

**Національний технічний університет України**

**“Київський політехнічний інститут”**

**Факультет прикладної математики**

**Кафедра системного програмування і спеціалізованих комп’ютерних систем**

**Лабораторна робота №3**

***з дисципліни “* *Бази даних і засоби управління”***

*Тема: “Засоби оптимізації роботи СУБД PostgreSQL”*

Виконав:

студент ІII курсу

групи КВ-01

Таранич Артем

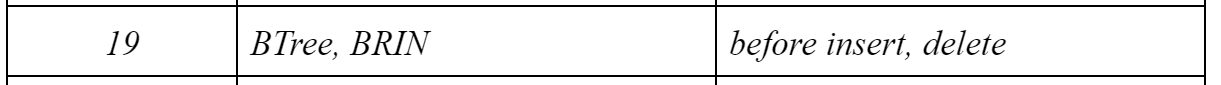
Перевірив: Павловський В. І.

Київ – 2022

Завдання роботи полягає у наступному:

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

**Варіант 19**

****

**Логічна модель предметної області «Аеропорт»**

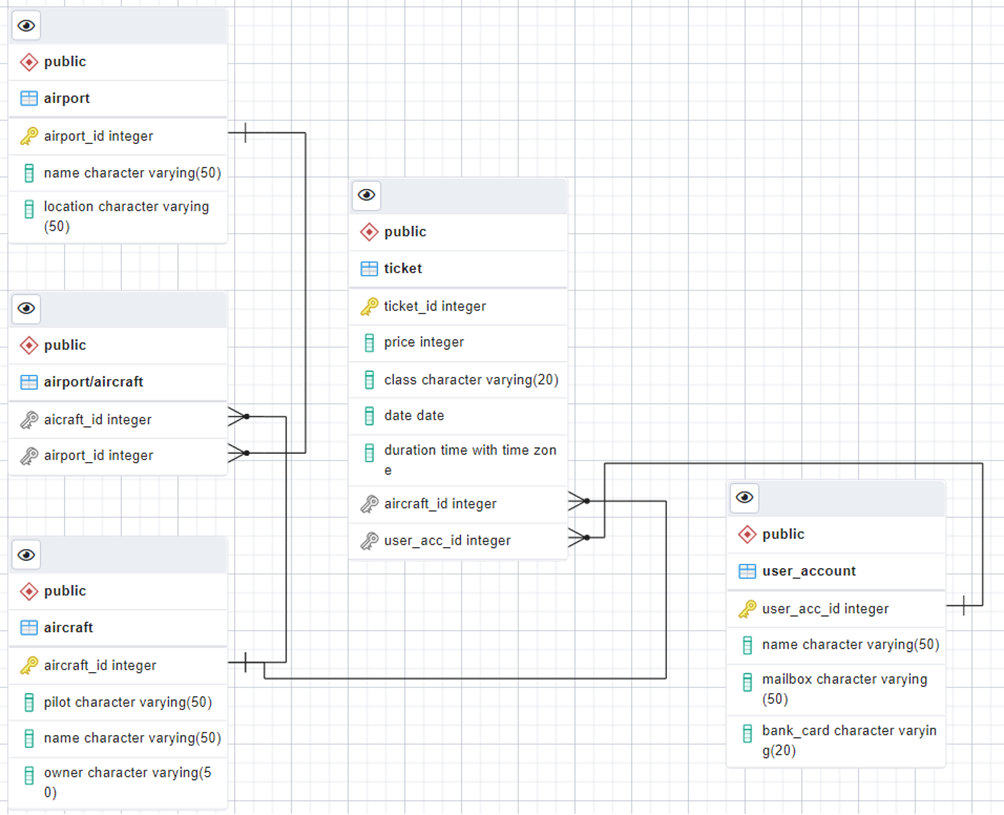


Рисунок 1. Схема бази даних, побудовано у [p](https://app.diagrams.net/)gAdmin 4.

**Завдання 1**

**Класи ORM**

**class** Airport(Base):

\_\_tablename\_\_ = 'airport'

columns = ['airport\_id','name','location']

airport\_id = Column(Integer, primary\_key=True)

name = Column(String)

location = Column(String)

**def** \_\_init\_\_(self, name, location):

self.name = name

self.location = location

super(Airport, self).\_\_init\_\_()

**def** id(self):

**return** self.airport\_id

**class** Aircraft(Base):

\_\_tablename\_\_ = 'aircraft'

columns = ['aircraft\_id','pilot','name','owner']

addition\_keys = ['airport\_id']

aircraft\_id = Column(Integer, primary\_key=True)

pilot = Column(String)

name = Column(String)

owner = Column(String)

**def** \_\_init\_\_(self, pilot, name, owner):

self.pilot = pilot

self.name = name

self.owner = owner

super(Aircraft, self).\_\_init\_\_()

**def** id(self):

**return** self.aircraft\_id

**class** AirportAircraft(Base):

\_\_tablename\_\_ = 'airport\_aircraft'

columns = ['aircraft\_id','airport\_id']

aircraft\_id = Column(Integer, ForeignKey('aircraft.aircraft\_id'), primary\_key=True)

airport\_id = Column(Integer, ForeignKey('airport.airport\_id'), primary\_key=True)

**def** \_\_init\_\_(self, aircraft\_id, airport\_id):

self.aircraft\_id = aircraft\_id

self.airport\_id = airport\_id

super(AirportAircraft, self).\_\_init\_\_()

