|  |  |  |
| --- | --- | --- |
| Національний технічний університет України  **«Київський політехнічний інститут»**  Кафедра автоматизованих систем обробки інформації і управління  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  КУРСОВА РОБОТА  З ДИСЦИПЛІНИ  " **ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ**” | | |
|  | Виконала:  Група:  Керівник: | Зражевська Г.С.  ІС-33  Жаріков Е.В. |
| Київ 2015 | | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Завдання  на курсову роботу по дисципліні "Організація баз даних та знань"  фахове спрямування -  "Інформаційні управляючі системи та технології" напрям 6.050101 "Комп'ютерні науки" Група ІС-33  (п.і.б.) ЗРАЖЕВСЬКА ГАННА СЕРГІЇВНА  Тема Курсової роботи:  **Інформаційна система продажу залізничних квитків**  Дата представлення роботи до захисту до « » 20\_\_ р.   |  |  |  | | --- | --- | --- | |  | Завдання видано: | 10.09.2015 | |  | Дата: | «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ р. | |  | Керівник: | (Е.В.Жаріков) |   Студент:\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_) |

Зміст

**Вступ4**

Опис предметної області4

Постановка задачі4

**Основна частина5**

Розробка ER-діаграми предметної області5

Створення бази даних та заповнення її тестовими данними5

Реалізація запитів12

Запити від двох чи більше таблиць12

Групуючі запити14

Запити на вставку, оновлення та видалення значень15

Розробка необхідних форм введення інформації17

Процес встановлення та розгортання додатку21

**Висновки23**

**Список літератури24**

**Додатки25**

# Вступ

* 1. **Опис предметної області**

Продаж залізничних квитків здійснюється таким чином. Клієнт запрошує наявність квитків на певну дату і напрям, від початкової до кінечної станції, додатково вказуючи бажаний тип вагону (купе, плацкарт і так далі). Касир здійснює пошук в базі по вказаних атрибутах, і повідомляє клієнтові можливі варіанти, які блокуються до закінчення операції. Клієнт вибирає з можливих варіантів квиток, після чого касир вносить дані до бази і отримує гроші за квиток.

Дані, введені касиром, потрапляють у базу даних у центральному апараті, що унеможливлює продажу одного і того ж квитка різним пасажирам.

У разі здачі квитка касир повертає клієнту гроші (за вирахуванням деякої пені), і забирає оформлений квиток, після чого позначає даний квиток як незайнятий.

**1.2 Постановка задачі**

На основі опису предметної області в кожній роботі необхідно виконати наступне:

1. Розробити ER-діаграму предметної області.
2. Створити базу даних і наповнити її тестовими даними.
3. До цієї бази розробити по 2 запити наступних типів:

* запити від двох або більше таблиць,
* групуючі запити,
* запити на вставку, оновлення, видалення значень.

1. Розробити необхідні форми введення інформації у вигляді застосування ASP.NET або подібного (крім Microsoft Access).

Розробити ескізи форм необхідних звітів.

# Основна частина

* 1. **Розробка ER-діаграми предметної області**

Я проаналізувала свою предметну область та виділила такі сутності:

* Станція
* Потяг
* Маршрутний лист
* Розклад
* Вагон
* Місце
* Замовлення
* Куплений квиток
* Варіант
* Підходящий квиток

Таблиця зв’язків між сутностями.

|  |  |  |
| --- | --- | --- |
| **Перша сутність** | **Тип зв’язку** | **Друга сутність** |
| Маршрутний лист | Один до багатьох | Станція |
| Маршрутний лист | Один до багатьох | Потяш |
| Замовлення | Один до багатьох | Станція |
| Потяг | Один до багатьох | Станція |
| Розклад | Один до багатьох | Потяг |
| Вагон | Багато до одного | Розклад |
| Вагон | Багато до одного | Потяг |
| Місце | Багато до одного | Вагон |
| Підходящий потяг | Один до одного | Потяг |
| Підходящий потяг | Багато до одного | Замовлення |
| Підходящий потяг | Багато до одного | Розклад |
| Куплений квиток | Один до одного | Замовлення |
| Куплений квиток | Багато до одного | Розклад |
| Куплений квиток | Багато до одного | Місце |
| Варіант | Один до багатьох | Потяг |
| Варіант | Багато до одного | Розклад |
| Варіант | Багато до одного | Замовлення |
| Варіант | Один до одного | Місце |

Побудована ER-діаграма

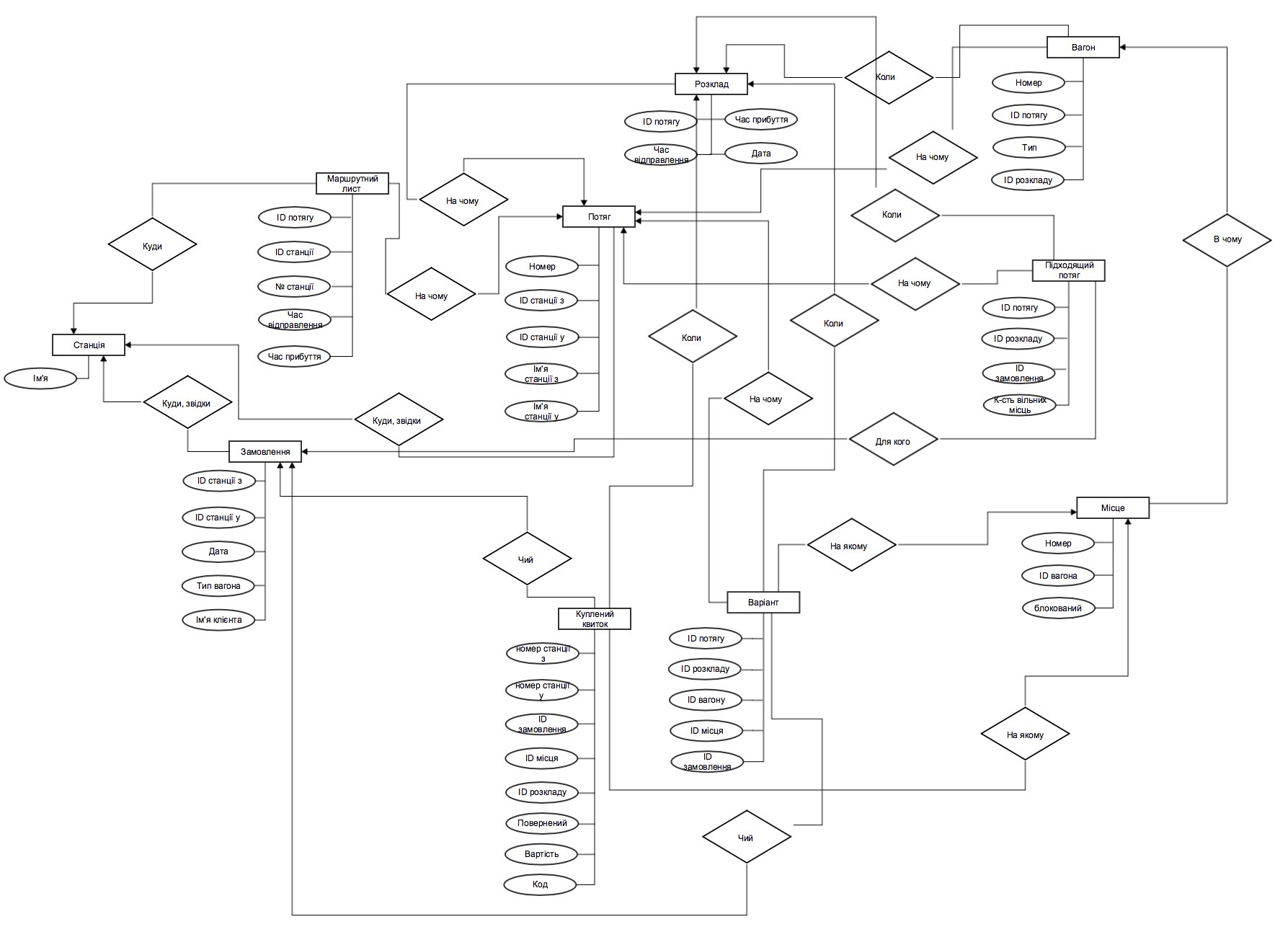


Рис.1

**2.2 Створення бази даних та заповнення її тестовими даними**

На основі розробленої мною ER-діаграми я створила базу даних у СУБД PostgreSQL.

Схема бази даних

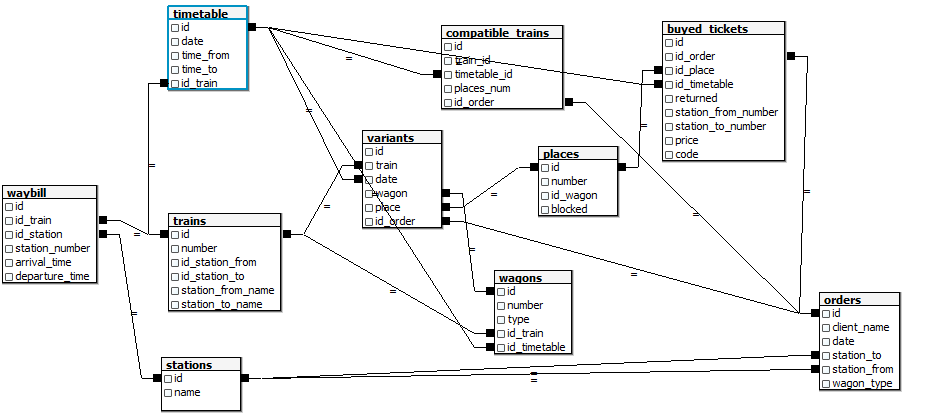


Рис.2

Усі запити на створення бази даних знаходяться в додатку А.

