НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

(повна назва інституту/факультету)

КАФЕДРА інформатики та програмної інженерії

(повна назва кафедри)

**КУРСОВА РОБОТА**

з дисципліни «Бази даних»

(назва дисципліни)

на тему: база даних аеропорту\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

Студента \_\_2\_\_ курсу \_ІП-XX групи

спеціальності 121 «Інженерія програмного забезпечення»

Xxx Xxx Xxx \_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник Xxxxx X.X.\_\_\_\_\_\_\_\_\_\_\_\_\_

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

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

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

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

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

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

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

Київ – 2021 рік

**Національний технічний університет України**

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

Інформатики та обчислювальної техніки

Факультет

Інформатики та програмної інженерії

Кафедра

Бази даних

Дисципліна

Курс 2 Група ІП-XX Семестр 1

**ЗАВДАННЯ**

**на курсову роботу студента**

Xxx Xxxxx Xxxxx

(прізвище, ім’я, по батькові)

1. Тема роботи база даних аеропорту

керівник роботи Xxxxxx X. X.

2. Строк здачі студентом закінченої роботи 20.12.2021

3. Вихідні дані до роботи створена база даних відповідно до умови, sql скрипти

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

Опис предметного середовища, постановка задачі, побудова ER-діаграми, побудова

Реляційної моделі бази даних, реалізація бази даних, створення користувачів, sql запити

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

ER-діаграма, реляційна схема бази даних, приклади використання SQL скриптів

6. Дата видачі завдання 31.10.2021

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів виконання курсового проекту | Строк виконання етапів проекту | Примітка |
| 1. | Вступ | 31.10.2021-01.11.2021 |  |
| 2. | Опис предметного середовища | 01.11.2021-03.11.2021 |  |
| 3. | Постановка задачі | 01.11.2021-03.11.2021 |  |
| 4. | Побудова ER-моделі | 03.11.2021-13.11.2021 |  |
| 5. | Побудова реляційної схеми | 13.11.2021-17.11.2021 |  |
| 6. | Створення бази даних з допомогою обраної СУБД | 17.11.2021-19.11.2021 |  |
| 7. | Імпортування даних | 17.11.2021-19.11.2021 |  |
| 8. | Створення користувачів та реалізація їх функціоналу | 19.11.2021-24.11.2021 |  |
| 9. | Створення SQL запитів | 19.11.2021-13.12.2021 |  |
| 10. | Висновок | 13.12.2021-14.12.2021 |  |
| 11. | Перелік посилань | 31.10.2021-14.12.2021 |  |

**Студент \_\_\_\_\_\_\_\_\_\_\_\_ Xxxx X. X.**

(підпис ) (прізвище та ініціали)

**Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(підпис ) (прізвище та ініціали)

ЗМІСТ

[ВСТУП 5](#_Toc90860049)

[1 Опис предметного середовища 6](#_Toc90860050)

[2 Постановка задачі 7](#_Toc90860051)

[3 ER-діаграма 8](#_Toc90860052)

[3.1 Бізнес-правила 8](#_Toc90860053)

[3.2 Вибір сутностей 8](#_Toc90860054)

[3.3 Набори атрибутів сутностей 9](#_Toc90860055)

[4 Реляційна модель бази даних 12](#_Toc90860056)

[4.1 Побудова необхідних відношень та визначення первинних і зовнішніх ключів 12](#_Toc90860057)

[5 Реалізація бази даних 13](#_Toc90860058)

[5.1 Створення бази даних у форматі системи управління базою даних mysql. 13](#_Toc90860059)

[5.2 Імпортування даних в таблицю 17](#_Toc90860060)

[6 Створення користувачів бази даних 18](#_Toc90860061)

[6.1 Створення пасажира 18](#_Toc90860062)

[6.2 Створення перевіряльника 18](#_Toc90860063)

[7 SQL запити 20](#_Toc90860064)

[7.1 Створення тригерів на таблиці, що будуть змінюватись користувачами. 20](#_Toc90860065)

[7.2 Створення процедур для взаємодії користувачів з базою даних 26](#_Toc90860066)

[7.2.1 Процедури пасажира 26](#_Toc90860067)

[7.2.2 Процедури перевіряльника багажу та пасажирів 28](#_Toc90860068)

[7.2.3 Інші sql запити 30](#_Toc90860069)

[7.3 Приклад роботи індексів 33](#_Toc90860070)

[Висновок 35](#_Toc90860071)

[Перелік посилань 36](#_Toc90860072)

ВСТУП

Розробка власної бази даних – це фундамент для будь-якого підприємства. Колись цими базами були архіви, з якими було дуже складно працювати, адже для того, щоб знайти інформацію, яка тебе цікавить потрібно було переглянути гори різних паперів. На їх заміну було створено цифрові бази даних, в яких дані згуртовано по таблицях, що набагато пришвидшило пошук необхідної інформації та її обробки.

Обрана мною тема є актуальною, тому що в аеропортах, зазвичай, дуже великі черги, що означає великі скупчення людей, сьогоднішня ситуація з вірусом COVID-19,

Метою цієї роботи є створити базу даних для аеропорту, яка може замінити роботу касира, на самообслуговування, тобто клієнт через онлайн сервіс вдома або ж будь-де з доступом до інтернету робить бронювання місця на будь-який рейс з вільними місцями.

# Опис предметного середовища

На сьогодення одними з найбезпечніших та найшвидших транспортів є літаки. Організація їх польотів є одним з найголовніших факторів безпечного та комфортного перельоту. Під кожен рейс варто виділити найбільш підходящий літак, обирати який можна за різними характеристиками: вантажопідйомність, кількість пасажирів, марка і чимало інших. Для цього інформацію про кожен літак зручно структурувати у вигляді таблиці. Для ефективного використання часу зайнятості смуги і завантаженості рейсів найкращим рішенням буде створити графік відбуття та прибуття літаків на рейсі. Також для безпеки пасажирів, в аеропорту перед посадкою на літак, кожна людина та її багаж повинні пройти перевірку.

# Постановка задачі

Метою даної роботи є розробка бази даних для аеропорту. Тобто організація даних таким чином, щоб робота аеропорту виконувалась максимально ефективно. Отже основні задачі:

Пасажир повинен мати змогу:

* Забронювати місце на рейс
* Додати багаж, який візьме на рейс
* Переглянути всі свої бронювання
* Переглянути всі зареєстровані багажі

Також, перед тим як зайти на літак, потрібно пройти перевірку, тому перевіряючий повинен мати змогу:

* Переглянути список пасажирів
* Переглянути список багажів
* Змінити статус перевірки певному пасажиру
* Змінити статус перевірки певному багажу

# ER-діаграма

Після аналізу було виділено такі сутності та зв’язки між ними:

## Бізнес-правила

1. Дата бронювання білета не може бути більшою ніж дата вильту літака.
2. На літак не можна забронювати більше білетів, ніж максимальна кількість місць.
3. На літак не можна зареєструвати багаж, який може призвести до перевантаження.
4. Результат перевірки є або позитивним або негативним.

## Вибір сутностей

* Пасажир
* Перевірка пасажира
* Багаж
* Перевірка багажу
* Бронювання
* Перевіряючий
* Результати(довідник)
* Графік рейсів
* Рейс
* Місця(початок і кінець рейсу)
* Літак
* Бренд літака

## Набори атрибутів сутностей

– Сутності та їхні атрибути

|  |  |
| --- | --- |
| Сутність | Атрибути |
| passanger | passanger\_id  baggage\_id  first\_name  last\_name  country\_of\_residence  passport\_number |
| passanger\_check | passanger\_check\_id  result\_id  checked\_at  passanger\_id  checker\_id |
| baggage | baggage\_id  weight  booking\_id |
| baggage\_check | baggage\_check\_id  baggage\_id  result\_id  checked\_at  date\_check  checker\_id |
| booking | booking\_id  passangers\_id  booking\_time  booking\_date  race\_id |

Продовження таблиці 3.1

|  |  |
| --- | --- |
| race | race\_id  graphic\_id  start\_place  end\_place |
| race\_graphic | graphic\_id  time\_daparture  time\_arrival  race\_date |
| plane | plane\_id  id\_brand  race\_id  max\_baggage\_weight  max\_passangers  last\_date\_of\_check |
| places | place\_id  place\_name |
| сhecker | checker\_id  first\_name  last\_name |
| brand | id\_brand  brand\_name |
| results | result\_id  result\_name |

Сутність passanger буде пов’язана один до багатьох з сутностями booking passanger\_check, адже один пасажир може в різні дати та час бронювати квитки та проходити перевірку.

Сутність baggage\_checks буде пов’язана один до багатьох з сутністю багаж, адже перевірки можуть відбуватися в різний час.

Сутність booking пов’язана багато до одного з сутністю race, тому що на один рейс є багато бронювань та один до багатьох з baggage , тому що один пасажир може взяти з собою багато багажів.

Сутність race\_graphic пов’язана один до багатьх з сутністю race, тому що всі рейси відбуваються за одним графіком.

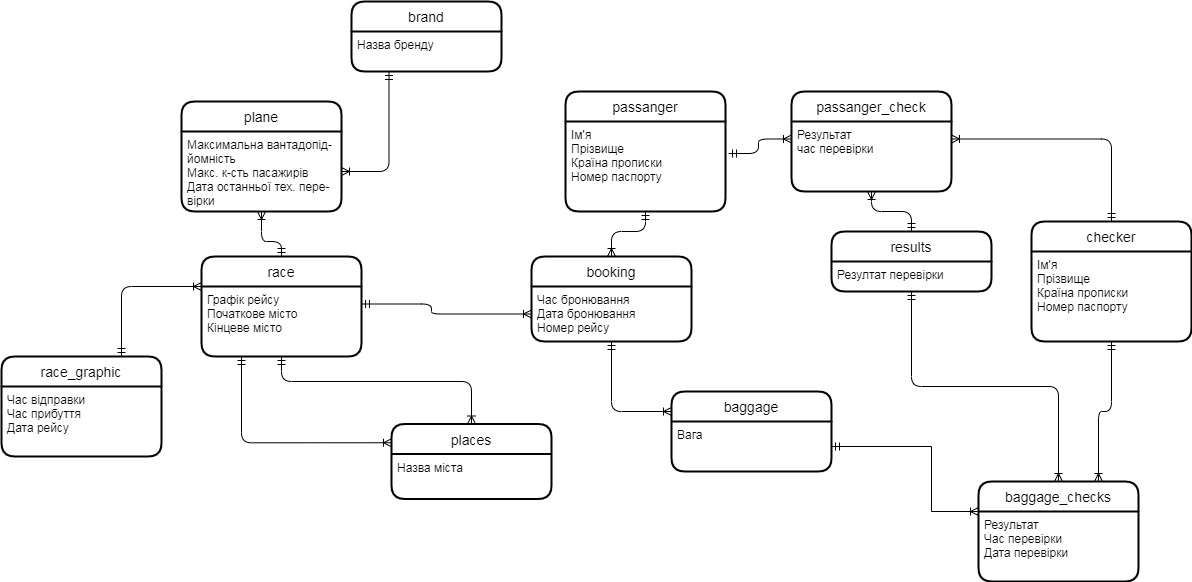
Сутність brand пов’язана один до багатьох з сутністю plane, тому що є багато літаків одного бренда.

Сутність plane пов’язана багато до одного з сутністю race, тому що на одному рейсі може працювати декілька літаків.

Сутність places пов’язана один до багатьх з сутністю race, тому що декілька рейсів можуть починатися або завершуватися в одному місті.

Сутність checker пов’язана один до багатьох з сутностями passanger\_check та baggage\_checks, бо одна людина може перевірити чимало багажів та пасажирів.