**def** id(self):

**return** self.aircraft\_id

**class** Ticket(Base):

\_\_tablename\_\_ = 'ticket'

columns = ['ticket\_id','price','seat\_type','date','duration','aircraft\_id','user\_acc\_id']

ticket\_id = Column(Integer, primary\_key=True)

price = Column(Integer)

seat\_type = Column(String)

date = Column(String)

duration = Column(Integer)

aircraft\_id = Column(Integer, ForeignKey('aircraft.aircraft\_id'))

user\_acc\_id = Column(Integer, ForeignKey('user\_account.user\_acc\_id'))

**def** \_\_init\_\_(self, price, seat\_type, date, duration, aircraft\_id, user\_acc\_id):

self.price = price

self.seat\_type = seat\_type

self.date = date

self.duration = duration

self.aircraft\_id = aircraft\_id

self.user\_acc\_id = user\_acc\_id

super(Ticket, self).\_\_init\_\_()

**def** id(self):

**return** self.ticket\_id

**class** UserAccount(Base):

\_\_tablename\_\_ = 'user\_account'

columns = ['user\_acc\_id','name','mailbox','bank\_card']

user\_acc\_id = Column(Integer, primary\_key=True)

name = Column(String)

mailbox = Column(String)

bank\_card = Column(String)

**def** \_\_init\_\_(self, name, mailbox, bank\_card):

self.name = name

self.mailbox = mailbox

self.bank\_card = bank\_card

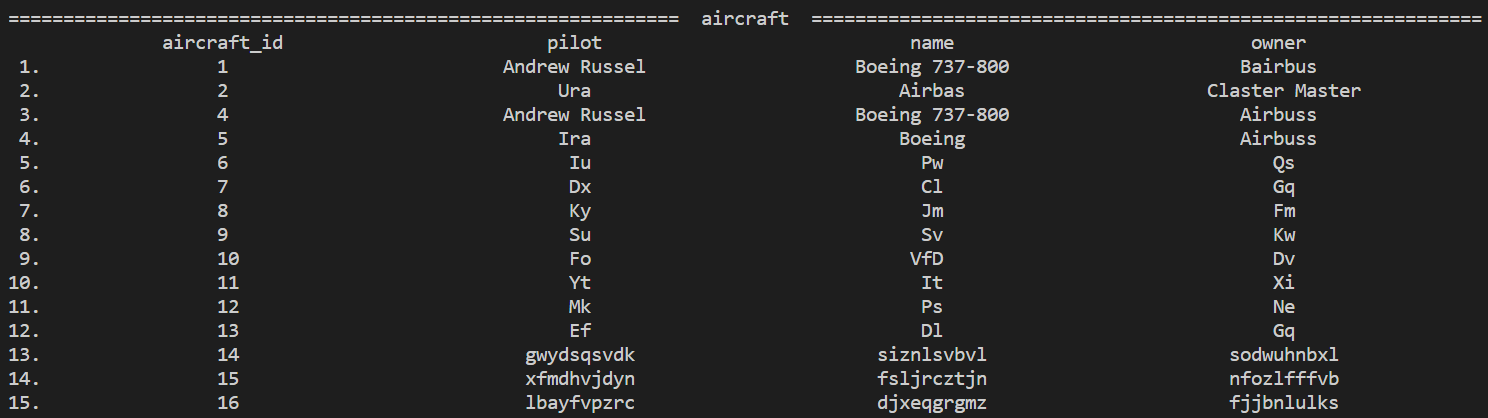
super(UserAccount, self).\_\_init\_\_()

