевірки результату:

INSERT INTO Bus\_tickets(bt\_price, idType\_tickets, idRoute)

VALUES ( 318, 10, 1),

( 212, 10, 2);

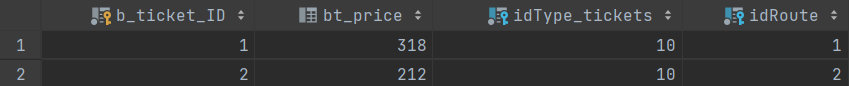


Рисунок 2.27 – Результат виконання тригера

Тригер, який викликає процедуру, що встановлює вартість квитка на поїзд (рисунок 2.28):

delimiter //

CREATE TRIGGER setPriceTrainTicket BEFORE INSERT ON Train\_tickets

FOR EACH ROW

BEGIN

CALL calc\_price\_tt(NEW.tt\_price,NEW.idType\_tickets);

END;//

delimiter ;

Введено тестові дані для перевірки результату:

INSERT INTO Train\_tickets( tt\_price, idType\_tickets, idRoute)

VALUES ( 394, 1, 31),

( 554, 1, 32);

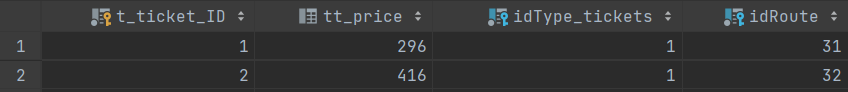


Рисунок 2.28 – Результат виконання тригера

**2.3 Створення запитів**

SELECT — оператор мови [SQL](https://uk.wikipedia.org/wiki/SQL_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), котрий повертає рядки з однієї чи багатьох таблиць. Повний синтаксис оператора SELECT є складним, проте його можна описати так:

SELECT список\_вибірки

[ INTO нова\_таблиця ]

FROM таблиця

[ WHERE умови\_пошуку ]

[ GROUP BY умова\_групування ]

[ HAVING умови\_пошуку ]

[ ORDER BY умова\_сортування [ ASC | DESC ] ]

Повертає нуль або більше рядків з однієї або більше таблиць, тимчасових таблиць, або ж представлень бази даних. У більшості застосунків, SELECT — найчастіша команда Data Manipulation Language (DML).[4]

У результаті аналізу завдання на курсове проєктування було створено наступні запити, які відображають усі можливі дії з даними, що містяться у таблицях.

Запит, який формує список пасажирів що придбали плацкартний білет (рисунок 2.29):

SELECT c\_name, concat(f\_from, ' - ', f\_to) AS Рейс, CONCAT(f\_from, ' - ', city\_name) AS Маршрут, f\_distance, tt\_price

FROM Customers INNER JOIN Train\_tickets\_has\_Customers TthC

ON Customers.customer\_ID = TthC.Customers\_customer\_ID INNER JOIN Train\_tickets Tt

ON TthC.Train\_tickets\_t\_ticket\_ID = Tt.t\_ticket\_ID INNER JOIN Routes R

ON Tt.idRoute = R.route\_ID INNER JOIN Flights\_has\_Cities FhC

ON R.Flights\_has\_Cities\_primaryID = FhC.primaryID INNER JOIN Flights F

ON FhC.idFlight = F.flight\_ID INNER JOIN Cities C

ON R.idCity = C.city\_ID INNER JOIN Type\_tickets T on Tt.idType\_tickets = T.typeticket\_ID

WHERE type\_afterpay = 'Плацкарт студентський'

ORDER BY tt\_price;

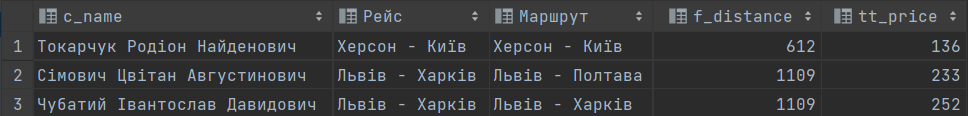


Рисунок 2.29 – Результат виконання запиту

Запит, який формує список пасажирів що придбали плацкартний білет (рисунок 2.30):

SELECT DISTINCT concat(f\_from, ' - ', f\_to) AS Рейс, starttime

FROM Customers INNER JOIN Bus\_tickets\_has\_Customers BthC

ON Customers.customer\_ID = BthC.Customers\_customer\_ID INNER JOIN Bus\_tickets Bt

ON BthC.Bus\_tickets\_b\_ticket\_ID = Bt.b\_ticket\_ID INNER JOIN Routes R

ON Bt.idRoute = R.route\_ID INNER JOIN Flights\_has\_Cities FhC

ON R.Flights\_has\_Cities\_primaryID = FhC.primaryID INNER JOIN Flights F

ON FhC.idFlight = F.flight\_ID INNER JOIN Cities C

ON R.idCity = C.city\_ID

WHERE DAYOFMONTH(starttime) BETWEEN 0 AND 15 AND MONTH(starttime) IN (6,7,8);

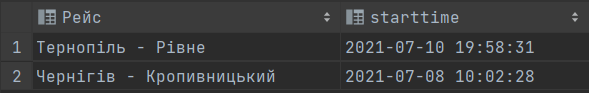


Рисунок 2.30 – Результат виконання запиту

Запит, який виводить кількість проданих та зарезервованих квитків кожного виду (рисунок 2.31):

SELECT type\_afterpay, LPAD(CONCAT(COUNT(typeticket\_ID), ' шт.'),10,' ') AS Sold

FROM Type\_tickets INNER JOIN Airplane\_tickets A

ON Type\_tickets.typeticket\_ID = A.idType\_tickets INNER JOIN Airplane\_tickets\_has\_Customers AthC

ON A.a\_ticket\_ID = AthC.Airplane\_tickets\_a\_ticket\_ID GROUP BY 1

UNION

SELECT type\_afterpay, LPAD(CONCAT(COUNT(typeticket\_ID), ' шт.'),10,' ')

FROM Type\_tickets INNER JOIN Train\_tickets T

ON Type\_tickets.typeticket\_ID = T.idType\_tickets INNER JOIN Train\_tickets\_has\_Customers TthC

ON T.t\_ticket\_ID = TthC.Train\_tickets\_t\_ticket\_ID GROUP BY 1

UNION

SELECT type\_afterpay, LPAD(CONCAT(COUNT(typeticket\_ID), ' шт.'),10,' ')

FROM Type\_tickets INNER JOIN Bus\_tickets B

ON Type\_tickets.typeticket\_ID = B.idType\_tickets INNER JOIN Bus\_tickets\_has\_Customers BthC

ON B.b\_ticket\_ID = BthC.Bus\_tickets\_b\_ticket\_ID GROUP BY 1

UNION

SELECT type\_afterpay, LPAD(CONCAT(COUNT(typeticket\_ID), ' шт.'),10,' ')

FROM Type\_tickets INNER JOIN Ship\_tickets S

ON Type\_tickets.typeticket\_ID = S.idType\_tickets INNER JOIN Ship\_tickets\_has\_Customers SthC

ON S.s\_ticket\_ID = SthC.Ship\_tickets\_s\_ticket\_ID GROUP BY 1;

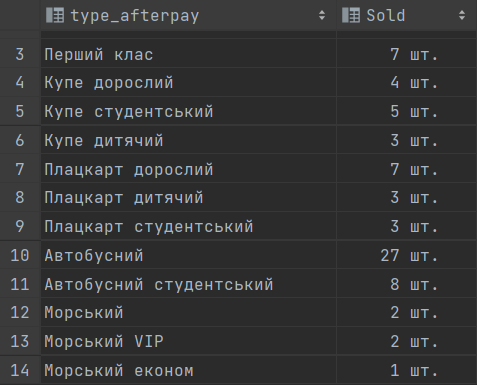


Рисунок 2.31 – Результат виконання запиту

Запит, який виводить інформацію про рейси, що відправляються за межі України (рисунок 2.32):

SELECT T.name AS Transport, CONCAT(f\_from, ' - ', f\_to) AS Flight, F.name AS Firm, type\_name AS Type\_trip

FROM Flights INNER JOIN Firms F

ON Flights.idFirm = F.firm\_ID INNER JOIN Type\_transportations Tt

ON F.idTransportation = Tt.transportation\_ID INNER JOIN Transport T

ON Flights.idTransport = T.transport\_ID INNER JOIN Flights\_has\_Cities

ON Flights.flight\_ID = FhC.idFlight INNER JOIN Cities C

ON FhC.idCity2 = C.city\_ID

WHERE country <> 'Україна';



Рисунок 2.32 – Результат виконання запиту

Запит, що виводить співвідношення наявної та загальної кількостей квитків на автобусні рейси (рисунок 2.33):

SELECT CONCAT(f\_from, ' - ', f\_to) AS Flight, CONCAT(T.name, '\t', T.number) AS Bus\_and\_number , type\_name,

CONCAT(T.countplace - (SELECT COUNT(Bus\_tickets\_b\_ticket\_ID)

FROM Bus\_tickets\_has\_Customers INNER JOIN Bus\_tickets Bt

ON Bus\_tickets\_has\_Customers.Bus\_tickets\_b\_ticket\_ID = Bt.b\_ticket\_ID

INNER JOIN Routes R ON Bt.idRoute = R.route\_ID INNER JOIN Flights\_has\_Cities FhC

ON R.Flights\_has\_Cities\_primaryID = FhC.primaryID

WHERE flight\_ID = idFlight), '/', T.countplace) AS Tickets

FROM Flights INNER JOIN Firms F

ON Flights.idFirm = F.firm\_ID INNER JOIN Type\_transportations Tt

ON F.idTransportation = Tt.transportation\_ID INNER JOIN Transport T

ON Flights.idTransport = T.transport\_ID

WHERE type\_name = 'Наземне';



Рисунок 2.33 – Результат виконання запиту

Запит, який вибирає назву міста в яке відправляється найбільша кількість залізничних рейсів (рисунок 2.34):

SELECT city\_name, COUNT(idCity2) AS count

FROM Routes INNER JOIN Flights\_has\_Cities FhC

ON Routes.Flights\_has\_Cities\_primaryID = FhC.primaryID INNER JOIN Cities C

ON FhC.idCity2 = C.city\_ID INNER JOIN Flights

ON FhC.idFlight = flight\_ID INNER JOIN Firms F

ON Flights.idFirm = firm\_ID INNER JOIN Type\_transportations Tt

ON F.idTransportation = Tt.transportation\_ID INNER JOIN Transport T

ON Flights.idTransport = T.transport\_ID

WHERE type\_name = 'Колійне'

GROUP BY 1

ORDER BY 2 DESC

LIMIT 1;



Рисунок 2.34 – Результат виконання запиту

Запит, який виводить рейси з середньою вартістю квитків більше 5000 грн., які прямують в будь-які країни світу окрім України, Франції і Болгарії (рисунок 2.35):

SELECT CONCAT(f\_from, ' - ', f\_to) AS Flight, AVG(at\_price) AS avg\_ticket\_price

FROM Flights INNER JOIN Flights\_has\_Cities FhC

ON Flights.flight\_ID = FhC.idFlight INNER JOIN Airplane\_tickets A

ON FhC.primaryID = A.Flights\_has\_Cities\_primaryID INNER JOIN Cities C

ON FhC.idCity2 = C.city\_ID

WHERE country NOT IN ('Україна', 'Франція', 'Болгарія')

GROUP BY 1

HAVING 5000 < avg\_ticket\_price

ORDER BY 2;

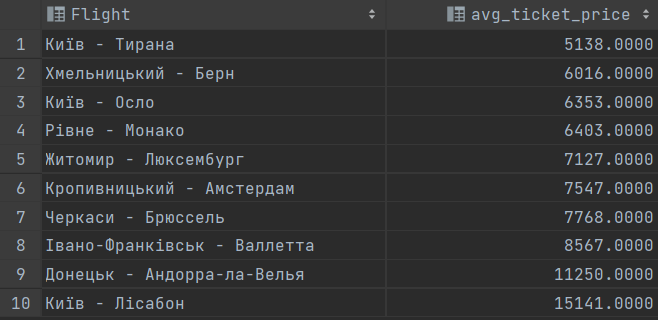


Рисунок 2.35 – Результат виконання запиту

Запит, який виводить ініціали пасажирів, що придбали більше одного квитка на автобусні рейси (рисунок 2.36):

SELECT c\_name, COUNT(b\_ticket\_ID) AS tickets, SUM(bt\_price) AS total\_sum

FROM Customers LEFT JOIN Bus\_tickets\_has\_Customers TthC

ON Customers.customer\_ID = TthC.Customers\_customer\_ID RIGHT JOIN Bus\_tickets Tt

ON Tt.b\_ticket\_ID = TthC.Bus\_tickets\_b\_ticket\_ID

GROUP BY 1

HAVING tickets <> 1

ORDER BY 3 DESC;

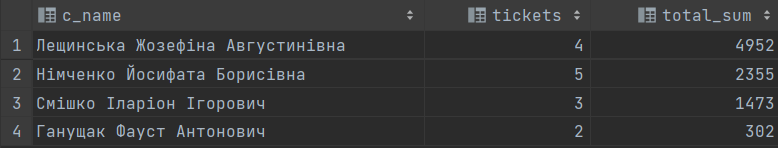


Рисунок 2.36 – Результат виконання запиту

Запит, який виводить найдовший та найкоротший за тривалістю рейси (рисунок 2.37):

(SELECT 'Найтриваліший рейс' AS Descrip, CONCAT(f\_from, ' - ', f\_to) AS Flight,

SEC\_TO\_TIME(TIMESTAMPDIFF(SECOND, starttime, endtime)) AS time\_interval

FROM Flights

ORDER BY 3 DESC

LIMIT 1)

UNION

(SELECT 'Найкоротший рейс', CONCAT(f\_from, ' - ', f\_to) AS Flight,

SEC\_TO\_TIME(TIMESTAMPDIFF(SECOND, starttime, endtime))

FROM Flights

ORDER BY 3

LIMIT 1);

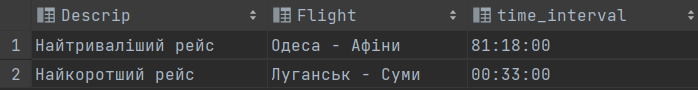


Рисунок 2.37 – Результат виконання запиту

Запит, який виводить автобусні рейси до Харкова та Києва з вартістю менше середньої (рисунок 2.38):

SELECT b\_ticket\_ID, CONCAT(f\_from, ' - ', city\_name) AS Flight, CONCAT(bt\_price, ' грн.')

FROM Flights INNER JOIN Flights\_has\_Cities FhC

ON Flights.flight\_ID = FhC.idFlight INNER JOIN Routes R

ON FhC.primaryID = R.Flights\_has\_Cities\_primaryID INNER JOIN Bus\_tickets Bt

ON R.route\_ID = Bt.idRoute INNER JOIN Bus\_tickets\_has\_Customers BthC

ON Bt.b\_ticket\_ID = BthC.Bus\_tickets\_b\_ticket\_ID INNER JOIN Cities C

ON C.city\_ID = FhC.idCity2

WHERE bt\_price < (SELECT AVG(bt\_price)

FROM Flights INNER JOIN Flights\_has\_Cities FhC

ON Flights.flight\_ID = FhC.idFlight INNER JOIN Routes R

ON FhC.primaryID = R.Flights\_has\_Cities\_primaryID INNER JOIN Bus\_tickets Bt

ON R.route\_ID = Bt.idRoute INNER JOIN Bus\_tickets\_has\_Customers BthC

ON Bt.b\_ticket\_ID = BthC.Bus\_tickets\_b\_ticket\_ID INNER JOIN Cities C

ON FhC.idCity2 = C.city\_ID)

HAVING Flight LIKE '%Київ' OR Flight LIKE '%Харків'

ORDER BY bt\_price DESC;

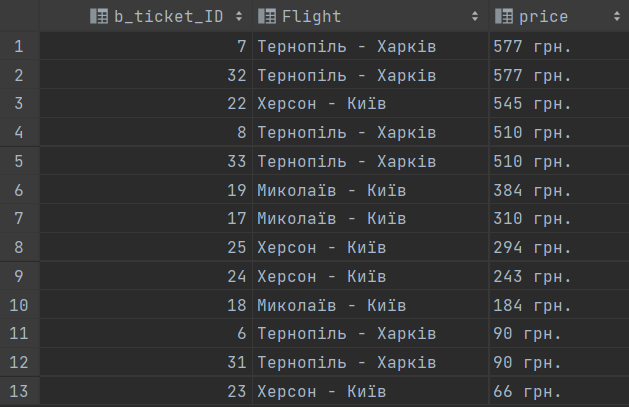


Рисунок 2.38 – Результат виконання запиту

**2.4 Створення представлень**

Представлення - об'єкт бази даних, що є результатом виконання запиту до бази даних, визначеного за допомогою оператора SELECT, в момент звернення до представлення.

Представлення іноді називають «віртуальними таблицями». Така назва пов'язана з тим, що представлення доступне для користувача як таблиця, але саме воно не містить даних, а витягує їх з таблиць в момент звернення до нього. Якщо дані змінені в базовій таблиці, то користувач отримає актуальні дані при зверненні до представлення, котрі використовують цю таблицю.

Представлення можуть грунтуватися як на таблицях, так і на інших представлення, тобто можуть бути вкладеними (до 32 рівнів вкладеності).

У відповідності до поставленого завдання на курсове проєктування було створено наступні представлення.