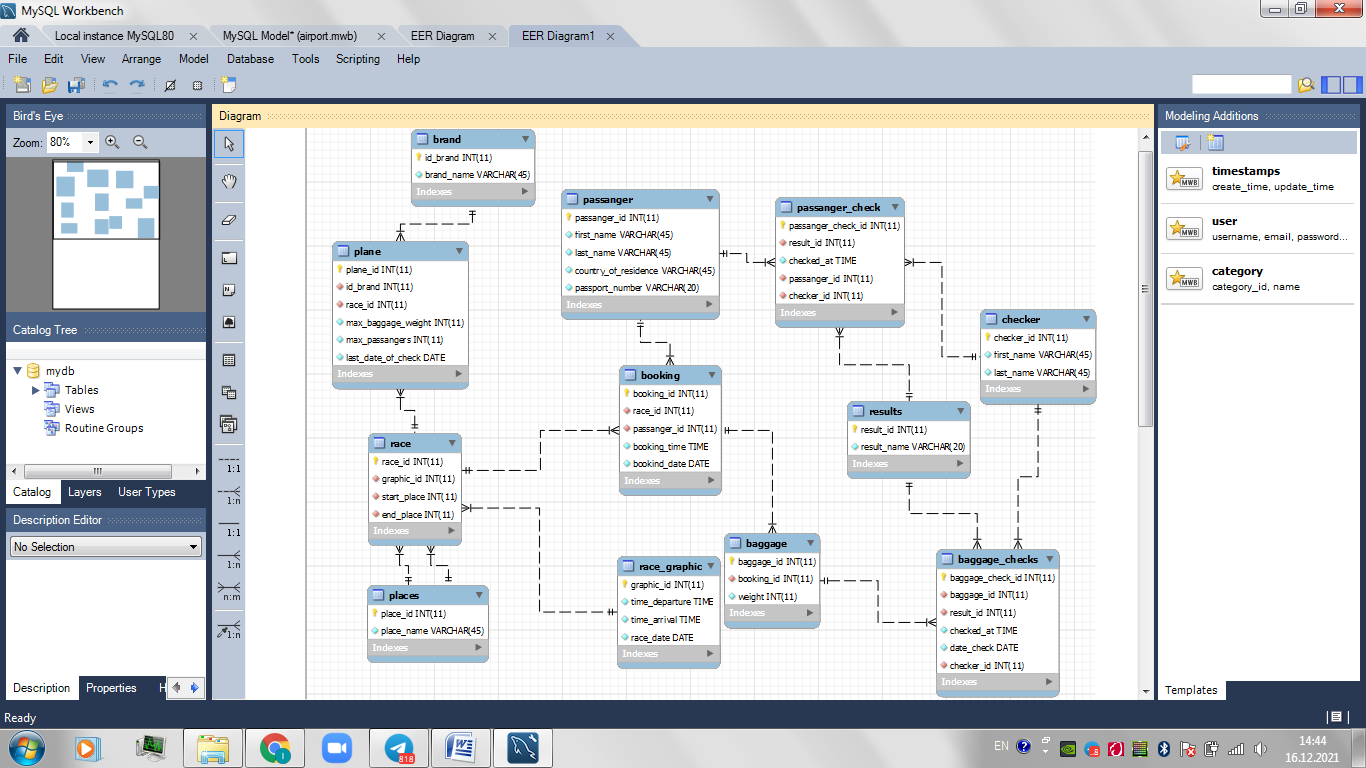
Сутність results пов’язана один до багатьох з сутностями passanger\_check та baggage\_checks, бо один результат може бути характерний для багатьох перевірок.



– ER-діаграма

# Реляційна модель бази даних

## Побудова необхідних відношень та визначення первинних і зовнішніх ключів



– Реляційна схема бази даних

На даній схемі видно, що база даних знаходиться у 3 нормальній формі, адже всі поля таблиць декомпозовані, також всі атрибути таблиць функціонально повно залежать від первинного ключа, кожен неключовий атрибут не є транзитивно залежним від первинного ключа.

1. Визначення обмежень цілісності для спроектованих відношень. Обмеження цілісності: Рядок батьківської таблиці може бути видалений лише у тому випадку, якщо немає зовнішніх ключів, що посилаються на значення преференційного ключа цього рядка. (Реалізовується відсутністю параметра ON DELETE {CASCADE|SET NULL} при створенні таблиці)
2. Обов’язкові атрибути таблиць мають обмеження NOT NULL, для запобіганню помилок при роботі з даними.

# Реалізація бази даних

## Створення бази даних у форматі системи управління базою даних mysql

CREATE TABLE IF NOT EXISTS results (

result\_id INT(11) NOT NULL AUTO\_INCREMENT,

result\_name VARCHAR(20) NOT NULL,

PRIMARY KEY (result\_id)

);

CREATE TABLE IF NOT EXISTS checker (

checker\_id INT(11) NOT NULL AUTO\_INCREMENT,

first\_name VARCHAR(45) NOT NULL,

last\_name VARCHAR(45) NOT NULL,

PRIMARY KEY (checker\_id)

);

CREATE TABLE IF NOT EXISTS passanger (

passanger\_id INT(11) NOT NULL AUTO\_INCREMENT,

first\_name VARCHAR(45) NOT NULL,

last\_name VARCHAR(45) NOT NULL,

country\_of\_residence VARCHAR(45) NOT NULL,

passport\_number VARCHAR(20) NOT NULL,

PRIMARY KEY (passanger\_id)

);

CREATE TABLE IF NOT EXISTS passanger\_check (

passanger\_check\_id INT(11) NOT NULL AUTO\_INCREMENT,

result\_id INT(11) NOT NULL,

checked\_at TIME NOT NULL,

passanger\_id INT(11) NOT NULL,

checker\_id INT(11) NOT NULL,

PRIMARY KEY (passanger\_check\_id),

FOREIGN KEY (passanger\_id) REFERENCES passanger(passanger\_id),

FOREIGN KEY (checker\_id) REFERENCES checker(checker\_id),

FOREIGN KEY (result\_id) REFERENCES results(result\_id)

);

CREATE TABLE IF NOT EXISTS brand (

id\_brand INT(11) NOT NULL AUTO\_INCREMENT,

brand\_name VARCHAR(45) NOT NULL,

PRIMARY KEY(id\_brand)

);

CREATE TABLE IF NOT EXISTS places (

place\_id INT(11) NOT NULL AUTO\_INCREMENT,

place\_name VARCHAR(45) NOT NULL,

PRIMARY KEY (place\_id)

);

CREATE TABLE IF NOT EXISTS race\_graphic(

graphic\_id INT(11) NOT NULL AUTO\_INCREMENT,

time\_departure TIME NOT NULL,

time\_arrival TIME NOT NULL,

race\_date DATE NOT NULL,

PRIMARY KEY (graphic\_id)

);

CREATE TABLE IF NOT EXISTS race (

race\_id INT(11) NOT NULL AUTO\_INCREMENT,

graphic\_id INT(11) NOT NULL,

start\_place INT(11) NOT NULL,

end\_place INT(11) NOT NULL,

PRIMARY KEY (race\_id),

FOREIGN KEY (start\_place) REFERENCES places(place\_id),

FOREIGN KEY (end\_place) REFERENCES places(place\_id),

FOREIGN KEY ( graphic\_id) REFERENCES race\_graphic(graphic\_id)

);

CREATE TABLE IF NOT EXISTS plane (

plane\_id INT(11) NOT NULL AUTO\_INCREMENT,

id\_brand INT(11) NOT NULL,

race\_id INT (11) NOT NULL,

max\_baggage\_weight INT(11) NOT NULL,

max\_passangers INT(11) NOT NULL,

last\_date\_of\_check DATE NOT NULL,

PRIMARY KEY (plane\_id),

FOREIGN KEY (id\_brand) REFERENCES brand(id\_brand),

FOREIGN KEY (race\_id) REFERENCES race(race\_id)

);

CREATE TABLE IF NOT EXISTS booking (

booking\_id INT(11) NOT NULL AUTO\_INCREMENT,

race\_id INT(11) NOT NULL,

passanger\_id INT(11) NOT NULL,

booking\_time TIME NOT NULL,

bookind\_date DATE NOT NULL,

PRIMARY KEY (booking\_id),

FOREIGN KEY (passanger\_id) REFERENCES passanger(passanger\_id),

FOREIGN KEY (race\_id) REFERENCES race(race\_id)

);

CREATE TABLE IF NOT EXISTS baggage(

baggage\_id INT(11) NOT NULL AUTO\_INCREMENT,

booking\_id INT(11) NOT NULL,

weight INT(11) NOT NULL,

PRIMARY KEY (baggage\_id),

FOREIGN KEY (booking\_id) REFERENCES booking(booking\_id)

);

CREATE TABLE IF NOT EXISTS baggage\_checks (

baggage\_check\_id INT(11) NOT NULL AUTO\_INCREMENT,

baggage\_id INT(11) NOT NULL,

result\_id INT(11) NOT NULL,

checked\_at TIME NOT NULL,

date\_check DATE NOT NULL,

checker\_id INT(11) NOT NULL,

PRIMARY KEY (baggage\_check\_id),

FOREIGN KEY (baggage\_id) REFERENCES baggage(baggage\_id),

FOREIGN KEY (checker\_id) REFERENCES checker(checker\_id),

FOREIGN KEY (result\_id) REFERENCES results(result\_id)

);

## Імпортування даних в таблицю

Для імпортування даних у СУБД mysql існує два методи:

* Імпортувати дані кожної таблиці з .csv файлів
* Імпортувати дані із файла скрипта .sql

Для першого методу було створено .csv файли для кожної таблиці.

Для другого методу було написано файл inserting.sql; його вміст:

INSERT INTO `passanger` `passanger\_id`,`first\_name`,`last\_name`,`country\_of\_residence`,`passport\_number`) VALUES (1,'Tommi','Cleave','Argentina','833214740');

INSERT INTO `passanger` (`passanger\_id`,`first\_name`,`last\_name`,`country\_of\_residence`,`passport\_number`) VALUES (2,'Reilly','Seson','Belgium','100678603');

INSERT INTO `passanger` (`passanger\_id`,`first\_name`,`last\_name`,`country\_of\_residence`,`passport\_number`) VALUES (3,'Elna','Dami','Ukraine','162100167');

……

INSERT INTO `booking` (`booking\_id`,`race\_id`,`passanger\_id`,`booking\_time`,`bookind\_date`) VALUES (16,2,12,'15:02:00','2021-01-21');

INSERT INTO `booking` (`booking\_id`,`race\_id`,`passanger\_id`,`booking\_time`,`bookind\_date`) VALUES (17,10,5,'23:30:00','2021-08-05');

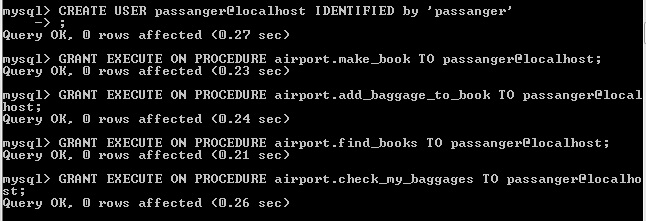
INSERT INTO `booking` (`booking\_id`,`race\_id`,`passanger\_id`,`booking\_time`,`bookind\_date`) VALUES (18,10,11,'09:42:00','2021-03-06');

INSERT INTO `booking` (`booking\_id`,`race\_id`,`passanger\_id`,`booking\_time`,`bookind\_date`) VALUES (19,16,16,'23:00:00','2021-12-01');

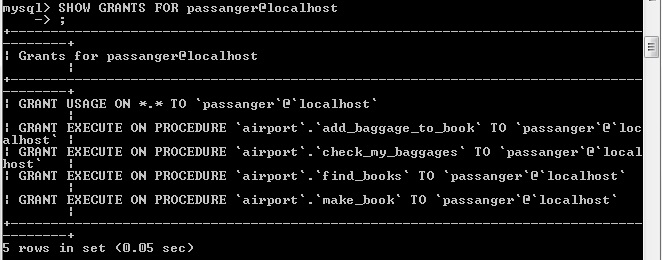
INSERT INTO `booking` (`booking\_id`,`race\_id`,`passanger\_id`,`booking\_time`,`bookind\_date`) VALUES (20,10,5,'13:43:00','2021-08-30');

# Створення користувачів бази даних

## Створення пасажира

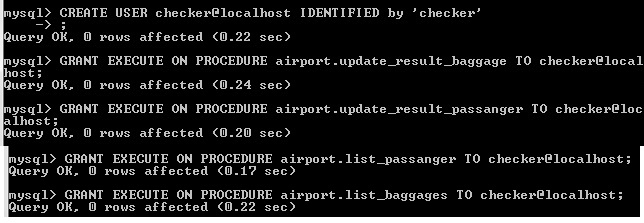


– Надання прав пасажиру

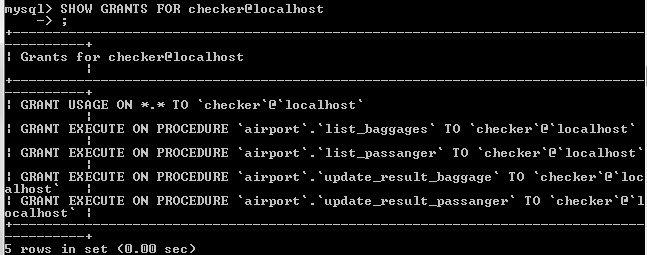


– Права пасажира

## Створення перевіряльника



– Надання прав перевіряльнику



– Права перевіряльника

# SQL запити

## Створення тригерів на таблиці, що будуть змінюватись користувачами.

-- booking

DROP TRIGGER IF EXISTS booking\_insert;