Далі я заповнила базу за допомогою запитів в середовищі pgAdmin III, які можна переглянути в додатку Б.

В результаті отримала заповнені такі таблиці:





Рис.3. Вміст таблиці станцій

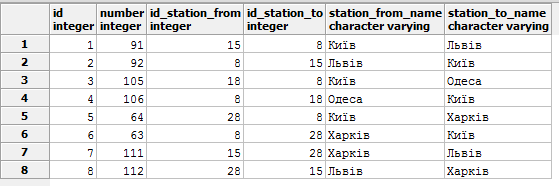


Рис. 4. Вміст таблиці потягів

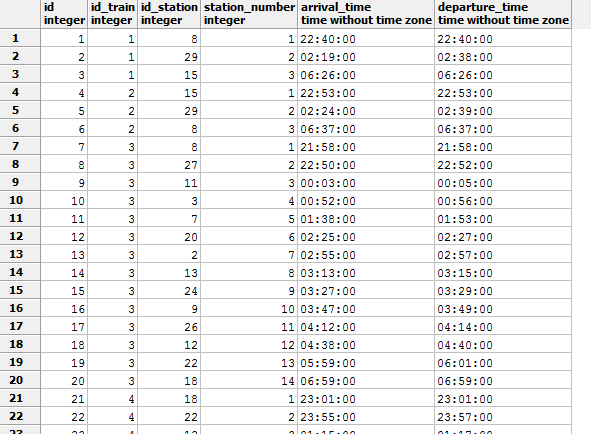


Рис. 5. Вміст таблиці маршрутного листа (загалом містить 80 рядків)

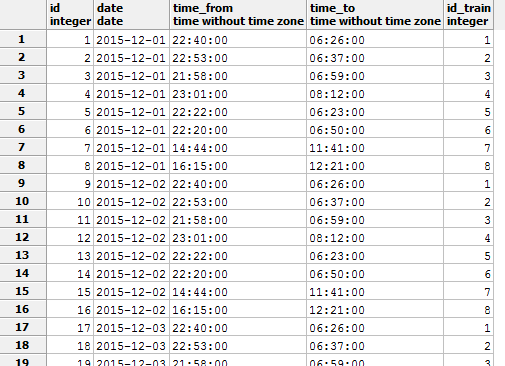


Рис.6. Вміст таблиці розкладу (загалом містить 248 рядків)

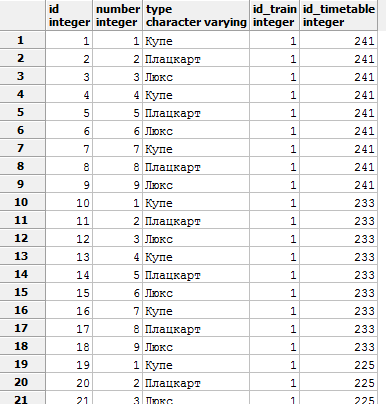


Рис.7. Вміст таблиці вагонів (загалом містить 2232 рядки)

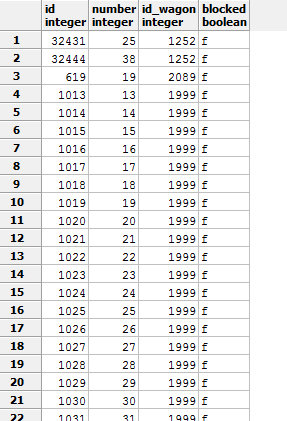


Рис.8. Вміст таблиці місць (загалом містить 84816 рядків)

Усі інші таблиці, а саме таблиці замовлень, варіантів, підходящих подягів, куплених білетів заповнювалися в ході користування клієнтом застосунка.

* 1. **Реалізація запитів**

**2.3.1 Запити від двох чи більше таблиць**

CREATE FUNCTION CompleteVariants(INTEGER) RETURNS void AS $$

DECLARE ord\_id INT := $1;

BEGIN

INSERT INTO Variants(train, date, wagon, place, id\_order) SELECT DISTINCT Timetable.id\_train, Timetable.id, Wagons.id, Places.id, ord\_id

FROM Timetable JOIN (Trains JOIN (Waybill JOIN Stations

ON Waybill.id\_station=Stations.id)

ON Waybill.id\_train=Trains.id)

ON Timetable.id\_train=Trains.id JOIN (Wagons JOIN Places

ON Places.id\_wagon=Wagons.id)

ON Wagons.id\_train=Trains.id AND Wagons.id\_timetable=Timetable.id

WHERE (Timetable.date=(SELECT date FROM Orders WHERE id = ord\_id)

AND EXISTS (SELECT \*

FROM Stations

WHERE Waybill.id\_train=Timetable.id\_train

AND Stations.id = (SELECT station\_from FROM Orders WHERE id=ord\_id))

AND EXISTS (SELECT \*

FROM Stations

WHERE Waybill.id\_train=Timetable.id\_train

AND Stations.id= (SELECT station\_to FROM Orders WHERE id=ord\_id))

AND (SELECT Waybill.station\_number

FROM Waybill JOIN Stations ON Waybill.id\_station=Stations.id

WHERE Waybill.id\_train=Timetable.id\_train

AND Stations.id=(SELECT station\_from FROM Orders WHERE id=ord\_id)) < (SELECT Waybill.station\_number

FROM Waybill JOIN Stations ON Waybill.id\_station=Stations.id

WHERE Waybill.id\_train=Timetable.id\_train

AND Stations.id = (SELECT station\_to FROM Orders WHERE id=ord\_id))

AND Wagons.type=(SELECT wagon\_type FROM Orders WHERE id=ord\_id)

AND Places.blocked <> 't'

AND NOT EXISTS (SELECT \*

FROM Buyed\_Tickets

WHERE id\_place=Places.id

AND id\_timetable = Timetable.id

AND returned <> 't'

AND station\_from\_number <= (SELECT Waybill.station\_number

FROM Waybill JOIN (Stations JOIN Orders ON Orders.station\_from = Stations.id) ON Waybill.id\_station=Stations.id

WHERE Waybill.id\_train=Timetable.id\_train

AND Orders.id = ord\_id)

AND station\_to\_number >= (SELECT Waybill.station\_number

FROM Waybill JOIN (Stations JOIN Orders ON Orders.station\_from = Stations.id) ON Waybill.id\_station=Stations.id

WHERE Waybill.id\_train=Timetable.id\_train

AND Orders.id = ord\_id))

AND NOT EXISTS (SELECT \*

FROM Buyed\_Tickets

WHERE id\_place=Places.id

AND id\_timetable = Timetable.id

AND returned <> 't'

AND station\_from\_number >= (SELECT Waybill.station\_number

FROM Waybill JOIN (Stations JOIN Orders ON Orders.station\_from = Stations.id) ON Waybill.id\_station=Stations.id

WHERE Waybill.id\_train=Timetable.id\_train

AND Orders.id = ord\_id)

AND station\_to\_number <= (SELECT Waybill.station\_number

FROM Waybill JOIN (Stations JOIN Orders ON Stations.id=Orders.station\_to) ON Waybill.id\_station=Stations.id

WHERE Waybill.id\_train=Timetable.id\_train

AND Orders.id=ord\_id))

AND NOT EXISTS (SELECT \*

FROM Buyed\_Tickets

WHERE id\_place=Places.id

AND id\_timetable = Timetable.id

AND returned <> 't'

AND station\_from\_number <= (SELECT Waybill.station\_number

FROM Waybill JOIN (Stations JOIN Orders ON Stations.id=Orders.station\_to) ON Waybill.id\_station=Stations.id

WHERE Waybill.id\_train=Timetable.id\_train

AND Orders.id=ord\_id)

AND station\_to\_number >= (SELECT Waybill.station\_number

FROM Waybill JOIN (Stations JOIN Orders ON Stations.id=Orders.station\_to) ON Waybill.id\_station=Stations.id

WHERE Waybill.id\_train=Timetable.id\_train

AND Orders.id=ord\_id))

);

UPDATE Places

SET blocked = 't'

WHERE id IN (SELECT Variants.place

FROM Variants

WHERE id\_order=ord\_id);

INSERT INTO Compatible\_trains(train\_id,timetable\_id,places\_num,id\_order)

SELECT DISTINCT train, date, COUNT(place), ord\_id

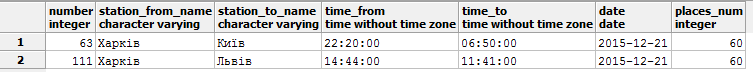
FROM Variants

WHERE id\_order = ord\_id

GROUP BY train, date, ord\_id;

END;

$$LANGUAGE plpgsql



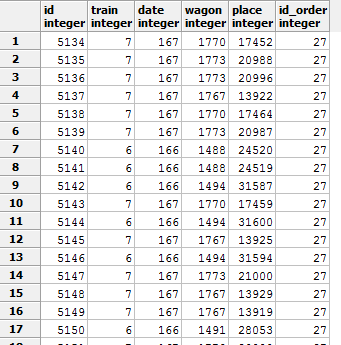


Рис.9,10. Заповнення таблиць варіантів та підходящих потягів

Запрос від декількох таблиць лежить в функції.