Представлення, яке містить інформацію про всі продані залізничні квитки (рисунок 2.39):

CREATE VIEW sold\_train\_tickets AS SELECT c\_name, concat(f\_from, ' - ', f\_to) AS Рейс,

CONCAT(f\_from, ' - ', city\_name) AS Маршрут, f\_distance, tt\_price

FROM Customers INNER JOIN Train\_tickets\_has\_Customers TthC

ON Customers.customer\_ID = TthC.Customers\_customer\_ID INNER JOIN Train\_tickets Tt

ON TthC.Train\_tickets\_t\_ticket\_ID = Tt.t\_ticket\_ID INNER JOIN Routes R

ON Tt.idRoute = R.route\_ID INNER JOIN Flights\_has\_Cities FhC

ON R.Flights\_has\_Cities\_primaryID = FhC.primaryID INNER JOIN Flights F

ON FhC.idFlight = F.flight\_ID INNER JOIN Cities C

ON R.idCity = C.city\_ID INNER JOIN Type\_tickets T on Tt.idType\_tickets = T.typeticket\_ID

ORDER BY tt\_price;



Рисунок 2.39 – Результат виконання представлення

Представлення, яке виводить рейси, що відправляються влітку (рисунок 2.40):

CREATE VIEW summer\_flights AS SELECT DISTINCT concat(f\_from, ' - ', f\_to) AS Рейс, starttime

FROM Customers INNER JOIN Bus\_tickets\_has\_Customers BthC

ON Customers.customer\_ID = BthC.Customers\_customer\_ID INNER JOIN Bus\_tickets Bt

ON BthC.Bus\_tickets\_b\_ticket\_ID = Bt.b\_ticket\_ID INNER JOIN Routes R

ON Bt.idRoute = R.route\_ID INNER JOIN Flights\_has\_Cities FhC

ON R.Flights\_has\_Cities\_primaryID = FhC.primaryID INNER JOIN Flights F

ON FhC.idFlight = F.flight\_ID INNER JOIN Cities C

ON R.idCity = C.city\_ID

WHERE MONTH(starttime) IN (6,7,8);

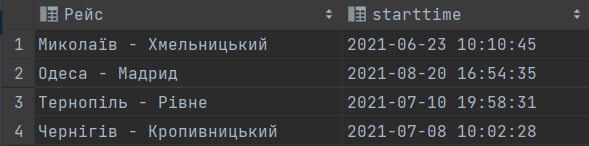


Рисунок 2.40 – Результат виконання представлення

Представлення, яке виводить інформацію про транспортні засоби, що не мають рейсів (рисунок 2.41):

CREATE VIEW unused\_transport AS SELECT \*

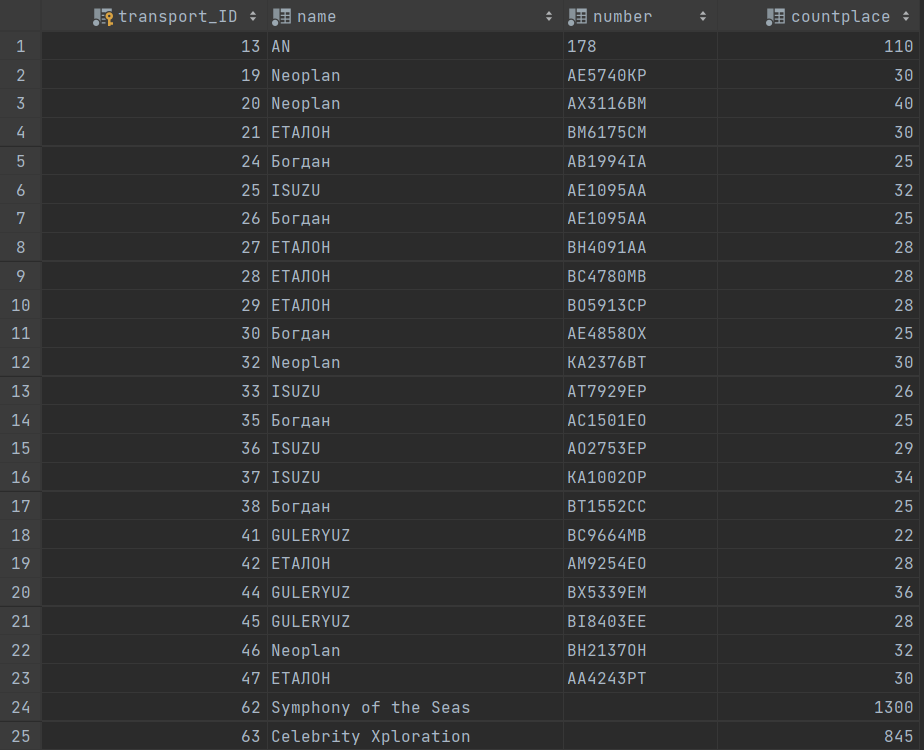
FROM Transport

WHERE transport\_ID NOT IN (

SELECT DISTINCT transport\_ID

FROM Transport INNER JOIN Flights F

ON Transport.transport\_ID = F.idTransport);



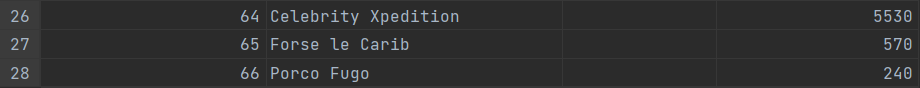


Рисунок 2.41 – Результат виконання представлення

Представлення, яке виводить рейси, що відправляються за межі України (рисунок 2.42):

CREATE VIEW foreign\_flights AS SELECT T.name AS Transport, CONCAT(f\_from, ' - ', f\_to) AS Flight, F.name AS Firm, type\_name AS Type\_trip

FROM Flights INNER JOIN Firms F

ON Flights.idFirm = F.firm\_ID INNER JOIN Type\_transportations Tt

ON F.idTransportation = Tt.transportation\_ID INNER JOIN Transport T

ON Flights.idTransport = T.transport\_ID INNER JOIN Flights\_has\_Cities FhC

ON Flights.flight\_ID = FhC.idFlight INNER JOIN Cities C

ON FhC.idCity2 = C.city\_ID

WHERE country <> 'Україна';



Рисунок 2.42 – Результат виконання представлення

Представлення, яке виводить п’ять найпопулярніших автобусних та залізничних рейсів. (рисунок 2.43):

CREATE VIEW the\_most\_popular\_travel\_city AS SELECT city\_name, COUNT(idCity2)

FROM Routes INNER JOIN Flights\_has\_Cities FhC

ON Routes.Flights\_has\_Cities\_primaryID = FhC.primaryID INNER JOIN Cities C

ON FhC.idCity2 = C.city\_ID INNER JOIN Flights

ON FhC.idFlight = flight\_ID INNER JOIN Firms F

ON Flights.idFirm = firm\_ID INNER JOIN Type\_transportations Tt

ON F.idTransportation = Tt.transportation\_ID INNER JOIN Transport T

ON Flights.idTransport = T.transport\_ID

WHERE type\_name IN ('Наземне', 'Колійне')

GROUP BY 1

ORDER BY 2 DESC

LIMIT 5;

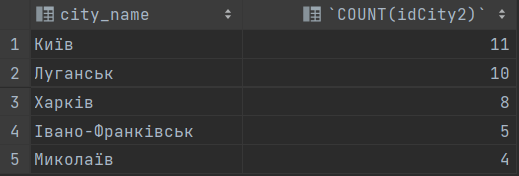


Рисунок 2.43 – Результат виконання представлення

**2.5 Створення процедур та функцій**

В MySQL 5 є багато нових можливостей, однією з найвагоміших із яких є створення збережених процедур та функцій. Збережені процедури і функції є підпрограмами, що створюються за допомогою операторів CREATE PROCEDURE і CREATE FUNCTION.

Процедура викликається за допомогою оператора call і може тільки передавати значення назад, використовуючи вихідні змінні.

Збережена процедура - це спосіб інкапсуляції повторюваних дій. В збережених процедурах можна оголошувати змінні, управляти потоками даних, а також застосовувати інші техніки програмування. Збережені процедури можуть викликати інші збережені процедури.

Функція - фрагмент програми, який повертає значення, а її виклик може використовуватися в програмі як вираз. Можна також розглядати функцію як окрему систему - чорну скриньку, на вхід якої надходять деякі управляючі дії у вигляді значень аргументів, а на виході системи одержуємо результат виконання.

Функції можуть повертати скалярне значення і викликаються з оператора точно так само, як і будь-які інші функції (тобто, через вказівку імені функції).

У відповідності до поставленого завдання на курсове проєктування було створено наступні процедури та функції.

Функція, для переведення чисел з плаваючою крапкою у формат “hh:mm:ss” (рисунок 2.44):

delimiter //

CREATE FUNCTION getHour (inHour FLOAT)

RETURNS TIME

DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE result VARCHAR(15);

IF ~~(ROUND(inHour,2)\*100)%100 >= 60

THEN SET result = CONCAT(FLOOR(inHour) + 1, ':', (~~(ROUND(inHour,2)\*100)%100) - 60, ':00');

ELSE SET result = CONCAT(FLOOR(inHour), ':', ~~(ROUND(inHour,2)\*100)%100, ':00');

END IF;

RETURN result;

END;//

delimiter ;

SELECT getHour(1.3);



Рисунок 2.44 – Результат виконання функції

Функція, яка генерує випадкову дату з проміжку (рисунок 2.45):

delimiter //

CREATE FUNCTION randDATETIME (from\_date DATETIME, to\_date DATETIME)

RETURNS DATETIME

DETERMINISTIC

READS SQL DATA

BEGIN

DECLARE result DATETIME;

SET result = FROM\_UNIXTIME(RAND()\*(UNIX\_TIMESTAMP(to\_date)-

UNIX\_TIMESTAMP(from\_date))+UNIX\_TIMESTAMP(from\_date));

RETURN result;

END;//

delimiter ;

SELECT randDATETIME('2000:01:01 00:00:00', CURRENT\_TIMESTAMP) AS rand\_datetime, randDATETIME('2000:01:01 00:00:00', CURRENT\_TIMESTAMP) AS rand\_datetime, randDATETIME('2000:01:01 00:00:00', CURRENT\_TIMESTAMP) AS rand\_datetime, randDATETIME('2000:01:01 00:00:00', CURRENT\_TIMESTAMP) AS rand\_datetime;



Рисунок 2.45 – Результат виконання функції

Функція, яка повертає всі числа, що знаходяться у вхідному рядку (рисунок 2.46):

DELIMITER $$

CREATE FUNCTION ExtractNumber(in\_string VARCHAR(50))

RETURNS INT

NO SQL

BEGIN

DECLARE ctrNumber VARCHAR(50);

DECLARE finNumber VARCHAR(50) DEFAULT '';

DECLARE sChar VARCHAR(1);

DECLARE inti INTEGER DEFAULT 1;

IF LENGTH(in\_string) > 0

THEN WHILE(inti <= LENGTH(in\_string)) DO

SET sChar = SUBSTRING(in\_string, inti, 1);

SET ctrNumber = FIND\_IN\_SET(sChar, '0,1,2,3,4,5,6,7,8,9');

IF ctrNumber > 0

THEN SET finNumber = CONCAT(finNumber, sChar);

END IF;

SET inti = inti + 1;

END WHILE;

RETURN CAST(finNumber AS UNSIGNED);

ELSE

RETURN 0;

END IF;

END; $$

DELIMITER ;

SELECT ExtractNumber(number) AS "Номери автобусів без регіонального кодування і з ним", number

FROM Transport INNER JOIN Flights F

ON Transport.transport\_ID = F.idTransport INNER JOIN Firms F2

ON F.idFirm = F2.firm\_ID INNER JOIN Type\_transportations Tt

ON F2.idTransportation = Tt.transportation\_ID

WHERE type\_name = 'Наземне';



Рисунок 2.46 – Результат виконання функції

Процедура, яка обчислює та встановлює вартість авіаквитків (рисунок 2.47):

DELIMITER $$

CREATE PROCEDURE `calc\_price\_at`(OUT new\_price DECIMAL, IN new\_FhCpID INTEGER, IN new\_idTt INTEGER)

BEGIN

SET new\_price = 4 \* (SELECT f\_distance FROM Flights INNER JOIN Flights\_has\_Cities FhC on Flights.flight\_ID = FhC.idFlight WHERE primaryID = new\_FhCpID) + ((SELECT f\_distance FROM Flights INNER JOIN Flights\_has\_Cities FhC

on Flights.flight\_ID = FhC.idFlight WHERE idFlight = new\_FhCpID) \*

(SELECT percent\_afterpay/100 FROM Type\_tickets WHERE new\_idTt = Type\_tickets.typeticket\_ID));

END; $$

CALL calc\_price\_at(@test,10,7);

SELECT ROUND(@test,2) AS "Ціна", f\_distance AS "Відстань", CONCAT(f\_from, ' - ', f\_to) AS "Рейс"

FROM Flights INNER JOIN Flights\_has\_Cities FhC

ON Flights.flight\_ID = FhC.idFlight

WHERE idFlight = 10;



Рисунок 2.47 – Результат виконання процедури

Процедура, яка обчислює та встановлює вартість квитків на морські рейси (рисунок 2.48):

DELIMITER $$

CREATE PROCEDURE `calc\_price\_st`(OUT new\_price DECIMAL, IN new\_FhCpID INTEGER, IN new\_idTt INTEGER)

BEGIN

SET new\_price = 5 \* (SELECT f\_distance FROM Flights INNER JOIN Flights\_has\_Cities FhC

on Flights.flight\_ID = FhC.idFlight WHERE primaryID = new\_FhCpID) +

((SELECT f\_distance FROM Flights INNER JOIN Flights\_has\_Cities FhC

on Flights.flight\_ID = FhC.idFlight WHERE idFlight = new\_FhCpID) \*

(SELECT percent\_afterpay/100 FROM Type\_tickets

WHERE new\_idTt = Type\_tickets.typeticket\_ID));

END; $$

DELIMITER ;

CALL calc\_price\_st(@test,32,12);

SELECT ROUND(@test,2) AS "Ціна", f\_distance AS "Відстань", CONCAT(f\_from, ' - ', f\_to) AS "Рейс"

FROM Flights INNER JOIN Flights\_has\_Cities FhC

ON Flights.flight\_ID = FhC.idFlight

WHERE idFlight = 32;



Рисунок 2.48 – Результат виконання процедури

Процедура, яка обчислює та встановлює вартість залізничних квитків (рисунок 2.49):

DELIMITER $$

CREATE PROCEDURE `calc\_price\_tt`(INOUT new\_price DECIMAL, IN new\_idTt INTEGER)

BEGIN

SET new\_price = new\_price/2 + (new\_price/2)\*(SELECT percent\_afterpay/100

FROM Type\_tickets WHERE typeticket\_ID = new\_idTt);

END; $$

DELIMITER ;

SET @test := 394;

CALL calc\_price\_tt(@test,1);

SELECT ROUND(@test,2) AS "Ціна", f\_distance AS "Відстань", CONCAT('Одеса', ' - ', f\_to) AS "Рейс"

FROM Flights INNER JOIN Flights\_has\_Cities FhC

ON Flights.flight\_ID = FhC.idFlight

WHERE idFlight = 31;



Рисунок 2.49 – Результат виконання процедури

Процедура, яка виконує пошук маршрутів за точкою відправлення і точкою прибуття (рисунок 2.50):

DELIMITER $$

CREATE PROCEDURE `find\_routes`(IN from\_city VARCHAR(45), IN to\_city VARCHAR(45))

BEGIN

DECLARE flight\_from VARCHAR(45);

DECLARE flight\_to VARCHAR(45);

DECLARE route\_from VARCHAR(45);

DECLARE route\_to VARCHAR(45);

DECLARE flight\_time DATETIME;

DECLARE myCondition BOOLEAN;

DECLARE success BOOLEAN DEFAULT FALSE;

DECLARE myCursor CURSOR FOR SELECT f\_from, f\_to, f\_from, city\_name, starttime

FROM Flights INNER JOIN Flights\_has\_Cities FhC

ON Flights.flight\_ID = FhC.idFlight INNER JOIN Routes R

ON FhC.primaryID = R.Flights\_has\_Cities\_primaryID INNER JOIN Cities C

ON C.city\_ID = R.idCity INNER JOIN Firms F

ON Flights.idFirm = F.firm\_ID INNER JOIN Type\_transportations Tt

ON F.idTransportation = Tt.transportation\_ID;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET myCondition = FALSE;

OPEN myCursor;

SET myCondition = TRUE;

WHILE myCondition DO

FETCH myCursor INTO flight\_from, flight\_to, route\_from, route\_to, flight\_time;

IF(route\_from = from\_city AND route\_to = to\_city) THEN

SELECT flight\_time AS "Час відправлення", CONCAT(flight\_from, ' - ', flight\_to) AS "Рейс",

CONCAT(route\_from, ' - ', route\_to) AS "Маршрут";

SET success = TRUE;

END IF;

END WHILE;

CLOSE myCursor;

IF NOT success

THEN SELECT 'Маршрутів не знайдено!' AS "Повідомлення";

END IF;

END; $$

DELIMITER ;

CALL find\_routes('Львів', 'Київ');



Рисунок 2.50 – Результат виконання процедури

Процедура, яка виконує пошук рейсів за датою відправлення (рисунок 2.51, рисунок 2.52):

DELIMITER $$

CREATE PROCEDURE `find\_flight`(IN flight\_datetime DATETIME)

BEGIN

DECLARE flight\_from VARCHAR(45);

DECLARE flight\_to VARCHAR(45);

DECLARE flight\_distance FLOAT;

DECLARE flight\_startTime DATETIME;

DECLARE flight\_endTime DATETIME;

DECLARE myCondition BOOLEAN;

DECLARE success BOOLEAN DEFAULT FALSE;

DECLARE myCursor CURSOR FOR SELECT f\_from, f\_to, f\_distance, starttime, endtime FROM Flights;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET myCondition = FALSE;

OPEN myCursor;

SET myCondition = TRUE;

WHILE myCondition DO

FETCH myCursor INTO flight\_from, flight\_to, flight\_distance, flight\_startTime, flight\_endTime;

IF(flight\_startTime = flight\_datetime) THEN

SELECT 'Точний збіг' AS ' ', flight\_from, flight\_to, flight\_distance, flight\_startTime, flight\_endTime;

SET success = TRUE;

ELSEIF(DATE(flight\_startTime) = DATE(flight\_datetime)) THEN

SELECT 'Схожий рейс' AS 'Точних збігів не знайдено!', flight\_from, flight\_to, flight\_distance, flight\_startTime, flight\_endTime;

SET success = TRUE;

END IF;

END WHILE;

CLOSE myCursor;

IF NOT success

THEN SELECT 'Рейсів не знайдено!' AS "Повідомлення";

END IF;

END; $$

DELIMITER ;

CALL find\_flight('2021-08-09 20:18:56');

CALL find\_flight('2021-08-09 00:00:00');



Рисунок 2.51 – Рейс з ідентичною датою та часом відправлення



Рисунок 2.52 – Рейс з ідентичною датою відправлення

Процедура, яка обчислює дату та час прибуття рейсу (рисунок 2.53):

DELIMITER $$

CREATE PROCEDURE `calc\_end\_time`(OUT flight\_endTime DATETIME, IN flight\_distance FLOAT, IN flight\_startTime DATETIME,

IN avg\_speed FLOAT)

BEGIN

SET flight\_endTime = ADDTIME(flight\_startTime, getHour(flight\_distance/avg\_speed));

END; $$

DELIMITER ;

SELECT 50 AS avg\_speed, 488 AS distance, '2021-10-24 21:22:46' AS startTime, @test AS endTime;SELECT @test;



Рисунок 2.53 – Результат виконання процедури

**2.6 Додавання користувачів та надання їм прав**

СКБД MySQL є багатокористувацьким середовищем, тому для доступу до таблиць БД можуть бути створені різні облікові записи з різним рівнем привілеїв. Обліковому запису редактора можна надати привілеї на перегляд таблиці, додавання нових записів і оновлення вже існуючих. Адміністратору БД можна надати більш широкі повноваження (можливість створення таблиць, редагування та видалення вже існуючих). Для користувача БД достатньо лише перегляду таблиць.

У результаті аналізу завдання на курсове проєктування було створено наступних користувачів.