**def** id(self):

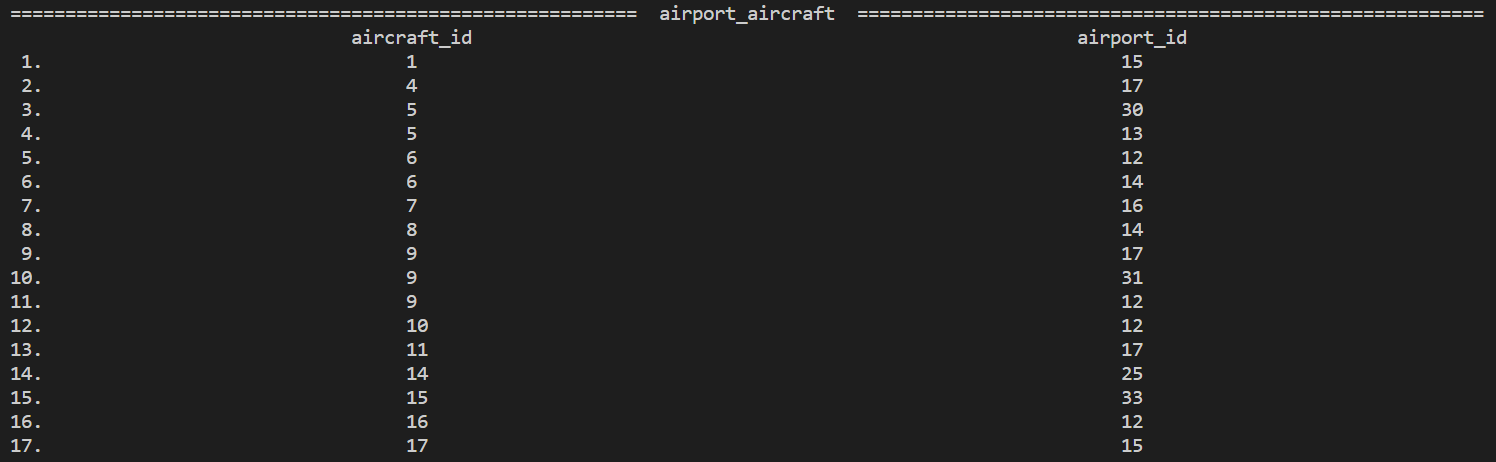
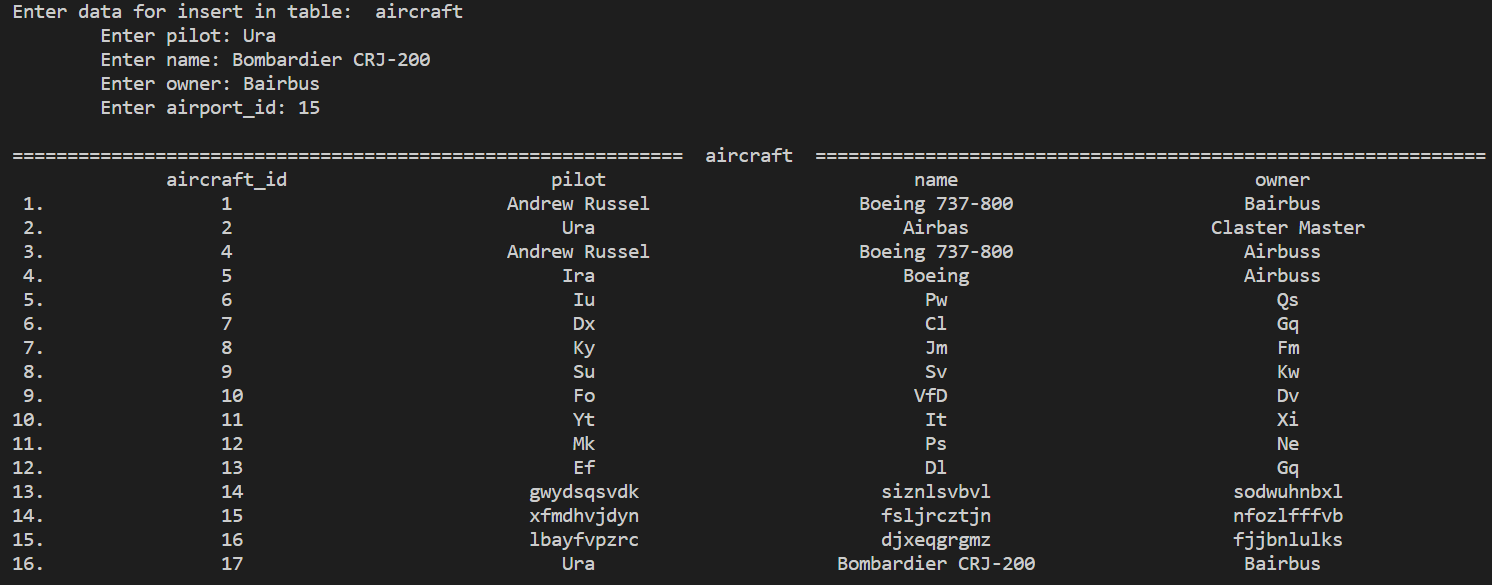
**return** self.user\_acc\_id