SELECT DISTINCT Trains.number, Trains.station\_from\_name, Trains.station\_to\_name, Timetable.time\_from,Timetable.time\_to, Timetable.date, Compatible\_trains.places\_num

FROM Compatible\_trains JOIN (Trains JOIN Waybill ON Waybill.id\_train=Trains.id JOIN Timetable ON Timetable.id\_train=Trains.id) ON Compatible\_trains.train\_id=Trains.id

WHERE Compatible\_trains.id\_order = '27' AND Timetable.id = Compatible\_trains.timetable\_id

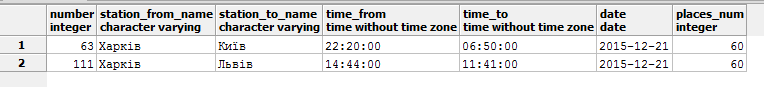


Рис.11 Вивід інформації про потяги, які влаштовують запиту клієнта

* + 1. **Групуючі запити**

place\_id = (SELECT place

FROM Variants

WHERE id\_order = ord\_id

AND train = tr\_id

ORDER BY RANDOM()

LIMIT 1);

Обирання будь-якого вільного місця, влаштовуючого запиту клієнта.

INSERT INTO Compatible\_trains(train\_id,timetable\_id,places\_num,id\_order)

SELECT DISTINCT train, date, COUNT(place), ord\_id

FROM Variants

WHERE id\_order = ord\_id

GROUP BY train, date, ord\_id;

Заповнення таблиці підходящих потягів.

* + 1. **Запити на вставку, оновлення та видалення значень**

UPDATE Places

SET blocked = 't'

WHERE id IN (SELECT Variants.place

FROM Variants

WHERE id\_order=ord\_id);

Оновлення таблиці місць.

CREATE FUNCTION ReturnTicket(UUID) RETURNS FLOAT AS $$

DECLARE sum FLOAT;

BEGIN

sum = (SELECT price FROM Buyed\_tickets

WHERE code = $1);

UPDATE Buyed\_tickets

SET returned = 't'

WHERE code = $1;

RETURN (sum-0.2\*sum);

END;

$$LANGUAGE plpgsql

Оновлення таблиці куплених квитків.

DELETE

FROM Variants

WHERE id\_order = ord\_id;

Видалення з таблиці варіантів.

DELETE

FROM Compatible\_trains

WHERE id\_order = ord\_id;

Видалення з таблиці підходящих потягів.

INSERT INTO Orders(client\_name, date, station\_to, station\_from, wagon\_type)

VALUES ('Зражевська Ганна Сергіївна', '20151221', '17', '16', 'Люкс')

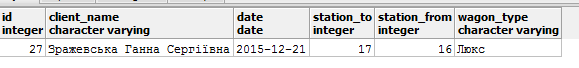


Рис.12 Запит на вставку в таблицю замовлень

INSERT INTO Compatible\_trains(train\_id,timetable\_id,places\_num,id\_order)

SELECT DISTINCT train, date, COUNT(place), ord\_id

FROM Variants

WHERE id\_order = ord\_id

GROUP BY train, date, ord\_id;

Запит на вставку в таблицю підходящих потягів.

Повний список запитів та функції для работи з базою даних представлені в додатку В.

* 1. **Розробка необхідних форм введення інформації**

Для роботи зі створеною базою даних, я створила ВЕБ-застосування та сервер.

Веб-сторінки генеруються сервером мовою jade та висилаються клієнту в браузер. Сервер написаний мовою JavaScript, виконуюче застосування якого Node.js.

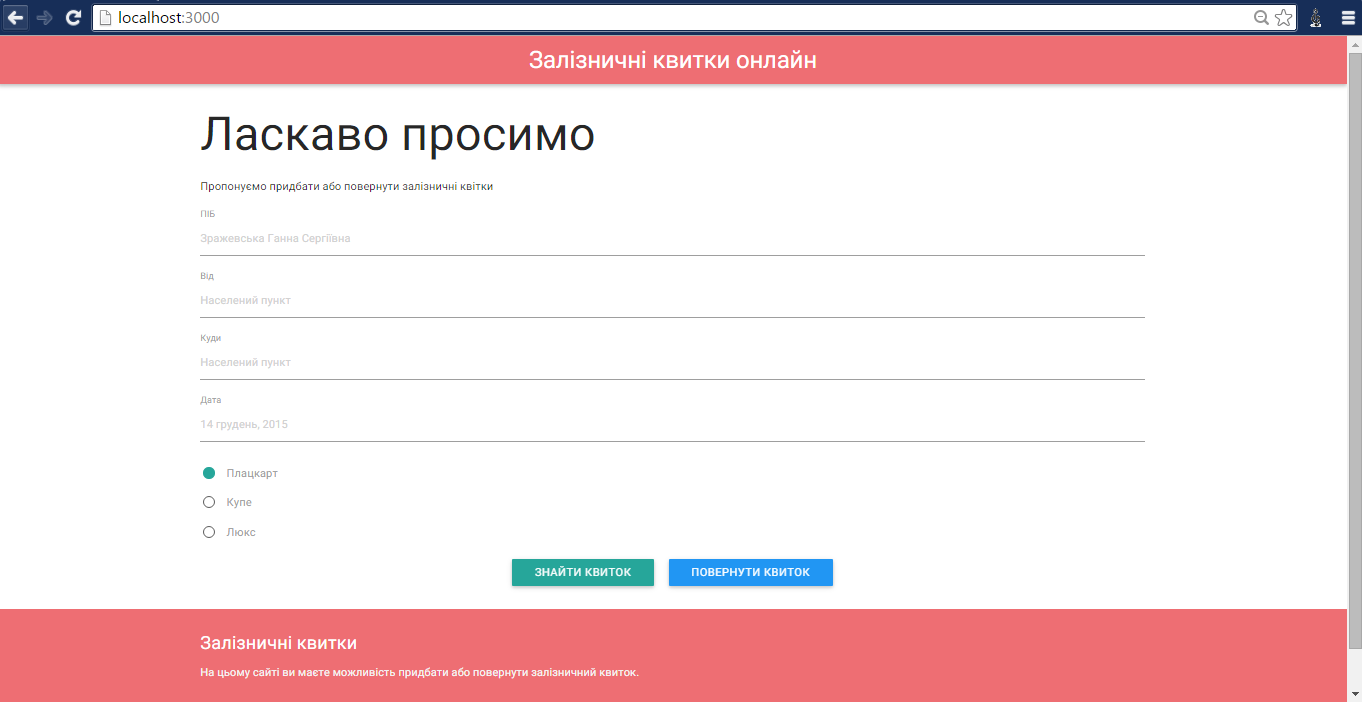


Рис. 13 Головна сторінка

Якщо форма не заповнена, виникає попередження при спробі знайти квиток.

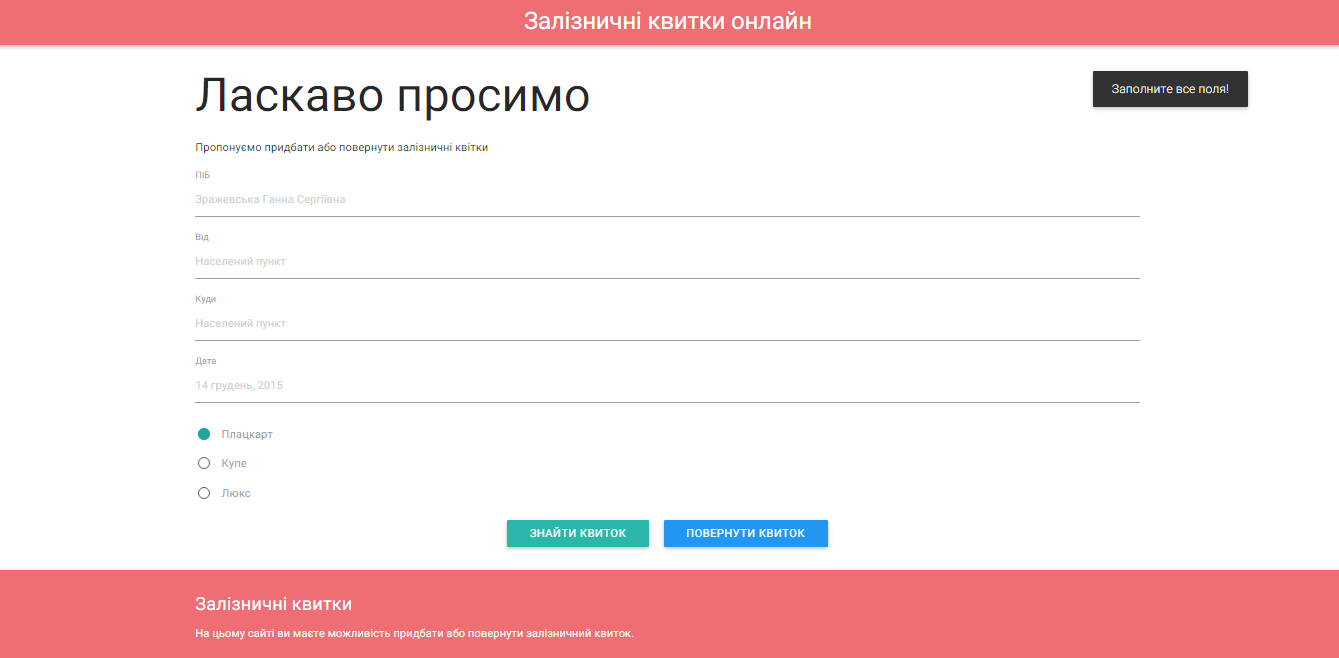


Рис. 14.