Користувач, який має всі права для бази Tickets\_booking (рисунок 2.54):

CREATE USER 'Developer'@'localhost:3306' IDENTIFIED BY 'helloWorld';

GRANT ALL PRIVILEGES ON Tickets\_booking.\* TO 'Developer'@'localhost:3306';

FLUSH PRIVILEGES;

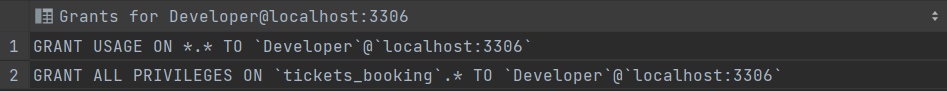


Рисунок 2.54 – Результат створення користувача

Користувач, який має доступ до редагування таблиць та створення представлень для бази Tickets\_booking (рисунок 2.55):

CREATE USER 'Admin'@'localhost:3306' IDENTIFIED BY 'myAdmin';

GRANT SELECT, INSERT, DELETE, UPDATE, CREATE VIEW, EXECUTE ON Tickets\_booking.\* TO 'Admin'@'localhost:3306';

FLUSH PRIVILEGES;

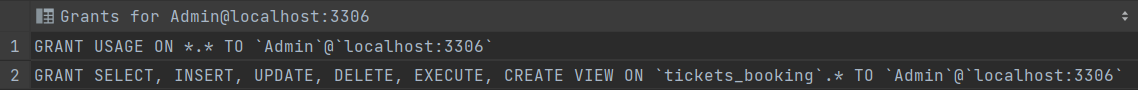


Рисунок 2.55 – Результат створення користувача

Користувач, який має доступ до вибірки даних з всіх таблиць бази Tickets\_booking (рисунок 2.56):

CREATE USER 'Customer'@'localhost:3306' IDENTIFIED BY 'qwerty123';

GRANT SELECT ON Tickets\_booking.\* TO 'Customer'@'localhost:3306';

FLUSH PRIVILEGES;

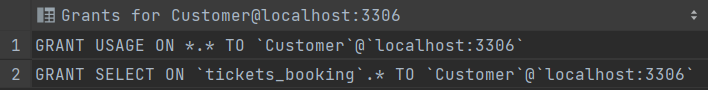


Рисунок 2.56 – Результат створення користувача

Повний код створеної бази даних у СУБД MySQL наведено у додатку В.

**3 РОЗРОБЛЕННЯ БАЗИ ДАНИХ ЗАСОБАМИ СКБД POSTGRESQL**

PostgreSQL - це об'єктно-реляційна система управління базами даних ( ОРСУБД, ORDBMS ), заснована на POSTGRES, Version 4.2 - програмою, розробленою на факультеті комп'ютерних наук Каліфорнійського університету в Берклі. У POSTGRES з'явилося безліч нововведень, які були реалізовані в деяких комерційних СУБД набагато пізніше.

**3.1 Створення таблиць**

Вся інформація таблиць PostgreSQL знаходиться в оперативній пам'яті. Не можливо створити таблицю, яка буде не в пам'яті. Записи таблиці упорядковано відповідно до індексу, це забезпечує можливість швидкої вибірки даних.

В цілому PostgreSQL працює швидше ніж MySQL, за виключенням використання первинних ключів.

У відповідності до розробленої схеми даних було створено SQL таблиці бази  даних.

Першим кроком створено типи перерахування:

CREATE TYPE FLIGHTSTATUS AS ENUM('Скасований', 'Відбувся', 'Очікується');

CREATE TYPE TYPETICKET AS ENUM('Купе дорослий', 'Купе дитячий', 'Купе студентський', 'Плацкарт дорослий', 'Плацкарт дитячий', 'Плацкарт студентський', 'Бізнес клас', 'Економ клас', 'Перший клас', 'Автобусний','Автобусний студентський', 'Морський', 'Морський VIP',

'Морський економ');

Створено таблицю "Type\_transportations". Структура таблиці наведена на рисунку 3.1.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Type\_tickets(

typeticket\_ID SERIAL NOT NULL,

type\_afterpay TYPETICKET NOT NULL,

percent\_afterpay INT NOT NULL,

PRIMARY KEY (typeticket\_ID));

Код заповнення таблиці:

BEGIN ;  
INSERT INTO Type\_transportations(type\_name)  
VALUES ('Повітряне'),('Колійне'),('Водне'),('Наземне');  
COMMIT ;

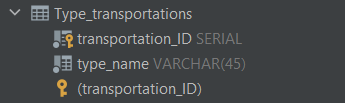


Рисунок 3.1 – Структура таблиці "Type\_transportations"

Створено таблицю " Firms". Структура таблиці наведена на рисунку 3.2.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Firms (

firm\_ID SERIAL NOT NULL,

idTransportation INT NOT NULL,

name VARCHAR(45) NOT NULL,

email VARCHAR(45) NOT NULL,

tel VARCHAR(20) NOT NULL,

region\_activity VARCHAR(20) NULL,

PRIMARY KEY (firm\_ID),

FOREIGN KEY (idTransportation) REFERENCES Type\_transportations (transportation\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;  
INSERT INTO Firms(idTransportation, name, email, tel, regin\_activity)  
VALUES (1,'МАУ','uia@flyuia.com','+38 (044) 581-50-50','Україна'),  
 (1,'KLM','KLM.Ukraine@klm.nl','+31 (0) 20 649 91 23','Нідерланди'),  
 (1,'Emirates Airlines','pr@emirates.com','600 555 555','ОАЕ'),  
 ...  
 (4,'Ecolines','help@ecolines.ua','+38 044 594 90 10',''),  
 (4,'EAST WEST EUROLINES','support@ewe.ua','+380988154444',''),  
 (4,'TransTempo','transtempo@ukr.net','+38 (067) 467-44-77','');  
COMMIT ;

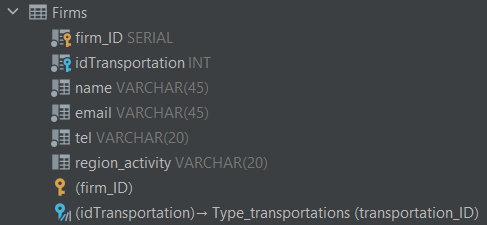


Рисунок 3.2 – Структура таблиці "Firms"

Створено таблицю "Flights". Структура таблиці наведена на рисунку 3.3.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Flights (

flight\_ID SERIAL NOT NULL,

idFirm INT NOT NULL,

idTransport INT NOT NULL,

f\_from VARCHAR(45) NULL,

f\_to VARCHAR(45) NULL,

f\_distance FLOAT NULL,

starttime TIMESTAMP NOT NULL,

endtime TIMESTAMP NULL,

status FLIGHTSTATUS DEFAULT 'Очікується',

PRIMARY KEY (flight\_ID),

FOREIGN KEY (idFirm) REFERENCES Firms(firm\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (idTransport) REFERENCES Transport(transport\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;  
INSERT INTO Flights(idFirm, f\_from, f\_to, f\_distance, idTransport, starttime)  
VALUES (8, 'Київ', 'Дніпро', 488, 51, '2021-10-24 21:22:46'),  
 (10, 'Херсон', 'Миколаїв', 80, 57, '2021-05-08 14:01:34'),  
 (11, 'Київ', 'Луганськ', 922, 6, '2021-08-09 20:18:56'),  
 ...  
 (10, 'Кропивницький', 'Миколаїв', 182, 53, '2021-03-10 02:42:48'),  
 (3, 'Хмельницький', 'Берн', 1444, 4, '2021-03-19 11:52:21'),  
 (1, 'Київ', 'Осло', 1630, 3, '2021-01-02 18:36:57');  
COMMIT ;

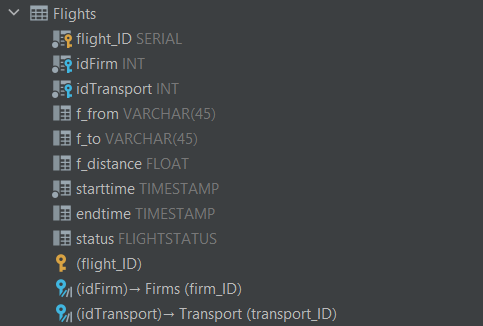


Рисунок 3.3 – Структура таблиці "Flights"

Створено таблицю "Customers". Структура таблиці наведена на рисунку 3.4.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Customers (

customer\_ID SERIAL NOT NULL,

c\_name VARCHAR(45) NOT NULL,

c\_birthday DATE NOT NULL,

PRIMARY KEY (customer\_ID));

Код заповнення таблиці:

BEGIN ;  
INSERT INTO Customers(c\_name, c\_birthday)  
VALUES ('Тарасович Ратимир Вадимович', '1952-08-10'),  
 ('Ніколенко Ольга Вікторівна', '1963-05-17'),  
 ('Кириленко Муховіст Полянович', '1987-03-19'),  
 ...  
 ( 'Мухопад Царук Адріанович', '1982-04-20'),  
 ( 'Бурбан Вернислав Артемович', '1993-03-05'),  
 ( 'Штинь Атрей Русланович', '1956-11-19'),  
 ( 'Деркач Лютобор Романович', '1966-07-25');  
COMMIT ;

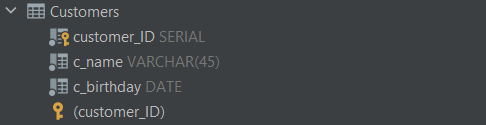


Рисунок 3.4 – Структура таблиці "Customers"

Створено таблицю "Type\_tickets". Структура таблиці наведена на рисунку 3.5.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Type\_tickets(

typeticket\_ID SERIAL NOT NULL,

type\_afterpay TYPETICKET NOT NULL,

percent\_afterpay INT NOT NULL,

PRIMARY KEY (typeticket\_ID));

Код заповнення таблиці:

BEGIN ;

INSERT INTO Type\_tickets (type\_afterpay, percent\_afterpay)

VALUES ( 'Купе дорослий', 50),

( 'Купе студентський', 30),

( 'Купе дитячий', 10),

( 'Плацкарт дорослий', 0),

( 'Плацкарт дитячий', -30),

( 'Плацкарт студентський', -50),

( 'Бізнес клас', 150),

( 'Економ клас', -10),

( 'Перший клас', 50),

( 'Автобусний', 0),

( 'Автобусний студентський', -20),

( 'Морський', 0),

( 'Морський VIP', 75),

( 'Морський економ',-15);

COMMIT ;

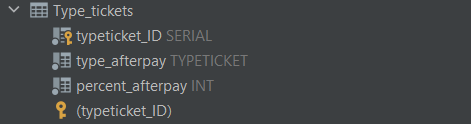


Рисунок 3.5 – Структура таблиці "Type\_tickets"

Створено таблицю "Cities". Структура таблиці наведена на рисунку 3.6.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Cities (

city\_ID SERIAL NOT NULL,

city\_name VARCHAR(45) NOT NULL,

country VARCHAR(45) NOT NULL,

PRIMARY KEY (city\_ID));

Код заповнення таблиці:

BEGIN ;

INSERT INTO Cities (country, city\_name)

VALUES ('Молдова', 'Кишинів'),

('Греція', 'Афіни'),

('Хорватія', 'Загреб'),

...

('Україна','Хмельницький'),

('Україна','Черкаси'),

('Україна','Чернівці'),

('Україна','Чернігів');

COMMIT ;

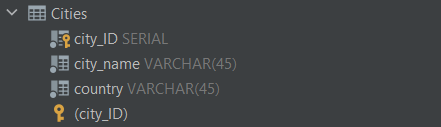


Рисунок 3.6 – Структура таблиці "Cities"

Створено таблицю "Flights\_has\_Cities". Структура таблиці наведена на рисунку 3.7.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Flights\_has\_Cities (

primaryID SERIAL NOT NULL,

idFlight INT NOT NULL,

idCity1 INT NOT NULL,

idCity2 INT NULL,

PRIMARY KEY (primaryID),

FOREIGN KEY (idFlight) REFERENCES Flights (flight\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (idCity1) REFERENCES Cities (city\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (idCity2) REFERENCES Cities (city\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Flights\_has\_Cities(idFlight, idCity1, idCity2)

(SELECT flight\_ID, city\_ID, (SELECT city\_ID FROM Cities WHERE f\_to = city\_name) FROM Flights, Cities

WHERE f\_from = city\_name);

COMMIT ;



Рисунок 3.7 – Структура таблиці "Flights\_has\_Cities"

Створено таблицю "Airplane\_tickets". Структура таблиці наведена на рисунку 3.8.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Airplane\_tickets (

a\_ticket\_ID SERIAL NOT NULL,

at\_price DECIMAL NULL,

idType\_tickets INT NOT NULL,

Flights\_has\_Cities\_primaryID INT NOT NULL,

PRIMARY KEY (a\_ticket\_ID),

FOREIGN KEY (idType\_tickets) REFERENCES Type\_tickets (typeticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Flights\_has\_Cities\_primaryID) REFERENCES Flights\_has\_Cities (primaryID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Airplane\_tickets(idType\_tickets, Flights\_has\_Cities\_primaryID)

VALUES (7, 10),

(9, 50),

(9, 13),

(8, 25),

...

(8, 37),

(9, 7),

(8, 6);

COMMIT ;

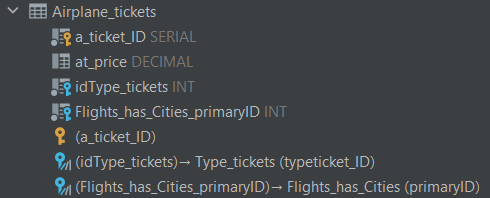


Рисунок 3.8 – Структура таблиці "Airplane\_tickets"

Створено таблицю "Routes". Структура таблиці наведена на рисунку 3.9.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Routes (

route\_ID SERIAL NOT NULL,

Flights\_has\_Cities\_primaryID INT NOT NULL,

idCity INT NOT NULL,

PRIMARY KEY (route\_ID),

FOREIGN KEY (Flights\_has\_Cities\_primaryID) REFERENCES Flights\_has\_Cities (primaryID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (idCity) REFERENCES Cities (city\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Routes(Flights\_has\_Cities\_primaryID, idCity)

VALUES ( 8, 60),

( 8, 63),

( 8, 50),

( 29, 43),

...

( 47, 64),

( 47, 55),

( 9, 50),

( 9, 60);

COMMIT ;

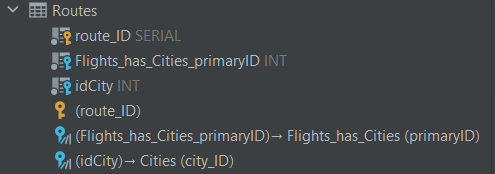


Рисунок 3.9 – Структура таблиці "Routes"

Створено таблицю "Train\_tickets". Структура таблиці наведена на рисунку 3.10.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Train\_tickets (

t\_ticket\_ID SERIAL NOT NULL,

tt\_price DECIMAL NULL,

idType\_tickets INT NOT NULL,

idRoute INT NOT NULL,

PRIMARY KEY (t\_ticket\_ID),

FOREIGN KEY (idType\_tickets) REFERENCES Type\_tickets (typeticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (idRoute) REFERENCES Routes (route\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Train\_tickets( tt\_price, idType\_tickets, idRoute)

VALUES ( 394, 1, 31),

( 554, 1, 32),

( 221, 2, 33),

...

( 185, 1, 53),

( 394, 4, 54),

( 472, 5, 55);

COMMIT ;

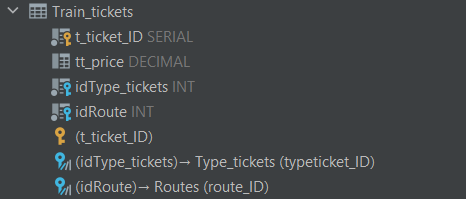


Рисунок 3.10 – Структура таблиці "Train\_tickets"

Створено таблицю "Bus\_tickets". Структура таблиці наведена на рисунку 3.11.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Bus\_tickets (

b\_ticket\_ID SERIAL NOT NULL,

bt\_price DECIMAL NULL,

idType\_tickets INT NOT NULL,

idRoute INT NOT NULL,

PRIMARY KEY (b\_ticket\_ID),

FOREIGN KEY (idRoute) REFERENCES Routes (route\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (idType\_tickets) REFERENCES Type\_tickets (typeticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Bus\_tickets(bt\_price, idType\_tickets, idRoute)

VALUES ( 318, 10, 1),

( 212, 10, 2),

( 448, 10, 3),

...

( 638, 11, 8),

( 953, 10, 9),

( 3754, 10, 10);

COMMIT ;

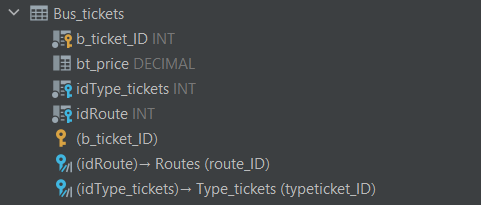


Рисунок 3.11 – Структура таблиці "Bus\_tickets"

Створено таблицю "Ship\_tickets". Структура таблиці наведена на рисунку 3.12.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Ship\_tickets (

s\_ticket\_ID SERIAL NOT NULL,

st\_price DECIMAL NULL,

idType\_tickets INT NOT NULL,

Flights\_has\_Cities\_primaryID INT NOT NULL,

PRIMARY KEY (s\_ticket\_ID),

FOREIGN KEY (idType\_tickets) REFERENCES Type\_tickets (typeticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Flights\_has\_Cities\_primaryID) REFERENCES Flights\_has\_Cities (primaryID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Ship\_tickets(idType\_tickets, Flights\_has\_Cities\_primaryID)

VALUES ( 12, 32),

( 13, 32),

( 14, 32),

...

( 13, 32),

( 13, 32),

( 13, 32);

COMMIT ;

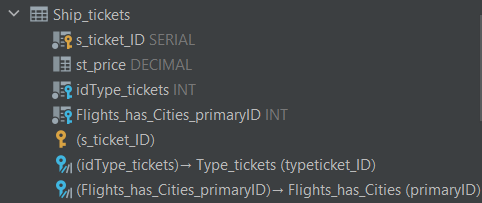


Рисунок 3.12 – Структура таблиці "Ship\_tickets"

Створено таблицю "Train\_tickets\_has\_Customers". Структура таблиці наведена на рисунку 3.13.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Train\_tickets\_has\_Customers (

Train\_tickets\_t\_ticket\_ID INT NOT NULL,

Customers\_customer\_ID INT NOT NULL,

t\_booking\_time TIMESTAMP NOT NULL,

t\_numwagon INT NOT NULL,

t\_seat INT NOT NULL,

comments VARCHAR(45) NULL,

PRIMARY KEY (Train\_tickets\_t\_ticket\_ID, Customers\_customer\_ID),

FOREIGN KEY (Train\_tickets\_t\_ticket\_ID) REFERENCES Train\_tickets (t\_ticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Customers\_customer\_ID) REFERENCES Customers (customer\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Train\_tickets\_has\_Customers(train\_tickets\_t\_ticket\_id, customers\_customer\_id, t\_booking\_time, t\_numwagon, t\_seat)

VALUES ( 1, 2, '2020-03-15 09:59:44', 1, 21),

( 2, 4, '2020-09-01 05:27:15', 6, 13),

( 3, 6, '2020-11-11 16:03:38', 4, 17),

...