**Результати та виконання операцій**

**Виведення таблиці ‘Aircraft’ в консоль**

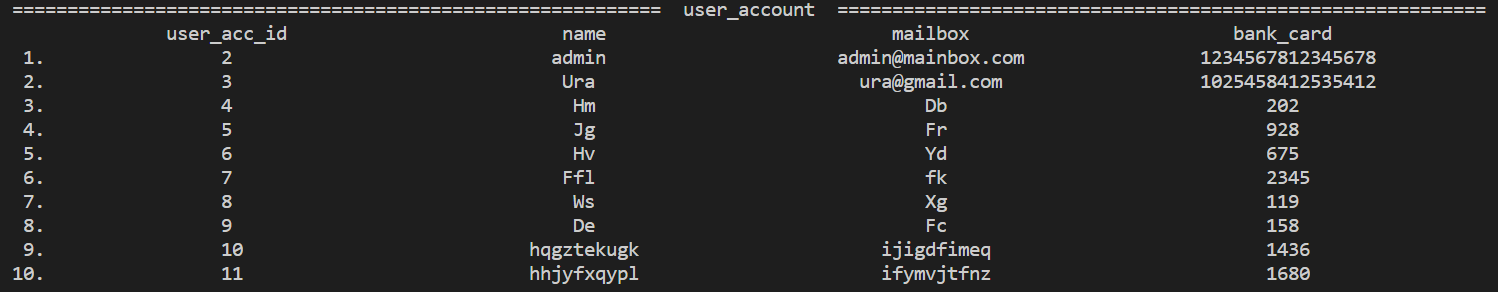
****

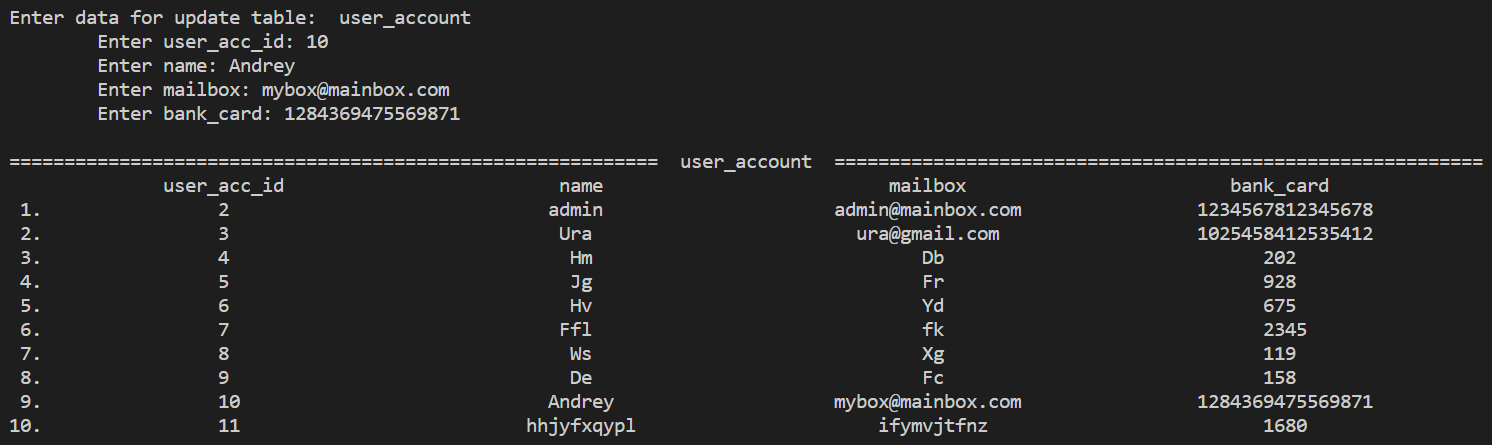
**Додавання нового рядку до таблиці “Aircraft”**

В результаті дані також оновилися в зв’язаній таблиці “Airport/Aircraft”