DELIMITER //

CREATE TRIGGER booking\_insert BEFORE INSERT

ON booking FOR EACH ROW

BEGIN

DECLARE max\_passangers INT;

IF (NEW.race\_id > (SELECT COUNT(race\_id) FROM race) OR NEW.race\_id<=0) THEN

SIGNAL SQLSTATE '99999'

SET MESSAGE\_TEXT = 'No such race';

END IF;

SET max\_passangers = (SELECT p.max\_passangers FROM plane p WHERE p.race\_id = NEW.race\_id);

IF (max\_passangers<(SELECT COUNT(booking\_id) FROM booking WHERE race\_id = NEW.race\_id)) THEN

SIGNAL SQLSTATE '99999'

SET MESSAGE\_TEXT = 'Have not free seats';

END IF;

END //

DELIMITER ;

-- checks

DROP TRIGGER IF EXISTS passanger\_check\_update;

DELIMITER //

CREATE TRIGGER passanger\_check\_update BEFORE UPDATE

ON passanger\_check FOR EACH ROW

BEGIN

IF (NEW.passanger\_id > (SELECT COUNT(passanger\_id) FROM passanger) OR NEW.passanger\_id<=0) THEN

SET NEW.passanger\_id = OLD.passanger\_id;

END IF;

IF (NEW.result\_id > 2 OR NEW.result\_id <= 0) THEN

SET NEW.result\_id = OLD.result\_id;

END IF;

END //

DELIMITER ;

UPDATE passanger\_check SET result\_id = 5 WHERE result\_id = 2;

DROP TRIGGER IF EXISTS passanger\_check\_insert;

DELIMITER //

CREATE TRIGGER passanger\_check\_insert BEFORE INSERT

ON passanger\_check FOR EACH ROW

BEGIN

IF (NEW.passanger\_id > (SELECT COUNT(passanger\_id) FROM passanger) OR NEW.passanger\_id<=0) THEN

SIGNAL SQLSTATE '99999'

SET MESSAGE\_TEXT = 'No such passanger';

END IF;

IF (NEW.result\_id > 2 OR NEW.result\_id <= 0) THEN

SIGNAL SQLSTATE '99999'

SET MESSAGE\_TEXT = 'No such result';

END IF;

SET NEW.checked\_at = current\_time();

IF (NEW.checker\_id > (SELECT COUNT(checker\_id) FROM checker) OR NEW.checker\_id<0) THEN

SIGNAL SQLSTATE '99999'

SET MESSAGE\_TEXT = 'Invalid number of checker!';

END IF;

END //

DELIMITER ;

DROP TRIGGER IF EXISTS baggage\_checks\_update;

DELIMITER //

CREATE TRIGGER baggage\_checks\_update BEFORE UPDATE

ON baggage\_checks FOR EACH ROW

BEGIN

IF (NEW.baggage\_check\_id != OLD.baggage\_check\_id OR NEW.baggage\_id != OLD.baggage\_id) THEN

SIGNAL SQLSTATE '99999'

SET MESSAGE\_TEXT = 'You cant change this field!';

END IF;

IF (NEW.result\_id > 2 OR NEW.result\_id <= 0) THEN

SET NEW.result\_id = OLD.result\_id;

END IF;

SET NEW.checked\_at = current\_time();

SET NEW.date\_check = current\_date();

IF (NEW.checker\_id > (SELECT COUNT(checker\_id) FROM checker) OR NEW.checker\_id<0) THEN

SIGNAL SQLSTATE '99999'

SET MESSAGE\_TEXT = 'Invalid number of checker!';

END IF;

END //

DELIMITER ;

UPDATE baggage\_checks SET result\_id = 5 WHERE result\_id = 2;

DROP TRIGGER IF EXISTS baggage\_checks\_insert;

DELIMITER //

CREATE TRIGGER baggage\_checks\_insert BEFORE INSERT

ON baggage\_checks FOR EACH ROW

BEGIN

IF (NEW.baggage\_id > (SELECT COUNT(baggage\_id) FROM baggage) OR NEW.baggage\_id<=0) THEN

SIGNAL SQLSTATE '99999'

SET MESSAGE\_TEXT = 'No such baggage';

END IF;

IF (NEW.result\_id > 2 OR NEW.result\_id <= 0) THEN

SIGNAL SQLSTATE '99999'

SET MESSAGE\_TEXT = 'No such result';

END IF;

END //

DELIMITER ;

-- baggage

DROP TRIGGER IF EXISTS baggage\_update;

DELIMITER //

CREATE TRIGGER baggage\_update BEFORE UPDATE

ON baggage FOR EACH ROW

BEGIN

DECLARE current\_sum\_weight INT;

DECLARE max\_weight INT;

DECLARE max\_passangers INT;

SET max\_passangers = (SELECT p.max\_passangers FROM plane p WHERE p.race\_id = (SELECT r.race\_id FROM booking b, race r WHERE b.race\_id = r.race\_id AND b.booking\_id=NEW.booking\_id));

SET max\_weight = (SELECT p.max\_baggage\_weight FROM plane p WHERE p.race\_id = (SELECT r.race\_id FROM booking b, race r WHERE b.race\_id = r.race\_id AND b.booking\_id=NEW.booking\_id));

SET current\_sum\_weight = (SELECT SUM(b.weight) FROM baggage b WHERE b.booking\_id = NEW.booking\_id)-OLD.weight + NEW.weight;

IF (current\_sum\_weight > (max\_weight/max\_passangers) OR current\_sum\_weight <= 0 OR NEW.weight <0) THEN

SET NEW.weight = OLD.weight;

END IF;

END //

DELIMITER ;

DROP TRIGGER IF EXISTS baggage\_insert;

DELIMITER //

CREATE TRIGGER baggage\_insert BEFORE INSERT

ON baggage FOR EACH ROW

BEGIN

DECLARE current\_sum\_weight INT;

DECLARE max\_weight INT;

DECLARE max\_passangers INT;

SET max\_passangers = (SELECT p.max\_passangers FROM plane p WHERE p.race\_id = (SELECT r.race\_id FROM booking b, race r WHERE b.race\_id = r.race\_id AND b.booking\_id=NEW.booking\_id));

SET max\_weight = (SELECT p.max\_baggage\_weight FROM plane p WHERE p.race\_id = (SELECT r.race\_id FROM booking b, race r WHERE b.race\_id = r.race\_id AND b.booking\_id=NEW.booking\_id));

IF (isnull((SELECT SUM(b.weight) FROM baggage b WHERE b.booking\_id = NEW.booking\_id))) THEN

SET current\_sum\_weight = NEW.weight;

ELSE

SET current\_sum\_weight = (SELECT SUM(b.weight) FROM baggage b WHERE b.booking\_id = NEW.booking\_id)+NEW.weight;

END IF;

IF (current\_sum\_weight > (max\_weight/max\_passangers)) THEN

SIGNAL SQLSTATE '99999'

SET MESSAGE\_TEXT = 'Too high weight of baggages';

ELSEIF (NEW.weight < 0) THEN

SIGNAL SQLSTATE '99999'

SET MESSAGE\_TEXT = 'Invalid value';

END IF;

END //

DELIMITER ;

DROP TRIGGER IF EXISTS baggage\_after\_insert;

DELIMITER //

CREATE TRIGGER baggage\_after\_insert AFTER INSERT

ON baggage FOR EACH ROW

BEGIN

INSERT INTO `baggage\_checks` (`baggage\_id`,`result\_id`,`checked\_at`,`date\_check`,`checker\_id`) VALUES (NEW.baggage\_id,1,current\_time(),current\_date(),(SELECT RAND()\*(SELECT COUNT(checker\_id)-1 FROM checker)+1));

END //

DELIMITER ;

DROP TRIGGER IF EXISTS passangers\_insert;

DELIMITER //

CREATE TRIGGER passangers\_insert BEFORE INSERT

ON passanger FOR EACH ROW

BEGIN

IF (SELECT EXISTS(SELECT passport\_number FROM passanger WHERE passport\_number = NEW.passport\_number)) THEN

SIGNAL SQLSTATE '99999'

SET MESSAGE\_TEXT = 'Current number of passport exist!';

END IF;

END //

DELIMITER ;

DROP TRIGGER IF EXISTS passangers\_after\_insert;

DELIMITER //

CREATE TRIGGER passangers\_after\_insert AFTER INSERT

ON passanger FOR EACH ROW

BEGIN

INSERT INTO `passanger\_check` (`result\_id`,`checked\_at`,`passanger\_id`,`checker\_id`) VALUES (1,current\_time(),NEW.passanger\_id,(SELECT RAND()\*(SELECT COUNT(checker\_id)-1 FROM checker)+1));

END //

DELIMITER ;

## Створення процедур для взаємодії користувачів з базою даних

### Процедури пасажира

DROP PROCEDURE IF EXISTS make\_book;

DELIMITER //

CREATE PROCEDURE make\_book(IN passanger INT(11), IN race INT(11))

BEGIN

DECLARE num\_of\_booking INT;

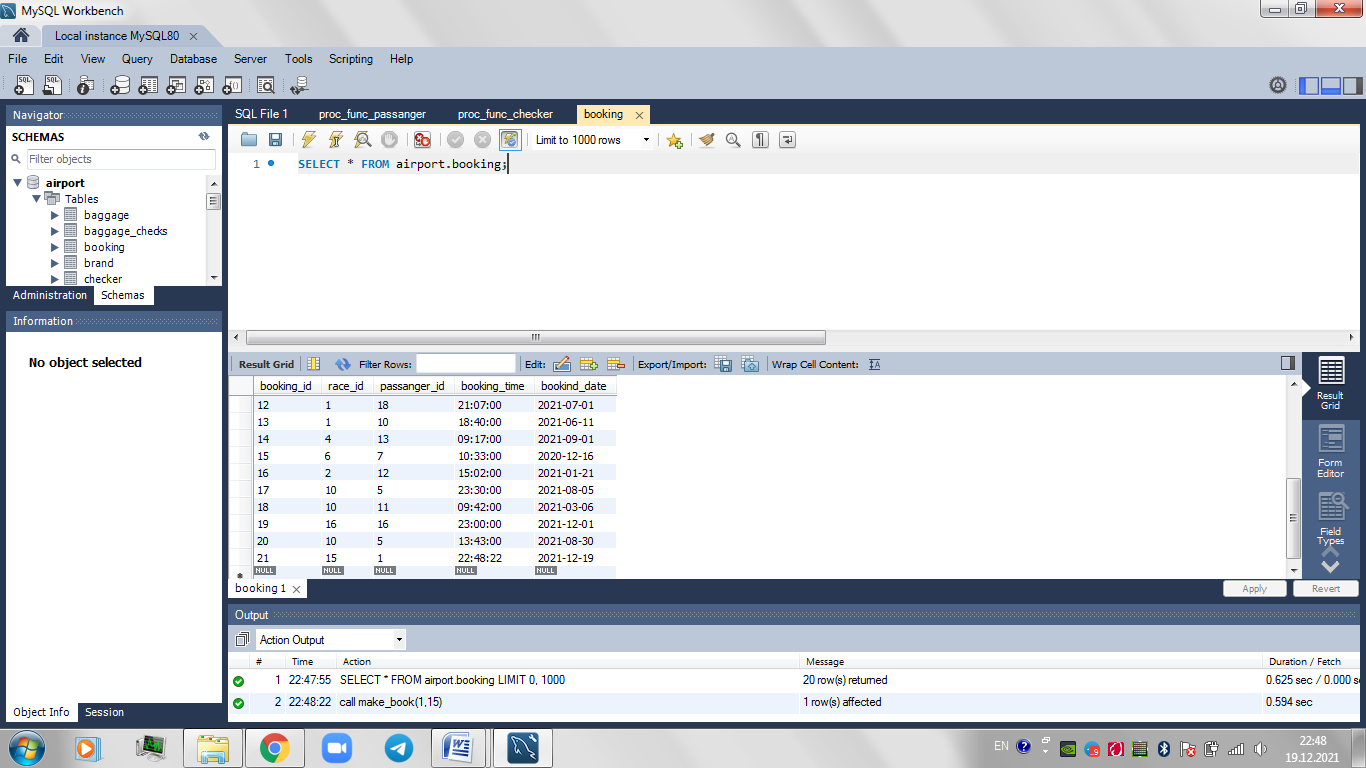
SET num\_of\_booking = (SELECT COUNT(booking\_id) FROM booking)+1;