( 23, 46, '2020-05-06 04:59:54', 7, 42),

( 24, 48, '2020-12-04 22:03:56', 1, 45),

( 25, 50, '2020-05-01 14:58:58', 2, 21);

COMMIT ;



Рисунок 3.13 – Структура таблиці "Train\_tickets\_has\_Customers"

Створено таблицю "Airplane\_tickets\_has\_Customers". Структура таблиці наведена на рисунку 3.14.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Airplane\_tickets\_has\_Customers (

Airplane\_tickets\_a\_ticket\_ID INT NOT NULL,

Customers\_customer\_ID INT NOT NULL,

a\_booking\_time TIMESTAMP NOT NULL,

at\_class VARCHAR(15) NULL,

a\_seat INT NOT NULL,

comments VARCHAR(45) NULL,

PRIMARY KEY (Airplane\_tickets\_a\_ticket\_ID, Customers\_customer\_ID),

FOREIGN KEY (Airplane\_tickets\_a\_ticket\_ID) REFERENCES Airplane\_tickets (a\_ticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Customers\_customer\_ID) REFERENCES Customers (customer\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

COMMIT ;

INSERT INTO Airplane\_tickets\_has\_Customers

(Airplane\_tickets\_a\_ticket\_ID, Customers\_customer\_ID, a\_booking\_time, a\_seat, comments)

VALUES ( 1, 1, '2020-10-09 20:21:36', 1, NULL),

( 2, 2, '2020-01-14 09:55:01', 2, NULL),

( 3, 3, '2020-06-23 14:27:41', 3, NULL),

...

( 23, 23, '2020-04-16 16:54:07', 23, NULL),

( 24, 24, '2020-06-25 04:22:31', 24, NULL),

( 25, 25, '2020-10-28 08:31:29', 25, NULL);

COMMIT ;

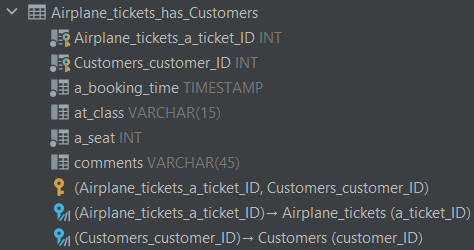


Рисунок 3.14 – Структура таблиці "Airplane\_tickets\_has\_Customers"

Створено таблицю "Bus\_tickets\_has\_Customers". Структура таблиці наведена на рисунку 3.15.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Bus\_tickets\_has\_Customers (

Bus\_tickets\_b\_ticket\_ID INT NOT NULL,

Customers\_customer\_ID INT NOT NULL,

b\_booking\_time TIMESTAMP NOT NULL,

b\_seat INT NOT NULL,

comments VARCHAR(45) NULL,

PRIMARY KEY (Bus\_tickets\_b\_ticket\_ID, Customers\_customer\_ID),

FOREIGN KEY (Bus\_tickets\_b\_ticket\_ID) REFERENCES Bus\_tickets (b\_ticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Customers\_customer\_ID) REFERENCES Customers (customer\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Bus\_tickets\_has\_Customers(bus\_tickets\_b\_ticket\_id, customers\_customer\_id, b\_booking\_time, b\_seat)

VALUES ( 1, 26, '2020-05-08 02:11:23', 1),

( 2, 27, '2020-10-06 21:24:28', 4),

( 3, 28, '2020-05-14 23:59:25', 7),

...

( 33, 29, '2020-08-12 08:28:02', 19),

( 34, 26, '2020-10-04 00:22:49', 1),

( 35, 29, '2020-01-25 05:00:16', 4);

COMMIT ;

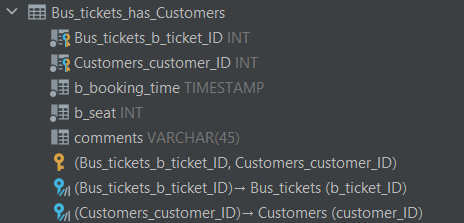


Рисунок 3.15 – Структура таблиці "Bus\_tickets\_has\_Customers"

Створено таблицю "Ship\_tickets\_has\_Customers". Структура таблиці наведена на рисунку 3.16.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Ship\_tickets\_has\_Customers (

Ship\_tickets\_s\_ticket\_ID INT NOT NULL,

Customers\_customer\_ID INT NOT NULL,

s\_booking\_time TIMESTAMP NOT NULL,

s\_class VARCHAR(15) NULL,

s\_seat INT NOT NULL,

comments VARCHAR(45) NULL,

PRIMARY KEY (Ship\_tickets\_s\_ticket\_ID, Customers\_customer\_ID),

FOREIGN KEY (Ship\_tickets\_s\_ticket\_ID) REFERENCES Ship\_tickets (s\_ticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Customers\_customer\_ID) REFERENCES Customers (customer\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Ship\_tickets\_has\_Customers(ship\_tickets\_s\_ticket\_id, customers\_customer\_id, s\_booking\_time, s\_seat, comments)

VALUES ( 1, 26, '2020-03-24 05:43:44', 23, NULL),

( 2, 27, '2020-02-25 13:56:49', 97, NULL),

( 3, 28, '2020-09-11 10:54:59', 45, NULL),

( 4, 29, '2020-11-19 02:10:09', 12, NULL),

( 5, 30, '2020-02-14 01:36:50', 88, NULL);

COMMIT ;

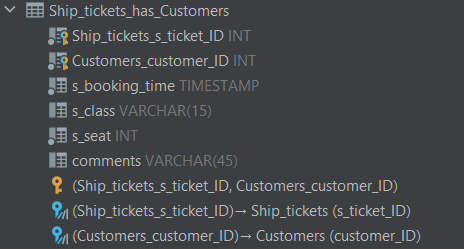


Рисунок 3.16 – Структура таблиці "Ship\_tickets\_has\_Customers"

Створено таблицю "Transport\_has\_Type\_tickets". Структура таблиці наведена на рисунку 3.17.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Transport\_has\_Type\_tickets (

Transport\_idTransport INT NOT NULL,

Type\_ticketa\_idType\_tickets INT NOT NULL,

count\_seats\_type INT NOT NULL,

PRIMARY KEY (Transport\_idTransport, Type\_ticketa\_idType\_tickets),

FOREIGN KEY (Transport\_idTransport) REFERENCES Transport (transport\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (Type\_ticketa\_idType\_tickets) REFERENCES Type\_tickets (typeticket\_ID)

ON DELETE CASCADE

ON UPDATE CASCADE);

Код заповнення таблиці:

BEGIN ;

INSERT INTO Transport\_has\_Type\_tickets(transport\_idtransport, type\_ticketa\_idtype\_tickets, count\_seats\_type)

VALUES (1, 7, 50),

(1, 8, 75),

...

(66,13, 50),

(66,14, 50);

COMMIT ;

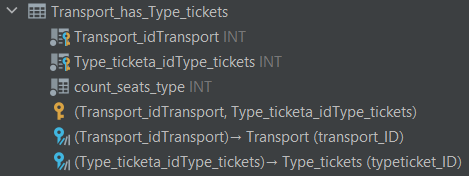


Рисунок 3.17 – Структура таблиці "Transport\_has\_Type\_tickets"

Створено таблицю "Transport". Структура таблиці наведена на рисунку 3.18.

Код створення таблиці:

CREATE TABLE IF NOT EXISTS Transport (

transport\_ID SERIAL NOT NULL,

name VARCHAR(45) NOT NULL,

number VARCHAR(10) NOT NULL,

countplace INT NOT NULL,

PRIMARY KEY (transport\_ID));

Код заповнення таблиці:

BEGIN ;

INSERT INTO Transport(name, number, countplace)

VALUES ( 'Boeing', '777', 250),

( 'Airbus', 'A340', 170),

( 'Airbus', 'A330', 125),

...

( 'Celebrity Xpedition', '', 5530),

( 'Forse le Carib', '', 570),

( 'Porco Fugo', '', 240);

COMMIT ;

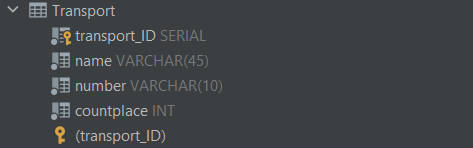


Рисунок 3.18 – Структура таблиці "Transport"

Вигляд таблиць подано у додатку Б.

**3.2 Створення тригерів**

CREATE TRIGGER створює новий тригер. Тригер буде пов'язаний з вказаною таблицею, представленням або сторонньою таблицею і буде виконувати задану функцію при певних подіях.

У результаті аналізу завдання на курсове проєктування було створено наступні тригери.

Тригер, який записує видалені дані з таблиці “Transpot” в таблицю “Del\_transport” (рисунок 3.19):

CREATE OR REPLACE FUNCTION insertInDel\_Transport() RETURNS trigger AS $saveDelTransport$

BEGIN

INSERT INTO Del\_transport(transport\_ID, name, number, countplace, time)

VALUES(OLD.transport\_ID, OLD.name, OLD.number, OLD.countplace, NOW());

RETURN NULL;

END;

$saveDelTransport$ LANGUAGE plpgsql;

CREATE TRIGGER saveDelTransport BEFORE DELETE ON Transport

FOR EACH ROW EXECUTE PROCEDURE insertInDel\_Transport();

Код створення допоміжної таблиці:

CREATE TABLE Del\_transport (LIKE Transport INCLUDING ALL);

ALTER TABLE Del\_transport ADD COLUMN time TIMESTAMP NULL;

Виконано тестове видалення для перевірки результату:

DELETE FROM Transport WHERE transport\_ID < 5;

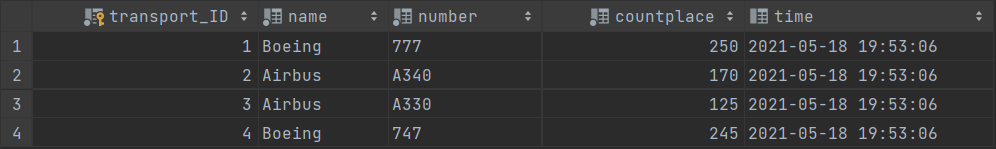


Рисунок 3.19 – Результат виконання тригера

Тригер, який обчислює приблизний час прибуття рейсу (рисунок 3.20):

CREATE OR REPLACE FUNCTION calc\_end\_time() RETURNS trigger AS $setEndTimeFlight$

BEGIN

CASE

WHEN NEW.idFirm <= 7

THEN NEW.endtime = NEW.starttime + to\_char(to\_timestamp((NEW.f\_distance/800.0) \* 60.0), 'MI:SS')::INTERVAL;

WHEN NEW.idFirm > 7 AND NEW.idFirm <= 11

THEN NEW.endtime = NEW.starttime + to\_char(to\_timestamp((NEW.f\_distance/50.0) \* 60.0), 'MI:SS')::INTERVAL;

WHEN NEW.idFirm > 11 AND NEW.idFirm <= 15

THEN NEW.endtime = NEW.starttime + to\_char(to\_timestamp((NEW.f\_distance/22.0) \* 60.0), 'MI:SS')::INTERVAL;

ELSE NEW.endtime = NEW.starttime + to\_char(to\_timestamp((NEW.f\_distance/65.0) \* 60.0), 'MI:SS')::INTERVAL;

END CASE;

RETURN NEW;

END;

$setEndTimeFlight$ LANGUAGE plpgsql;

CREATE TRIGGER setEndTimeFlight BEFORE INSERT ON Flights

FOR EACH ROW EXECUTE PROCEDURE calc\_end\_time();

Введено тестові дані для перевірки результату:

INSERT INTO Flights(idFirm, f\_from, f\_to, f\_distance, idTransport, starttime)

VALUES (8, 'Київ', 'Дніпро', 488, 51, '2021-10-24 21:22:46'),

(10, 'Херсон', 'Миколаїв', 80, 57, '2021-05-08 14:01:34');

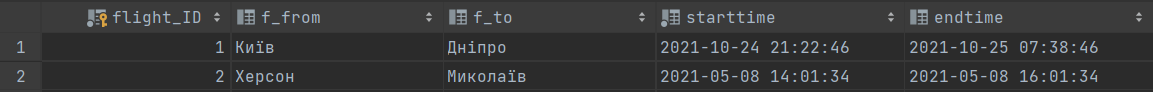


Рисунок 3.20 – Результат виконання тригера

Тригер, який виводить користувацьке повідомлення про помилку, якщо введено від’ємне значення відстані або цифри на місці міста відправлення чи прибуття (рисунок 3.21):

CREATE OR REPLACE FUNCTION validFlight() RETURNS trigger AS $checkDistance$

BEGIN

CASE

WHEN NEW.f\_distance <= 0

THEN RAISE 'Invalid distance!';

WHEN NEW.f\_from ~ '[0-9]' OR NEW.f\_to ~ '[0-9]'

THEN RAISE 'This string can`t include digits!';

ELSE RETURN NEW;

END CASE;

END;

$checkDistance$ LANGUAGE plpgsql;

CREATE TRIGGER checkDistance BEFORE INSERT ON Flights

FOR EACH ROW EXECUTE PROCEDURE validFlight();

Введено тестові дані для перевірки результату:

INSERT INTO Flights(idFirm, f\_from, f\_to, f\_distance, idTransport, starttime)

VALUES (8, 'Київ', 'Дніпро', -488, 51, '2021-10-24 21:22:46');

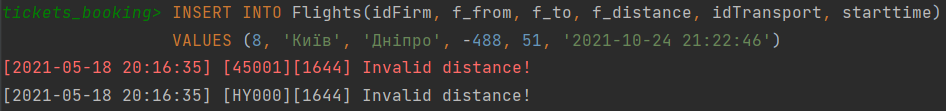


Рисунок 3.21 – Результат виконання тригера

Тригер, який встановлює статус рейсу відносно поточної дати та часу (рисунок 3.22):

CREATE OR REPLACE FUNCTION setStatus() RETURNS trigger AS $checkStatus$

BEGIN

CASE

WHEN CURRENT\_TIMESTAMP < NEW.endtime AND OLD.status = 'Відбувся'

THEN NEW.status = 'Очікується';

WHEN NEW.endtime < CURRENT\_TIMESTAMP AND OLD.status = 'Очікується'

THEN NEW.status = 'Відбувся';

ELSE RETURN NEW;

END CASE;

RETURN NEW;

END;

$checkStatus$ LANGUAGE plpgsql;

CREATE TRIGGER checkStatus BEFORE UPDATE OR INSERT ON Flights

FOR EACH ROW EXECUTE PROCEDURE setStatus();

Введено тестові дані для перевірки результату:

UPDATE Flights SET endtime = '2021-01-08 22:00:00' WHERE flight\_ID = 6;



Рисунок 3.22 – Результат виконання тригера

Тригер, який встановлює вартість авіаквитка (рисунок 3.23):

CREATE FUNCTION generate\_air\_ticket\_price() RETURNS trigger AS $setPriceAirTicket$

BEGIN

NEW.at\_price = 4.0 \* (SELECT f\_distance FROM Flights INNER JOIN Flights\_has\_Cities FhC

on Flights.flight\_ID = FhC.idFlight WHERE primaryID = NEW.Flights\_has\_Cities\_primaryID) +

((SELECT f\_distance FROM Flights INNER JOIN Flights\_has\_Cities FhC

on Flights.flight\_ID = FhC.idFlight WHERE idFlight = NEW.Flights\_has\_Cities\_primaryID) \*

(SELECT percent\_afterpay/100.0 FROM Type\_tickets

WHERE NEW.idType\_tickets = Type\_tickets.typeticket\_ID));

RETURN NEW;

END;

$setPriceAirTicket$ LANGUAGE plpgsql;

CREATE TRIGGER setPriceAirTicket BEFORE INSERT ON Airplane\_tickets

FOR EACH ROW EXECUTE PROCEDURE generate\_air\_ticket\_price();

Введено тестові дані для перевірки результату:

INSERT INTO Airplane\_tickets(idType\_tickets, Flights\_has\_Cities\_primaryID)

VALUES (7, 10),

(9, 50),

(9, 13);

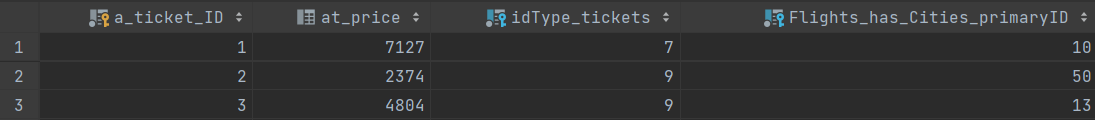


Рисунок 3.23 – Результат виконання тригера

Тригер, який визначає в якому класі знаходиться місце пасажира за його квитком (рисунок 3.24):

CREATE FUNCTION set\_air\_class() RETURNS trigger AS $setClassAirHasCus$

BEGIN

CASE

WHEN (SELECT idType\_tickets FROM Airplane\_tickets WHERE NEW.Airplane\_tickets\_a\_ticket\_ID = a\_ticket\_ID) = 7

THEN NEW.at\_class = 'Бізнес';

WHEN (SELECT idType\_tickets FROM Airplane\_tickets WHERE NEW.Airplane\_tickets\_a\_ticket\_ID = a\_ticket\_ID) = 8

THEN NEW.at\_class = 'Економ';

WHEN (SELECT idType\_tickets FROM Airplane\_tickets WHERE NEW.Airplane\_tickets\_a\_ticket\_ID = a\_ticket\_ID) = 9

THEN NEW.at\_class = 'Перший';

END CASE;

RETURN NEW;

END;

$setClassAirHasCus$ LANGUAGE plpgsql;

CREATE TRIGGER setClassAirHasCus BEFORE INSERT ON Airplane\_tickets\_has\_Customers

FOR EACH ROW EXECUTE PROCEDURE set\_air\_class();

Введено тестові дані для перевірки результату:

INSERT INTO Airplane\_tickets\_has\_Customers(Airplane\_tickets\_a\_ticket\_ID, Customers\_customer\_ID, a\_booking\_time, a\_seat, comments)

VALUES ( 1, 1, '2020-10-09 20:21:36', 1, NULL),

( 2, 2, '2020-01-14 09:55:01', 2, NULL);

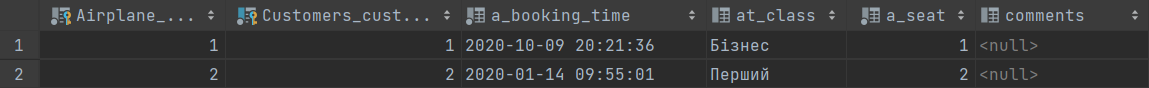


Рисунок 3.24 – Результат виконання тригера

Тригер, який встановлює вартість квитка на водні види транспорту (рисунок 3.25):

CREATE FUNCTION generate\_ship\_ticket\_price() RETURNS trigger AS $setPriceShipTicket$

BEGIN

NEW.st\_price = 5.0 \* (SELECT f\_distance FROM Flights INNER JOIN Flights\_has\_Cities FhC

on Flights.flight\_ID = FhC.idFlight WHERE primaryID = NEW.Flights\_has\_Cities\_primaryID) +

((SELECT f\_distance FROM Flights INNER JOIN Flights\_has\_Cities FhC

on Flights.flight\_ID = FhC.idFlight WHERE idFlight = NEW.Flights\_has\_Cities\_primaryID) \*

(SELECT percent\_afterpay/100.0 FROM Type\_tickets

WHERE NEW.idType\_tickets = Type\_tickets.typeticket\_ID));

RETURN NEW;

END;

$setPriceShipTicket$ LANGUAGE plpgsql;

CREATE TRIGGER setPriceShipTicket BEFORE INSERT ON Ship\_tickets

FOR EACH ROW EXECUTE PROCEDURE generate\_ship\_ticket\_price();

Введено тестові дані для перевірки результату:

INSERT INTO Ship\_tickets(idType\_tickets, Flights\_has\_Cities\_primaryID)

VALUES ( 12, 32),

( 13, 32);

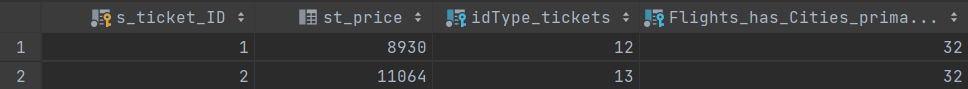


Рисунок 3.25 – Результат виконання тригера

Тригер, який визначає в якому класі знаходиться місце пасажира за його квитком (рисунок 3.26):

CREATE FUNCTION set\_ship\_class() RETURNS trigger AS $setClassShipHasCus$

BEGIN

CASE

WHEN (SELECT idType\_tickets

FROM Ship\_tickets

WHERE NEW.Ship\_tickets\_s\_ticket\_ID = Ship\_tickets.s\_ticket\_ID) = 12

THEN NEW.s\_class = 'Стандарт';

WHEN (SELECT idType\_tickets

FROM Ship\_tickets

WHERE NEW.Ship\_tickets\_s\_ticket\_ID = Ship\_tickets.s\_ticket\_ID) = 13

THEN NEW.s\_class = 'Преміум';

WHEN (SELECT idType\_tickets

FROM Ship\_tickets

WHERE NEW.Ship\_tickets\_s\_ticket\_ID = Ship\_tickets.s\_ticket\_ID) = 14

THEN NEW.s\_class = 'Економ';

END CASE;

RETURN NEW;

END;

$setClassShipHasCus$ LANGUAGE plpgsql;

CREATE TRIGGER setClassShipHasCus BEFORE INSERT ON Ship\_tickets\_has\_Customers

FOR EACH ROW EXECUTE PROCEDURE set\_ship\_class();

Введено тестові дані для перевірки результату:

INSERT INTO Ship\_tickets\_has\_Customers(ship\_tickets\_s\_ticket\_id, customers\_customer\_id, s\_booking\_time, s\_seat, comments)

VALUES ( 1, 26, '2020-03-24 05:43:44', 23, NULL),

( 2, 27, '2020-02-25 13:56:49', 97, NULL);

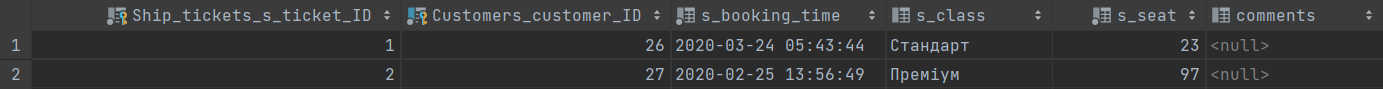


Рисунок 3.26 – Результат виконання тригера

Тригер, який викликає процедуру, що встановлює вартість автобусного квитка (рисунок 3.27):

CREATE FUNCTION generate\_bus\_ticket\_price() RETURNS trigger AS $setPriceBusTicket$

BEGIN

NEW.bt\_price = NEW.bt\_price + NEW.bt\_price\*(SELECT percent\_afterpay/100.0

FROM Type\_tickets WHERE typeticket\_ID = NEW.idType\_tickets);

RETURN NEW;

END;

$setPriceBusTicket$ LANGUAGE plpgsql;

CREATE TRIGGER setPriceBusTicket BEFORE INSERT ON Bus\_tickets

FOR EACH ROW EXECUTE PROCEDURE generate\_bus\_ticket\_price();

Введено тестові дані для перевірки результату:

INSERT INTO Bus\_tickets(bt\_price, idType\_tickets, idRoute)

VALUES ( 318, 10, 1),

( 212, 10, 2);

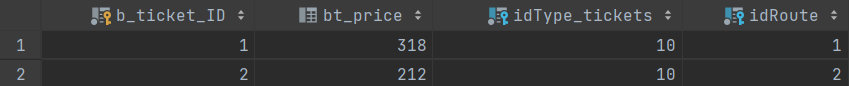


Рисунок 3.27 – Результат виконання тригера

Тригер, який встановлює вартість квитка на поїзд (рисунок 3.28):

CREATE FUNCTION generate\_train\_ticket\_price() RETURNS trigger AS $setPriceTrainTicket$

BEGIN

NEW.tt\_price = NEW.tt\_price/2.0 + (NEW.tt\_price/2.0)\*(SELECT percent\_afterpay/100.0

FROM Type\_tickets WHERE typeticket\_ID = NEW.idType\_tickets);

RETURN NEW;

END;

$setPriceTrainTicket$ LANGUAGE plpgsql;

CREATE TRIGGER setPriceTrainTicket BEFORE INSERT ON Train\_tickets

FOR EACH ROW EXECUTE PROCEDURE generate\_train\_ticket\_price();

Введено тестові дані для перевірки результату:

INSERT INTO Train\_tickets( tt\_price, idType\_tickets, idRoute)

VALUES ( 394, 1, 31),

( 554, 1, 32);

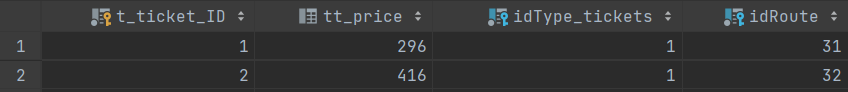


Рисунок 3.28 – Результат виконання тригера

**3.3 Створення запитів**

PostgreSQL надає можливість використовувати оператор WITH у запитах. WITH надає спосіб записувати додаткові оператори для застосування у великих запитах. Ці оператори, які також називають загальними табличними виразами (Common Table Expressions, CTE), можна уявити як визначення тимчасових таблиць, які існують лише для одного запиту. Основне призначення SELECT в реченні WITH полягає в розбитті складних запитів на прості частини [9].

У результаті аналізу завдання на курсове проєктування було створено наступні запити, які відображають усі можливі дії з даними, що містяться у таблицях.

Запит, який формує список фірми, що проводять тільки закордонні рейси та обчислює їхню кількість (рисунок 3.29):

SELECT name, COUNT(flight\_ID) AS Foreign\_flights

FROM Flights INNER JOIN Firms F

ON Flights.idFirm = F.firm\_ID INNER JOIN Flights\_has\_Cities FhC

ON Flights.flight\_ID = FhC.idFlight INNER JOIN Cities C

ON C.city\_ID = FhC.idCity2

WHERE country <> 'Україна' AND NOT EXISTS( SELECT COUNT(flight\_ID)

FROM Flights INNER JOIN Firms Fi

ON Flights.idFirm = Fi.firm\_ID INNER JOIN Flights\_has\_Cities FhC

ON Flights.flight\_ID = FhC.idFlight INNER JOIN Cities C

ON C.city\_ID = FhC.idCity2

WHERE country = 'Україна' AND F.name = Fi.name

HAVING COUNT(flight\_ID) <> 0)

GROUP BY 1;

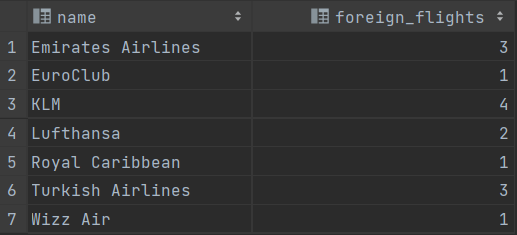


Рисунок 3.29 – Результат виконання запиту

Запит, який виводить ініціали всіх пасажирів і вартості їхніх квитків на авіа, автобуні, залізничні та морські рейси (рисунок 3.30):

SELECT customer\_ID, c\_name, at\_price, Bt.bt\_price, Tt.tt\_price, st\_price

FROM Customers LEFT JOIN Airplane\_tickets\_has\_Customers AthC

ON Customers.customer\_ID = AthC.Customers\_customer\_ID LEFT JOIN Airplane\_tickets A

ON A.a\_ticket\_ID = AthC.Airplane\_tickets\_a\_ticket\_ID LEFT JOIN Bus\_tickets\_has\_Customers BthC

ON Customers.customer\_ID = BthC.Customers\_customer\_ID LEFT JOIN Bus\_tickets Bt

ON Bt.b\_ticket\_ID = BthC.Bus\_tickets\_b\_ticket\_ID LEFT JOIN Train\_tickets\_has\_Customers TthC

ON Customers.customer\_ID = TthC.Customers\_customer\_ID LEFT JOIN Train\_tickets Tt

ON Tt.t\_ticket\_ID = TthC.Train\_tickets\_t\_ticket\_ID LEFT JOIN Ship\_tickets\_has\_Customers SthC

ON Customers.customer\_ID = SthC.Customers\_customer\_ID LEFT JOIN Ship\_tickets S

ON S.s\_ticket\_ID = SthC.Ship\_tickets\_s\_ticket\_ID LEFT JOIN Flights\_has\_Cities FhC

ON A.Flights\_has\_Cities\_primaryID = FhC.primaryID LEFT JOIN Routes R

ON FhC.primaryID = R.Flights\_has\_Cities\_primaryID LEFT JOIN Bus\_tickets B

ON R.route\_ID = B.idRoute LEFT JOIN Train\_tickets T

ON R.route\_ID = T.idRoute;

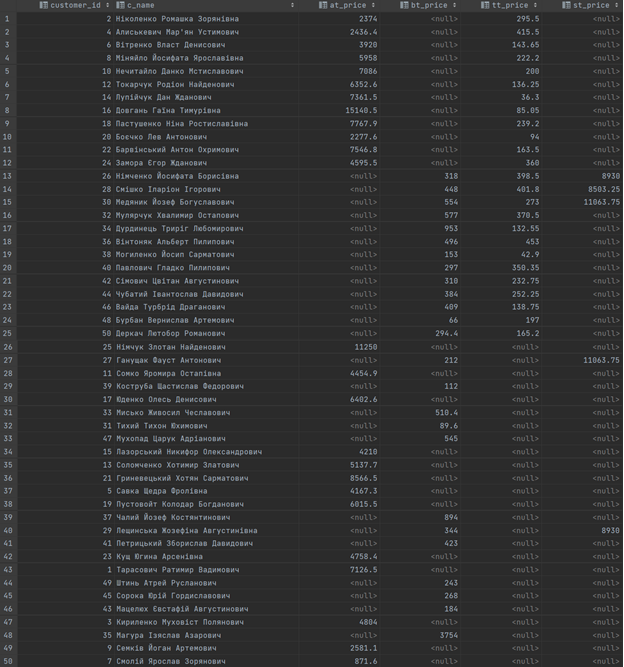


Рисунок 3.30 – Результат виконання запиту

Запит, який виводить всі можливі автобусні та залізничні маршрути до Києва з різних міст України (рисунок 3.31):

SELECT flight\_ID, route\_ID, f\_from, city\_name,idCity

FROM Flights INNER JOIN Flights\_has\_Cities FhC

ON Flights.flight\_ID = FhC.idFlight INNER JOIN Routes R

ON FhC.primaryID = R.Flights\_has\_Cities\_primaryID INNER JOIN Cities C

ON C.city\_ID = R.idCity

WHERE city\_name = 'Київ' AND country = 'Україна';

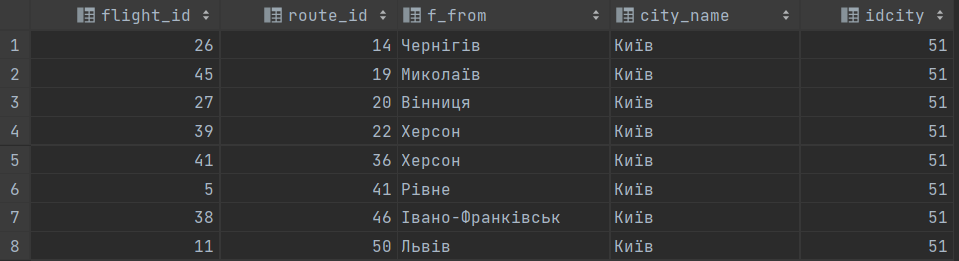


Рисунок 3.31 – Результат виконання запиту

Запит, який виводить фірми, що займаються повітряним або наземним перевезеннями та кількість рейсів яких більше трьох і водночас менше тринадцяти (рисунок 3.32):

SELECT F.name AS firm\_name, type\_name AS transition, COUNT(flight\_ID) AS number\_of\_flights

FROM Flights INNER JOIN Firms F

ON F.firm\_ID = Flights.idFirm INNER JOIN Type\_transportations Tt

ON Tt.transportation\_ID = F.idTransportation

WHERE type\_name IN ('Наземне', 'Повітряне')

GROUP BY 1, 2

HAVING COUNT(flight\_ID) BETWEEN 3 AND 13;

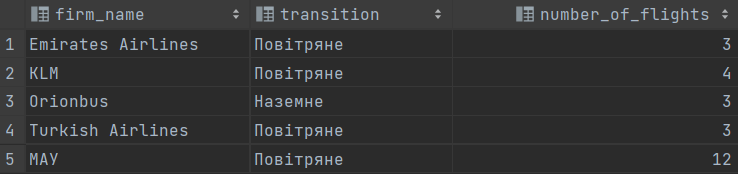


Рисунок 3.32 – Результат виконання запиту

Запит, що виводить список авіаквитків зарезервованих протягом останніх шести місяців (рисунок 3.33):

SELECT a\_booking\_time, at\_price, at\_class, a\_seat

FROM Airplane\_tickets INNER JOIN Airplane\_tickets\_has\_Customers AthC

ON Airplane\_tickets.a\_ticket\_ID = AthC.Airplane\_tickets\_a\_ticket\_ID

WHERE CURRENT\_TIMESTAMP - INTERVAL '6 MONTH' < a\_booking\_time;

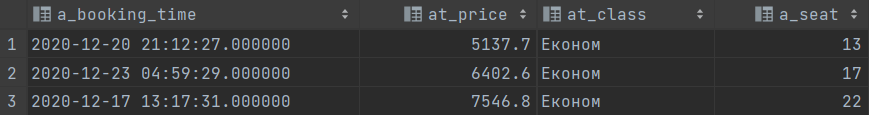


Рисунок 3.33 – Результат виконання запиту

Запит, що визначає тривалість рейсів, які виходять за межі України (рисунок 3.34):

SELECT CONCAT(f\_from, ' - ', f\_to) AS Flight,

CONCAT(ROUND((EXTRACT(EPOCH FROM (endtime - starttime)) / 3600.00)::NUMERIC, 1), ' год.') AS Hours,

CASE

WHEN EXTRACT(EPOCH FROM (endtime - starttime)) / 3600.00 <= 3.0

THEN 'Швидкий'

WHEN EXTRACT(EPOCH FROM (endtime - starttime)) / 3600.00 <= 9.0

THEN 'Середній'

ELSE 'Тривалий'

END AS Тривалість\_рейсу

FROM Flights INNER JOIN Flights\_has\_Cities FhC

ON Flights.flight\_ID = FhC.idFlight INNER JOIN Cities C

ON C.city\_ID = FhC.idCity2

WHERE country <> 'Україна'

ORDER BY EXTRACT(EPOCH FROM (endtime - starttime)) / 3600.00 DESC;



Рисунок 3.34 – Результат виконання запиту

Запит, який виводить список транспортних засобів та загальну кількість кожного виду квитків (рисунок 3.35):

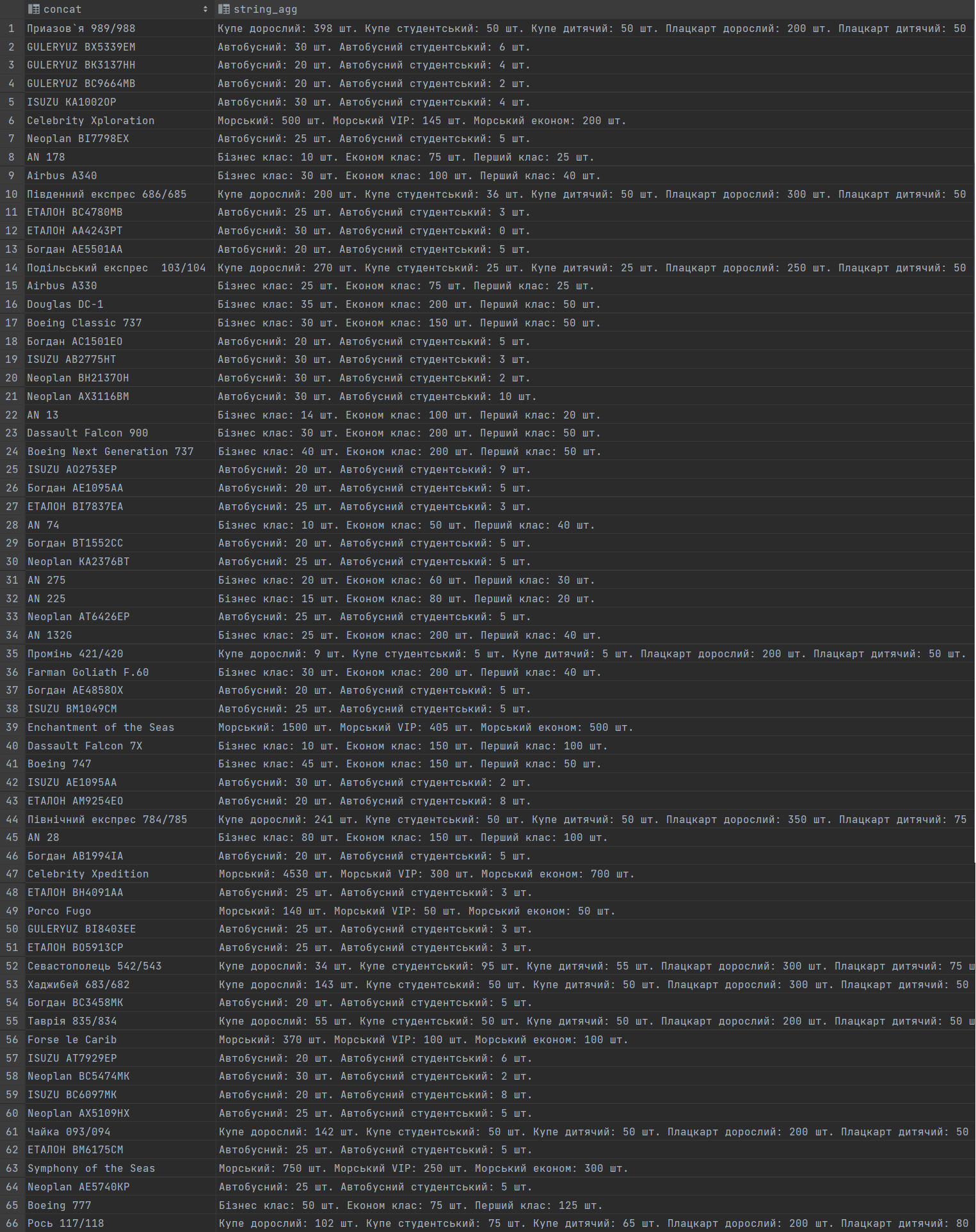
SELECT CONCAT(name, ' ', number), STRING\_AGG(CONCAT(type\_afterpay, ': ', count\_seats\_type, ' шт.'), ' ')