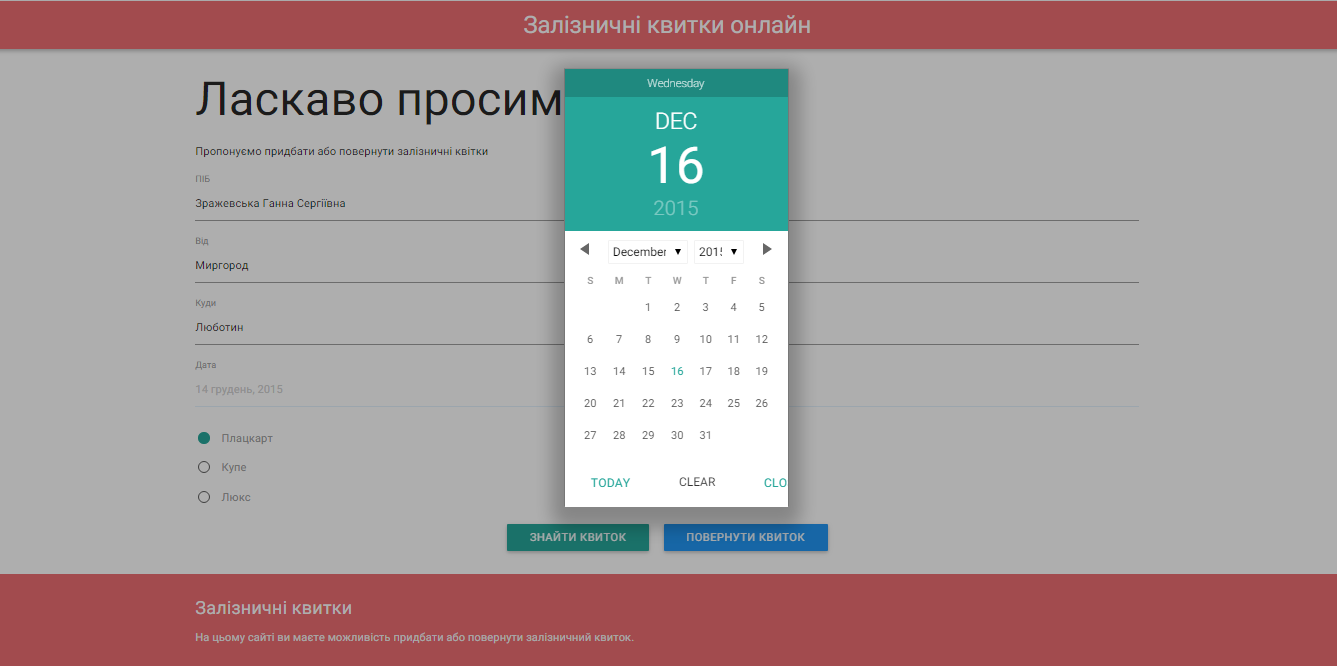


Рис.15 Введення даних

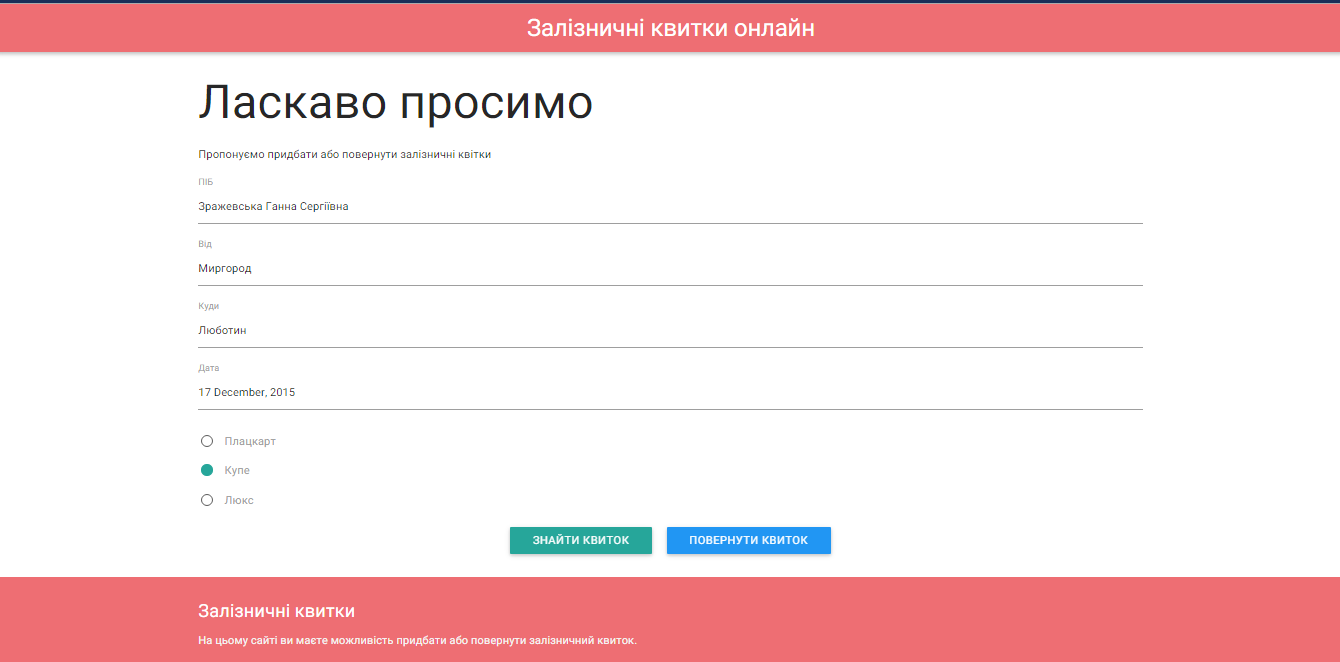


Рис.16

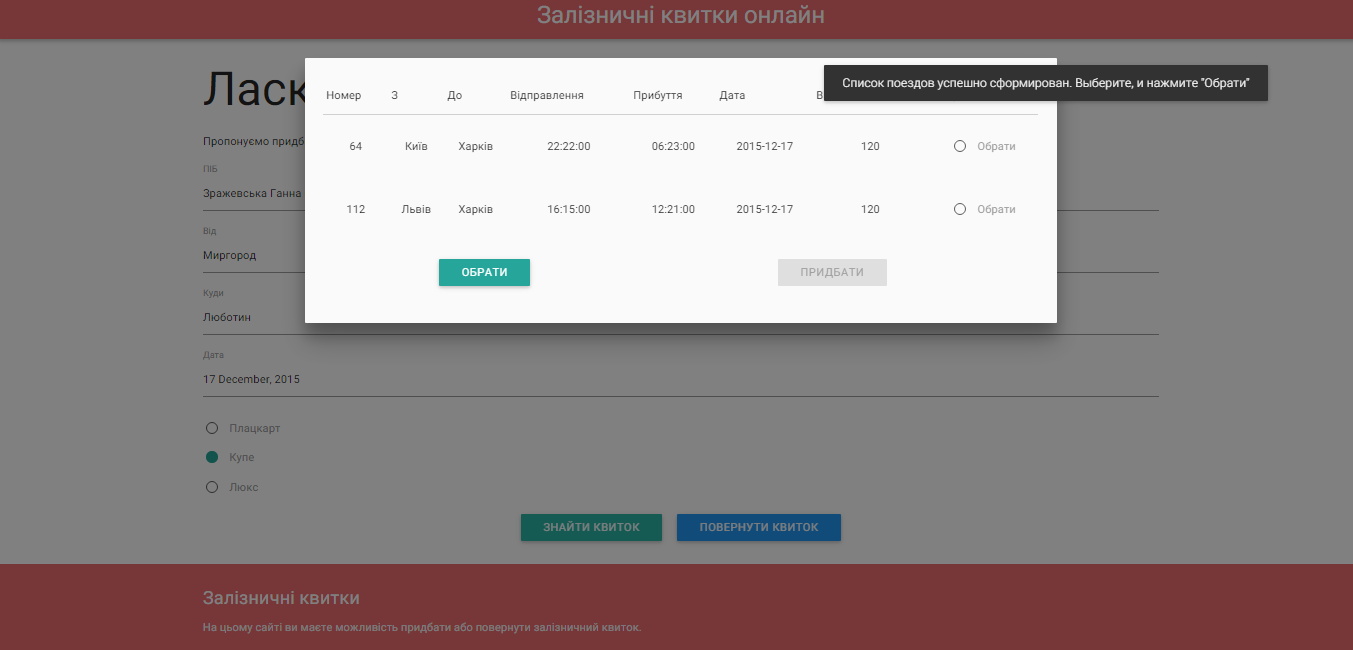


Рис.17 Сформовані варіантів потягів

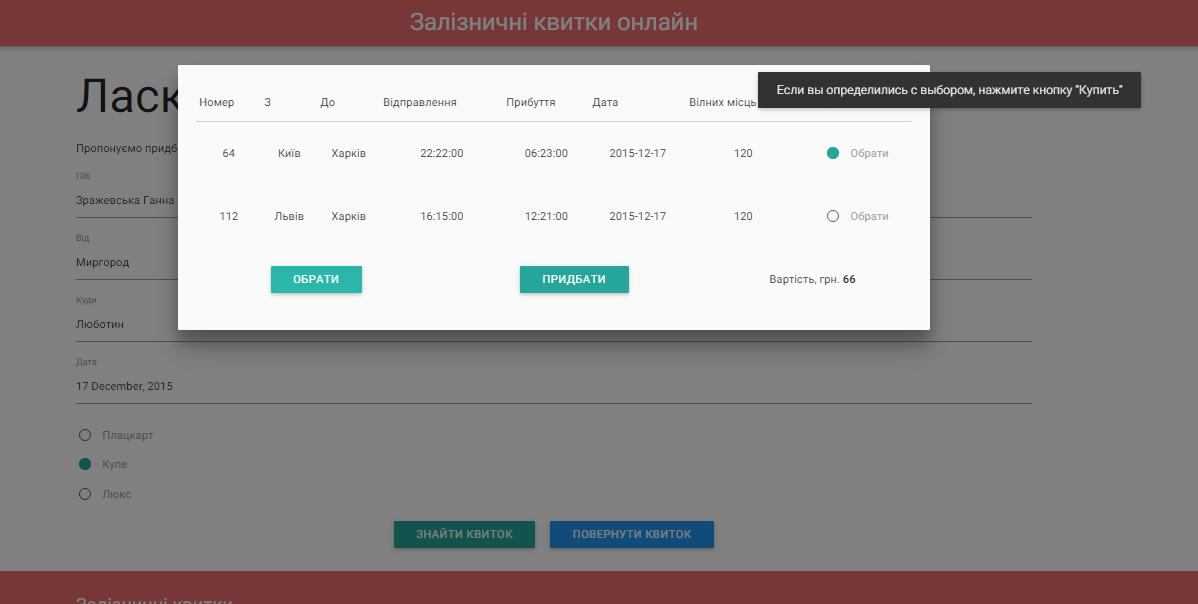


Рис.18 Вартість обраного варіанта

