A black and white drawing of a building

Description automatically generated

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського» ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Лабораторна робота №2**

з дисципліни **Бази даних і засоби управління**

*на тему: “Засоби оптимізації роботи СУБД PostgreSQLQL”*

Виконав:

студент ІІI курсу

групи КВ-12

Гусельніков А.О.

Перевірив:

Павловский В. І.

Київ – 2023

**Мета:** здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

**Завдання:**

1. Перетворити модуль “Модель” з шаблону MVC РГР 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.

**Виконання роботи**

**Сутності предметної області**

Згідно цієї області для побудови бази даних було виділено наступні сутності:

1. Організатори (Оrganizers).

Атрибути: ідентифікатор організатора, ім’я організатора, прізвище організатора, електронна пошта організатора.

Призначення: збереження даних щодо організаторів заходу.

1. Туристи (Tourists).

Атрибути: ідентифікатор туриста, ім’я туриста, прізвище туриста, електронна пошта туриста.

Призначення: збереження даних щодо туристів.

1. Заходи (Events).

Атрибути: ідентифікатор заходу, назва заходу, тип заходу, дата коли відбудеться, ідентифікатор адреси.

Призначення: збереження даних щодо заходів.

1. Номери телефонів (Phone\_nums).

Атрибути: ідентифікатор номеру телефону, номер телефону, ідентифікатор туриста, якому він належить.

Призначення: збереження даних щодо контактних номерів телефонів туристів.

**Зв’язки між сутностями предметної області**

1. Зв’язок «Оrganizers» - «Events» є зв’язком N:M. Організатор може мати багато створених заходів, а може й не мати, у одного заходу може бути один або більше організаторів.
2. Зв’язок «Tourists» - «Events» є зв’язком N:M. Турист може забронювати місця у багатьох заходах, а може й не забронювати, захід може мати багато заброньованих місць, а може й не мати.
3. Зв’язок «Tourists» - «Phone\_nums» є зв’язком 1:N. Кожен турист має ввести хоча б один контактний номер телефону та номер телефону має належати тільки одній людині.

Графічне подання концептуальної моделі «Сутність-зв’язок» зображено на рисунку 1.

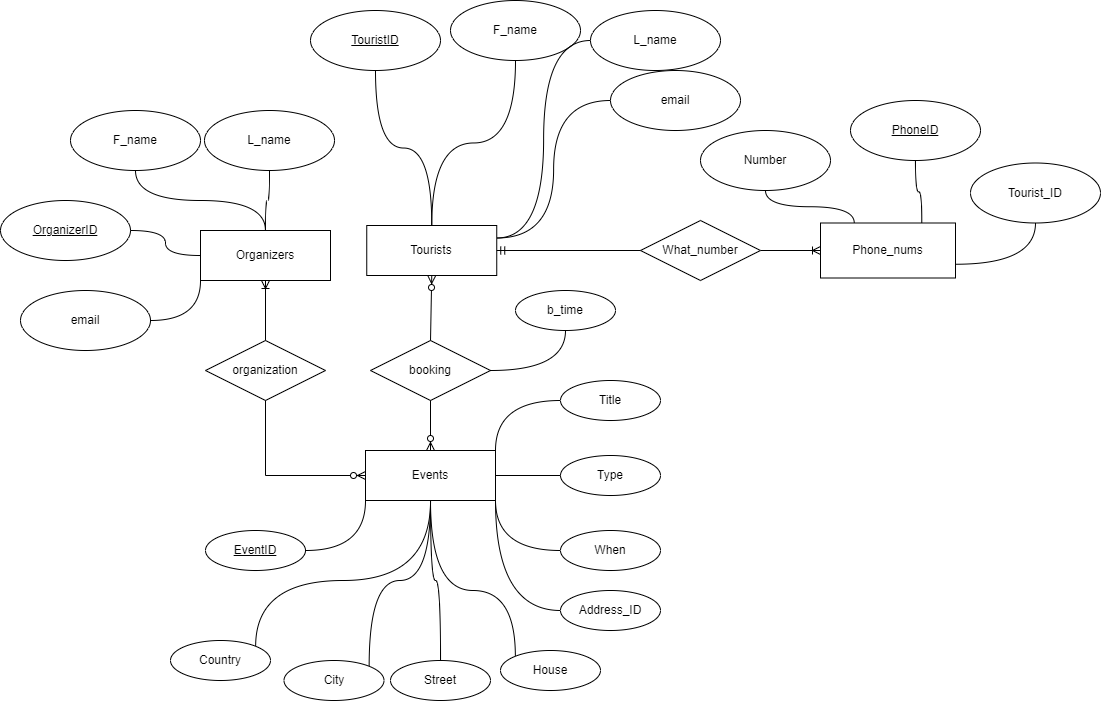


Рисунок 1 – ER-діаграма, побудована за нотацією Чена

Графічне подання логічної моделі «Сутність-зв’язок» зображено на рисунку 2.

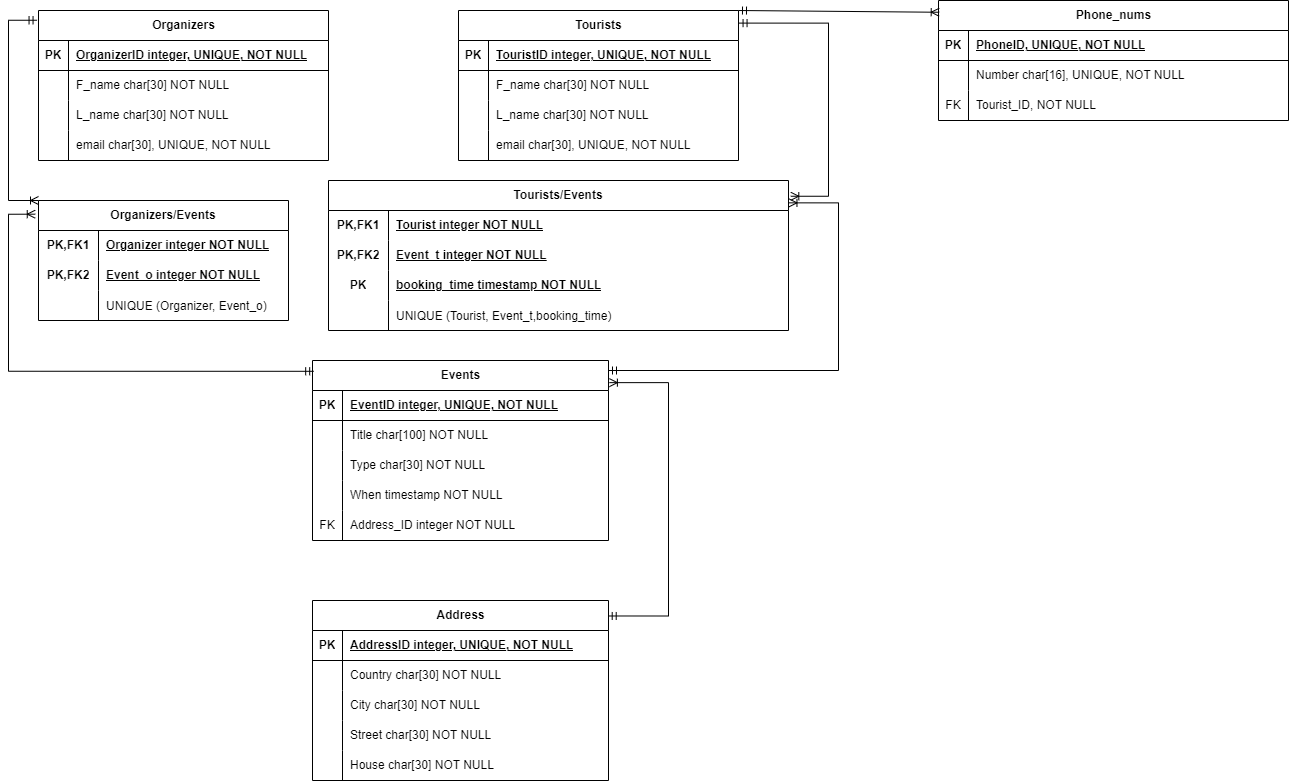
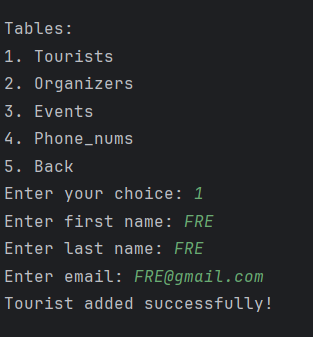
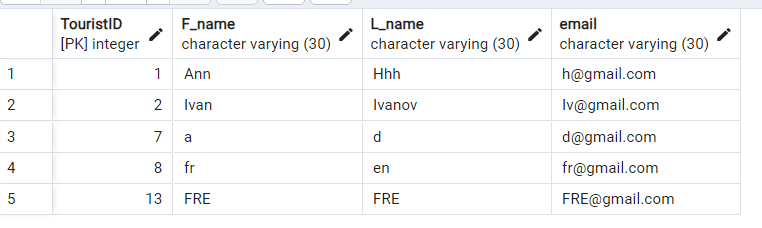