FROM Transport INNER JOIN Transport\_has\_Type\_tickets ThTt

ON Transport.transport\_ID = ThTt.Transport\_idTransport INNER JOIN Type\_tickets Tt

ON Tt.typeticket\_ID = ThTt.Type\_ticketa\_idType\_tickets

GROUP BY 1;

Рисунок 3.35 – Результат виконання запиту

Запит, який виводить три найзбитковіші атобусні рейси (рисунок 3.36):

SELECT flight\_ID, CONCAT(f\_from, ' - ', f\_to) AS Flight, starttime::TIMESTAMP(0), endtime::TIMESTAMP(0),

CONCAT(T.name, ' ', T.number) AS Bus\_and\_number,

(SELECT COUNT(Bus\_tickets\_b\_ticket\_ID)

FROM Bus\_tickets\_has\_Customers INNER JOIN Bus\_tickets Bt

ON Bus\_tickets\_has\_Customers.Bus\_tickets\_b\_ticket\_ID = Bt.b\_ticket\_ID

INNER JOIN Routes R ON Bt.idRoute = R.route\_ID INNER JOIN Flights\_has\_Cities FhC

ON R.Flights\_has\_Cities\_primaryID = FhC.primaryID

WHERE flight\_ID = idFlight) AS Sold\_tickets

FROM Flights INNER JOIN Firms F

ON Flights.idFirm = F.firm\_ID INNER JOIN Type\_transportations Tt

ON F.idTransportation = Tt.transportation\_ID INNER JOIN Transport T

ON Flights.idTransport = T.transport\_ID

WHERE type\_name = 'Наземне'

ORDER BY 6

LIMIT 3;

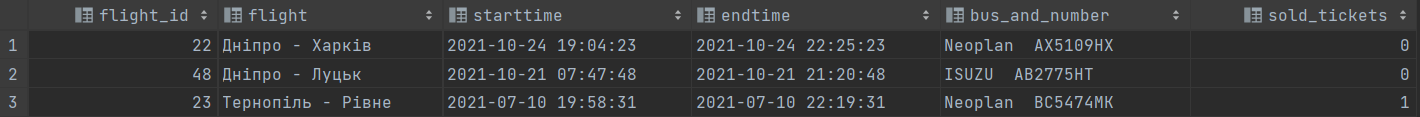


Рисунок 3.36 – Результат виконання запиту

Запит, який виводить пасажирів, що витратили найбільше грошей для покупку квитків на будь-який вид перевезень (рисунок 3.37):

SELECT customer\_ID, c\_name, LPAD(CONCAT(ROUND(COALESCE(at\_price, 0) + COALESCE(Bt.bt\_price, 0) +

COALESCE(Tt.tt\_price, 0) + COALESCE(st\_price, 0), 2), ' грн.'), 18, ' ') AS total

FROM Customers LEFT JOIN Airplane\_tickets\_has\_Customers AthC

ON Customers.customer\_ID = AthC.Customers\_customer\_ID LEFT JOIN Airplane\_tickets A

ON A.a\_ticket\_ID = AthC.Airplane\_tickets\_a\_ticket\_ID LEFT JOIN Bus\_tickets\_has\_Customers BthC

ON Customers.customer\_ID = BthC.Customers\_customer\_ID LEFT JOIN Bus\_tickets Bt

ON Bt.b\_ticket\_ID = BthC.Bus\_tickets\_b\_ticket\_ID LEFT JOIN Train\_tickets\_has\_Customers TthC

ON Customers.customer\_ID = TthC.Customers\_customer\_ID LEFT JOIN Train\_tickets Tt

ON Tt.t\_ticket\_ID = TthC.Train\_tickets\_t\_ticket\_ID LEFT JOIN Ship\_tickets\_has\_Customers SthC

ON Customers.customer\_ID = SthC.Customers\_customer\_ID LEFT JOIN Ship\_tickets S

ON S.s\_ticket\_ID = SthC.Ship\_tickets\_s\_ticket\_ID LEFT JOIN Flights\_has\_Cities FhC

ON A.Flights\_has\_Cities\_primaryID = FhC.primaryID LEFT JOIN Routes R

ON FhC.primaryID = R.Flights\_has\_Cities\_primaryID LEFT JOIN Bus\_tickets B

ON R.route\_ID = B.idRoute LEFT JOIN Train\_tickets T

ON R.route\_ID = T.idRoute

ORDER BY 3 DESC

LIMIT 10;

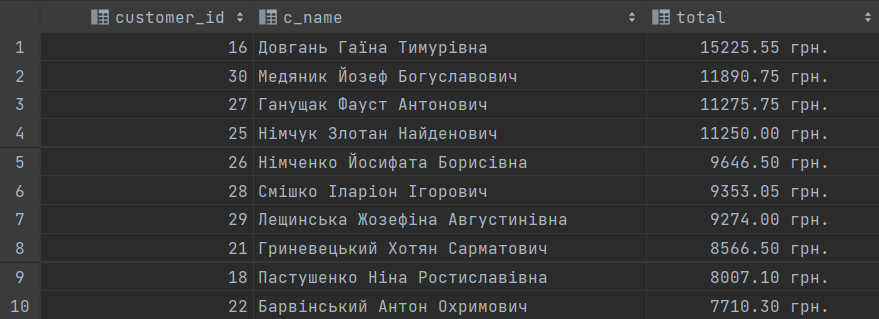


Рисунок 3.37 – Результат виконання запиту

Запит, який виводить кількість маршрутів на кожному рейсі (рисунок 3.38):

SELECT type\_name AS Transportation, CONCAT(f\_from, ' - ', f\_to) AS Flight, COUNT(route\_ID) AS number\_of\_routes,

STRING\_AGG(CONCAT(f\_from, ' - ', city\_name), ', ')

FROM Flights INNER JOIN Flights\_has\_Cities

FhC

ON Flights.flight\_ID = FhC.idFlight INNER JOIN Routes R

ON FhC.primaryID = R.Flights\_has\_Cities\_primaryID LEFT JOIN

Bus\_tickets Bt

ON R.route\_ID = Bt.idRoute LEFT JOIN Train\_tickets Tt

ON R.route\_ID = Tt.idRoute INNER JOIN Firms F

ON F.firm\_ID = Flights.idFirm INNER JOIN Type\_transportations T

ON T.transportation\_ID = F.idTransportation

INNER JOIN Cities C

ON C.city\_ID = R.idCity

GROUP BY 1, 2

HAVING 1 < COUNT(route\_ID)

ORDER BY 3 DESC;

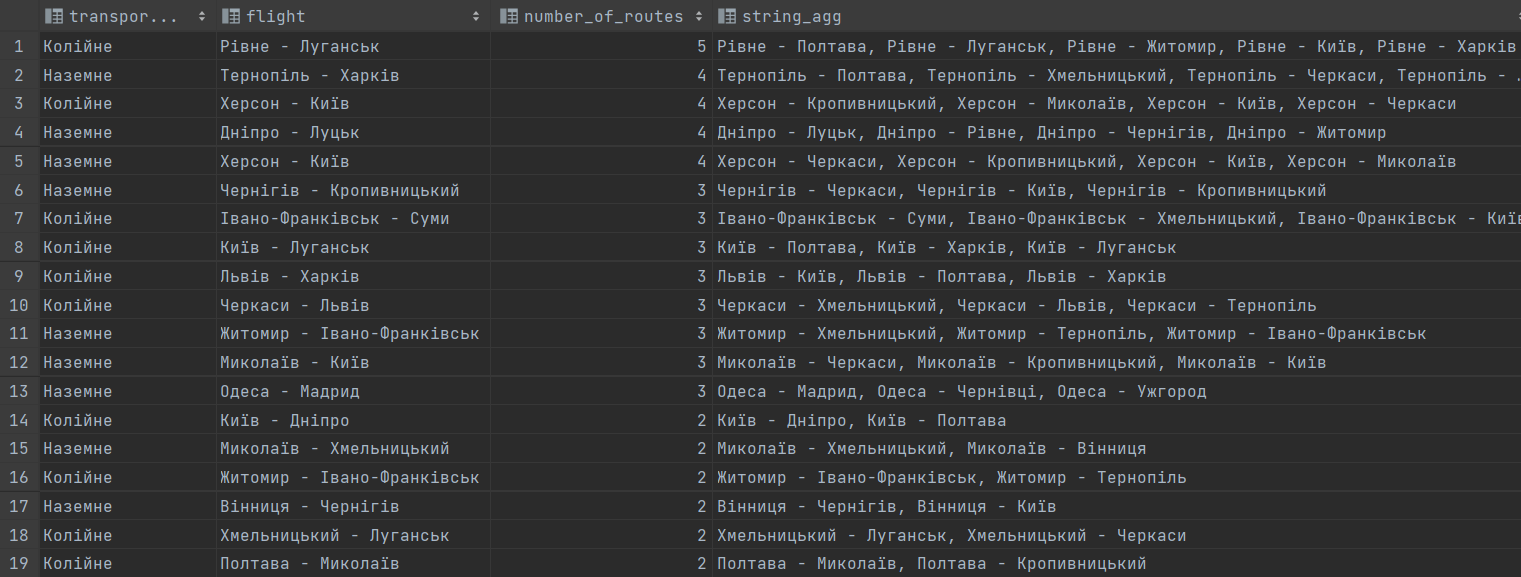


Рисунок 3.38 – Результат виконання запиту

**3.4 Створення представлень**

Представлення в PostgreSQL реалізовані на основі системи правил. Фактично по суті немає ніякої відмінності

CREATE VIEW myview AS SELECT \* FROM mytab;

від наступних двох команд:

CREATE TABLE myview ( same column list as mytab);

CREATE RULE "\_RETURN" AS ON SELECT TO myview DO INSTEAD

SELECT \* FROM mytab;

так як саме ці дії CREATE VIEW виконує всередині. Це має деякі побічні ефекти. Зокрема, інформація про подання в системних каталогах PostgreSQL нічим не відрізняється від інформації про таблиці. Тому при аналізі запиту немає абсолютно ніякої різниці між таблицями і представленнями. Вони являють собою одне й те ж - відношення.

У відповідності до поставленого завдання на курсове проєктування було створено наступні представлення.

Представлення, яке містить інформацію про фірми, що проводять тільки закордонні рейси та кількість цих рейсів (рисунок 3.39):

CREATE VIEW foreign\_firms AS SELECT firm\_ID, name, COUNT(flight\_ID) AS amount\_of\_flights

FROM Flights INNER JOIN Firms F

ON Flights.idFirm = F.firm\_ID INNER JOIN Flights\_has\_Cities FhC

ON Flights.flight\_ID = FhC.idFlight INNER JOIN Cities C

ON C.city\_ID = FhC.idCity2

WHERE country <> 'Україна' AND NOT EXISTS( SELECT COUNT(flight\_ID)

FROM Flights INNER JOIN Firms Fi

ON Flights.idFirm = Fi.firm\_ID INNER JOIN Flights\_has\_Cities FhC

ON Flights.flight\_ID = FhC.idFlight INNER JOIN Cities C

ON C.city\_ID = FhC.idCity2

WHERE country = 'Україна' AND F.name = Fi.name

HAVING COUNT(flight\_ID) <> 0)

GROUP BY 1;

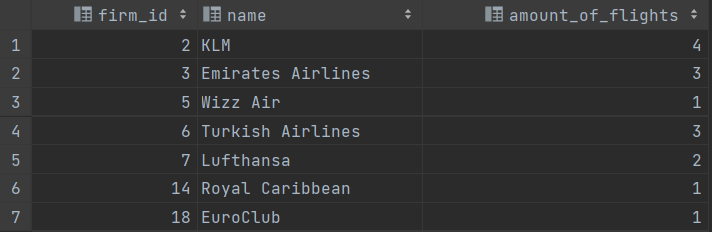


Рисунок 3.39 – Результат виконання представлення

Представлення, яке об`єднує найважливішу інформацію про авіаквитки (рисунок 3.40):

CREATE VIEW tickets\_between\_date AS SELECT c\_name AS Name, CONCAT(f\_from, ' - ', f\_to) AS Flight,

CONCAT(starttime, ' - ', endtime) AS Time, at\_class AS Class, a\_seat AS Seat,

at\_price AS Price, a\_booking\_time::TIMESTAMP(0) AS Booking

FROM Customers LEFT JOIN Airplane\_tickets\_has\_Customers AthC

ON Customers.customer\_ID = AthC.Customers\_customer\_ID LEFT JOIN Airplane\_tickets A

ON A.a\_ticket\_ID = AthC.Airplane\_tickets\_a\_ticket\_ID INNER JOIN Flights\_has\_Cities FhC

ON A.Flights\_has\_Cities\_primaryID = FhC.primaryID INNER JOIN Flights F

ON FhC.idFlight = F.flight\_ID INNER JOIN Firms F2

ON F.idFirm = F2.firm\_ID INNER JOIN Type\_transportations Tt

ON F2.idTransportation = Tt.transportation\_ID

WHERE type\_name = 'Повітряне';

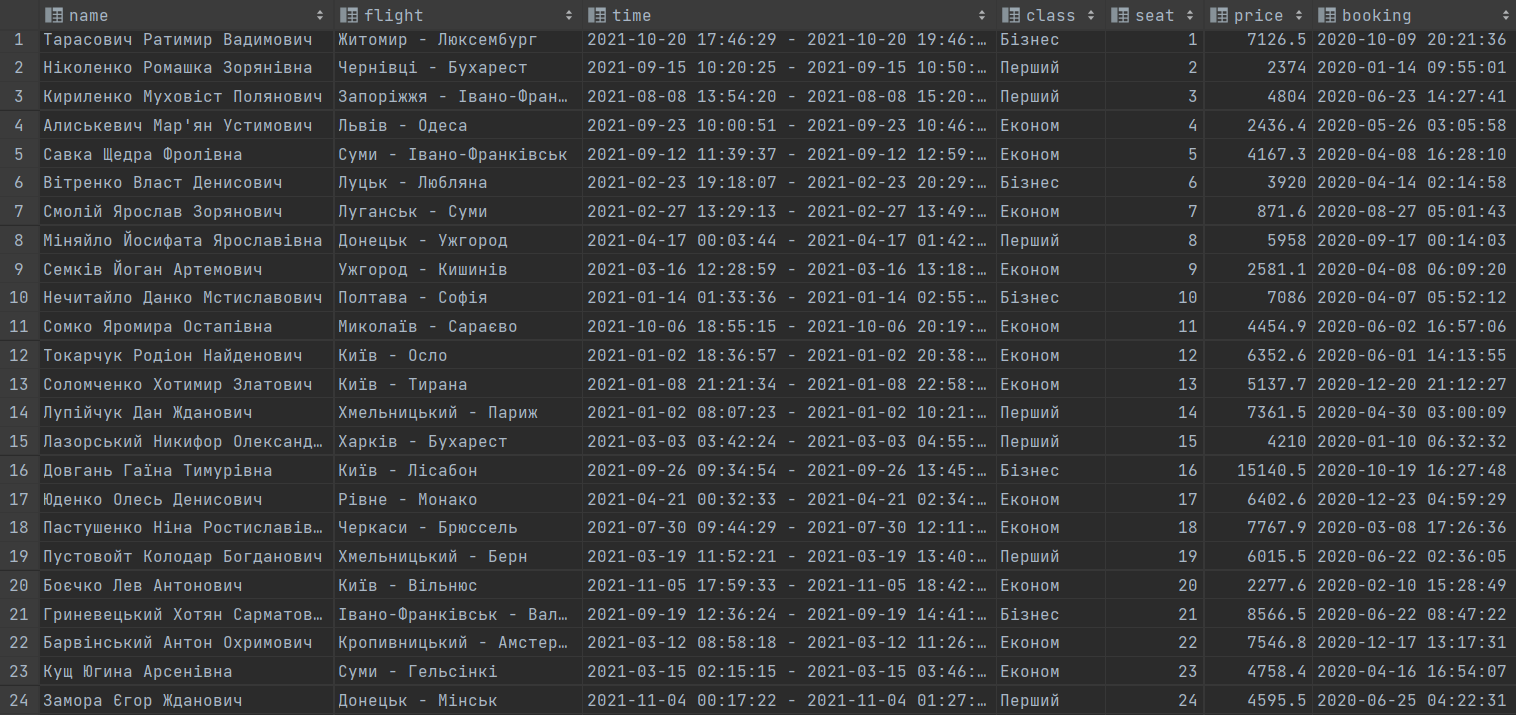


Рисунок 3.40 – Результат виконання представлення

Представлення, яке зберігає загальну суму витрачених коштів кожним пасажиром (рисунок 3.41):

CREATE VIEW customer\_money AS SELECT customer\_ID, c\_name, LPAD(CONCAT(ROUND(COALESCE(at\_price, 0) +

COALESCE(Bt.bt\_price, 0) + COALESCE(Tt.tt\_price, 0) +

COALESCE(st\_price, 0), 2), ' грн.'), 18, ' ') AS total

FROM Customers LEFT JOIN Airplane\_tickets\_has\_Customers AthC

ON Customers.customer\_ID = AthC.Customers\_customer\_ID LEFT JOIN Airplane\_tickets A

ON A.a\_ticket\_ID = AthC.Airplane\_tickets\_a\_ticket\_ID LEFT JOIN Bus\_tickets\_has\_Customers BthC

ON Customers.customer\_ID = BthC.Customers\_customer\_ID LEFT JOIN Bus\_tickets Bt

ON Bt.b\_ticket\_ID = BthC.Bus\_tickets\_b\_ticket\_ID LEFT JOIN Train\_tickets\_has\_Customers TthC

ON Customers.customer\_ID = TthC.Customers\_customer\_ID LEFT JOIN Train\_tickets Tt

ON Tt.t\_ticket\_ID = TthC.Train\_tickets\_t\_ticket\_ID LEFT JOIN Ship\_tickets\_has\_Customers SthC

ON Customers.customer\_ID = SthC.Customers\_customer\_ID LEFT JOIN Ship\_tickets S

ON S.s\_ticket\_ID = SthC.Ship\_tickets\_s\_ticket\_ID LEFT JOIN Flights\_has\_Cities FhC

ON A.Flights\_has\_Cities\_primaryID = FhC.primaryID LEFT JOIN Routes R

ON FhC.primaryID = R.Flights\_has\_Cities\_primaryID LEFT JOIN Bus\_tickets B

ON R.route\_ID = B.idRoute LEFT JOIN Train\_tickets T

ON R.route\_ID = T.idRoute

ORDER BY 3 DESC;



Рисунок 3.41 – Результат виконання представлення

Представлення, яке зберігає список маршрутів на кожному рейсі (рисунок 3.42):

CREATE VIEW routes\_on\_flight AS SELECT CONCAT(f\_from, ' - ', f\_to) AS Flight, COUNT(route\_ID) AS number\_of\_routes,

STRING\_AGG(CONCAT(f\_from, ' - ', city\_name), ', ')

FROM Flights INNER JOIN Flights\_has\_Cities FhC

ON Flights.flight\_ID = FhC.idFlight INNER JOIN Routes R

ON FhC.primaryID = R.Flights\_has\_Cities\_primaryID LEFT JOIN Bus\_tickets Bt

ON R.route\_ID = Bt.idRoute LEFT JOIN Train\_tickets Tt

ON R.route\_ID = Tt.idRoute INNER JOIN Firms F

ON F.firm\_ID = Flights.idFirm INNER JOIN Type\_transportations T

ON T.transportation\_ID = F.idTransportation INNER JOIN Cities C

ON C.city\_ID = R.idCity

GROUP BY 1

ORDER BY 2 DESC;