INSERT INTO `booking` (`booking\_id`,`race\_id`,`passanger\_id`,`booking\_time`,`bookind\_date`) VALUES (num\_of\_booking,race,passanger,current\_time(),current\_date());

END //

DELIMITER ;

call make\_book(1,15)



– Приклад виконання замовлення

DROP PROCEDURE IF EXISTS add\_baggage\_to\_book;

DELIMITER //

CREATE PROCEDURE add\_baggage\_to\_book(IN passanger INT, IN booking INT(11), IN weight INT(11))

BEGIN

IF (SELECT EXISTS(SELECT booking\_id FROM booking WHERE passanger\_id = passanger AND booking\_id = booking)) THEN

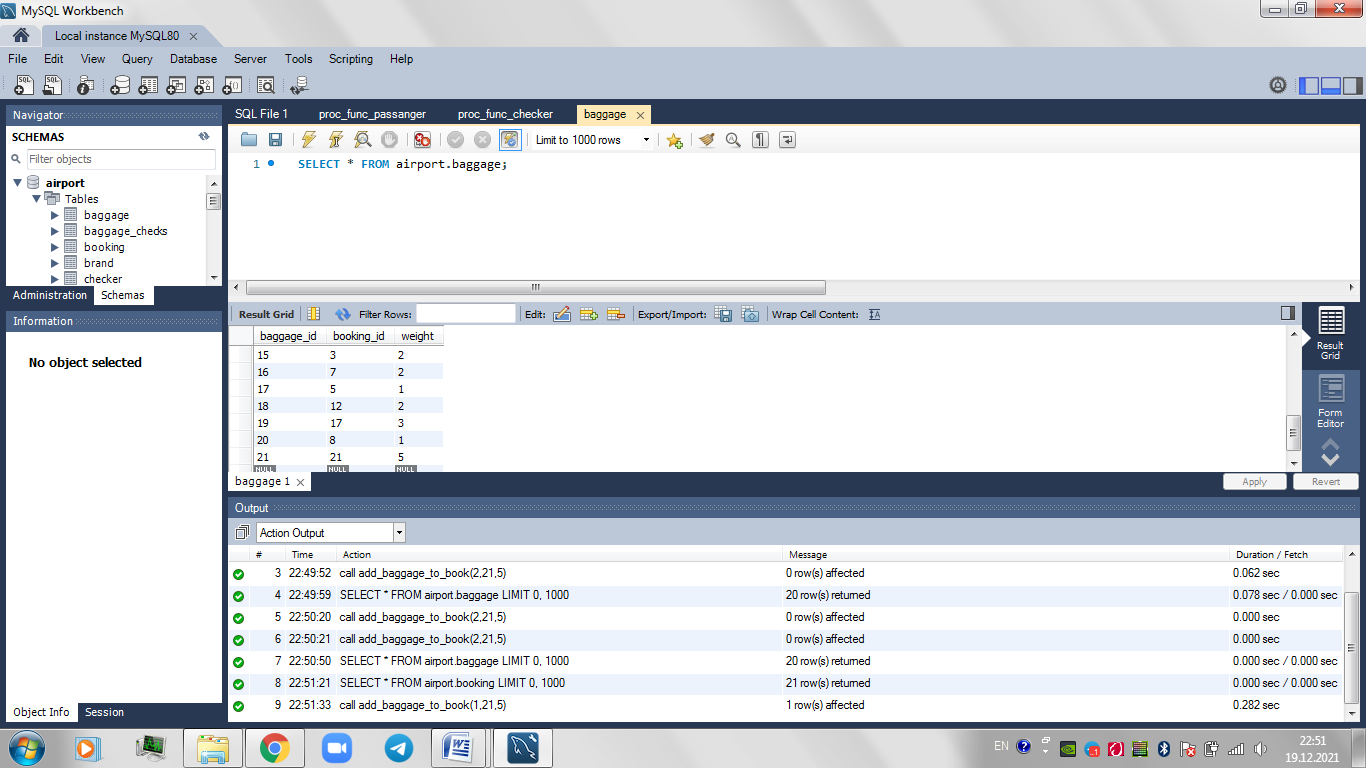
INSERT INTO `baggage` (`booking\_id`,`weight`) VALUES (booking,weight);

END IF;

END //

DELIMITER ;

call add\_baggage\_to\_book(1,21,5);



– Додавання багажу

DROP PROCEDURE IF EXISTS find\_books;

DELIMITER //

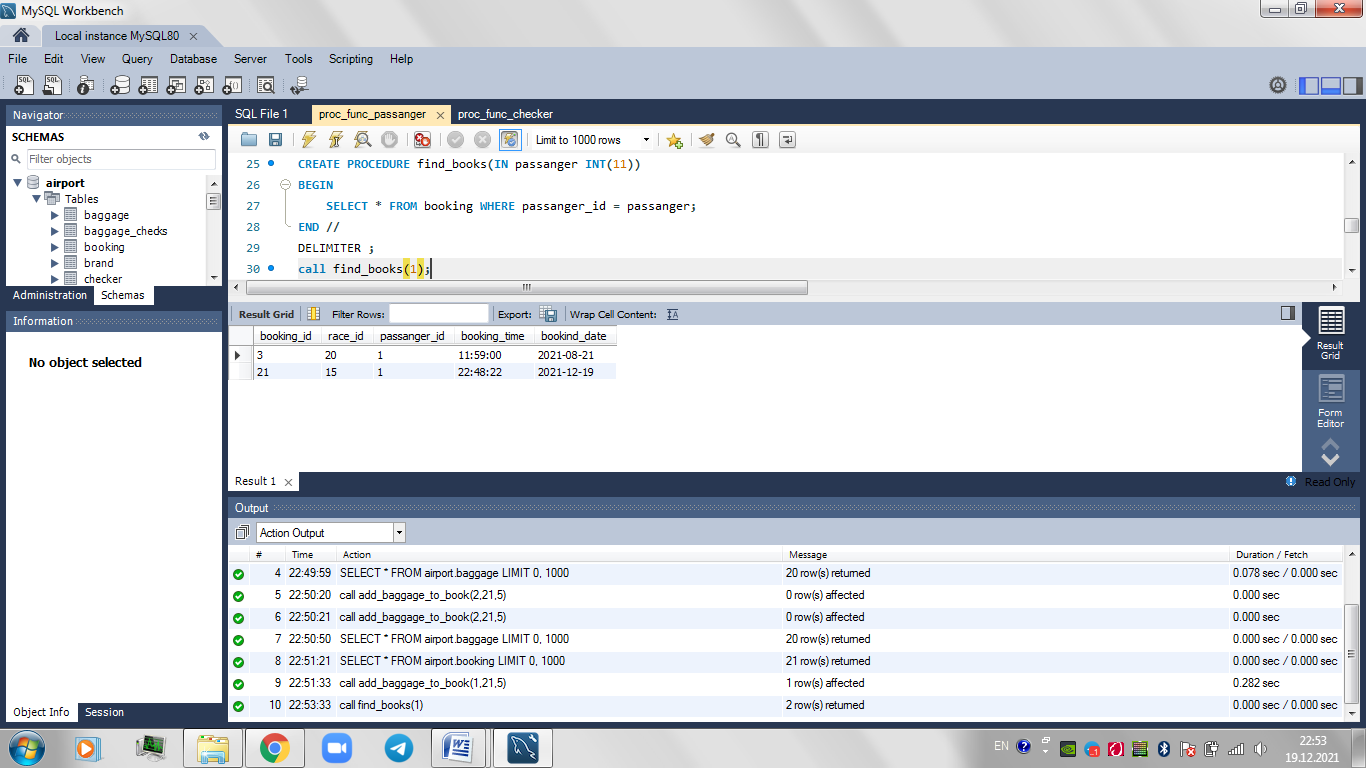
CREATE PROCEDURE find\_books(IN passanger INT(11))

BEGIN

SELECT \* FROM booking WHERE passanger\_id = passanger;

END //

DELIMITER ;



– Пошук бронювань першим пасажиром

DROP PROCEDURE IF EXISTS check\_my\_baggages;

DELIMITER //

CREATE PROCEDURE check\_my\_baggages(IN passanger INT(11))

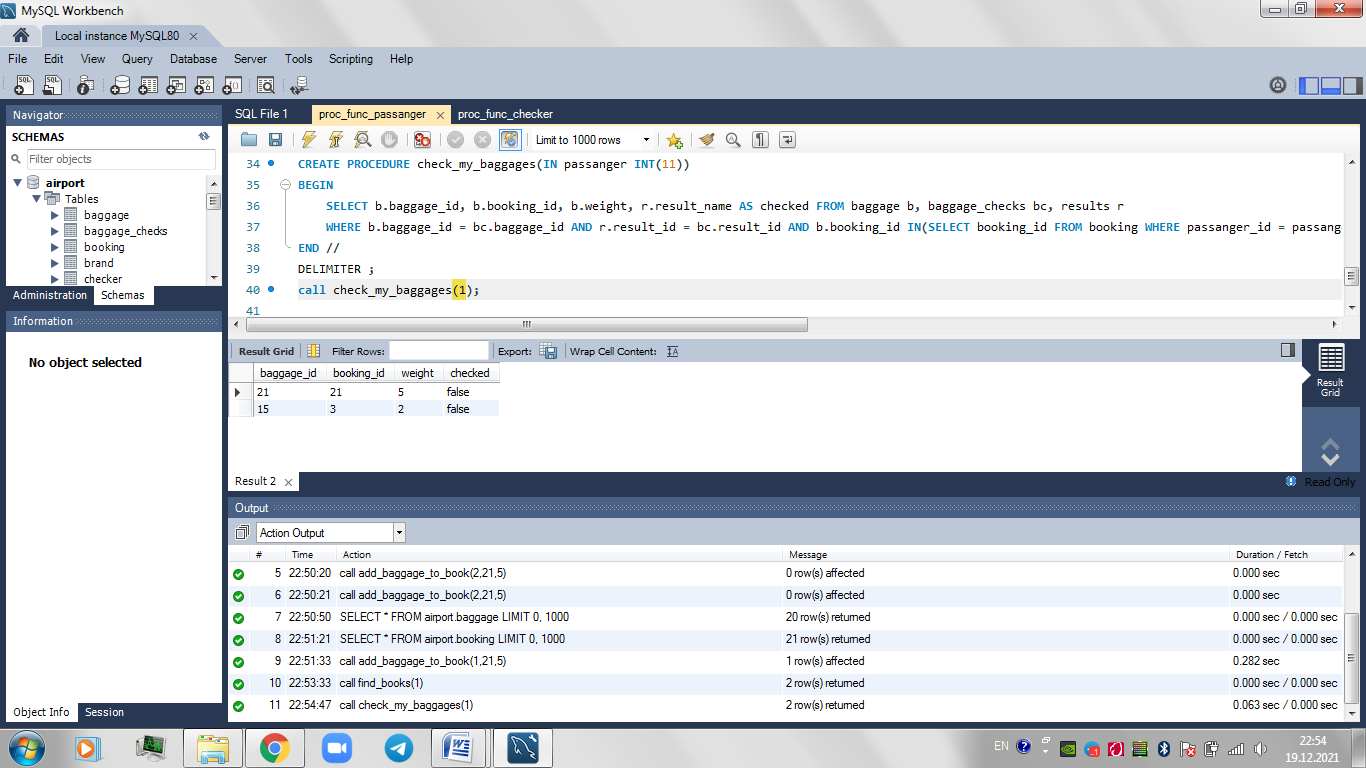
BEGIN

SELECT b.baggage\_id, b.booking\_id, b.weight, r.result\_name AS checked FROM baggage b, baggage\_checks bc, results r

WHERE b.baggage\_id = bc.baggage\_id AND r.result\_id = bc.result\_id AND b.booking\_id IN(SELECT booking\_id FROM booking WHERE passanger\_id = passanger);

END //

DELIMITER ;



– Пошук своїх багажів першим пасажиром

### Процедури перевіряльника багажу та пасажирів

DROP PROCEDURE IF EXISTS update\_result\_baggage;

DELIMITER //

CREATE PROCEDURE update\_result\_baggage(IN checker INT, IN baggage INT(11), IN result INT)