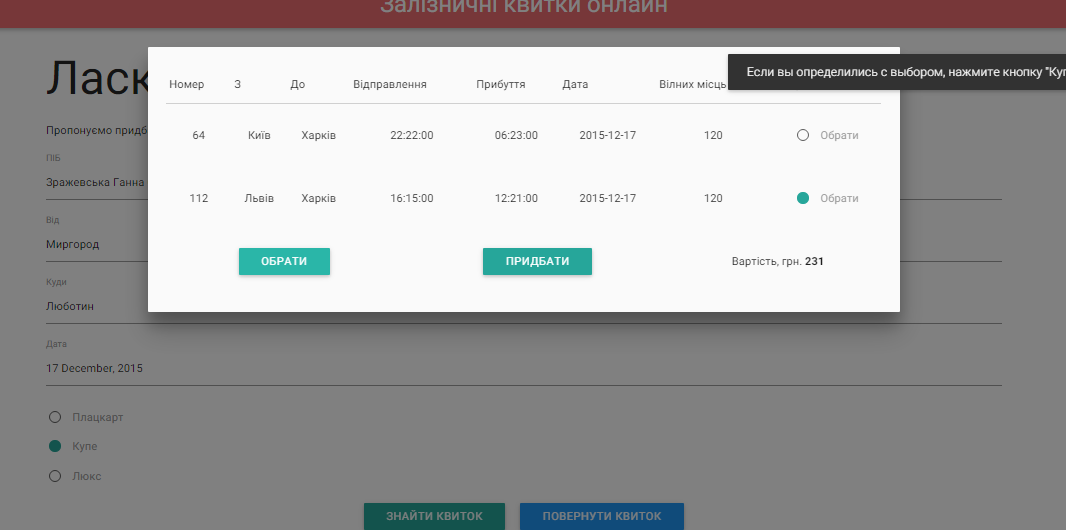


Рис.19

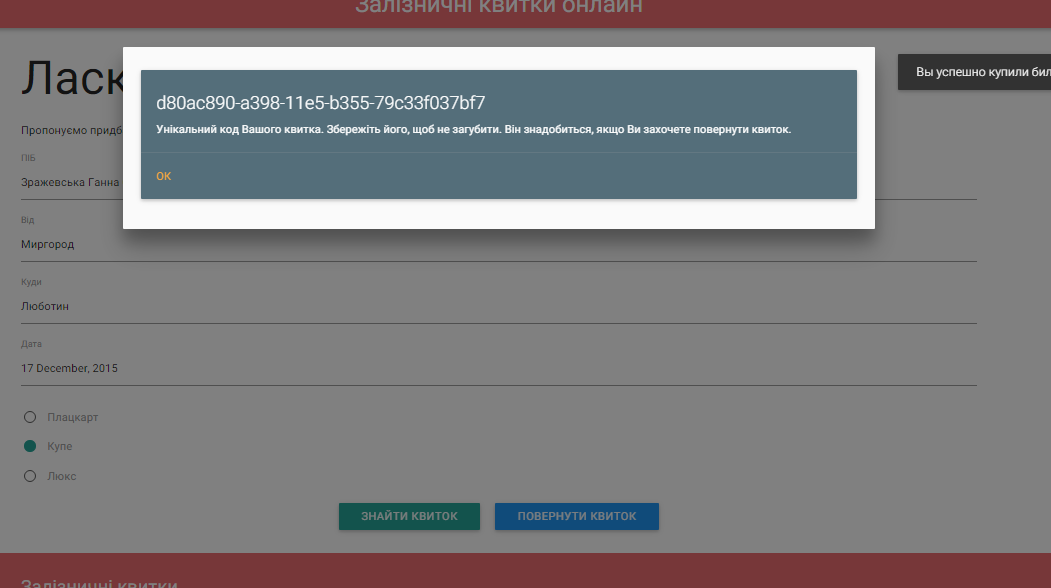


Рис.20 Успішна купівля квитка

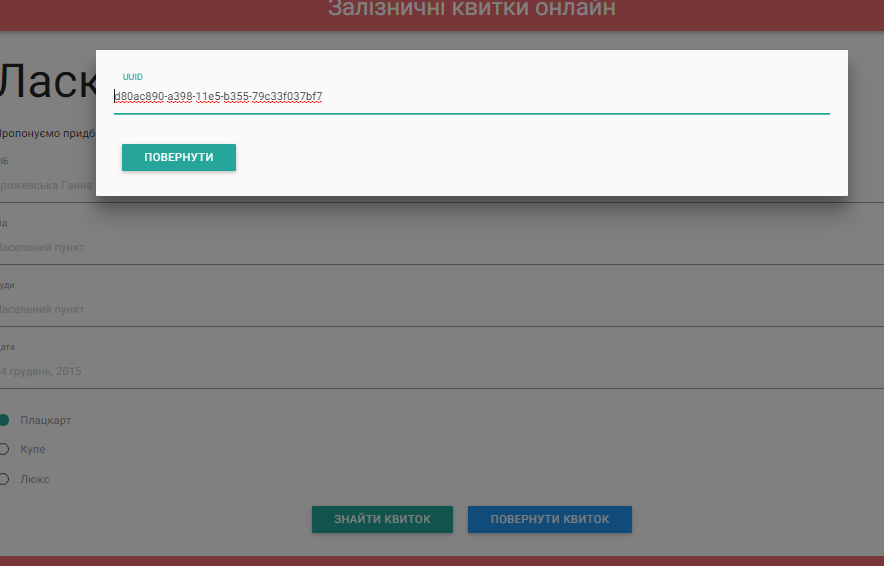


Рис.21 Повернення квитка

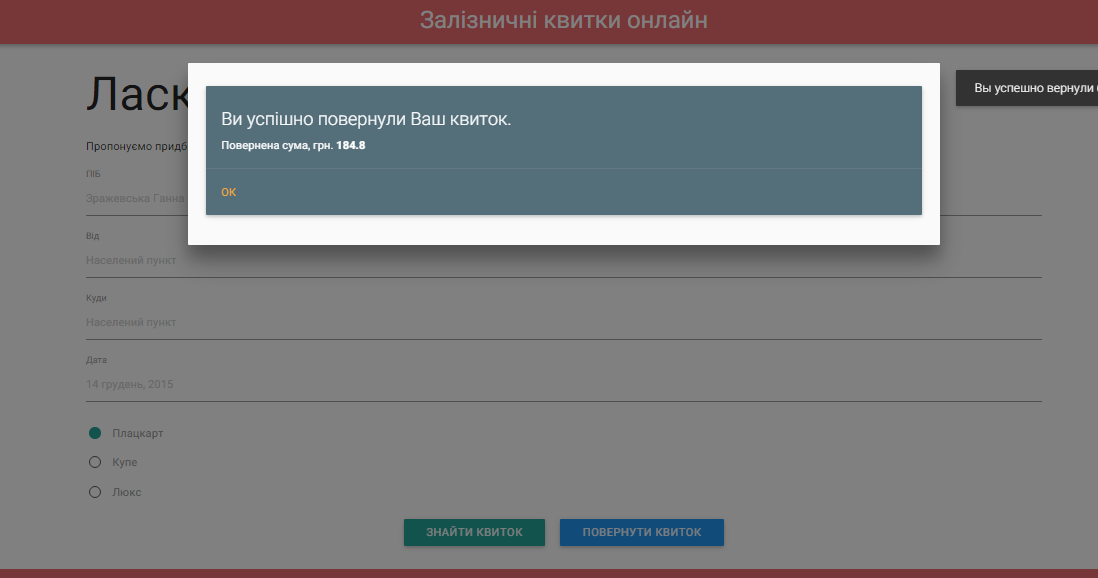


Рис.22 Повернена сума з вирахуванням 20% від початкової вартості квитка.