Рисунок 3.42 – Результат виконання представлення

Представлення, яке зберігає інформацію по кількість квитків кожного типу на будь-який транспортний засіб (рисунок 3.43):

CREATE VIEW transport\_tickets AS SELECT CONCAT(name, ' ', number), STRING\_AGG(CONCAT(type\_afterpay, ': ', count\_seats\_type, ' шт.'), ' ')

FROM Transport INNER JOIN Transport\_has\_Type\_tickets ThTt

ON Transport.transport\_ID = ThTt.Transport\_idTransport INNER JOIN Type\_tickets Tt

ON Tt.typeticket\_ID = ThTt.Type\_ticketa\_idType\_tickets

GROUP BY 1;

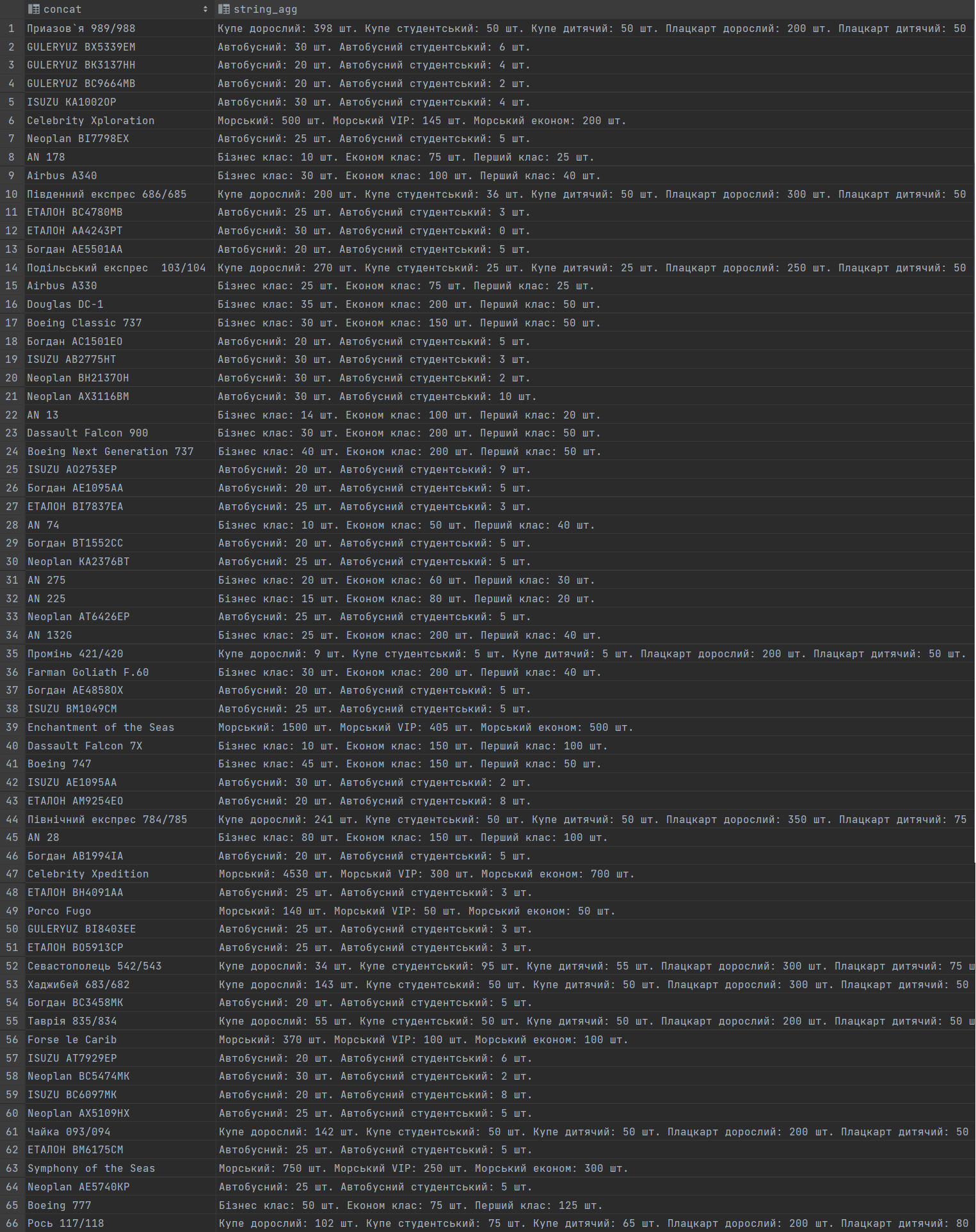


Рисунок 3.43 – Результат виконання представлення

**3.5 Створення процедур та функцій**

PostgreSQL допускає перевантаження функцій; тобто, дозволяє використовувати одне ім'я для кількох різних функцій, якщо у них різняться типи вхідних аргументів.

У відповідності до поставленого завдання на курсове проєктування було створено наступні процедури та функції.

Функція, яка повертає різницю між датами в днях (рисунок 3.44):

CREATE OR REPLACE FUNCTION DATEDIFF(timestamp, timestamp)

RETURNS INTEGER

AS $$

SELECT ABS(CAST($1 AS date) - CAST($2 AS date)) as DateDifference

$$ LANGUAGE SQL IMMUTABLE;

SELECT DATEDIFF('2000-01-01 00:00:00','2000-01-15 00:00:00');



Рисунок 3.44 – Результат виконання функції

Функція, яка виконує пошук авіквитків за ПІБ пасажира (рисунок 3.45):

CREATE OR REPLACE FUNCTION search\_customer\_tickets(search\_name TEXT)

RETURNS SETOF text AS $$

DECLARE

result TEXT DEFAULT '';

records RECORD;

myCursor CURSOR(search\_name TEXT)

FOR SELECT c\_name AS Name, CONCAT(f\_from, ' - ', f\_to) AS Flight,

CONCAT(starttime, ' - ', endtime) AS Time, at\_class AS Class, a\_seat AS Seat,

at\_price AS Price, a\_booking\_time::TIMESTAMP(0) AS Booking

FROM Customers LEFT JOIN Airplane\_tickets\_has\_Customers AthC

ON Customers.customer\_ID = AthC.Customers\_customer\_ID LEFT JOIN Airplane\_tickets A

ON A.a\_ticket\_ID = AthC.Airplane\_tickets\_a\_ticket\_ID INNER JOIN Flights\_has\_Cities FhC

ON A.Flights\_has\_Cities\_primaryID = FhC.primaryID INNER JOIN Flights F

ON FhC.idFlight = F.flight\_ID;

BEGIN

OPEN myCursor(search\_name);-- відкриваєм курсор

LOOP

FETCH myCursor INTO records;

EXIT WHEN NOT FOUND;

IF records.Name = search\_name

THEN result := 'Пасажир: ' || records.Name || ' Рейс: ' || records.Flight || ' Час: (' || records.Time || ') Клас: ' || records.Class

|| ' Місце: ' || records.Seat || ' Ціна: ' || records.Price || ' Час бронювання: ' || records.Booking;

RETURN NEXT result;

END IF;

END LOOP;

CLOSE myCursor;-- закриваєм курсор

END;

$$ LANGUAGE plpgsql;

SELECT search\_customer\_tickets('Сомко Яромира Остапівна');

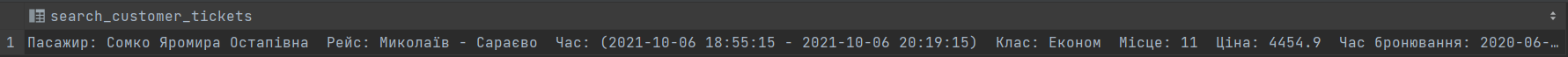


Рисунок 3.45 – Результат виконання функції

Функція, яка змінює статус рейсу, якщо дата прибуття менша за поточну дату та повертає оновлені дані (рисунок 3.46):

CREATE OR REPLACE FUNCTION update\_status ()

RETURNS TABLE(flight\_ID INT, idFirm INT, idTransport INT, from\_city VARCHAR(45), to\_city VARCHAR(45), distance FLOAT, timeStart TIMESTAMP(0), timeEnd TIMESTAMP(0), flight\_status FLIGHTSTATUS)

AS $$

BEGIN

RETURN QUERY

UPDATE Flights

SET status = 'Відбувся'

WHERE endtime < CURRENT\_TIMESTAMP AND status = 'Очікується'

RETURNING \*;

END;

$$ LANGUAGE plpgsql;

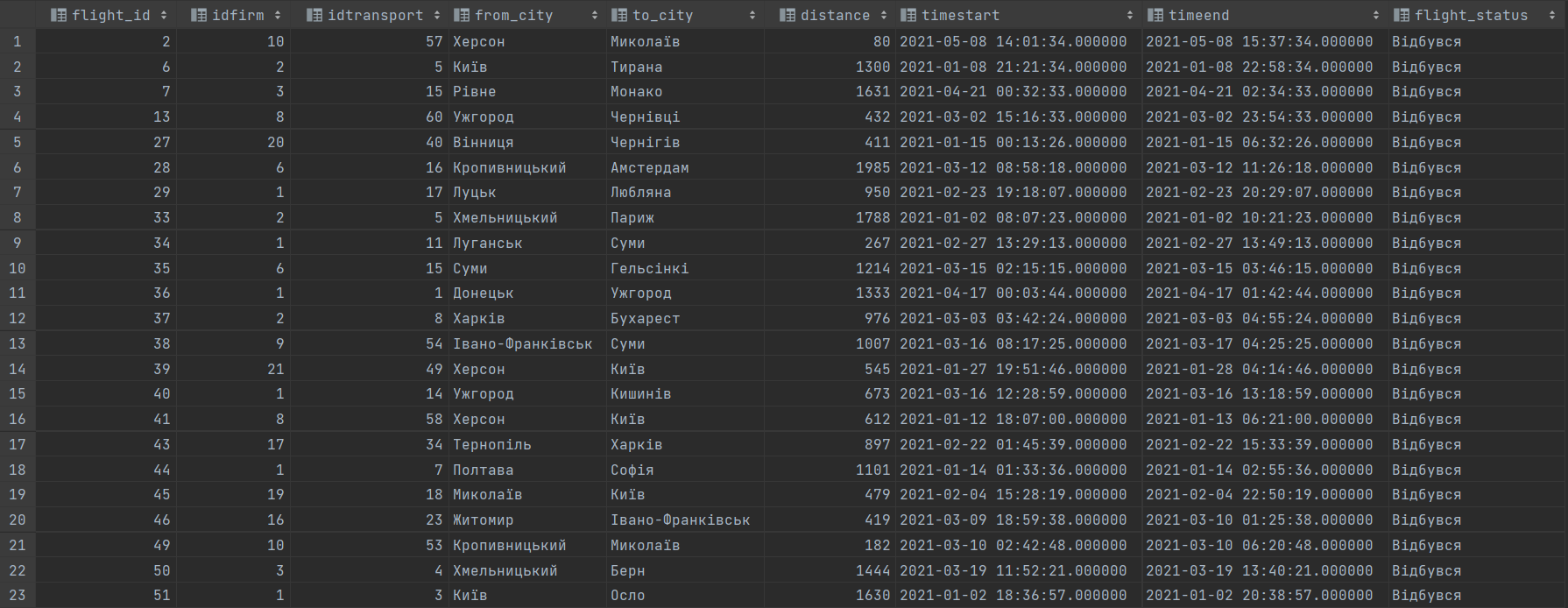


Рисунок 3.46 – Результат виконання функції

Перевантажена функція, яка повертає дату або масив дат відправлення введеного рейсу з заданого часового проміжку, або без нього. (рисунок 3.47, рисунок 3.48):

CREATE OR REPLACE FUNCTION getStartTime(TEXT, TEXT)

RETURNS TEXT

AS $$

SELECT string\_agg(starttime::TEXT, ', ')

FROM Flights

WHERE f\_from = $1 AND f\_to = $2

ORDER BY 1

LIMIT 1;

$$ LANGUAGE SQL IMMUTABLE;

CREATE OR REPLACE FUNCTION getStartTime(TEXT, TEXT, DATE, DATE)

RETURNS TEXT

AS $$

SELECT string\_agg(starttime::TEXT, ', ')

FROM Flights

WHERE f\_from = $1 AND f\_to = $2 AND (starttime BETWEEN $3::TIMESTAMP AND $4::TIMESTAMP)

ORDER BY 1

LIMIT 1;

$$ LANGUAGE SQL IMMUTABLE;

SELECT f\_from, f\_to, starttime AS from\_table, getStartTime(f\_from, f\_to, '2021-01-01', '2021-09-01') AS from\_function

FROM Flights

ORDER BY 1;

SELECT f\_from, f\_to, starttime AS from\_table, getStartTime(f\_from, f\_to, '2021-01-01', '2021-09-01') AS from\_function

FROM Flights

LIMIT 3;



Рисунок 3.47 – Результат виконання функції з двома аргументами

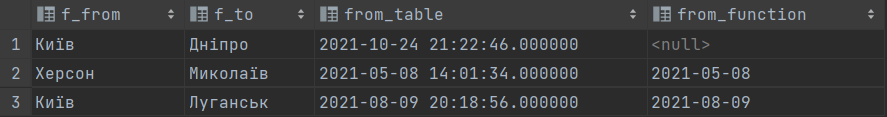


Рисунок 3.48 – Результат виконання функції з чотирьма аргументами

Перевантажена функція, яка формує графік відправлення рейсів між обласними центрами України (рисунок 3.49, рисунок 3.50, рисунок 3.51, рисунок 3.52):

CREATE OR REPLACE FUNCTION timeTable()

RETURNS TABLE(Cities VARCHAR(45), "Івано-Франківськ" TEXT, Вінниця TEXT, Дніпро TEXT , Донецьк TEXT, Житомир TEXT, Запоріжжя TEXT,

Київ TEXT, Кропивницький TEXT, Луганськ TEXT, Луцьк TEXT, Львів TEXT,

Миколаїв TEXT, Одеса TEXT, Полтава TEXT, Рівне TEXT, Суми TEXT,

Тернопіль TEXT, Ужгород TEXT, Харків TEXT, Херсон TEXT, Хмельницький TEXT,

Черкаси TEXT, Чернівці TEXT, Чернігів TEXT)

AS $$

BEGIN

RETURN QUERY

SELECT DISTINCT city\_name, getStartTime(city\_name,'Івано-Франківськ') AS "Івано-Франківськ",

getStartTime(city\_name,'Вінниця') AS "Вінниця", getStartTime(city\_name,'Дніпро') AS "Дніпро",

getStartTime(city\_name,'Донецьк') AS "Донецьк",

getStartTime(city\_name,'Житомир') AS "Житомир", getStartTime(city\_name,'Запоріжжя') AS "Запоріжжя",

getStartTime(city\_name,'Київ') AS "Київ", getStartTime(city\_name,'Кропивницький') AS "Кропивницький",

getStartTime(city\_name,'Луганськ') AS "Луганськ", getStartTime(city\_name,'Луцьк') AS "Луцьк",

getStartTime(city\_name,'Львів') AS "Львів", getStartTime(city\_name,'Миколаїв') AS "Миколаїв",

getStartTime(city\_name,'Одеса') AS "Одеса", getStartTime(city\_name,'Полтава') AS "Полтава",

getStartTime(city\_name,'Рівне') AS "Рівне", getStartTime(city\_name,'Суми') AS "Суми",

getStartTime(city\_name,'Тернопіль') AS "Тернопіль", getStartTime(city\_name,'Ужгород') AS "Ужгород",

getStartTime(city\_name,'Харків') AS "Харків", getStartTime(city\_name,'Херсон') AS "Херсон",

getStartTime(city\_name,'Хмельницький') AS "Хмельницький", getStartTime(city\_name,'Черкаси') AS "Черкаси",

getStartTime(city\_name,'Чернівці') AS "Чернівці", getStartTime(city\_name,'Чернігів') AS "Чернігів"

FROM Flights LEFT JOIN Flights\_has\_Cities FhC on Flights.flight\_ID = FhC.idFlight

INNER JOIN Cities C on C.city\_ID = FhC.idCity1 OR C.city\_ID = FhC.idCity2

WHERE country = 'Україна'

ORDER BY 1;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION timeTable(DATE, DATE)

RETURNS TABLE(Cities VARCHAR(45), "Івано-Франківськ" TEXT, Вінниця TEXT, Дніпро TEXT , Донецьк TEXT, Житомир TEXT, Запоріжжя TEXT,

Київ TEXT, Кропивницький TEXT, Луганськ TEXT, Луцьк TEXT, Львів TEXT,

Миколаїв TEXT, Одеса TEXT, Полтава TEXT, Рівне TEXT, Суми TEXT,

Тернопіль TEXT, Ужгород TEXT, Харків TEXT, Херсон TEXT, Хмельницький TEXT,

Черкаси TEXT, Чернівці TEXT, Чернігів TEXT)

AS $$

BEGIN

IF $1 > $2

THEN RAISE 'ERROR: timeTable(N DATE,M DATE); N must be <= M';

ELSE RETURN QUERY

SELECT DISTINCT city\_name, getStartTime(city\_name,'Івано-Франківськ', $1, $2) AS "Івано-Франківськ",

getStartTime(city\_name,'Вінниця', $1, $2) AS "Вінниця", getStartTime(city\_name,'Дніпро', $1, $2) AS "Дніпро",

getStartTime(city\_name,'Донецьк', $1, $2) AS "Донецьк",

getStartTime(city\_name,'Житомир', $1, $2) AS "Житомир", getStartTime(city\_name,'Запоріжжя', $1, $2) AS "Запоріжжя",

getStartTime(city\_name,'Київ', $1, $2) AS "Київ", getStartTime(city\_name,'Кропивницький', $1, $2) AS "Кропивницький",

getStartTime(city\_name,'Луганськ', $1, $2) AS "Луганськ", getStartTime(city\_name,'Луцьк', $1, $2) AS "Луцьк",

getStartTime(city\_name,'Львів', $1, $2) AS "Львів", getStartTime(city\_name,'Миколаїв', $1, $2) AS "Миколаїв",

getStartTime(city\_name,'Одеса', $1, $2) AS "Одеса", getStartTime(city\_name,'Полтава', $1, $2) AS "Полтава",

getStartTime(city\_name,'Рівне', $1, $2) AS "Рівне", getStartTime(city\_name,'Суми', $1, $2) AS "Суми",

getStartTime(city\_name,'Тернопіль', $1, $2) AS "Тернопіль", getStartTime(city\_name,'Ужгород', $1, $2) AS "Ужгород",

getStartTime(city\_name,'Харків', $1, $2) AS "Харків", getStartTime(city\_name,'Херсон', $1, $2) AS "Херсон",

getStartTime(city\_name,'Хмельницький', $1, $2) AS "Хмельницький", getStartTime(city\_name,'Черкаси', $1, $2) AS "Черкаси",

getStartTime(city\_name,'Чернівці', $1, $2) AS "Чернівці", getStartTime(city\_name,'Чернігів', $1, $2) AS "Чернігів"