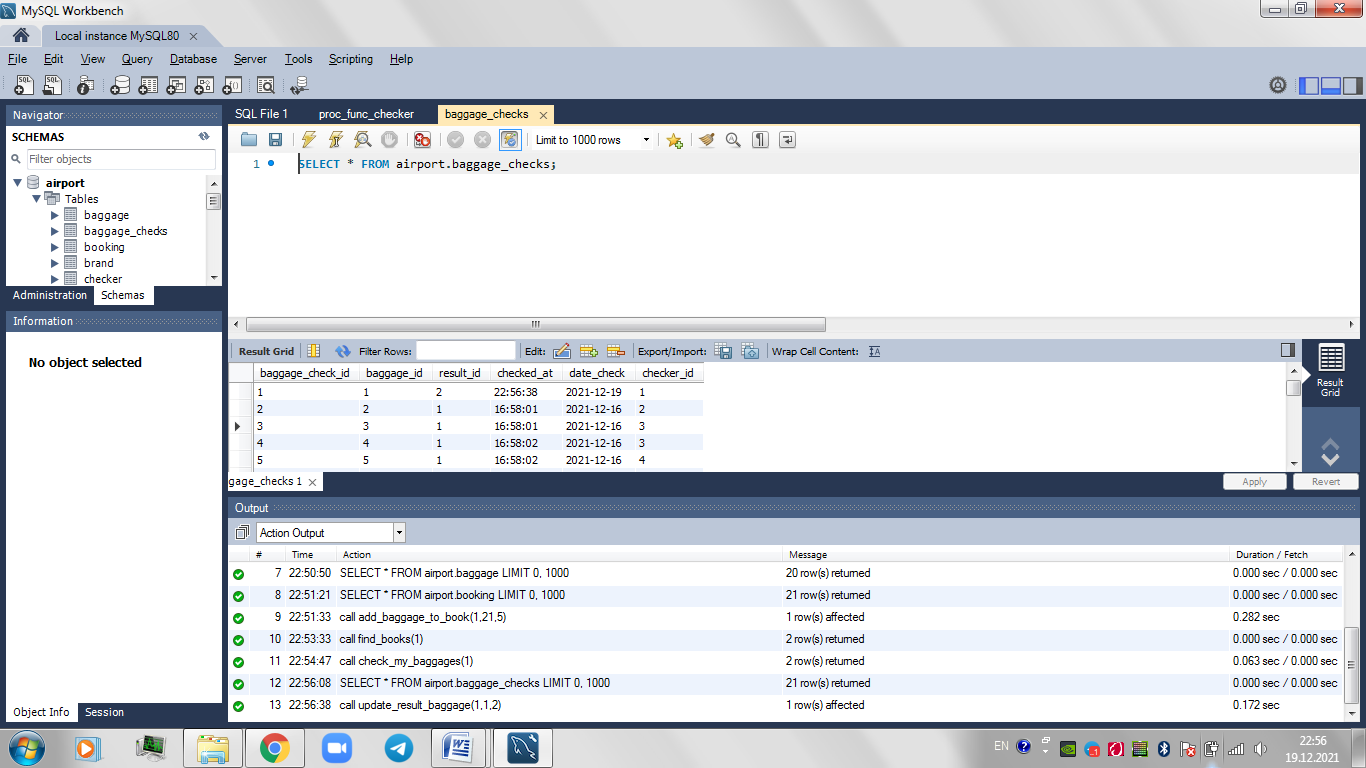
BEGIN

UPDATE baggage\_checks SET result\_id = result,checker\_id = checker WHERE baggage\_id = baggage;

END //

DELIMITER ;

call update\_result\_baggage(1,1,2);



– Оновлення даних про переіфрку багажу

DROP PROCEDURE IF EXISTS update\_result\_passanger;

DELIMITER //

CREATE PROCEDURE update\_result\_passanger(IN checker INT, IN passanger INT(11), IN result INT)

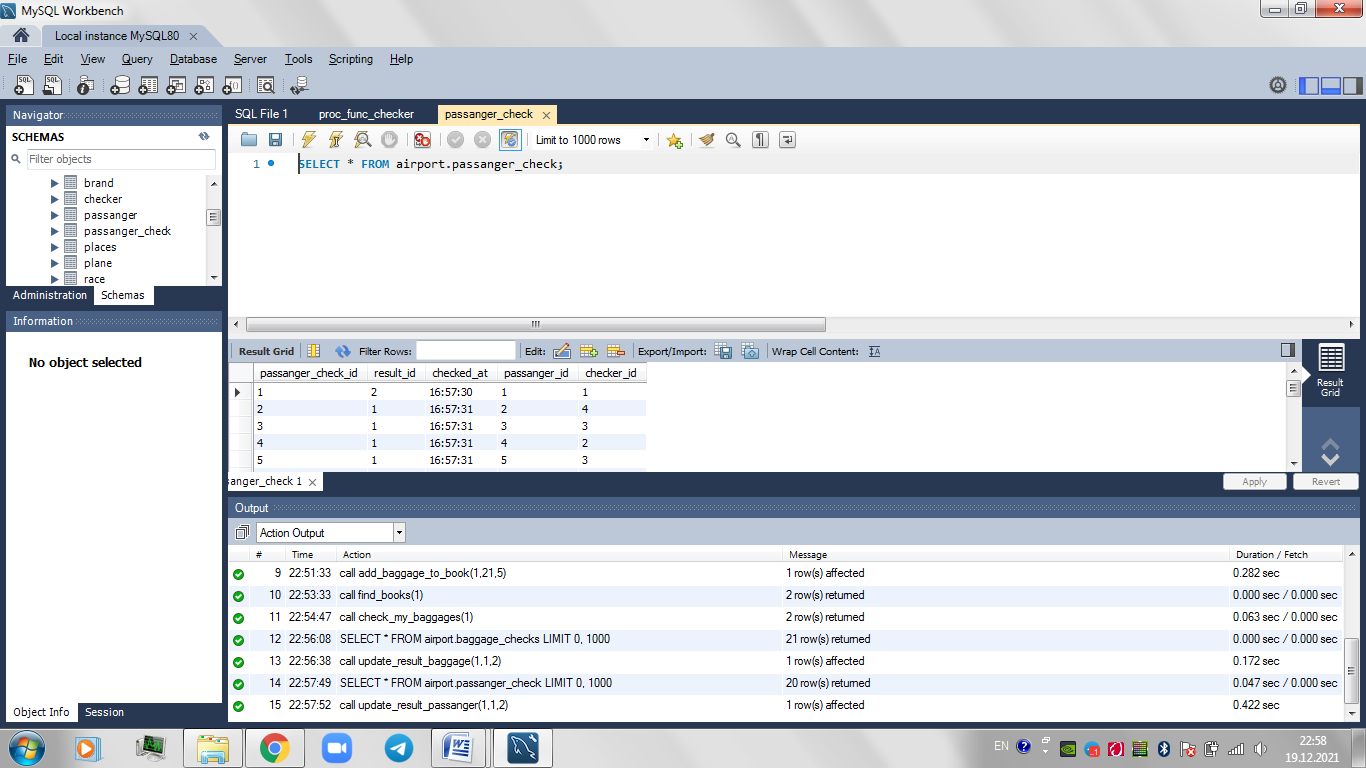
BEGIN

UPDATE passanger\_check SET result\_id = result, checker\_id = checker WHERE passanger\_id = passanger;

END //

DELIMITER ;

call update\_result\_passanger(1,1,2);



– Оновлення даних про перевірку пасажира

DROP PROCEDURE IF EXISTS list\_passanger;

DELIMITER //

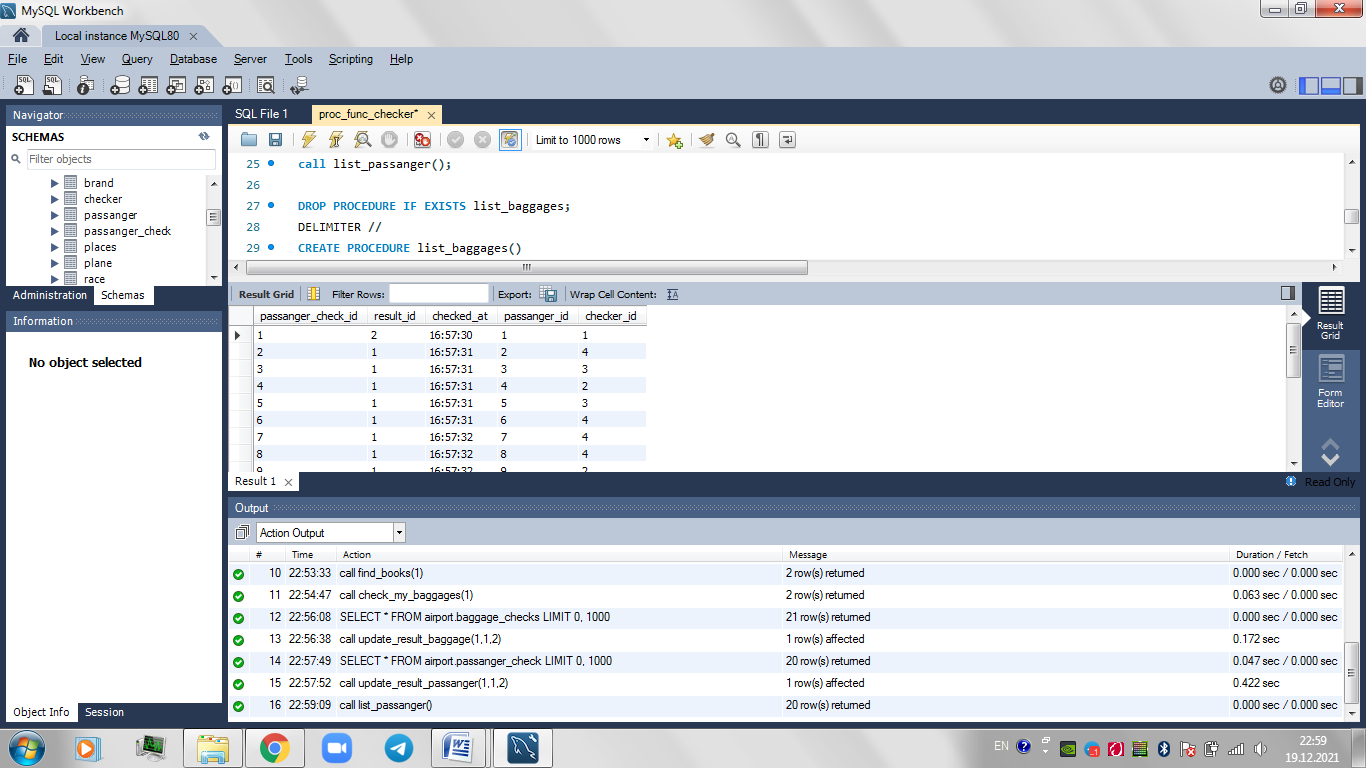
CREATE PROCEDURE list\_passanger()

BEGIN

SELECT \* FROM passanger\_check;

END //

DELIMITER ;



– Перегляд списку пасажирів

DROP PROCEDURE IF EXISTS list\_baggages;