Рисунок 2 – Схема бази даних

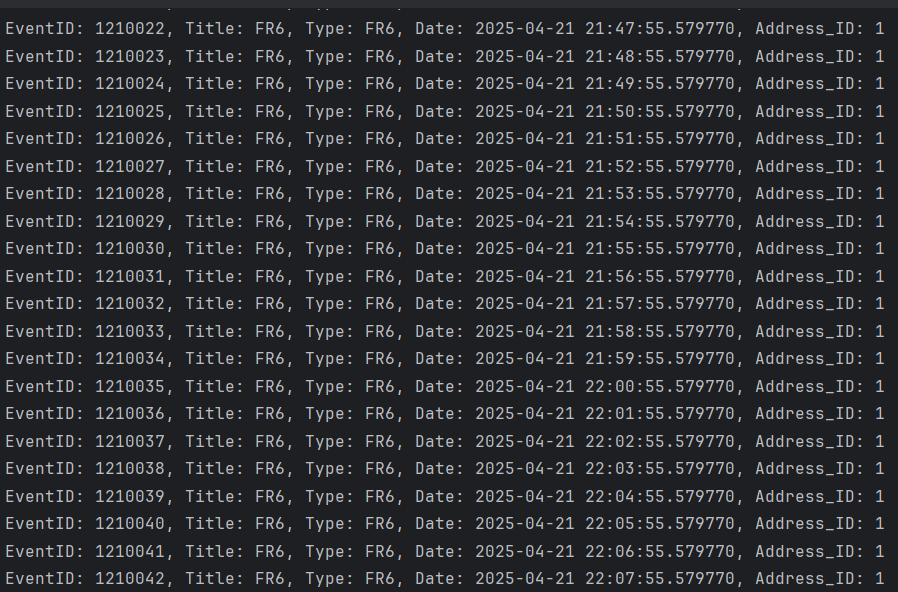
**Робота ORM**

Add Tourist

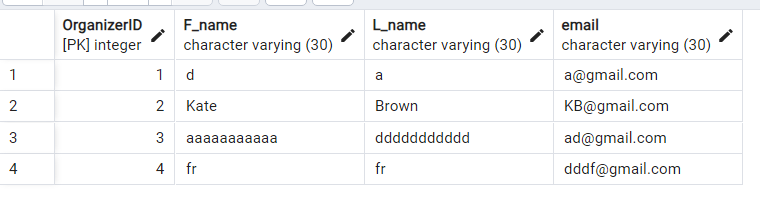


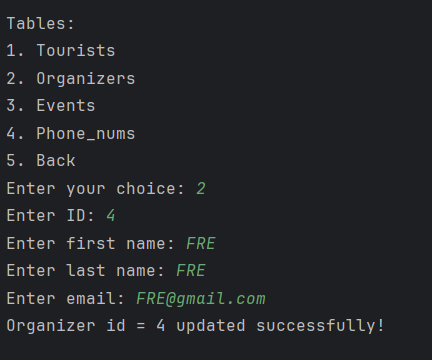


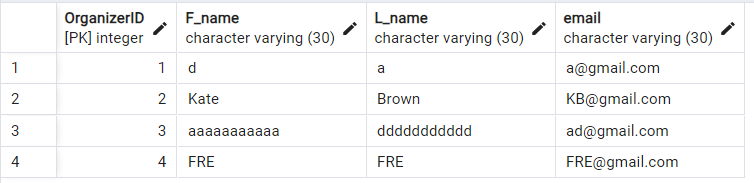
View Events



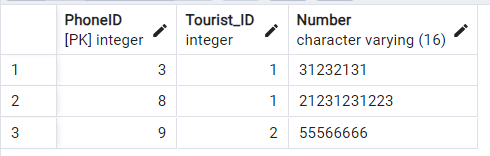
Update Organizer

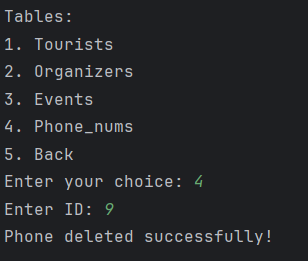


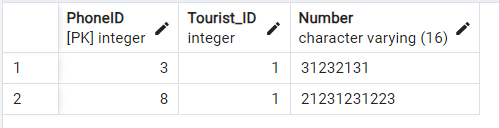




Delete Phone

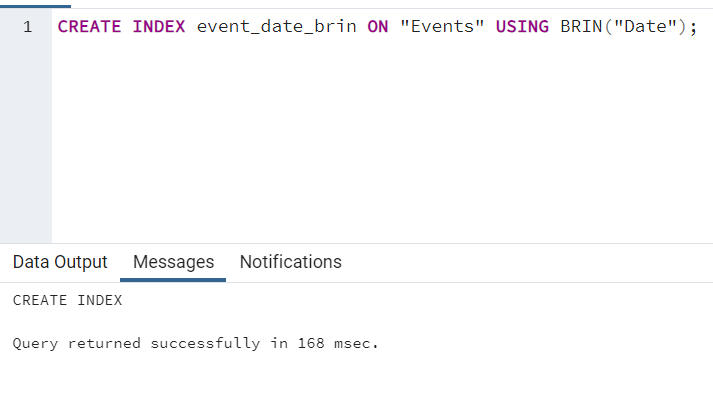




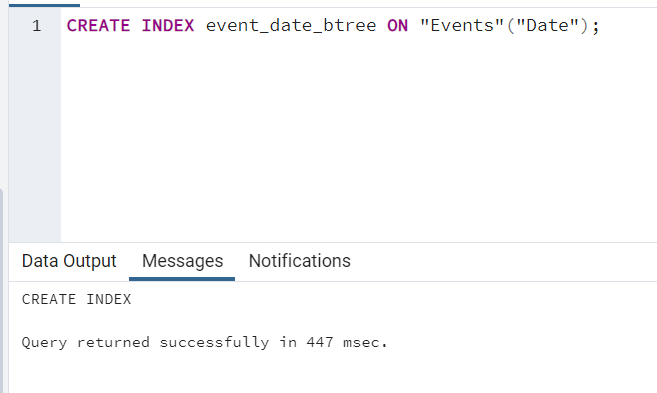


**Порівняння індексів**

CREATE INDEX event\_date\_brin ON "Events" USING BRIN("Date");