FROM Flights LEFT JOIN Flights\_has\_Cities FhC on Flights.flight\_ID = FhC.idFlight

INNER JOIN Cities C on C.city\_ID = FhC.idCity1 OR C.city\_ID = FhC.idCity2

WHERE country = 'Україна'

ORDER BY 1;

END IF;

END;

$$ LANGUAGE plpgsql;

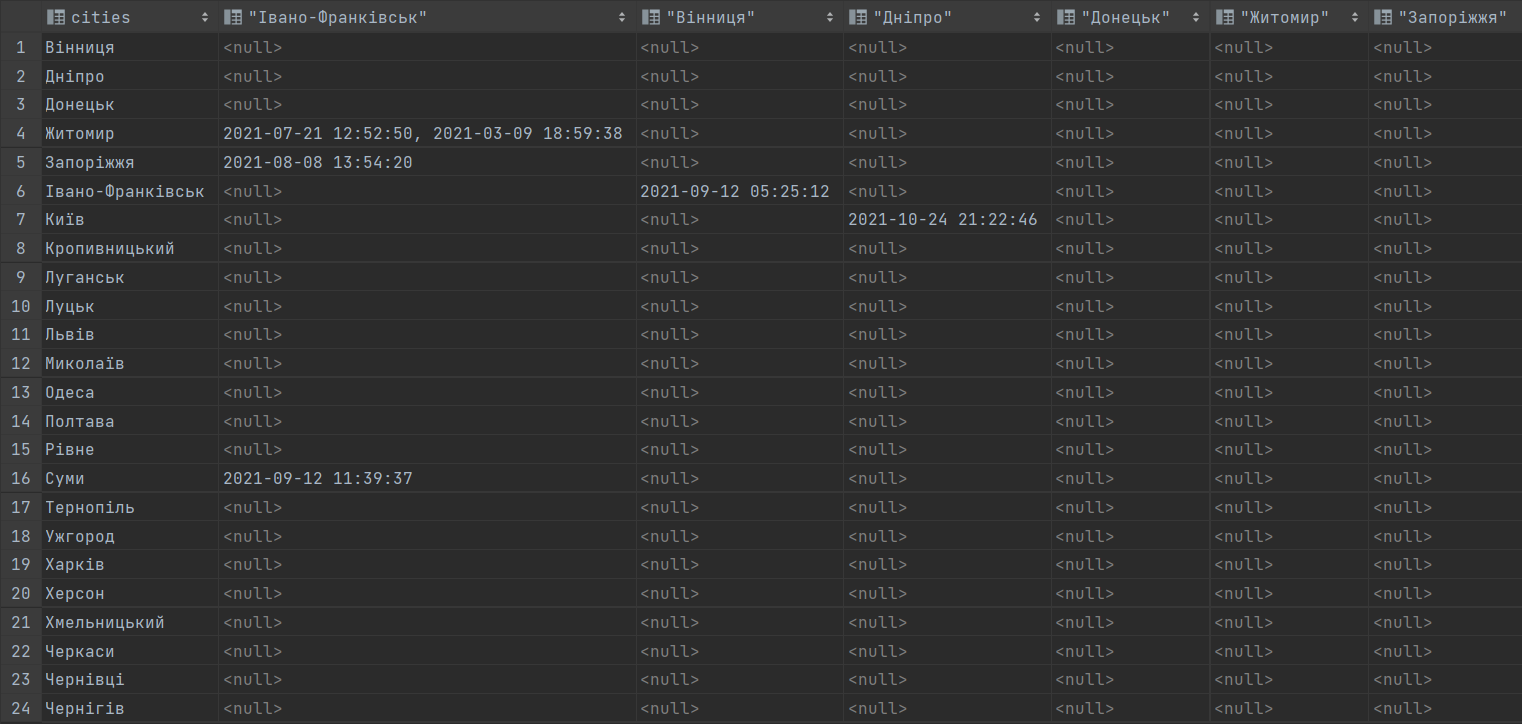


Рисунок 3.49 – Результат виконання функції

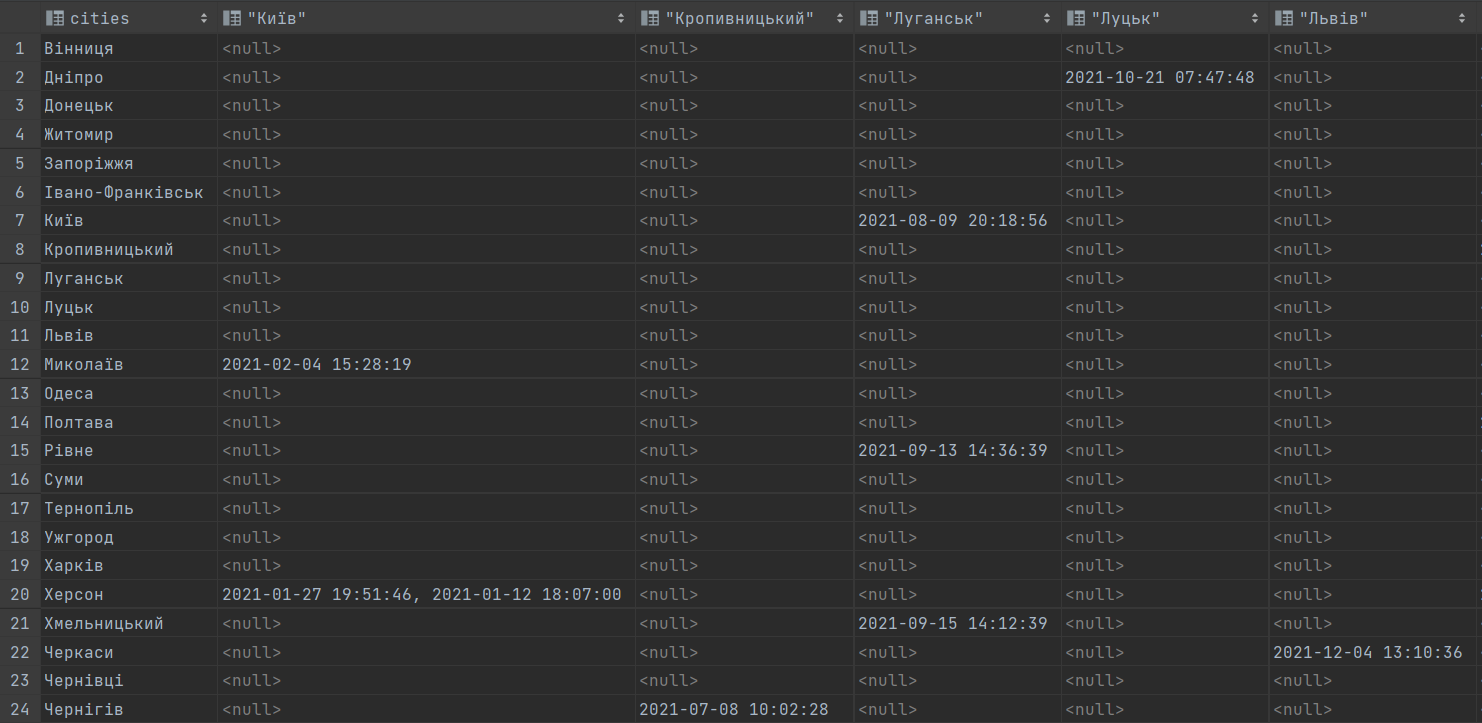


Рисунок 3.50 – Результат виконання функції

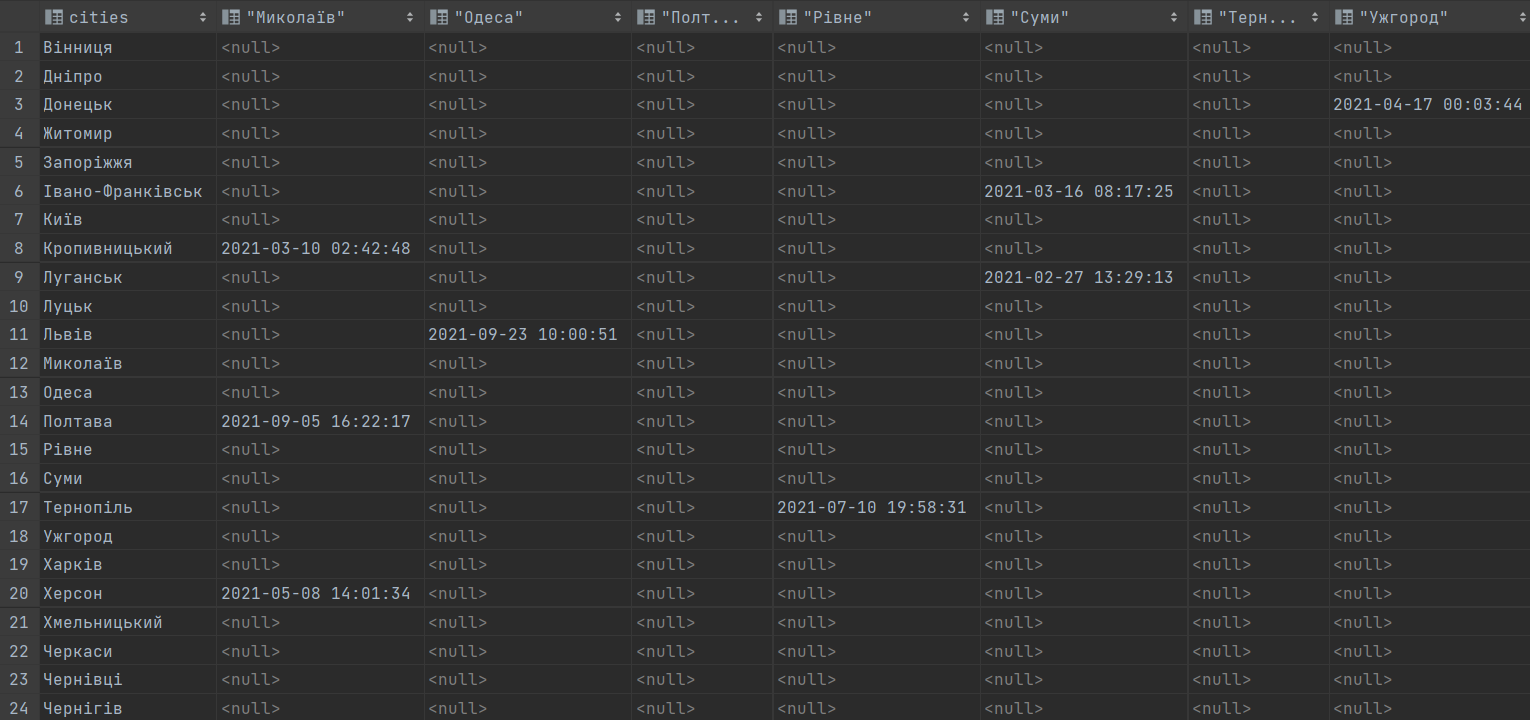


Рисунок 3.51 – Результат виконання функції

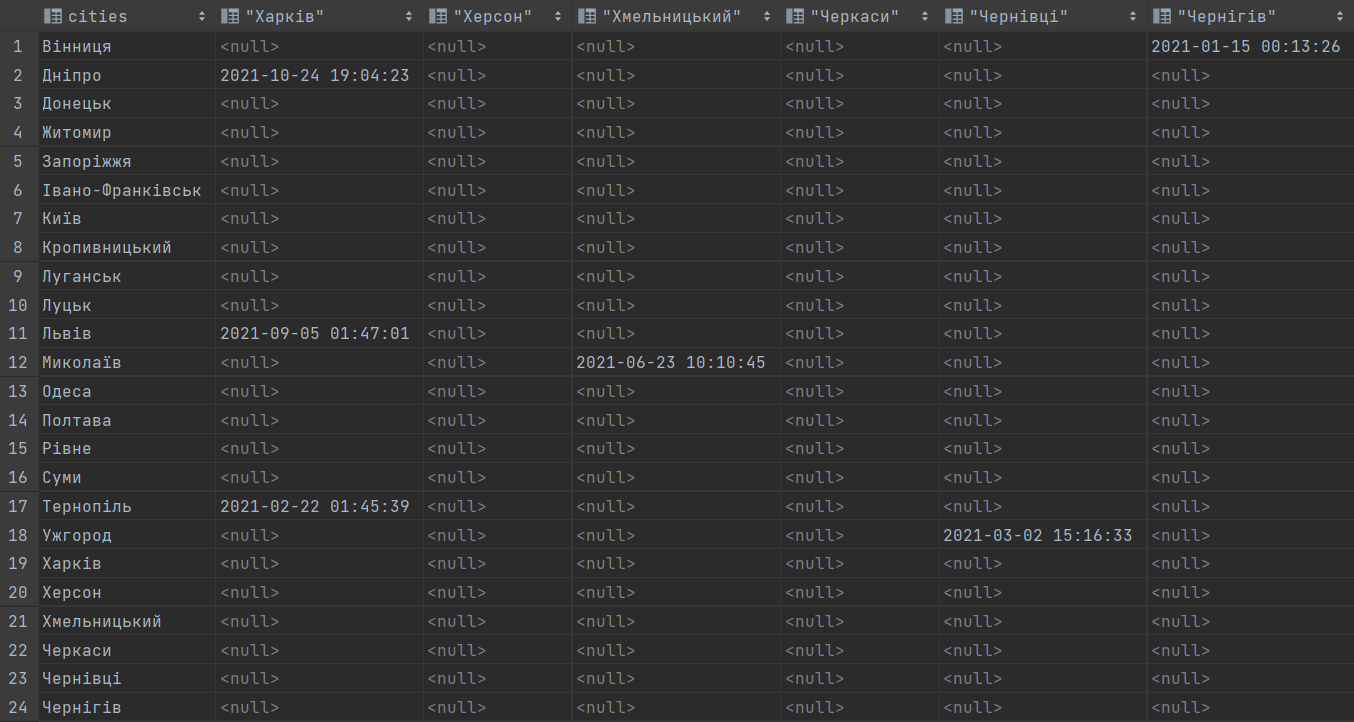


Рисунок 3.52 – Результат виконання функції

Результат виконання функції з двома аргументами наведено на рисунку 3.53. Для зручності перегляду було приховано стовпці, які містили тільки null значення.



Рисунок 3.53 – Результат виконання функції з двома аргументами

**3.6 Додавання користувачів та надання їм прав**

Команда CREATE USER є просто синонімом CREATE ROLE . Єдина відмінність в тому, що для команди, записаної у вигляді CREATE USER, за замовчуванням мається на увазі LOGIN, а у вигляді CREATE ROLE мається на увазі NOLOGIN.

Оператор CREATE USER є розширенням PostgreSQL . У стандарті SQL визначення користувачів вважається залежним від реалізації.

У результаті аналізу завдання на курсове проєктування було створено наступних користувачів.

Користувач, який має всі права для бази Tickets\_booking (рисунок 3.54):

CREATE USER Developer WITH PASSWORD 'Developer123';

GRANT ALL PRIVILEGES ON DATABASE Tickets\_booking TO Developer;

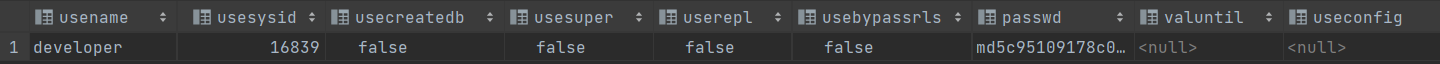


Рисунок 3.54 – Результат створення користувача

Користувач, який має доступ до редагування таблиць та створення представлень для бази Tickets\_booking (рисунок 3.55):

CREATE USER Administrator WITH PASSWORD 'Administrator123';

GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO Administrator;



Рисунок 3.55 – Результат створення користувача

Користувач, який має доступ до вибірки даних з всіх таблиць бази Tickets\_booking (рисунок 3.56):

CREATE USER Customer WITH PASSWORD 'Customer123';

GRANT SELECT ON \* TO Customer;



Рисунок 3.56 – Результат створення користувача

Повний код створеної бази даних у СУБД PostgreSQL наведено у додатку Г.

**ВИСНОВКИ**

Під час виконання курсового проєкту було засвоєно основні тенденції та перспективи розвитку систем керування базами даних. На основі предметної області «Вступна кампанія у ВНЗ» змодельовано фізичну модель бази даних, що відображає структуру та зв’язки даної системи. Для даної предметної області створено 10 запитів, 5 тригерів, 5 представлень, 5 процедур у обох СКБД різного рівня складності. Набуто навики щодо використання різноманітних функцій, операторів та нових можливостей обох СКБД.

Проаналізувавши дану предметну область, створено тринадцять таблиця, які чітко описують дану структуру. Робота над даним проєктом дала змогу познайомитися із способами створення баз даних і таблиць, їх заповненням, виведенням і обробкою записів. Розглянуто вбудовані функції, транзакції, збережені процедури, тригери, курсори. Опрацьовано способи забезпечення цілісності та безпеки даних.

Заповнення здійснювалося за допомогою транзакцій. Працюючи із транзакціями було прослідковано функціональність та корисність даної опції. Заповнення чи то видалення даних у базі даних значно полегшело використання саме транзакцій.

При розробці запитів опрацьовану усі функції для полегшення роботи із БД, а також продемонстровано різноманітні можливості використання тої чи іншої операції. Було створено запити на об’єднання усіх видів та відстежено користь застосування того чи іншого випадку об’єднання. Також було співставлено можливості обох середовищ і зроблено висновок щодо суттєвих відмінностей між ними. Зроблено висновки, що запити у СУБД PostgreSQL значно зручніше і функціональніше реалізовувати за допомогою конструкції WITH, яка розвантажує роботу складних запитів та полегшує роботу з ними. Для забезпечення роботи із значенням року в даті у СУБД PostgreSQL було використанно оператор EXTRACT, що полегшив доступ саме року із дати. У MySQL натомість служить оператор YEAR, чого немає у PostgreSQL.

Використання збережених процедур забезпечило можливість опрацювання та модифікації даних. Дана опція дала можливість об'єднати послідовність однакових запитів і зберегти їх на сервері.

Розглянуто функції у PostgreSQL, за допомогою яких значно легше отримати доступ до потрібних даних. Також простежено різні варіанти щодо повернення даних функцією, що говорить про функціональність даного інструменту.

Для того щоб передбачити певні умови щодо валідації даних, що будуть вводитися, застосовано тригери. Передбачено усі можливості неправильного або не можливого способу модифікації даних за допомогою реалізації тригерів для заданої предметної області.

Для збереження тимчасових та наочно потрібних даних використано представлення, які навідміну від простих запитів зберігають дані. У СУБД PostgreSQL було створено матеріалізовані представлення, основною перевагою і відмінністю який є те, що дані які вони містять можна оновлювати.

Отже, при розробці курсового проєкту було засвоєно велику кількість навиків щодо роботи із базами даних, опрацьовано нові можливості кожного із середовищ MySQL Workbench та PostgreSQL pgAdmin при створені та керуванні базою даних. Під час розроблення курсового проєкту отримано навики щодо забезпечення захищеності даних від несанкціонованих дій. Отримано знання про інформаційні моделі даних, навики щодо створення програмного забезпечення для зберігання, видобування та опрацьовування даних.