DELIMITER //

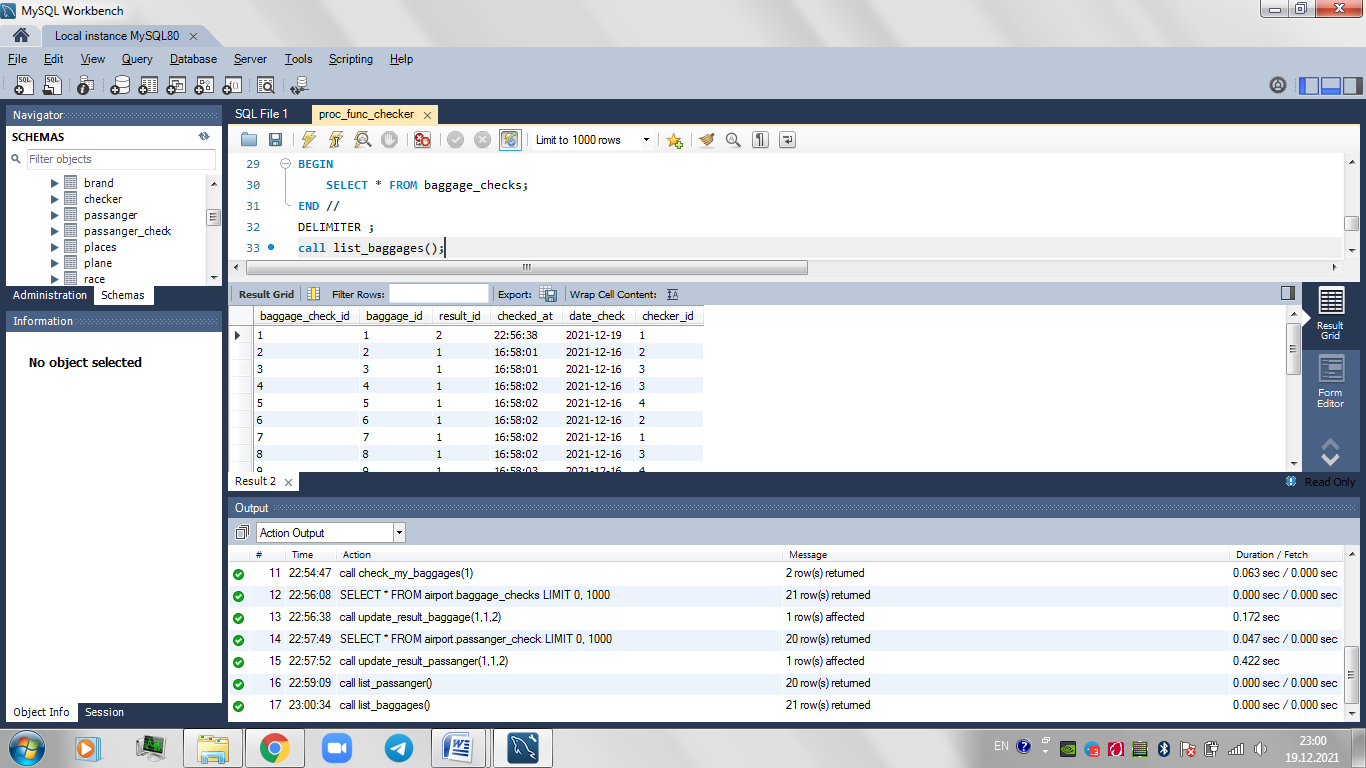
CREATE PROCEDURE list\_baggages()

BEGIN

SELECT \* FROM baggage\_checks;

END //

DELIMITER ;



– Перегляд списку багажів

### Інші sql запити

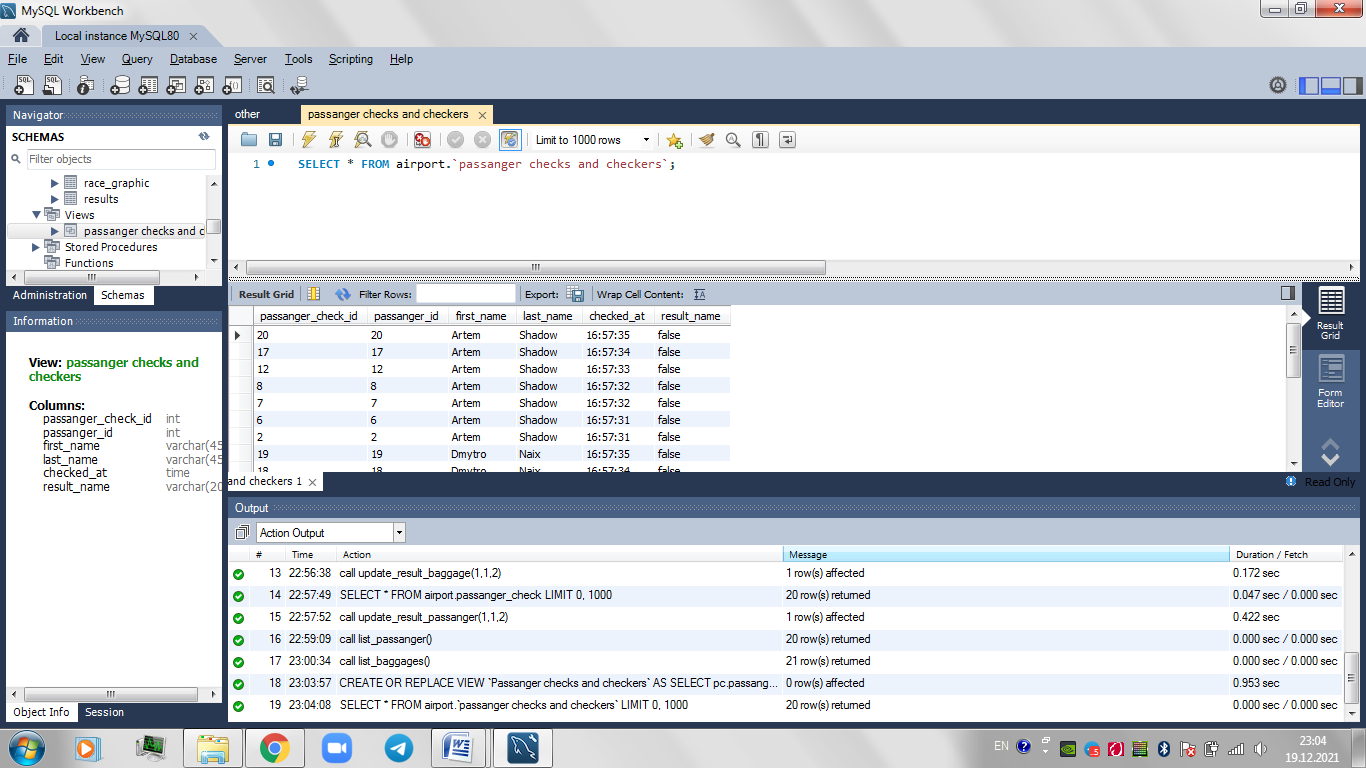
Створення VIEW для відображення номеру перевірки пасажира, її результат та час, і ім’я перевіряю чого.

CREATE OR REPLACE VIEW `Passanger checks and checkers` AS

SELECT pc.passanger\_check\_id, passanger\_id, c.first\_name, c.last\_name, pc.checked\_at, r.result\_name

FROM checker c, passanger\_check pc, results r

WHERE c.checker\_id = pc.checker\_id AND pc.result\_id = r.result\_id



– Запит представлення

Створення запиту з використанням INNER та OUTER JOIN для того, щоб дізнатись, результати перевірки пасажирів та тих хто їх перевіряв.

SELECT myData.passanger\_check\_id, myData.result\_name, CONCAT(myData.first\_name,' ',myDAta.last\_name) AS checker\_full\_name

FROM (

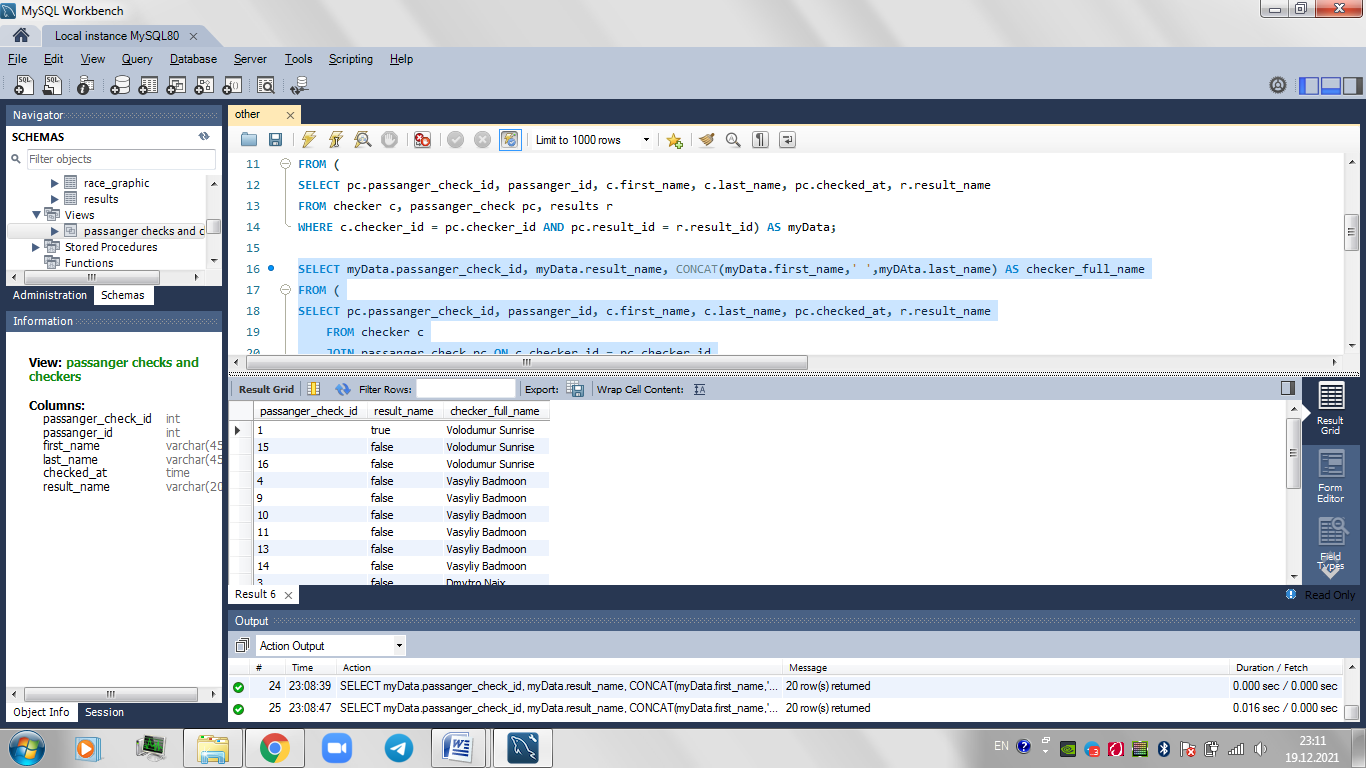
SELECT pc.passanger\_check\_id, passanger\_id, c.first\_name, c.last\_name, pc.checked\_at, r.result\_name

FROM checker c

JOIN passanger\_check pc ON c.checker\_id = pc.checker\_id

LEFT JOIN results r ON pc.result\_id = r.result\_id

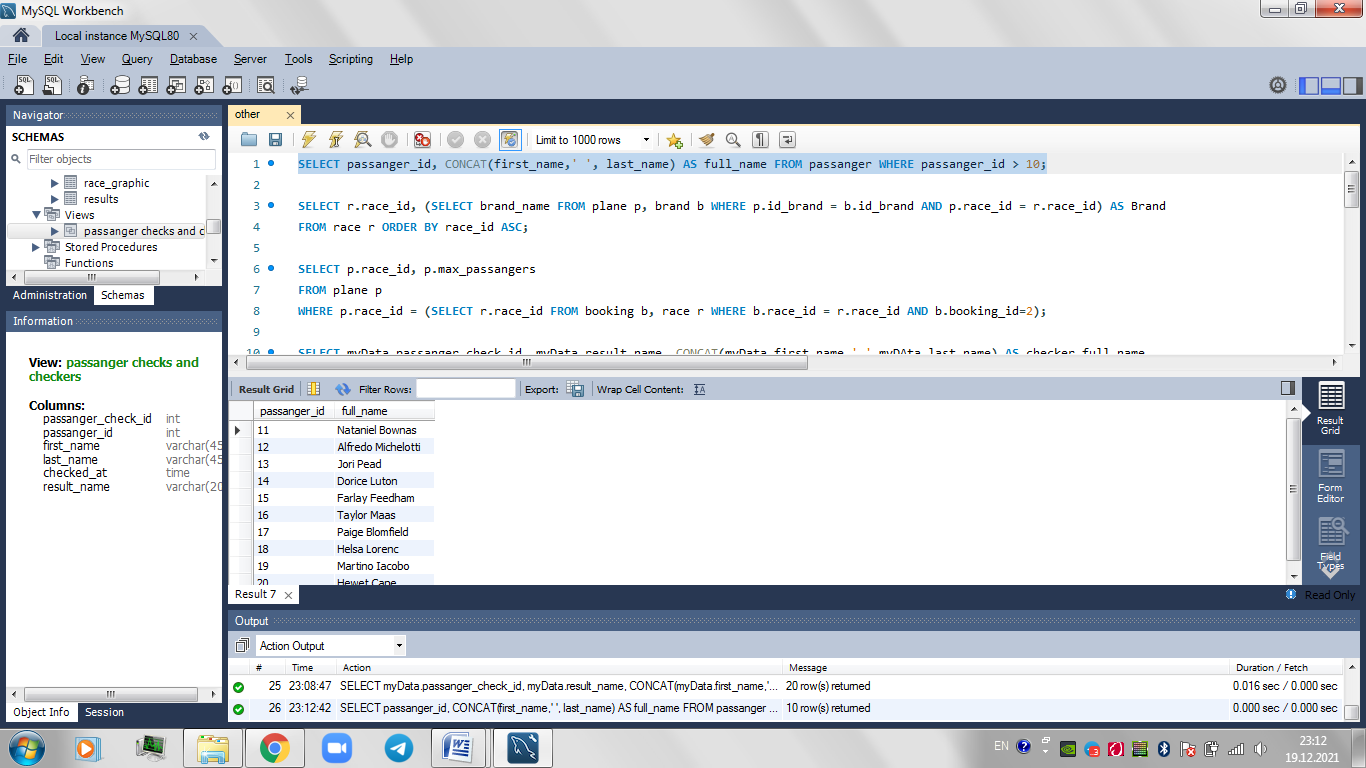
) AS myData;



– Результат запиту

Створення запиту з використанням функції CONCAT для визначення імен пасажирів, які зареєструвалися пізніше 10 пасажира.