**Оновлення рядку у таблиці “Users Account”**

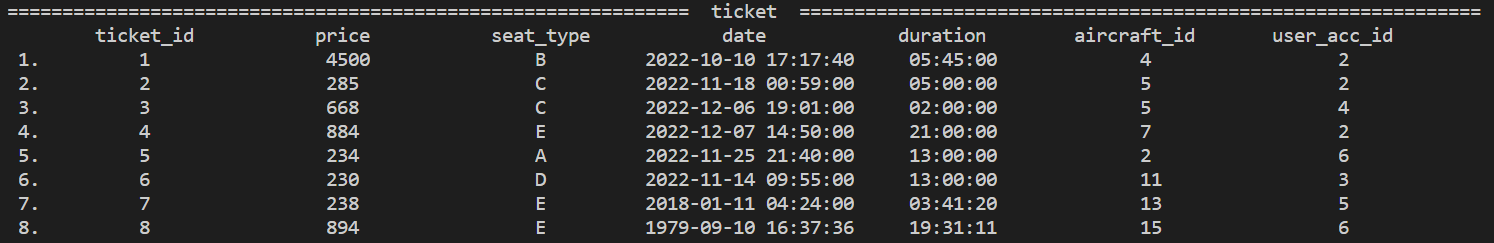
До

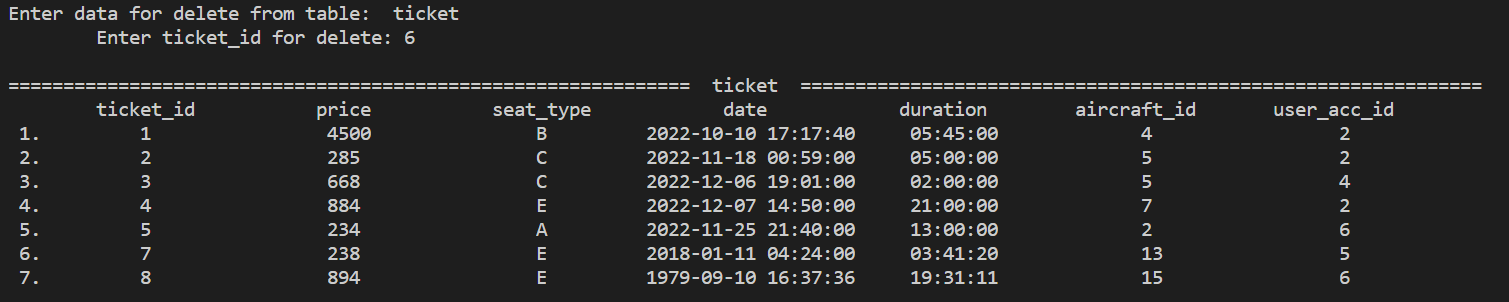
Після



**Видалення рядку з таблиці “Ticket”**

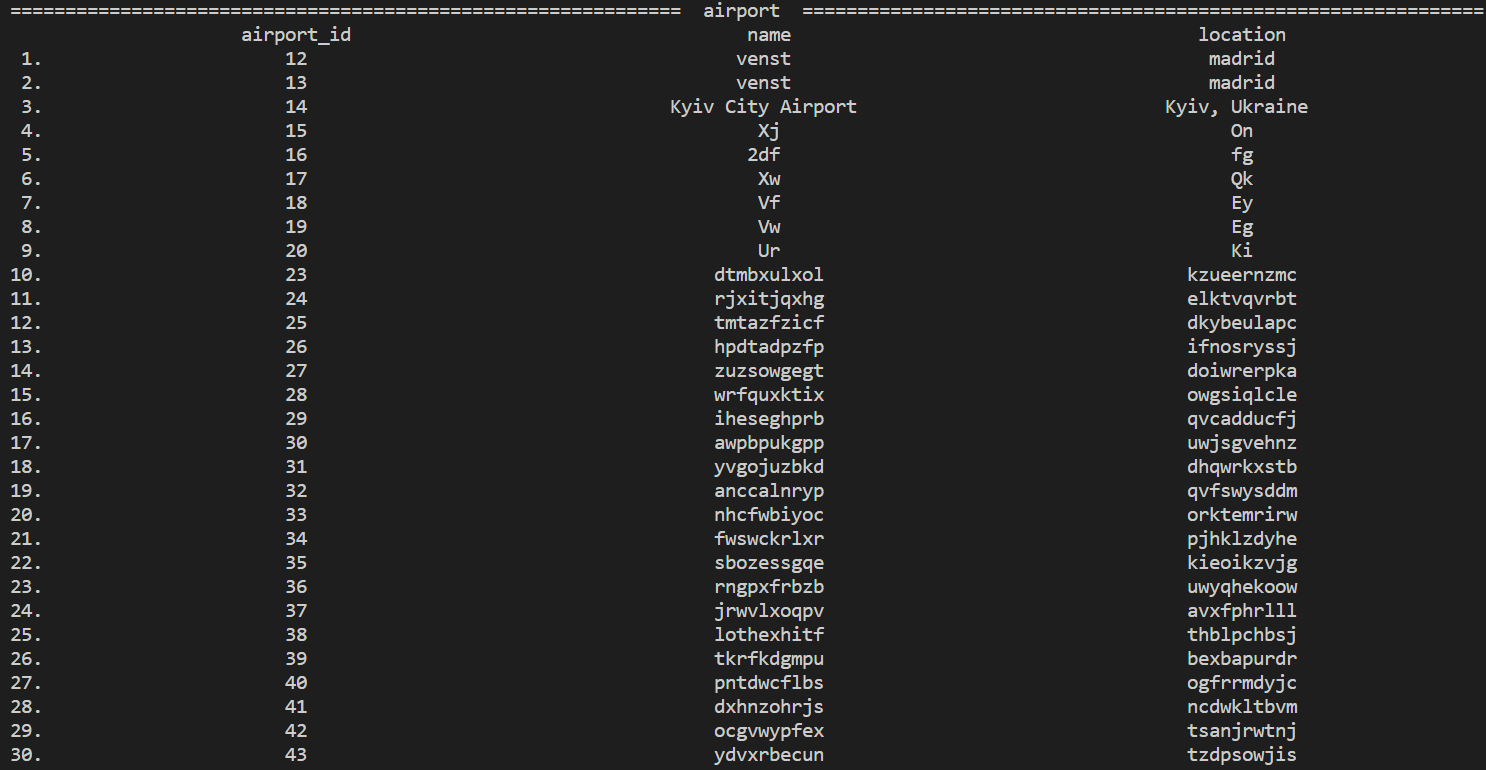
До

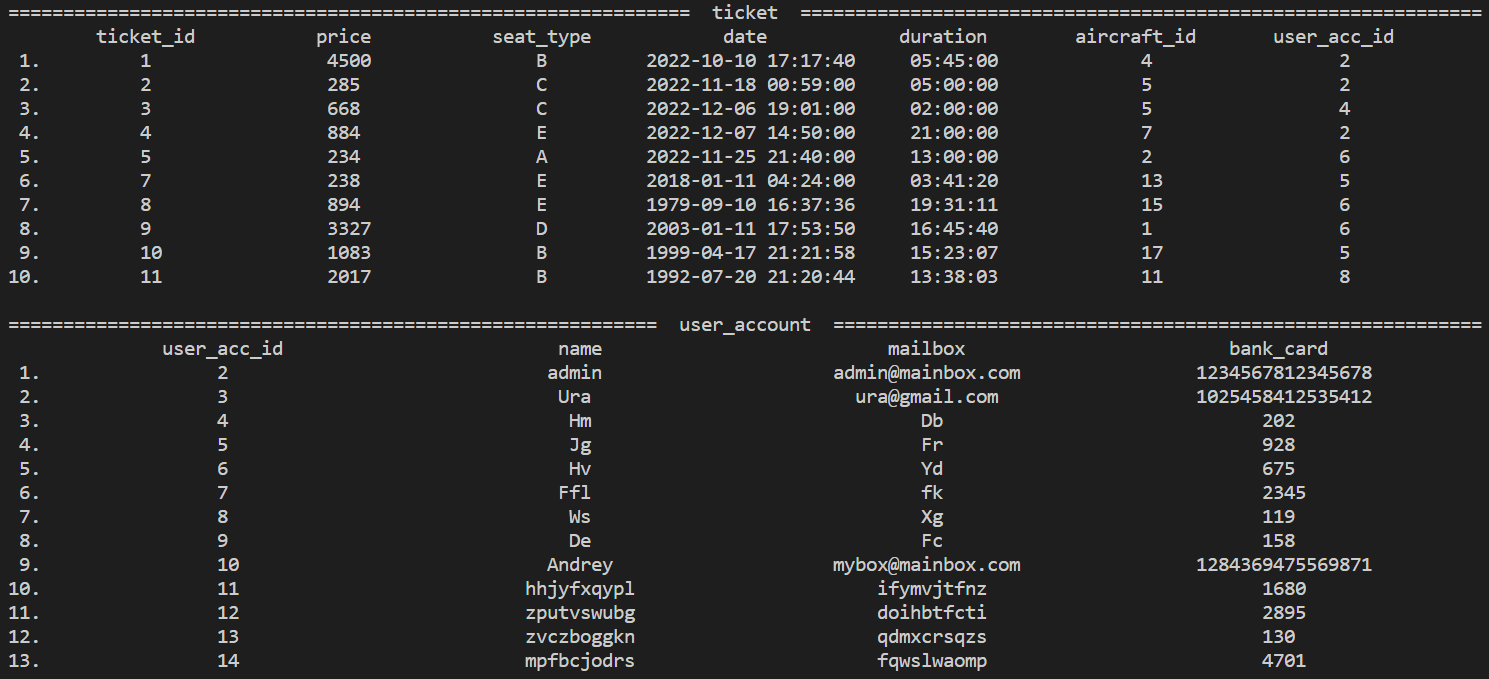
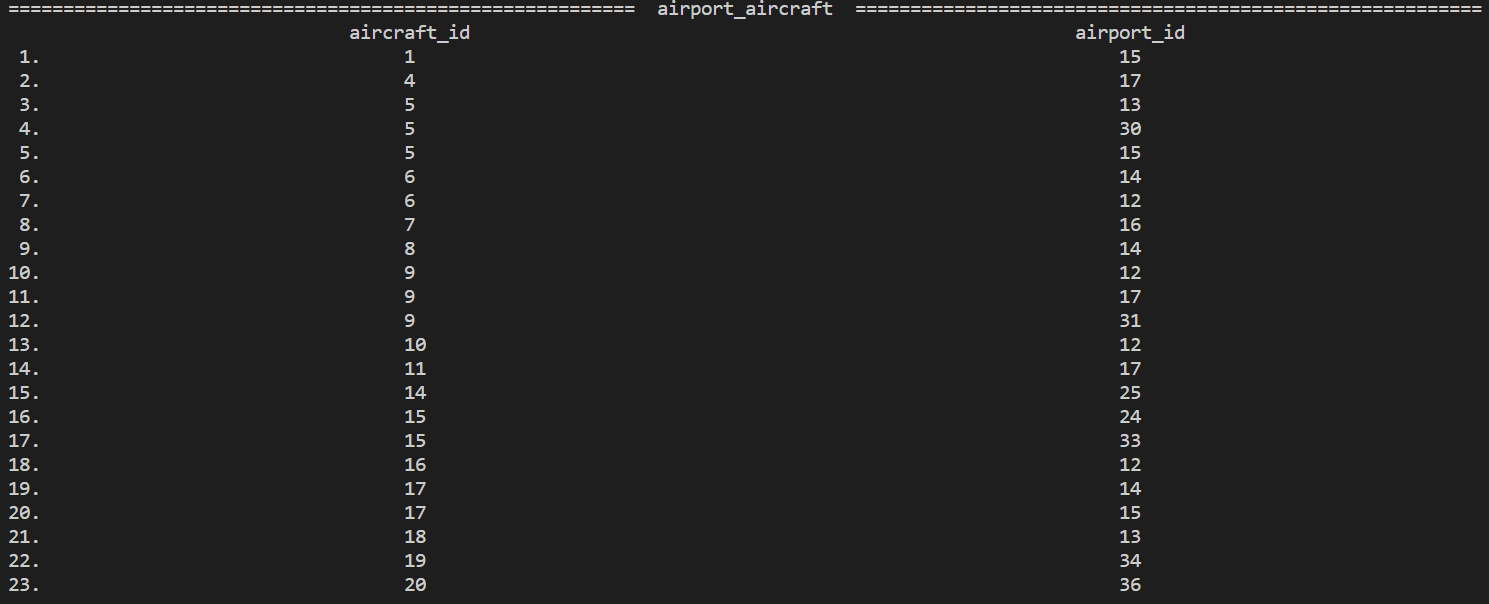
Після