**2.5 Процес встановлення та розгортання додатку**

Для роботи додатку необхідний будь-який web-сервер, який здатний виконувати код на JS. Для розробки та налагодження програми я використовувала сервер NodeJs 5.2.0, тому я можу гарантувати роботу програми саме на ньому.

Перед початком розгортання додатку потрібно встановити програму NodeJs, мінімальная версія якої потрібна бути 5.2.0. Після встановлення ваш комьютер наділяється змогою виконувати NodeJs код. Далі потрібно в к командній строці перейти у папку с проєктом, таким чином:

1. cd C:\Users\Anna\Desktop\project

Якщо це є Windows.

Далі потрібно виконати средствами NodeJs головний файл нашого серверу app.js, який опрацевує всю логіку, в папці проекта, написав в командну строку так:

1. node app

Якщо все вірно, ви побачити помилку, по типу такої:

1. Listen on port 3000
2. events.js:142
3. throw er; // Unhandled 'error' event
4. ^
5. Error: connect ECONNREFUSED 127.0.0.1:5432

Це виникає тому, що ще не создана, та не підключена база данних.

Для підключення бази данних треба встановити засоби PostgreSQL. Запит для створення бази знаходиться в додатку. Сервер бази треба розташовувати на порту 5432 - це стандартний порт PostgreSQL. Далі необхідно налаштувати додаток для роботи з щойно створеною базою. Для цього необхідно внести зміни до файлу postgres.js в корені каталога. Необхідно змінити рядок підключення до бази даних. Він описаний у 3 рядку цього файлу.



Рис.23

Рис.24 Загальний вид рядка підключення до бази з оф.сайту

Тепер, коли база готова ми повинні знову підключитися там ж самим способом, з використанням команди:

node app

находячись у папці з проектом, як описанно вище.

Все! В команднії строці повинно з’явитися повідомлення:

Listen on port 3000

Яке означатиме, що додаток успішно запушений - сервер слухає порт 3000 (стандартний порт для розробки)

Далі необхідно запустити любий браузер, та набрати в ньому адресу:

*localhost:3000*

В робочій області браузера повинен з’явитися сайт онлайн продажу ЖД квитків.

На цьому розгортання та перший запуск завершені.

# 3. Висновки

В даній роботі я розробила систему продажу зализничних квітків. Свій додаток я побудувала на базі реляційної системи керування базами даних PostgreSQL, ця СУБД дуже пишаєтся своїми можливостями, завдяки яким я розробляла складні умови запитів до бази.

Основным завданням при розробці було створити гідну по функціоналу систему, яка здатна виконувати основні функції взаємодії з білетами ЖД онлайн.

Взаємодія з кліентом виконується через веб-сайт, який має функціонал з тренду web-comunity - це AJAX, засіб для відправлення запросів до сервера не перегружаючи сторінку. Цим заходом я добилася приемного вигляду, плавного використання всього функціоналу, дуже зручного та гнучкого інтерфейсу сайту.

Я вважаю, що у майбутньому було б доцільним:

* ввести онлайн резервування білетів.
* додати до функціоналу сайту продаж білетів на літаки та автобуси.
* реалізувати форму зворотьного зв’язку для клієнтів (ций захід широко використовується у подібних великих проєктах)

# 4. Список використаної літератури

1. Офіційний веб-сайт NodeJs - засоб для написання серверних додатків на мові програмування JS. [Електронний ресурс]  
   <https://nodejs.org/en/>
2. Офіціїна документація до фрейморку JQuery - який ндає змоги виконувати складні єлементи функціоналу та стилю. [Електронний ресурс]  
   <http://api.jquery.com>
3. AJAX — Вікіпідручник [Електронний ресурс]  
   <https://ru.wikibooks.org/wiki/AJAX>
4. Архітектура REST Вікіпедія [Електронний ресурс]  
   <https://en.wikipedia.org/wiki/Representational_state_transfer>
5. API Вікіпедія [Електронний ресурс]   
   <https://en.wikipedia.org/wiki/Application_programming_interface>
6. Офіційний веб-сайт шаблонізатора Jade [Електронний ресурс]   
   <http://jade-lang.com>
7. Офіційний веб сайт фреймворка Materialize [Електронний ресурс]   
   <http://materializecss.com/>
8. Офіційний веб сайт фреймворка Bootstrap [Електронний ресурс]  
   <http://getbootstrap.com/>

# Додаток А

**Створення бази даних**

CREATE TABLE Stations

(

id SERIAL PRIMARY KEY,

name VARCHAR

)

CREATE TABLE Trains

(

id SERIAL PRIMARY KEY,

number INT,

id\_station\_from INT,

id\_station\_to INT,

station\_from\_name VARCHAR,

station\_to\_name VARCHAR,

CONSTRAINT fk\_Trains\_Stations\_from

FOREIGN KEY (id\_station\_from)

REFERENCES Stations(id),

CONSTRAINT fk\_Trains\_Stations\_to

FOREIGN KEY (id\_station\_to)

REFERENCES Stations(id)

)

CREATE TABLE Waybill

(

id SERIAL PRIMARY KEY,

id\_train INT,

id\_station INT,

station\_number INT,

arrival\_time TIME,

departure\_time TIME,

CONSTRAINT fk\_Waybill\_Trains

FOREIGN KEY (id\_train)

REFERENCES Trains(id),

CONSTRAINT fk\_Waybill\_Stations

FOREIGN KEY (id\_station)

REFERENCES Stations(id)

)

CREATE TABLE Timetable

(

id SERIAL PRIMARY KEY,

date DATE,

time\_from TIME,

time\_to TIME,

id\_train INT,

CONSTRAINT fk\_Timetable\_Trains

FOREIGN KEY (id\_train)

REFERENCES Trains(id)

)

CREATE TABLE Wagons

(

id SERIAL PRIMARY KEY,

number INT,

type VARCHAR,

id\_train INT,

id\_timetable INT,

CONSTRAINT fk\_Wagons\_Trains

FOREIGN KEY (id\_train)

REFERENCES Trains(id),

CONSTRAINT fk\_Wagons\_Timetable

FOREIGN KEY (id\_timetable)

REFERENCES Timetable(id)

)

CREATE TABLE Places

(

id SERIAL PRIMARY KEY,

number INT,

id\_wagon INT,

blocked BOOLEAN,

CONSTRAINT fk\_Places\_Wagons

FOREIGN KEY (id\_wagon)

REFERENCES Wagons(id)

)

CREATE TABLE Orders

(

id SERIAL PRIMARY KEY,

client\_name VARCHAR,

date DATE,

station\_to INT,

station\_from INT,

wagon\_type VARCHAR,

CONSTRAINT fk\_Orders\_Stations\_f

FOREIGN KEY (station\_from)

REFERENCES Stations(id),

CONSTRAINT fk\_Orders\_Stations\_t

FOREIGN KEY (station\_to)

REFERENCES Stations(id)

)

CREATE TABLE Buyed\_Tickets

(

id SERIAL PRIMARY KEY,

id\_order INT,

id\_place INT,

id\_timetable INT,

returned BOOLEAN,

station\_from\_number INT,

station\_to\_number INT,

price INT,

code UUID,

CONSTRAINT fk\_Buyed\_Orders

FOREIGN KEY (id\_order)

REFERENCES Orders(id),

CONSTRAINT fk\_Buyed\_Places

FOREIGN KEY (id\_place)

REFERENCES Places(id),

CONSTRAINT fk\_Buyed\_Timetable

FOREIGN KEY (id\_timetable)

REFERENCES Timetable(id)

)

CREATE TABLE Variants

(

id SERIAL PRIMARY KEY,

train INT,

date INT,

wagon INT,

place INT,

id\_order INT,

CONSTRAINT fk\_Variants\_Trains

FOREIGN KEY (train)

REFERENCES Trains(id),

CONSTRAINT fk\_Variants\_Timetable

FOREIGN KEY (date)

REFERENCES Timetable(id),

CONSTRAINT fk\_Variants\_Wagons

FOREIGN KEY (wagon)

REFERENCES Wagons(id),

CONSTRAINT fk\_Variants\_Places

FOREIGN KEY (place)

REFERENCES Places(id),

CONSTRAINT fk\_Variants\_Orders

FOREIGN KEY (id\_order)

REFERENCES Orders(id)

)

CREATE TABLE Compatible\_trains

(

id SERIAL PRIMARY KEY,

train\_id INT,

timetable\_id INT,

places\_num INT,

id\_order INT,

CONSTRAINT fk\_CompTrains\_Orders

FOREIGN KEY (id\_order)

REFERENCES Orders(id),

CONSTRAINT fk\_CompTrains\_Trains

FOREIGN KEY (train\_id)

REFERENCES Trains(id),

CONSTRAINT fk\_CompTrains\_Timetable

FOREIGN KEY (timetable\_id)

REFERENCES Timetable(id)

)

# Додаток Б

**Заповнення таблиць**

------------ЗАПОЛНЕНИЕ ТАБЛИЦЫ СТАНЦИЙ---------------

INSERT INTO Stations(name)