SELECT passanger\_id, CONCAT(first\_name,' ', last\_name) AS full\_name FROM passanger WHERE passanger\_id > 10;



– Результат запиту

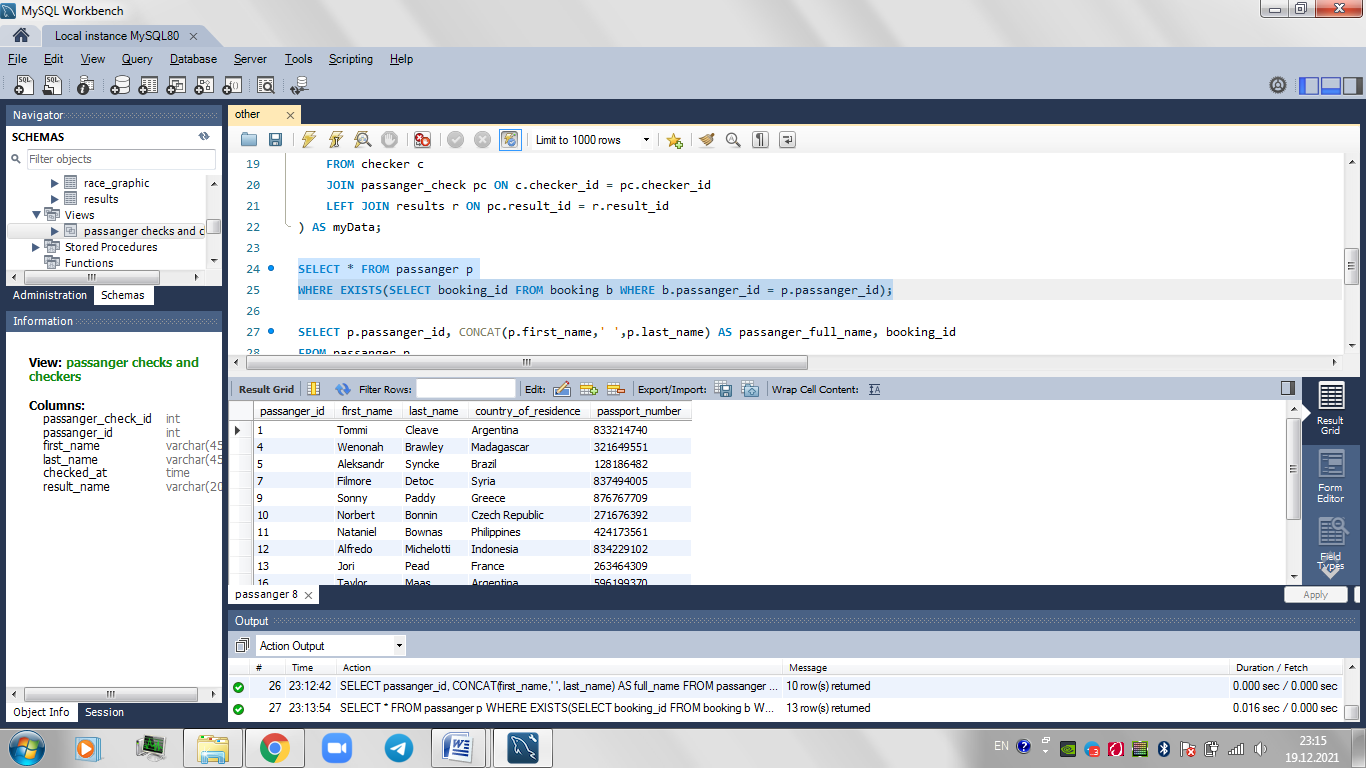
Запит для знаходження інформації про пасажирів, які зробили хоч одне бронювання

SELECT \* FROM passanger p

WHERE EXISTS(SELECT booking\_id FROM booking b WHERE b.passanger\_id = p.passanger\_id);

SELECT \* FROM passanger p

WHERE IN(SELECT booking\_id FROM booking b WHERE b.passanger\_id = p.passanger\_id);

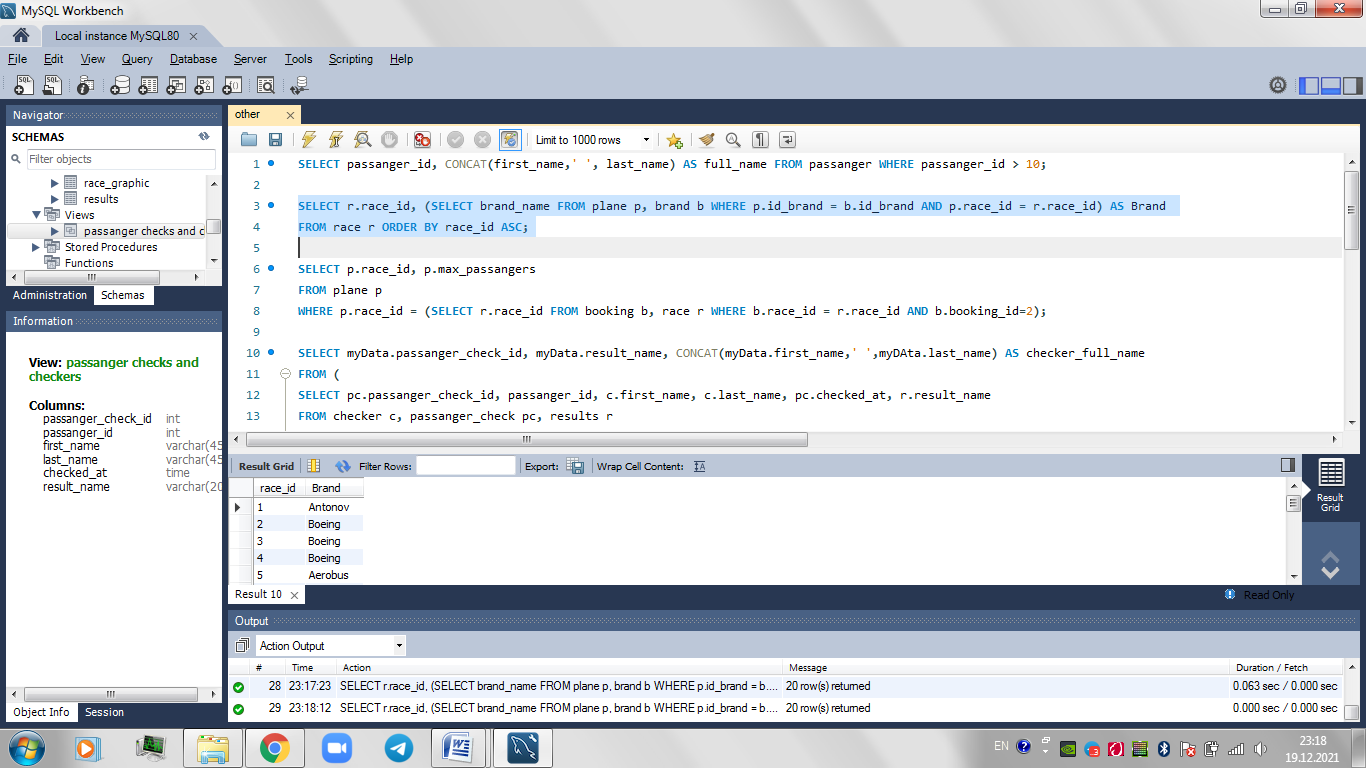


– Результат запиту

Створення запиту для пошуку бренда літака на рейсі

SELECT r.race\_id, (SELECT brand\_name FROM plane p, brand b WHERE p.id\_brand = b.id\_brand AND p.race\_id = r.race\_id) AS Brand

FROM race r ORDER BY race\_id ASC;



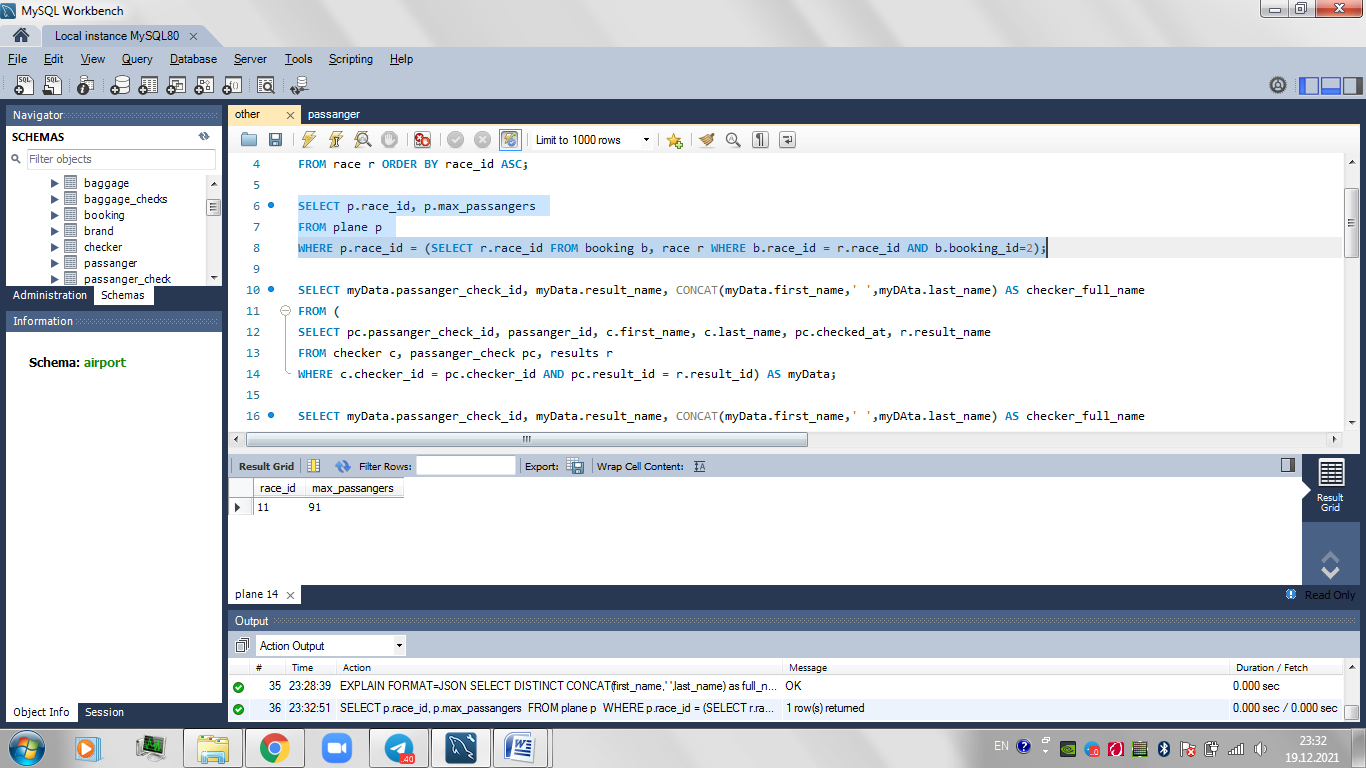
– Результат запиту

Запит для знаходження максимальної кількості пасажирів

SELECT p.race\_id, p.max\_passangers

FROM plane p

WHERE p.race\_id = (SELECT r.race\_id FROM booking b, race r WHERE b.race\_id = r.race\_id AND b.booking\_id=2);