****

**Додавання нових випадкових даних до кожної таблиці БД.**



****

****

**Завдання 2**

**BTree**

Для дослідження індексу була створена таблиця, яка має дві колонки: str типу varchar і str\_indexed типу varchat. Колонка str\_indexed проіндексована як BTree.

Заповнимо обидва стовбця 500.000 випадковими даними.

**create** **table** strs(

str varchar,

str\_indexed varchar

);

**CREATE** **INDEX** strs\_indexed **ON** strs **using** btree (str\_indexed);

**insert** **into** strs **SELECT**

md5(random()::text),

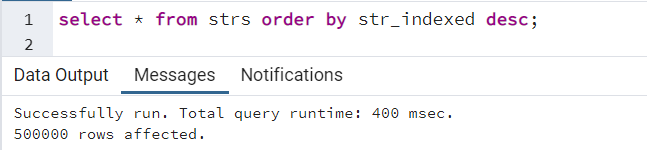
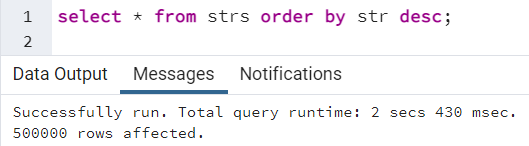
md5(random()::text)

**from** (

**SELECT** \* **FROM** generate\_series(1,500000) **AS** id

) **AS** ser;

Тепер за допомогою Select виберемо і відсортуємо всі значення за спаданням і зрівняємо час виконання кожної з команд.



Різниця при використанні індекса очевидна, ми отримуємо таку швидкодію через те, що дані в індексі впорядковані через незменшення , а сторінки одного рівня пов'язані між собою двонаправленим списком. Тому отримати впорядкований набір даних ми можемо просто проходячи по списку в одну або в іншу сторону, не повертаючись щоразу до кореня.

**BRIN**

Створимо та заповнимо таблицю з запису швидкості вітра кожної секунди протягом 2 років.