VALUES ('Бориспіль'),

('Вапнярка'),

('Вінниця'),

('Гоголеве'),

('Гребінка'),

('Дарниця'),

('Жмеринка'),

('Київ'),

('Кодима'),

('Коломак'),

('Козятин'),

('Котовськ'),

('Крижопіль'),

('Лубни'),

('Львів'),

('Люботин'),

('Миргород'),

('Одеса'),

('Полтава'),

('Рахни'),

('Решетилівка'),

('Роздільна'),

('Ромодан'),

('Рудниця'),

('Сагайдак'),

('Слобідка'),

('Фастів'),

('Харків'),

('Шепетівка'),

('Яготин'),

('Яреськи')

--------------ЗАПОЛНЕНИЕ ТАБЛИЦЫ ПОЕЗДОВ-------------

INSERT INTO Trains(number,id\_station\_to, id\_station\_from, station\_from\_name, station\_to\_name)

VALUES ('91','8','15','Київ','Львів'),

('92','15','8','Львів','Київ'),

('105','8','18','Київ','Одеса'),

('106','18','8','Одеса','Київ'),

('64','8','28','Київ','Харків'),

('63','28','8','Харків','Київ'),

('111','28','15','Харків','Львів'),

('112','15','28','Львів','Харків')

-------------ЗАПОЛНЕНИЕ ТАБЛИЦЫ МАРШРУТОВ-------------

INSERT INTO Waybill(id\_train, id\_station, station\_number, arrival\_time, departure\_time)

VALUES ('1','8','1','22:40','22:40'),

('1','29','2','02:19','02:38'),

('1','15','3','06:26','06:26'),

('2','15','1','22:53','22:53'),

('2','29','2','02:24','02:39'),

('2','8','3','06:37','06:37'),

('3','8','1','21:58','21:58'),

('3','27','2','22:50','22:52'),

('3','11','3','00:03','00:05'),

('3','3','4','00:52','00:56'),

('3','7','5','01:38','01:53'),

('3','20','6','02:25','02:27'),

('3','2','7','02:55','02:57'),

('3','13','8','03:13','03:15'),

('3','24','9','03:27','03:29'),

('3','9','10','03:47','03:49'),

('3','26','11','04:12','04:14'),

('3','12','12','04:38','04:40'),

('3','22','13','05:59','06:01'),

('3','18','14','06:59','06:59'),

('4','18','1','23:01','23:01'),

('4','22','2','23:55','23:57'),

('4','12','3','01:15','01:17'),

('4','26','4','01:42','01:44'),

('4','9','5','02:06','02:08'),

('4','24','6','02:27','02:29'),

('4','13','7','02:43','02:45'),

('4','2','8','03:03','03:05'),

('4','20','9','03:34','03:36'),

('4','7','10','04:08','04:23'),

('4','3','11','05:02','05:05'),

('4','11','12','05:53','05:55'),

('4','27','13','07:07','07:09'),

('4','8','14','08:12','08:12'),

('5','8','1','22:22','22:22'),

('5','6','2','22:40','22:42'),

('5','1','3','23:00','23:02'),

('5','5','4','00:20','00:22'),

('5','14','5','01:00','01:02'),

('5','17','6','01:53','01:55'),

('5','19','7','03:16','03:36'),

('5','16','8','05:41','05:43'),

('5','28','9','06:23','06:23'),

('6','28','1','22:20','22:20'),

('6','16','2','22:58','23:00'),

('6','19','3','01:09','01:29'),

('6','17','4','02:56','03:01'),

('6','14','5','03:57','03:59'),

('6','5','6','04:40','04:43'),

('6','1','7','06:03','06:05'),

('6','6','8','06:28','06:31'),

('6','8','9','06:50','06:50'),

('7','28','1','14:44','14:44'),

('7','16','2','15:20','15:22'),

('7','10','3','16:38','16:40'),

('7','19','4','17:32','18:39'),

('7','21','5','19:25','19:27'),

('7','25','6','19:44','19:46'),

('7','31','7','19:59','20:01'),

('7','4','8','20:17','20:40'),

('7','17','9','21:13','21:15'),

('7','24','10','21:39','21:41'),

('7','14','11','22:11','22:13'),

('7','5','12','22:53','22:56'),

('7','30','13','23:34','23:36'),

('7','8','14','01:12','01:12'),

('8','8','1','02:35','02:35'),

('8','30','2','03:51','03:53'),

('8','5','3','04:29','04:33'),

('8','14','4','05:13','05:15'),

('8','24','5','05:47','05:49'),

('8','17','6','6:14','6:16'),

('8','4','7','06:31','06:33'),

('8','31','8','06:46','06:48'),

('8','25','9','07:01','07:03'),

('8','21','10','07:20','07:22'),

('8','19','11','07:56','09:10'),

('8','10','12','10:07','10:09'),

('8','16','13','11:49','11:51'),

('8','28','14','12:21','12:21')

-------------ЗАПОЛНЕНИЕ ТАБЛИЦЫ РОЗКЛАДУ--------------

CREATE FUNCTION completeTimetable() RETURNS void AS $$

DECLARE i INT := 1;

DECLARE this\_day DATE;

BEGIN

WHILE i < 32

LOOP

IF i < 10 THEN this\_day := CAST(CONCAT('15120',CAST(i AS VARCHAR(1))) AS DATE);

ELSE this\_day := CAST(CONCAT('1512',CAST(i AS VARCHAR(2))) AS DATE);

END IF;

INSERT INTO Timetable(date, time\_from, time\_to, id\_train)

VALUES (this\_day,'22:40','6:26','1'),

(this\_day,'22:53','06:37','2'),

(this\_day,'21:58','06:59','3'),

(this\_day,'23:01','08:12','4'),

(this\_day,'22:22','06:23','5'),

(this\_day,'22:20','06:50','6'),

(this\_day,'14:44','11:41','7'),

(this\_day,'16:15','12:21','8');

i := i + 1;

END LOOP;

END;

$$ LANGUAGE plpgsql

SELECT completeTimetable()

----------ЗАПОЛНЕНИЕ ТАБЛИЦЫ ВАГОНОВ-----------

CREATE FUNCTION completeWagons() RETURNS void AS $$

DECLARE train\_id INT := 1;

DECLARE timetable\_count INT;

DECLARE timetable\_id INT;

DECLARE j INT := 1;

BEGIN

WHILE train\_id <> 9 LOOP

CREATE TABLE trains\_in\_time(id SERIAL PRIMARY KEY, train\_id\_time INT);

INSERT INTO trains\_in\_time(train\_id\_time) SELECT id FROM Timetable WHERE id\_train = train\_id;

timetable\_count := (SELECT COUNT(\*) FROM trains\_in\_time);

WHILE timetable\_count <> 0

LOOP

j := 1;

WHILE j <> 10

LOOP

timetable\_id := (SELECT train\_id\_time FROM trains\_in\_time WHERE id = timetable\_count);

INSERT INTO Wagons(number, type, id\_train, id\_timetable)

VALUES (j,'Купе',train\_id,timetable\_id),

(j+1,'Плацкарт',train\_id,timetable\_id),

(j+2,'Люкс',train\_id,timetable\_id);

j := j + 3;

END LOOP;

timetable\_count := timetable\_count - 1;

END LOOP;

DROP TABLE trains\_in\_time;

train\_id := train\_id + 1;

END LOOP;

END;

$$ LANGUAGE plpgsql

SELECT completeWagons()

----------ЗАПОЛНЕНИЕ ТАБЛИЦЫ МЕСТ---------

CREATE FUNCTION CompletePlaces() RETURNS void AS $$

DECLARE train\_number INT := 8;

DECLARE wagon\_number INT;

DECLARE wag\_time\_count INT;

DECLARE wagon\_id\_temp INT;

DECLARE places\_number\_coupe INT;

DECLARE places\_number\_luxe INT;

DECLARE places\_number\_res INT;

BEGIN

WHILE train\_number <> 0

LOOP

wagon\_number := 1;

WHILE wagon\_number <> 10

LOOP

CREATE TABLE wagons\_time(id SERIAL PRIMARY KEY, time\_id INT);

INSERT INTO wagons\_time(time\_id) SELECT id\_timetable FROM Wagons WHERE id\_train = train\_number AND number = wagon\_number;

wag\_time\_count := (SELECT COUNT(\*) FROM wagons\_time);

WHILE wag\_time\_count <> 0

LOOP

wagon\_id\_temp := (SELECT id FROM Wagons WHERE id\_train = train\_number AND number = wagon\_number AND id\_timetable = (SELECT time\_id FROM wagons\_time WHERE id = wag\_time\_count));

IF (SELECT type FROM Wagons WHERE id = wagon\_id\_temp) = 'Купе'

THEN

places\_number\_coupe := 1;

WHILE places\_number\_coupe <> 41

LOOP

INSERT INTO Places(number, id\_wagon, blocked)

VALUES (places\_number\_coupe,wagon\_id\_temp, '0');

places\_number\_coupe := places\_number\_coupe + 1;

END LOOP;

ELSIF (SELECT type FROM Wagons WHERE id = @wagon\_id\_temp) = 'Люкс'

THEN

places\_number\_luxe := 1;

WHILE places\_number\_luxe <> 21

LOOP

INSERT INTO Places(number, id\_wagon, blocked)

VALUES (places\_number\_luxe,wagon\_id\_temp, '0');

places\_number\_luxe := places\_number\_luxe + 1;

END LOOP;

ELSIF (SELECT type FROM Wagons WHERE id = @wagon\_id\_temp) = 'Плацкарт'

THEN

places\_number\_res := 1;