– Результат запиту

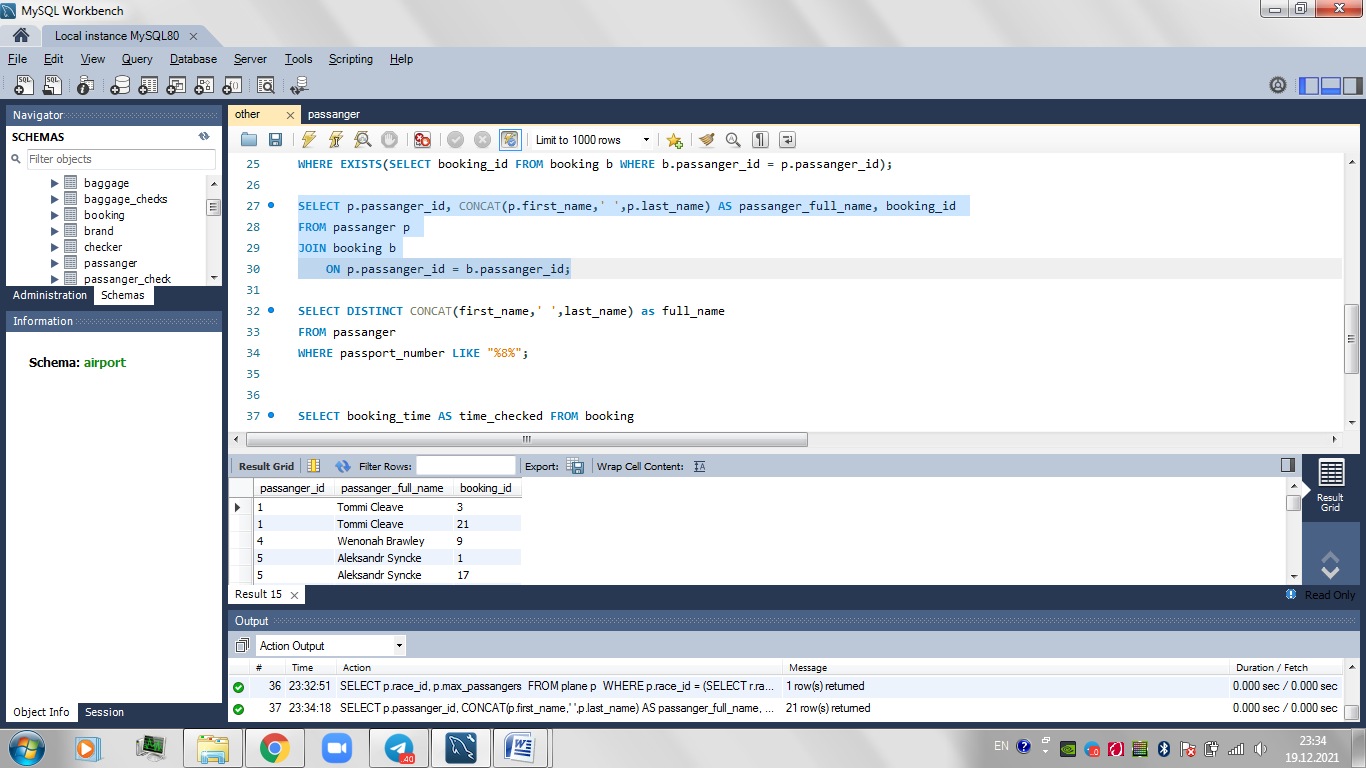
Запит для знаходження пасажирів та їх бронювань

SELECT p.passanger\_id, CONCAT(p.first\_name,' ',p.last\_name) AS passanger\_full\_name, booking\_id

FROM passanger p

JOIN booking b

ON p.passanger\_id = b.passanger\_id;



– Результат запиту

## Приклад роботи індексів

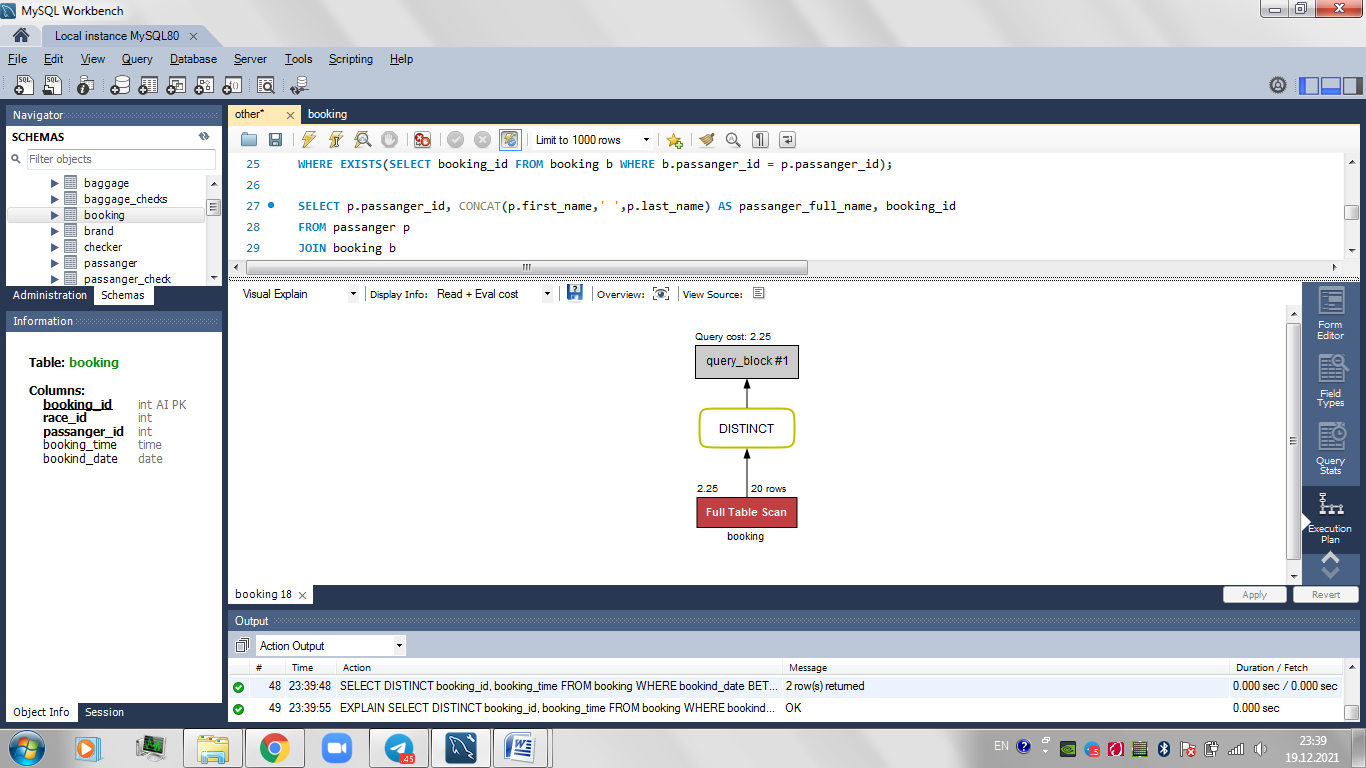
Для прикладу розглянемоо простий SQL запит типу:

SELECT DISTINCT booking\_id, booking\_time

FROM booking

WHERE bookind\_date BETWEEN CONVERT('2000-01-01',DATE) AND CONVERT('2021-01-01',DATE);

З допомогою даного запиту ми можемо обрати бронювання виконані, між 2000-01-01 та 2021-01-01. Схема даного запиту без використання індексів виглядає так:

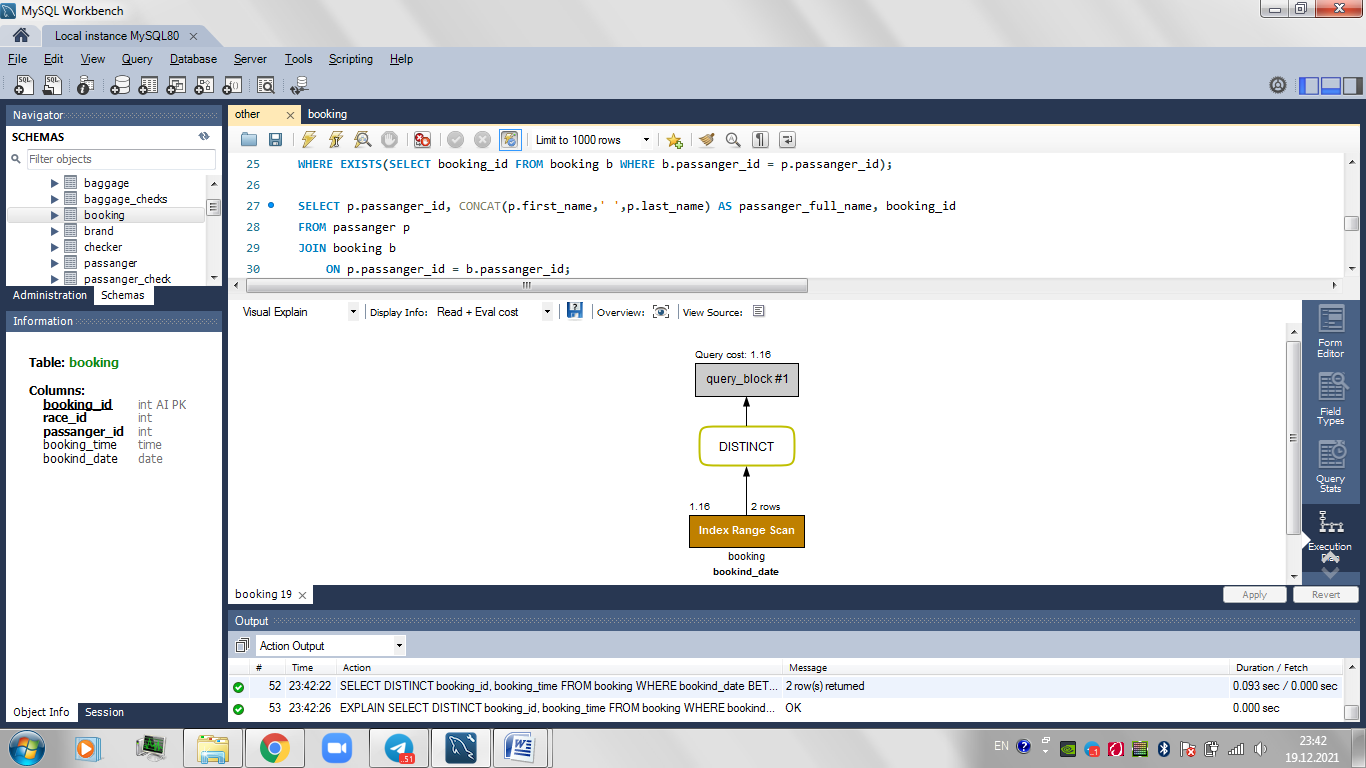


– Схема запиту без використання індексів

А зараз створимо індекс на поле `booking\_date` виконивши такий скрипт:

CREATE INDEX bookind\_date ON booking(bookind\_date);

Після повторного виконання запиту уже із створеним індексом отримаємо таку схему:



– Схема запиту з використанням індексів

Бачимо, що для результату без та з використанням індексів потрібно перевірити різну к-сть рядків:

* для запиту без використання індексів 20 рядків
* для запиту з використанням індексів 2 рядків

Ефективність збільшилась в 10. Отже використання індексів – невід’ємна частина роботи з базою даних.

Висновок

Створення бази даних – це один із фундаментів для початку ефективного та продуктивного бізнесу, адже від якості групування даних напряму залежать можливості обробки даних та кількість реалізованих одиниць товару.

У даній роботі було спроектовано базу даних для аеропорту. Спочатку було вивчено та описано предметне середовище, після чого визначені задачі для виконання.

Наступним кроком було створення ER-моделі бази даних, для кращого розуміння та представлення, які сутності будуть брати участь в роботі аеропорту.

Після чого, було створено реляційну модель бази даних приведену до третьої нормальної форми.

В завершення, для створення бази даних було виконано:

* написано SQL скрипти для створення таблиць
* написано тригери для перевірки коректності введених даних, де це необхідно
* створено користувачів бази даних та функції для них

Перелік посилань

1. <https://it-black.ru/manipuljacija-s-polzovateljami-v-mysql-workbench/>
2. <https://habr.com/ru/post/47031/>
3. <https://dev.mysql.com/doc/refman/8.0/en/functions.html>
4. <https://studopedia.com.ua/1_204901_korotka-harakteristika-strukturi-i-funktsiy-medichnih-zakladiv.html>
5. <https://ips.ligazakon.net/document/JF78Q00A?an=49>
6. <https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html>
7. <https://dev.mysql.com/doc/>
8. https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html