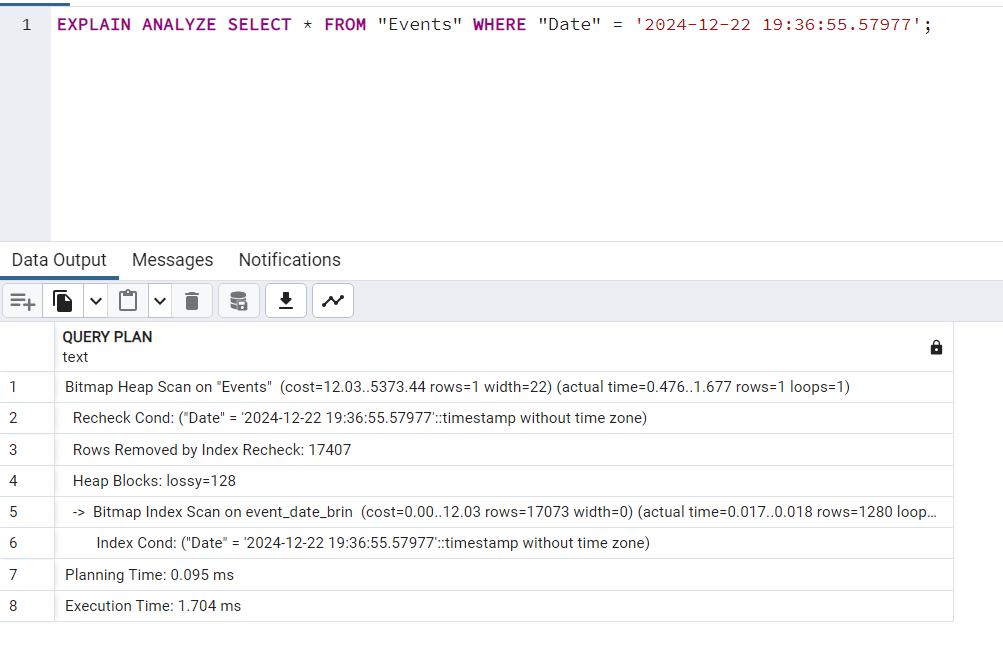


CREATE INDEX event\_date\_btree ON "Events"("Date");

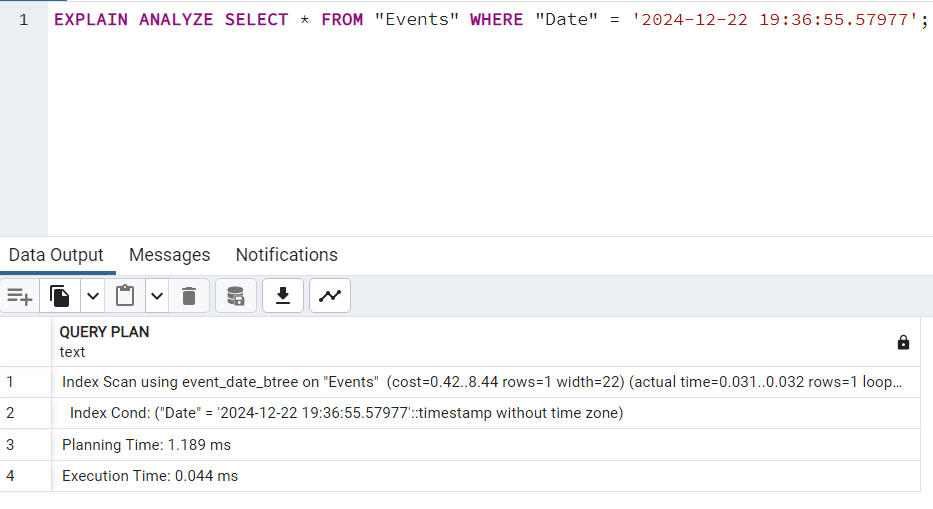


EXPLAIN ANALYZE SELECT \* FROM "Events" WHERE "Date" = '2024-12-22 19:36:55.57977';

BRIN:

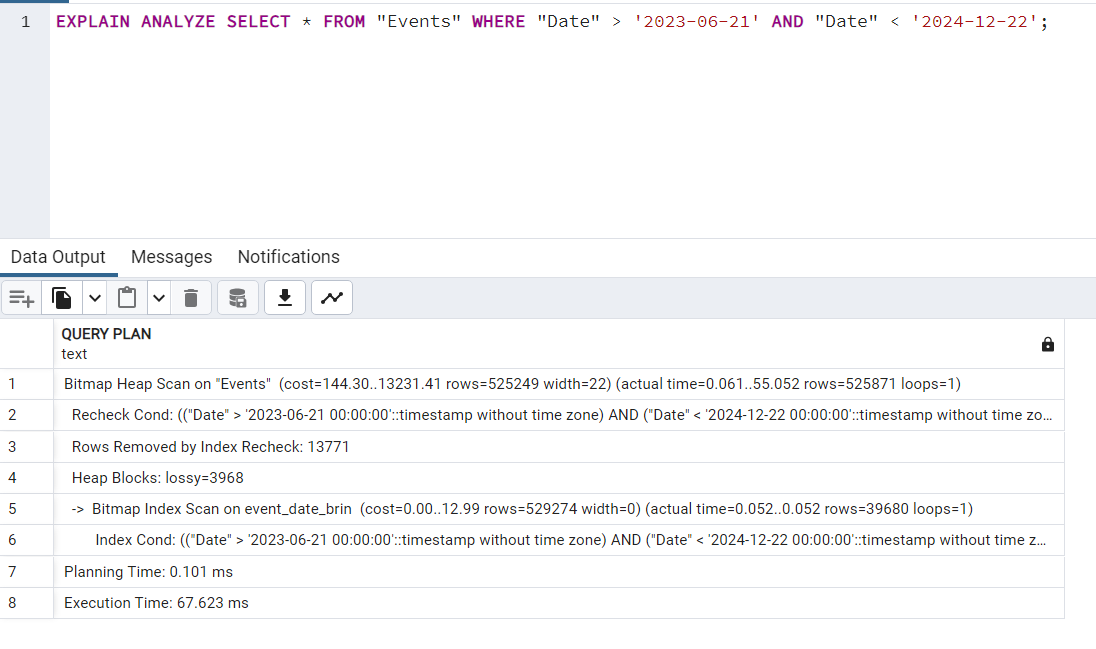


BTREE:

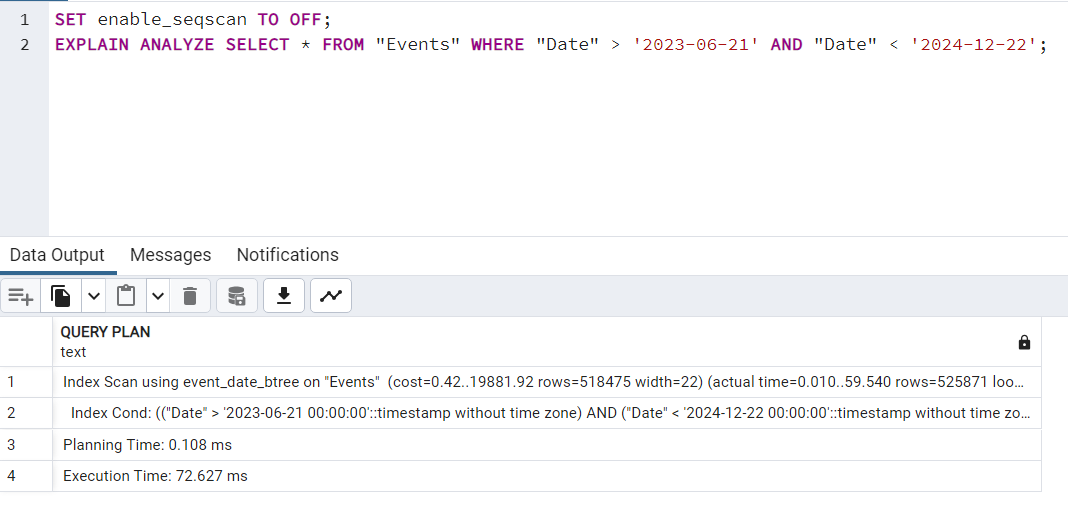


EXPLAIN ANALYZE SELECT \* FROM "Events" WHERE "Date" > '2023-06-21' AND "Date" < '2024-12-22';

BRIN:

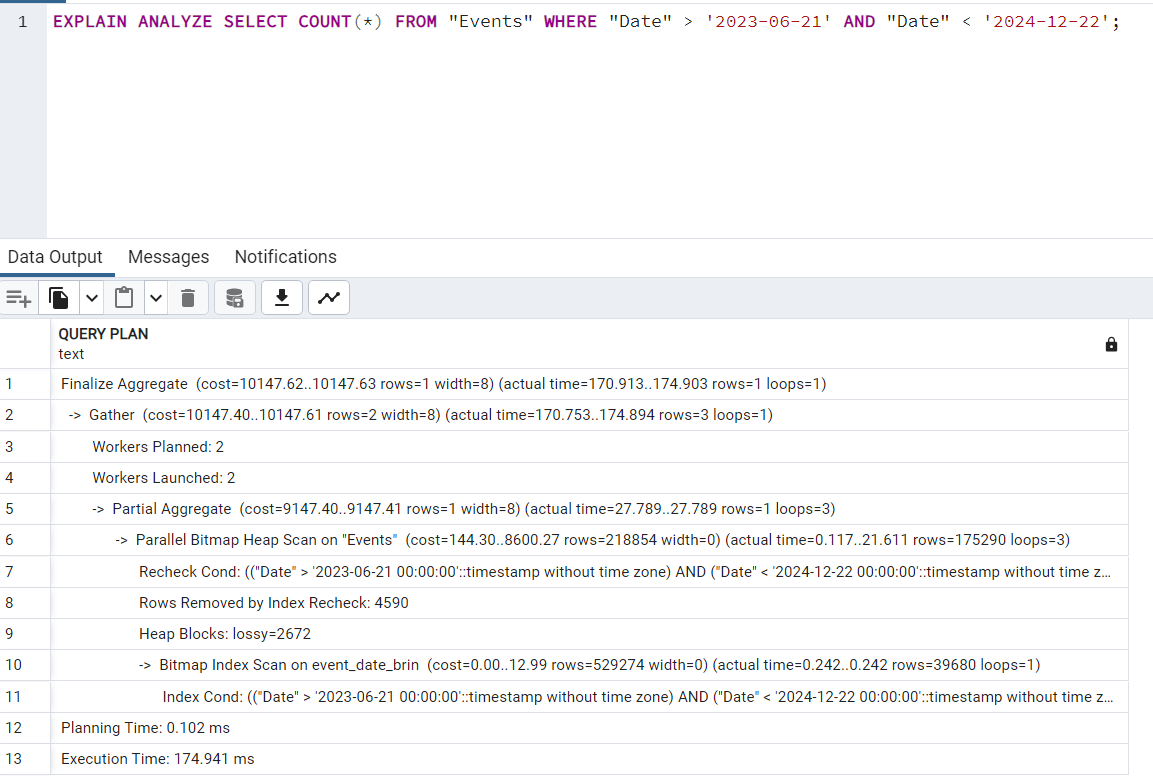


BTREE:



EXPLAIN ANALYZE SELECT COUNT(\*) FROM "Events" WHERE "Date" > '2023-06-21' AND "Date" < '2024-12-22';

BRIN:

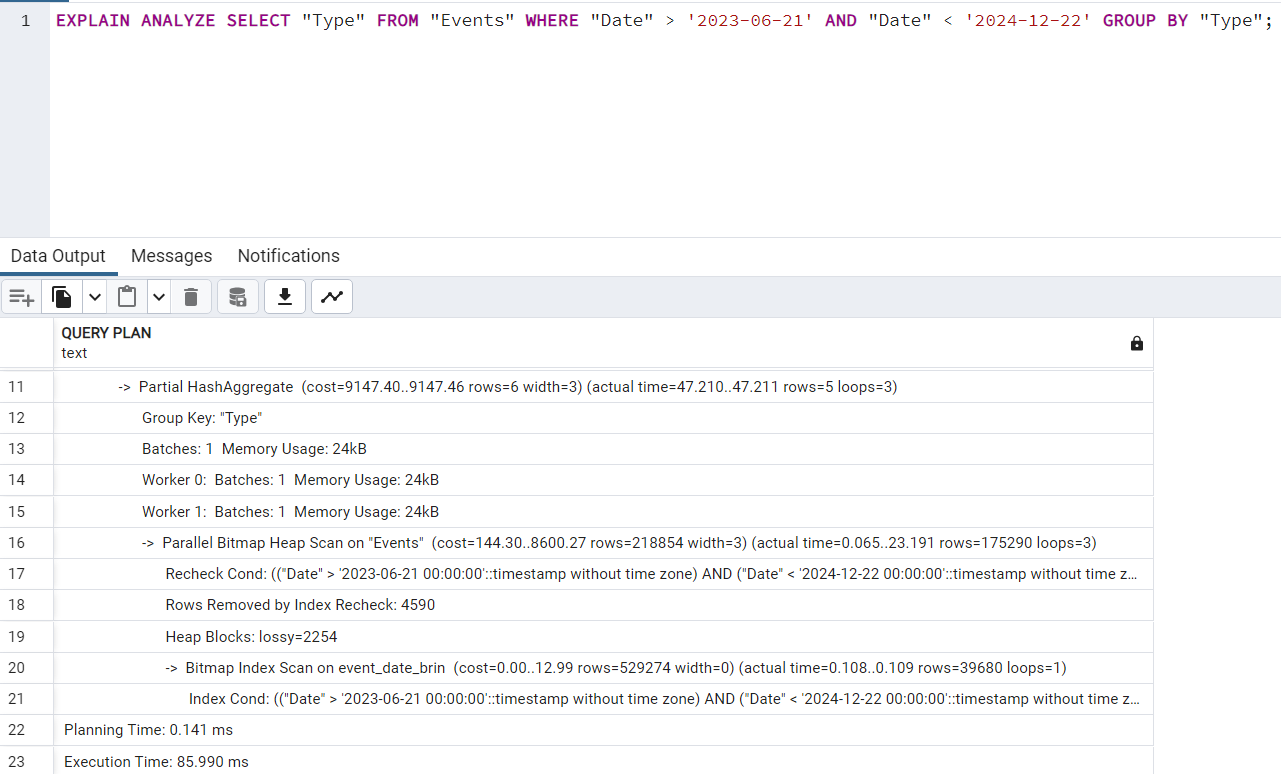


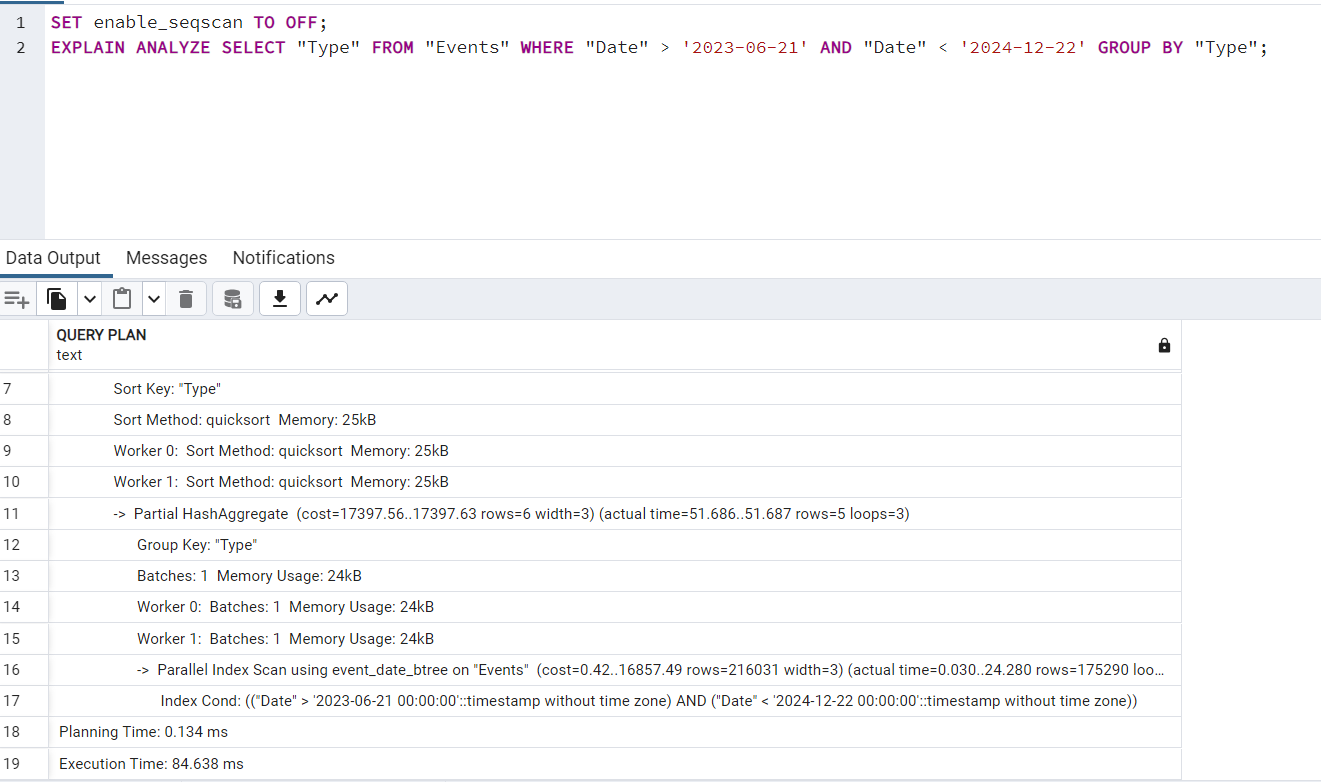
BTREE:



EXPLAIN ANALYZE SELECT "Type" FROM "Events" WHERE "Date" > '2023-06-21' AND "Date" < '2024-12-22' GROUP BY "Type";

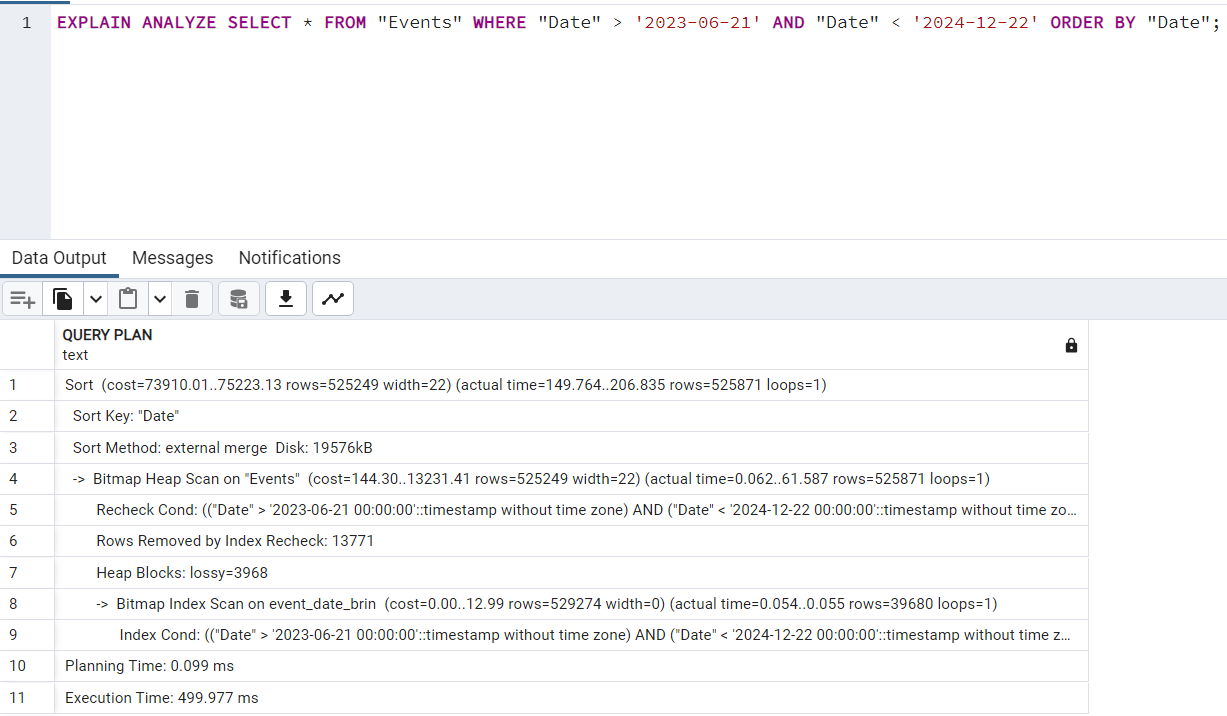
BRIN:



BTREE:  


EXPLAIN ANALYZE SELECT \* FROM "Events" WHERE "Date" > '2023-06-21' AND "Date" < '2024-12-22' ORDER BY "Date";

BRIN:



BTREE:



**Структура даних:**

**B-tree:** Це збалансоване дерево, де кожен вузол містить ключі даних і покажчики на піддерева. Структура B-tree забезпечує ефективний пошук ключів і підходить для широкого спектра запитів.

**BRIN:** Це індекс, орієнтований на блоки даних. Замість того щоб зберігати ключі та покажчики на записи, він зберігає інформацію про діапазони значень у кожному блоці даних. BRIN добре підходить для таблиць із великими обсягами даних, коли запити обмежуються певними діапазонами значень.

**Типи даних:**

**B-tree:** Підходить для широкого спектра типів даних і підтримує порівняння, сортування та пошук точних значень.

**BRIN:** Зазвичай використовується для впорядкованих даних, як-от часові ряди або дані з добре вираженою локальністю.

**Розмір індексу:**

**B-tree:** Зазвичай потребує більше місця в пам'яті, оскільки кожен вузол дерева містить ключі та покажчики.

**BRIN:** Займає менше місця, оскільки він фокусується на діапазонах значень у блоках даних.

**Продуктивність:**

**B-tree:** Ефективний для точних запитів за ключем і діапазонами значень, але може бути менш ефективний для великих обсягів даних.

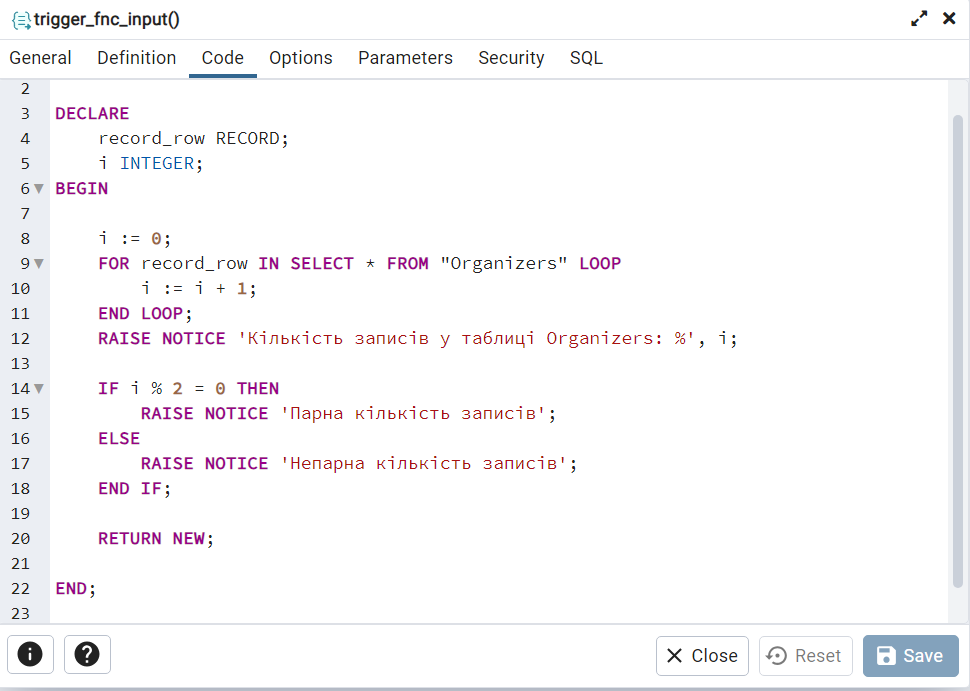
**BRIN:** Забезпечує хорошу продуктивність для запитів, які обмежуються діапазонами значень, особливо коли таблиця містить велику кількість даних.