**CREATE** **TABLE** wind\_log (

id int,

time\_st timestamp without time zone,

speed int

);

**INSERT** **INTO** wind\_log

(

id,

time\_st,

speed

)

**VALUES**

(

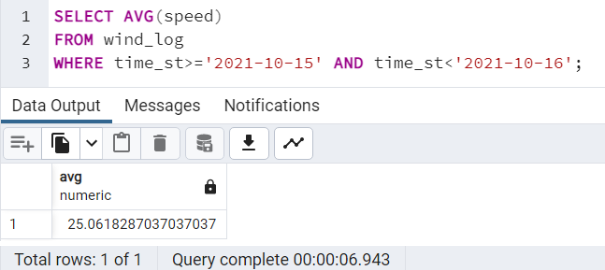
round(random()\*1000)::int,

generate\_series('2020-01-01'::timestamp,'2022-01-01'::timestamp,'1 second'),

round(random()\*50)::int

);

Порахуємо середню температуру за 15 жовтня

Без індекса отримали час виконання: 6.943 секунди

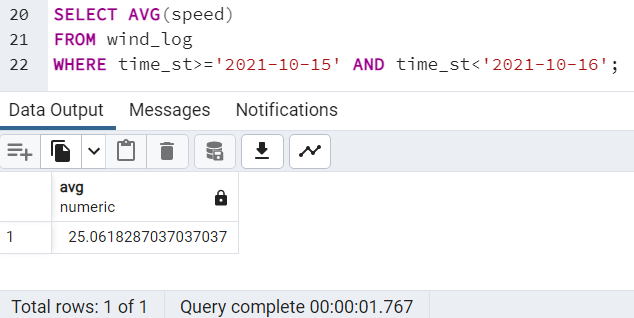
Створимо Breen індекс:

**DROP** **INDEX** **IF** **EXISTS** average\_speed;

**CREATE** **INDEX** average\_speed **ON** wind\_log

**USING** BRIN (time\_st) **WITH** (pages\_per\_range = 128);

і запустимо аналогічний select

Отримали час виконання: 1.767 секунди

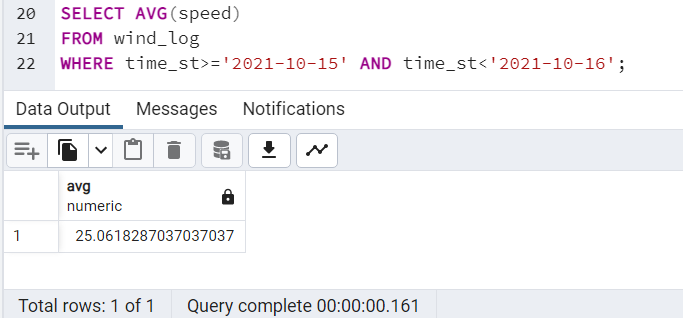
Створимо BTree індекс:

**DROP** **INDEX** **IF** **EXISTS** average\_speed;

**CREATE** **INDEX** average\_speed **ON** wind\_log

**USING** btree (time\_st);

і запустимо аналогічний select

 Отримали час виконання: 0.161 секунди

**Висновок**

Додавання індексів значно підвищує швидкодію, але збільшує використання пам’яті, також варто пам'ятати, що для різних операцій треба обирати різні типи індекса, наприклад для надвеликих таблиць чудово підійде BTREE, в той же час BRIN добре працює для тих стовпців, значення в яких корелюють з їх фізичним розташуванням у таблиці.

**Завдання 3**

**Умова для тригера – before insert, delete**

Створення 2 додаткових таблиць для більш наглядної роботи тригерів

**CREATE** **TABLE** **IF** NOT **EXISTS** public.users\_stats

(

total\_sign\_up integer NOT NULL,

last\_time\_sign\_up timestamp NOT NULL

)

TABLESPACE pg\_default;

**ALTER** **TABLE** **IF** **EXISTS** public.users\_stats

OWNER **to** postgres;

--

**INSERT** **INTO** public.users\_stats(

total\_sign\_up, last\_time\_sign\_up)

**VALUES** (0, current\_timestamp);

**CREATE** **TABLE** **IF** NOT **EXISTS** public.deleted\_users

(

id integer NOT NULL

)

TABLESPACE pg\_default;

**ALTER** **TABLE** **IF** **EXISTS** public.deleted\_users

OWNER **to** postgres;

**Тригери**

**before insert**

**CREATE** OR **REPLACE** **FUNCTION** insert\_func()

**RETURNS** trigger **AS** $insert\_func$

**BEGIN**

**UPDATE** users\_stats **SET** total\_sign\_up = total\_sign\_up + 1, last\_time\_sign\_up = current\_timestamp

**WHERE** id = 1;

RETURN NULL;

**END**;

$insert\_func$ LANGUAGE plpgsql;

**CREATE** OR **REPLACE** TRIGGER UsersInsertTrigger

BEFORE **INSERT** **ON** user\_account

**FOR** EACH **ROW** EXECUTE **FUNCTION** insert\_func();

**before delete**

**CREATE** OR **REPLACE** **FUNCTION** delete\_func()

**RETURNS** trigger **AS** $delete\_func$

**BEGIN**

**INSERT** **INTO** deleted\_users (id) **VALUES** (OLD.user\_acc\_id);

RETURN NULL;

**END**;

$delete\_func$ LANGUAGE plpgsql;

**CREATE** OR **REPLACE** TRIGGER UsersDeleteTrigger

BEFORE **DELETE** **ON** user\_account

**FOR** EACH **ROW** EXECUTE **FUNCTION** delete\_func();

**Принцип роботи**

Тригер спрацьовує перед додаванням або видаленням в таблиці ‘user\_account’.

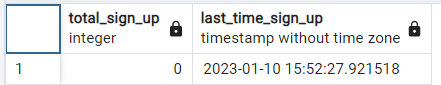
При додаванні нового рядка в таблицю ‘user\_account’, таблиця ‘users\_stats’ оновлює свій зміст, а саме інкрементує значення total\_sign\_up та оновлює часову мітку на теперішнє значення.