WHILE places\_number\_res <> 55

LOOP

INSERT INTO Places(number, id\_wagon, blocked)

VALUES (places\_number\_res,wagon\_id\_temp, '0');

places\_number\_res := places\_number\_res + 1;

END LOOP;

END IF;

wag\_time\_count := wag\_time\_count - 1;

END LOOP;

DROP TABLE wagons\_time;

wagon\_number := wagon\_number + 1;

END LOOP;

train\_number := train\_number - 1;

END LOOP;

END;

$$LANGUAGE plpgsql

SELECT CompletePlaces()

# Додаток В

**Запити для роботи з базою**

------------ЗАПОЛНЕНИЕ ВАРИАНТОВ---------------------

CREATE FUNCTION CompleteVariants(INTEGER) RETURNS void AS $$

DECLARE ord\_id INT := $1;

BEGIN

INSERT INTO Variants(train, date, wagon, place, id\_order) SELECT DISTINCT Timetable.id\_train, Timetable.id, Wagons.id, Places.id, ord\_id

FROM Timetable JOIN (Trains JOIN (Waybill JOIN Stations

ON Waybill.id\_station=Stations.id)

ON Waybill.id\_train=Trains.id)

ON Timetable.id\_train=Trains.id JOIN (Wagons JOIN Places

ON Places.id\_wagon=Wagons.id)

ON Wagons.id\_train=Trains.id AND Wagons.id\_timetable=Timetable.id

WHERE (Timetable.date=(SELECT date FROM Orders WHERE id = ord\_id)

AND EXISTS (SELECT \*

FROM Stations

WHERE Waybill.id\_train=Timetable.id\_train

AND Stations.id = (SELECT station\_from FROM Orders WHERE id=ord\_id))

AND EXISTS (SELECT \*

FROM Stations

WHERE Waybill.id\_train=Timetable.id\_train

AND Stations.id= (SELECT station\_to FROM Orders WHERE id=ord\_id))

AND (SELECT Waybill.station\_number

FROM Waybill JOIN Stations ON Waybill.id\_station=Stations.id

WHERE Waybill.id\_train=Timetable.id\_train

AND Stations.id=(SELECT station\_from FROM Orders WHERE id=ord\_id)) < (SELECT Waybill.station\_number

FROM Waybill JOIN Stations ON Waybill.id\_station=Stations.id

WHERE Waybill.id\_train=Timetable.id\_train

AND Stations.id = (SELECT station\_to FROM Orders WHERE id=ord\_id))

AND Wagons.type=(SELECT wagon\_type FROM Orders WHERE id=ord\_id)

AND Places.blocked <> 't'

AND NOT EXISTS (SELECT \*

FROM Buyed\_Tickets

WHERE id\_place=Places.id

AND id\_timetable = Timetable.id

AND returned <> 't'

AND station\_from\_number <= (SELECT Waybill.station\_number

FROM Waybill JOIN (Stations JOIN Orders ON Orders.station\_from = Stations.id) ON Waybill.id\_station=Stations.id

WHERE Waybill.id\_train=Timetable.id\_train

AND Orders.id = ord\_id)

AND station\_to\_number >= (SELECT Waybill.station\_number

FROM Waybill JOIN (Stations JOIN Orders ON Orders.station\_from = Stations.id) ON Waybill.id\_station=Stations.id

WHERE Waybill.id\_train=Timetable.id\_train

AND Orders.id = ord\_id))

AND NOT EXISTS (SELECT \*

FROM Buyed\_Tickets

WHERE id\_place=Places.id

AND id\_timetable = Timetable.id

AND returned <> 't'

AND station\_from\_number >= (SELECT Waybill.station\_number

FROM Waybill JOIN (Stations JOIN Orders ON Orders.station\_from = Stations.id) ON Waybill.id\_station=Stations.id

WHERE Waybill.id\_train=Timetable.id\_train

AND Orders.id = ord\_id)

AND station\_to\_number <= (SELECT Waybill.station\_number

FROM Waybill JOIN (Stations JOIN Orders ON Stations.id=Orders.station\_to) ON Waybill.id\_station=Stations.id

WHERE Waybill.id\_train=Timetable.id\_train

AND Orders.id=ord\_id))

AND NOT EXISTS (SELECT \*

FROM Buyed\_Tickets

WHERE id\_place=Places.id

AND id\_timetable = Timetable.id

AND returned <> 't'

AND station\_from\_number <= (SELECT Waybill.station\_number

FROM Waybill JOIN (Stations JOIN Orders ON Stations.id=Orders.station\_to) ON Waybill.id\_station=Stations.id

WHERE Waybill.id\_train=Timetable.id\_train

AND Orders.id=ord\_id)

AND station\_to\_number >= (SELECT Waybill.station\_number

FROM Waybill JOIN (Stations JOIN Orders ON Stations.id=Orders.station\_to) ON Waybill.id\_station=Stations.id

WHERE Waybill.id\_train=Timetable.id\_train

AND Orders.id=ord\_id))

);

UPDATE Places

SET blocked = 't'

WHERE id IN (SELECT Variants.place

FROM Variants

WHERE id\_order=ord\_id);

INSERT INTO Compatible\_trains(train\_id,timetable\_id,places\_num,id\_order)

SELECT DISTINCT train, date, COUNT(place), ord\_id

FROM Variants

WHERE id\_order = ord\_id

GROUP BY train, date, ord\_id;

END;

$$LANGUAGE plpgsql

------ВЫВЕДЕНИЕ ПОДХОДЯЩИХ ПОЕЗДОВ-------

SELECT DISTINCT Trains.number, Trains.station\_from\_name, Trains.station\_to\_name, Timetable.time\_from,Timetable.time\_to, Timetable.date, Compatible\_trains.places\_num

FROM Compatible\_trains JOIN (Trains JOIN Waybill ON Waybill.id\_train=Trains.id JOIN Timetable ON Timetable.id\_train=Trains.id) ON Compatible\_trains.train\_id=Trains.id

WHERE Compatible\_trains.id\_order = '27' AND Timetable.id = Compatible\_trains.timetable\_id

-------ЗАПОЛНЕНИЕ ЗАКАЗА-------------

INSERT INTO Orders(client\_name, date, station\_to, station\_from, wagon\_type)

VALUES ('Зражевська Ганна Сергіївна', '20151221', '17', '16', 'Люкс')

--------ЦЕНА БИЛЕТА---------

CREATE FUNCTION calculatePrice(INTEGER, INTEGER) RETURNS integer AS $$

DECLARE train\_numb INTEGER := $1;

DECLARE ord\_id INTEGER := $2;

BEGIN

RETURN 33\*(( SELECT station\_number FROM Waybill WHERE id\_station = (SELECT station\_to FROM Orders WHERE id = ord\_id)

AND id\_train = (SELECT id FROM Trains WHERE number = train\_numb))

- ( SELECT station\_number FROM Waybill WHERE id\_station = (SELECT station\_from FROM Orders WHERE id = ord\_id)

AND id\_train = (SELECT id FROM Trains WHERE number = train\_numb)));

END;

$$LANGUAGE plpgsql

-----------ЗАПОЛНЕНИЕ КУПЛЕННЫХ БИЛЕТОВ-----------

CREATE FUNCTION completeBuyedTickets(INTEGER, INTEGER, INTEGER, UUID) RETURNS void AS $$

DECLARE ord\_id INTEGER := $2;

DECLARE place\_id INT;

DECLARE tr\_id INT;

DECLARE stat\_fr\_num INT;

DECLARE stat\_to\_num INT;

BEGIN

tr\_id = (SELECT id

FROM Trains

WHERE number = $1);

place\_id = (SELECT place

FROM Variants

WHERE id\_order = ord\_id

AND train = tr\_id

ORDER BY RANDOM()

LIMIT 1);

stat\_fr\_num = (SELECT Waybill.station\_number

FROM Waybill

WHERE Waybill.id\_station = (SELECT station\_from FROM Orders WHERE id = ord\_id)

AND Waybill.id\_train = tr\_id);

stat\_to\_num = (SELECT Waybill.station\_number

FROM Waybill

WHERE Waybill.id\_station = (SELECT station\_to FROM Orders WHERE id = ord\_id)

AND Waybill.id\_train = tr\_id);

INSERT INTO Buyed\_Tickets(id\_order,id\_place,id\_timetable,returned,station\_from\_number,station\_to\_number, price, code)

SELECT ord\_id, place\_id, Timetable.id, 'f', stat\_fr\_num, stat\_to\_num, $3, $4

FROM Timetable JOIN Orders ON Orders.date = Timetable.date

WHERE Orders.id = ord\_id AND Timetable.id\_train = tr\_id;

UPDATE Places

SET blocked = 'f'

WHERE id IN (SELECT Variants.place

FROM Variants

WHERE id\_order=ord\_id);

DELETE

FROM Variants

WHERE id\_order = ord\_id;

DELETE

FROM Compatible\_trains

WHERE id\_order = ord\_id;

END;

$$LANGUAGE plpgsql

------ВОЗВРАТ БИЛЕТОВ-------

CREATE FUNCTION ReturnTicket(UUID) RETURNS FLOAT AS $$

DECLARE sum FLOAT;

BEGIN

sum = (SELECT price FROM Buyed\_tickets

WHERE code = $1);

UPDATE Buyed\_tickets

SET returned = 't'

WHERE code = $1;

RETURN (sum-0.2\*sum);

END;

$$LANGUAGE plpgsql