**Типи запитів:**

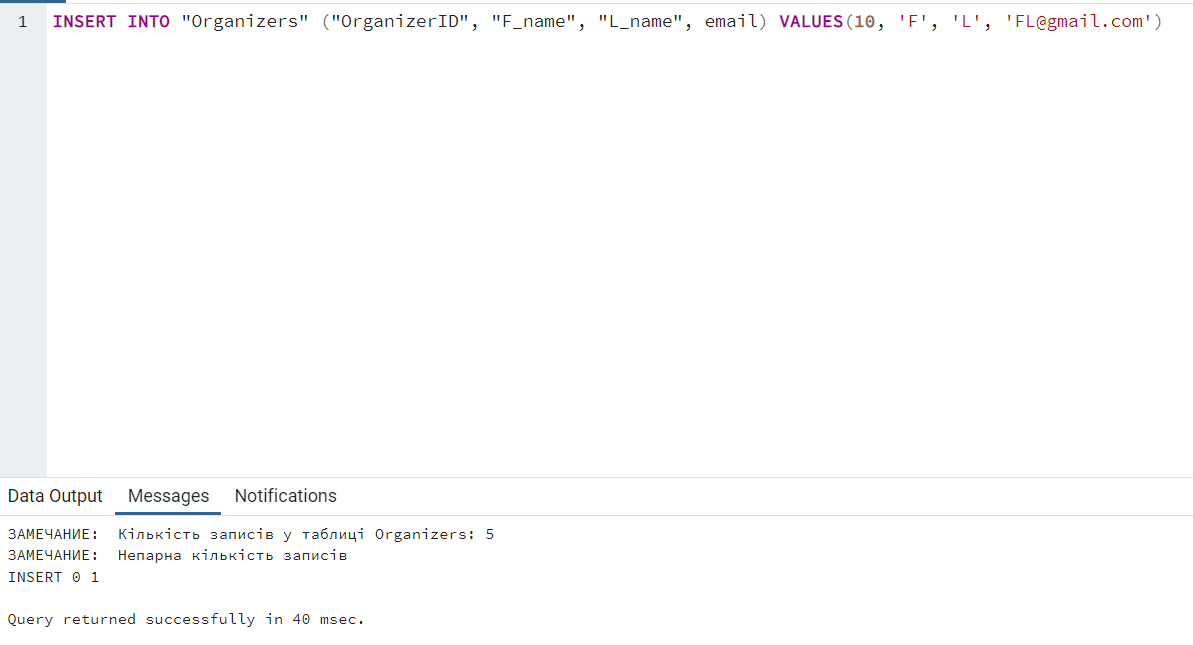
**B-tree:** Гарний для точних запитів за ключем, сортування і діапазонних запитів.

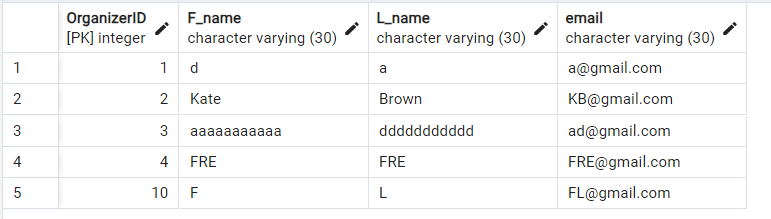
**BRIN:** Гарний для запитів, які фільтрують дані на основі діапазонів значень.

**Тригери**



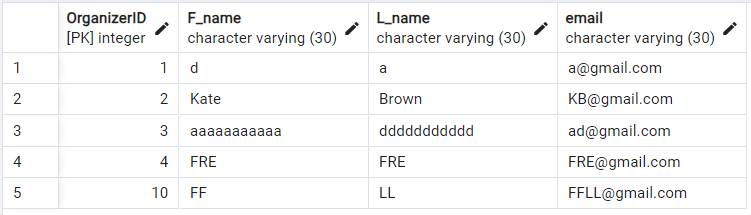
INSERT:



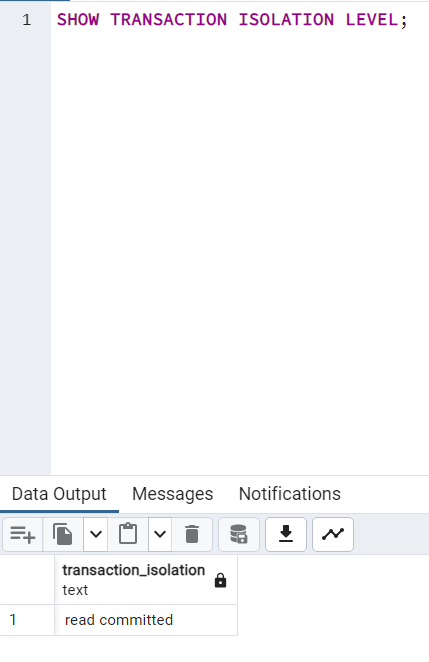


UPDATE:

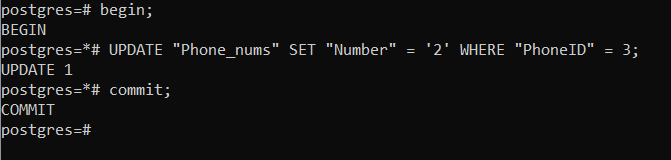


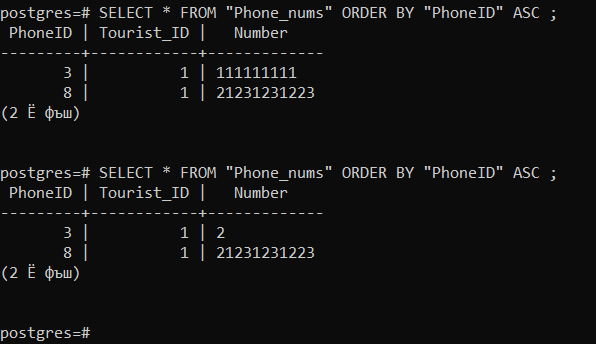


**Транзакції**

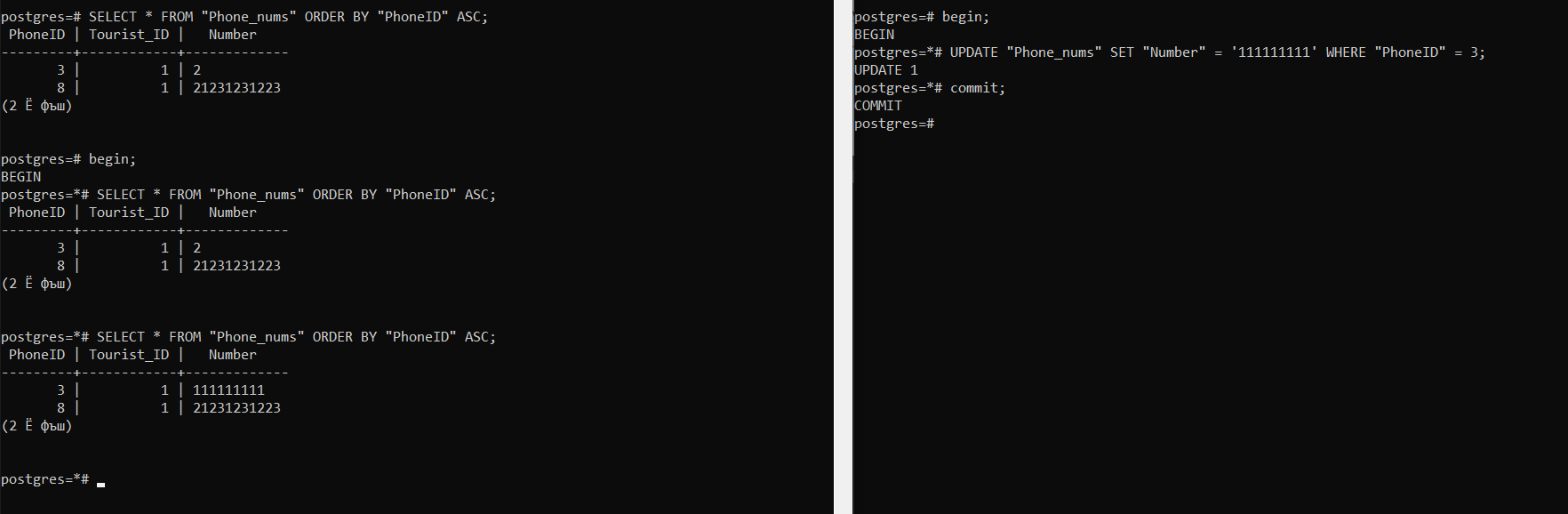


**Read committed:**





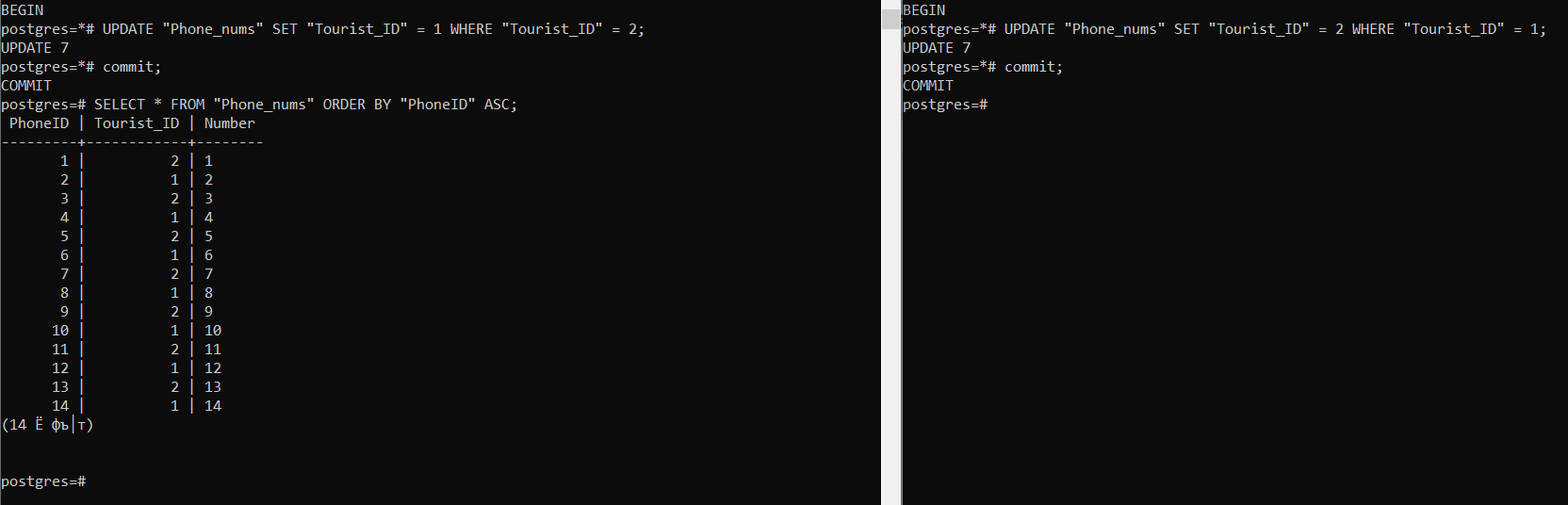
Феномен **Dirty Read** неможливий



Феномен **Nonrepeatable read** можливий

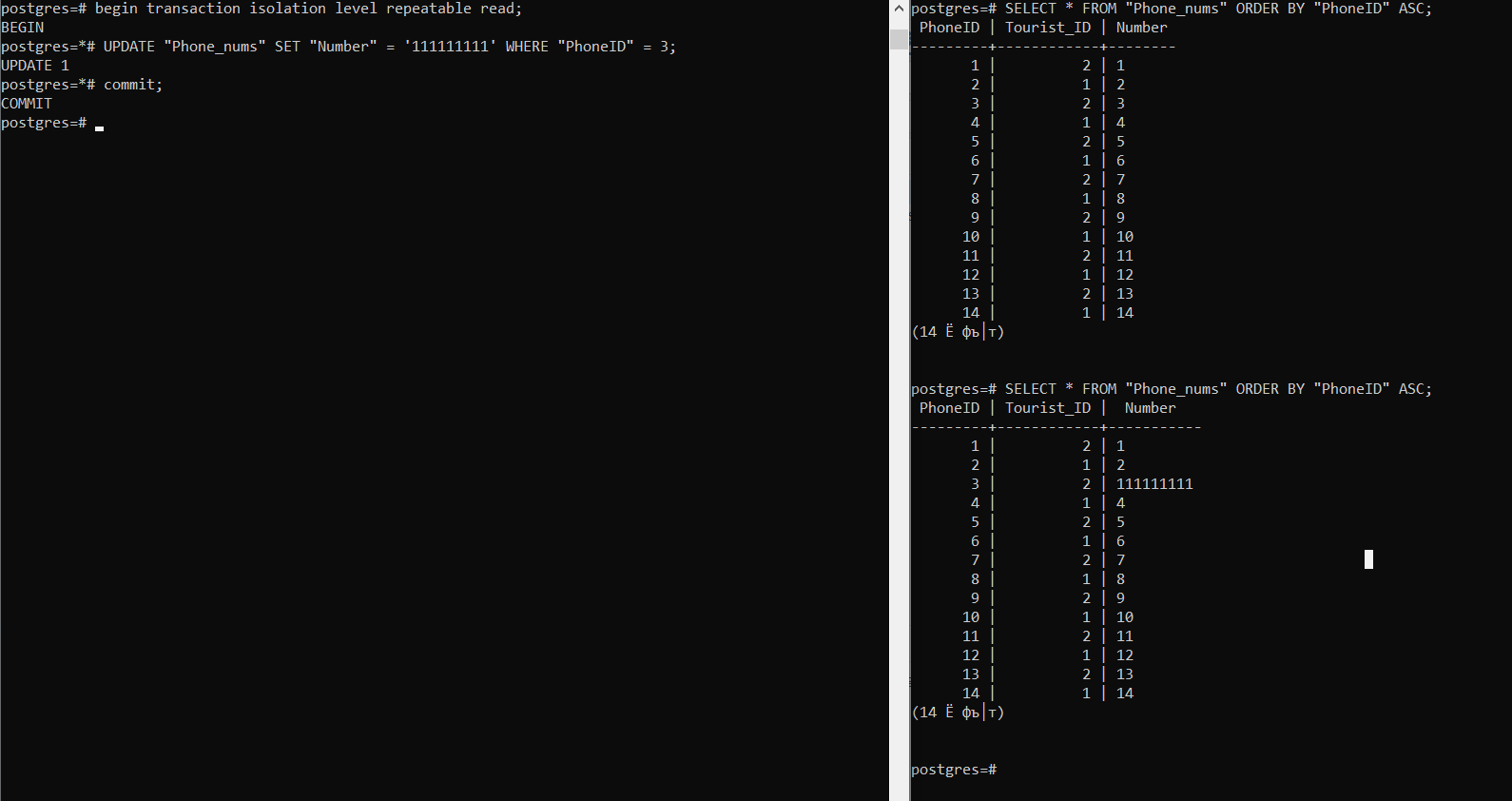