При видаленні рядка в таблиці ‘user\_account’, в таблицю ‘deleted\_users’ заноситься новий зміст, а саме додається видалене значення user\_acc\_id в колонку deleted\_users.id

***Приклад***

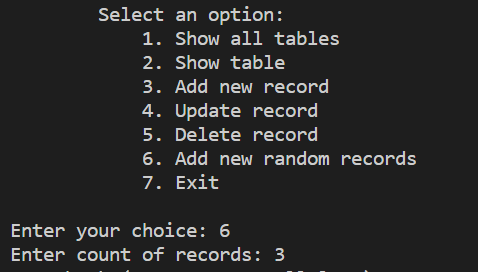
**Таблиці до змін:**

users\_stats

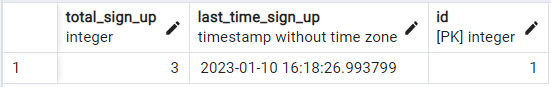
deleted\_users



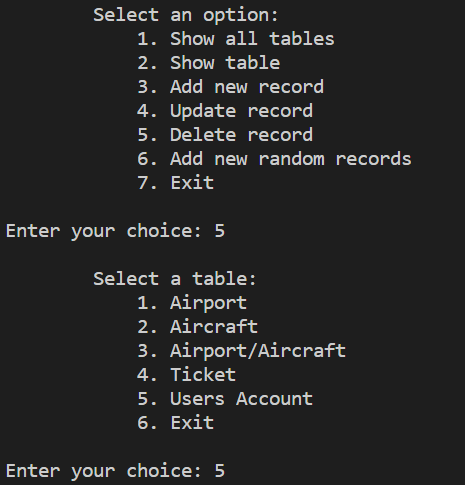
Введемо три записи до кожної таблиці



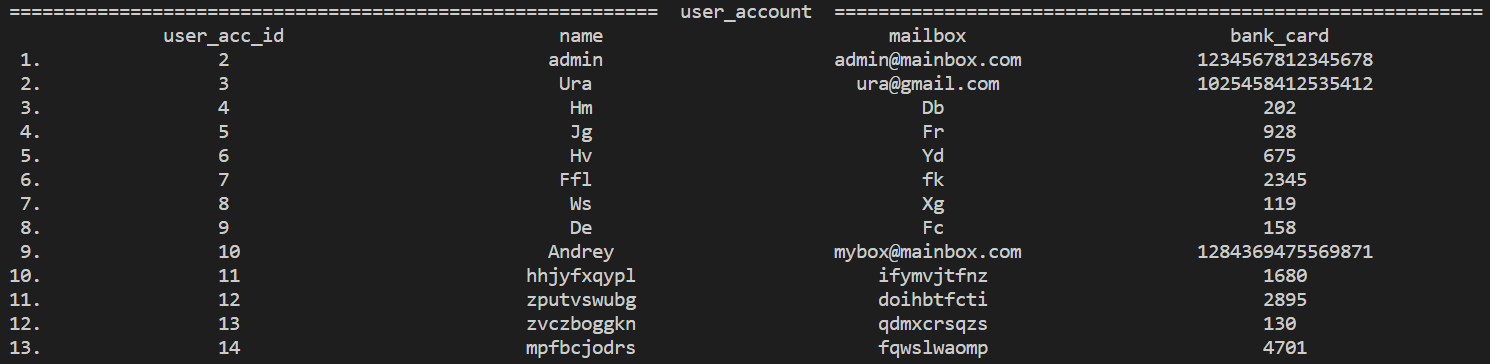
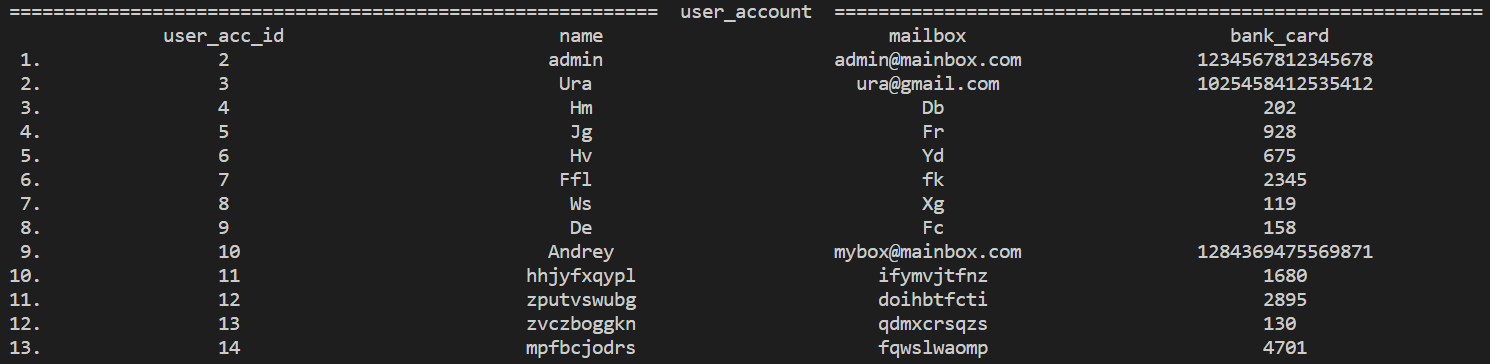
Таблиця ‘users\_stats’ оновилася



Видалимо три записи з таблиці ‘User Account’



Таблиця ‘User\_Account’ після видалення

Таблиця ‘deleted\_users’ після спрацювання тригерів