Феномен **Phantom reads** можливий

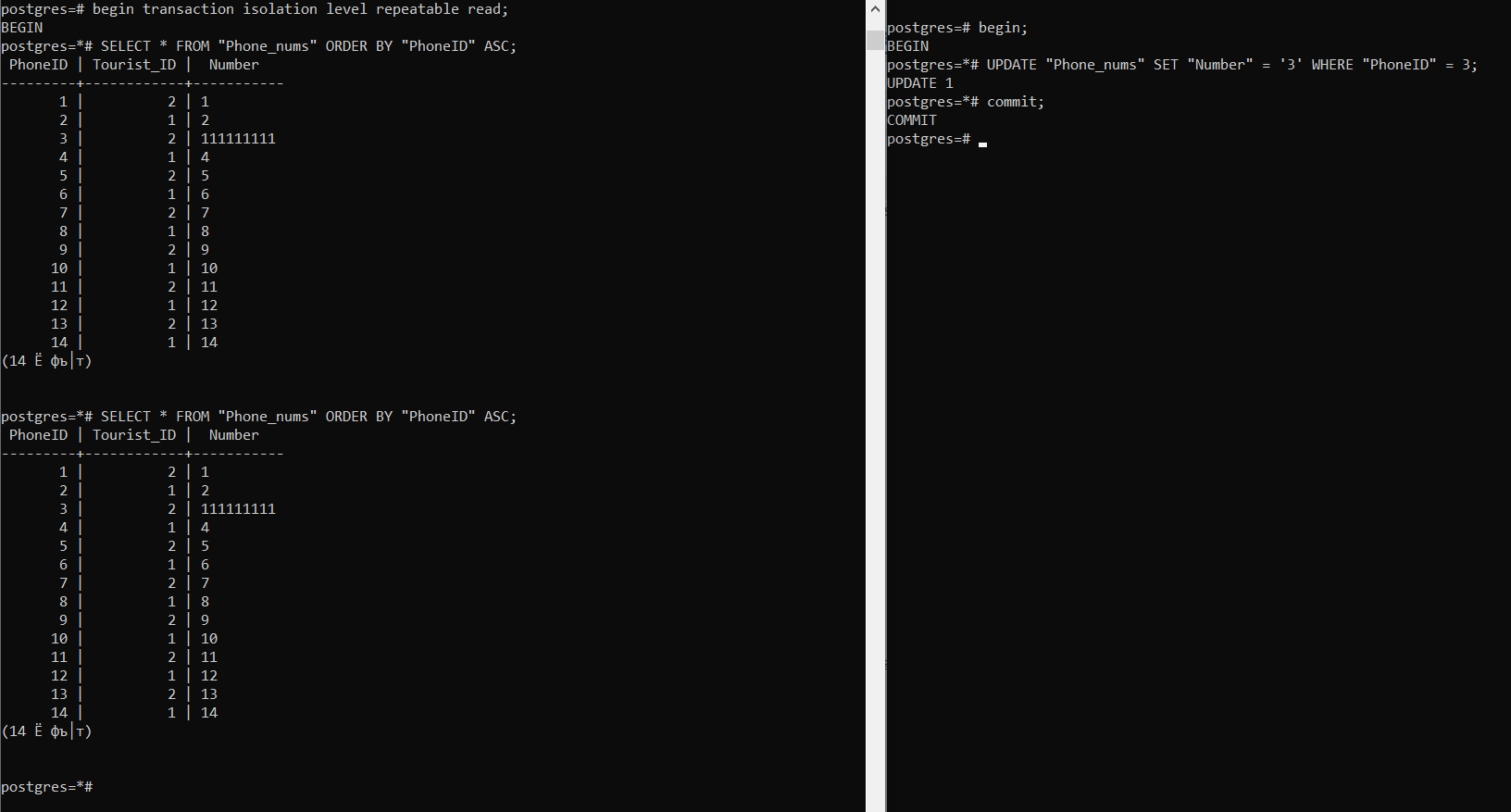


Феномен **Serialization Anomaly** можливий

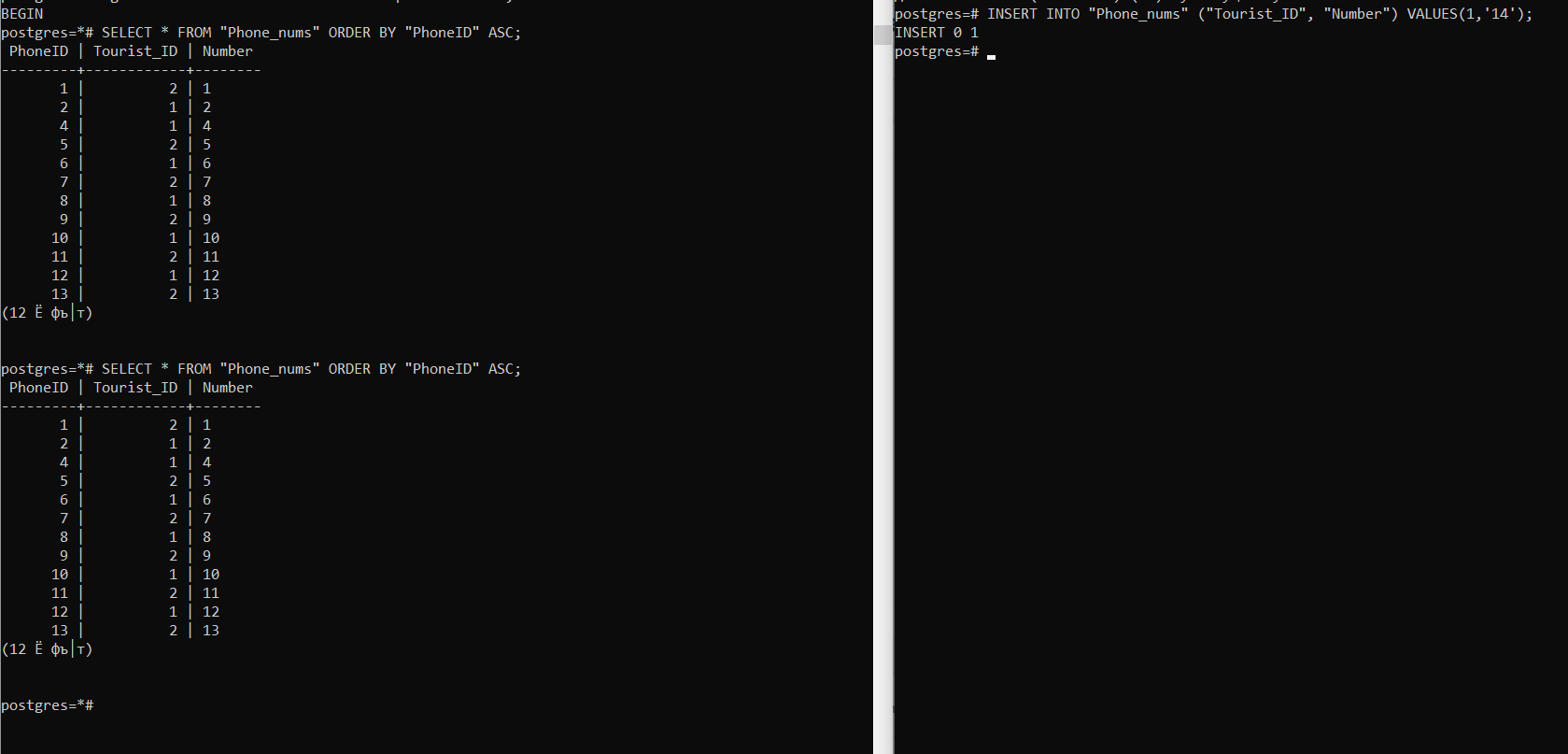
**Repeatable read:**



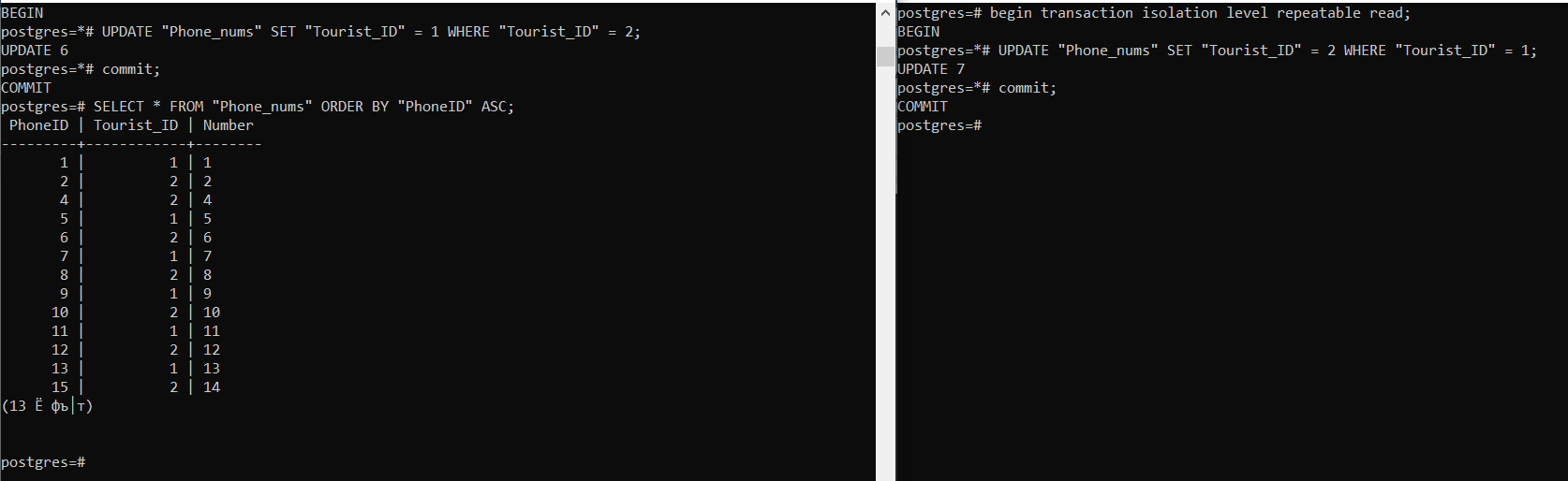
Феномен **Dirty Read** неможливий



Феномен **Nonrepeatable read** неможливий

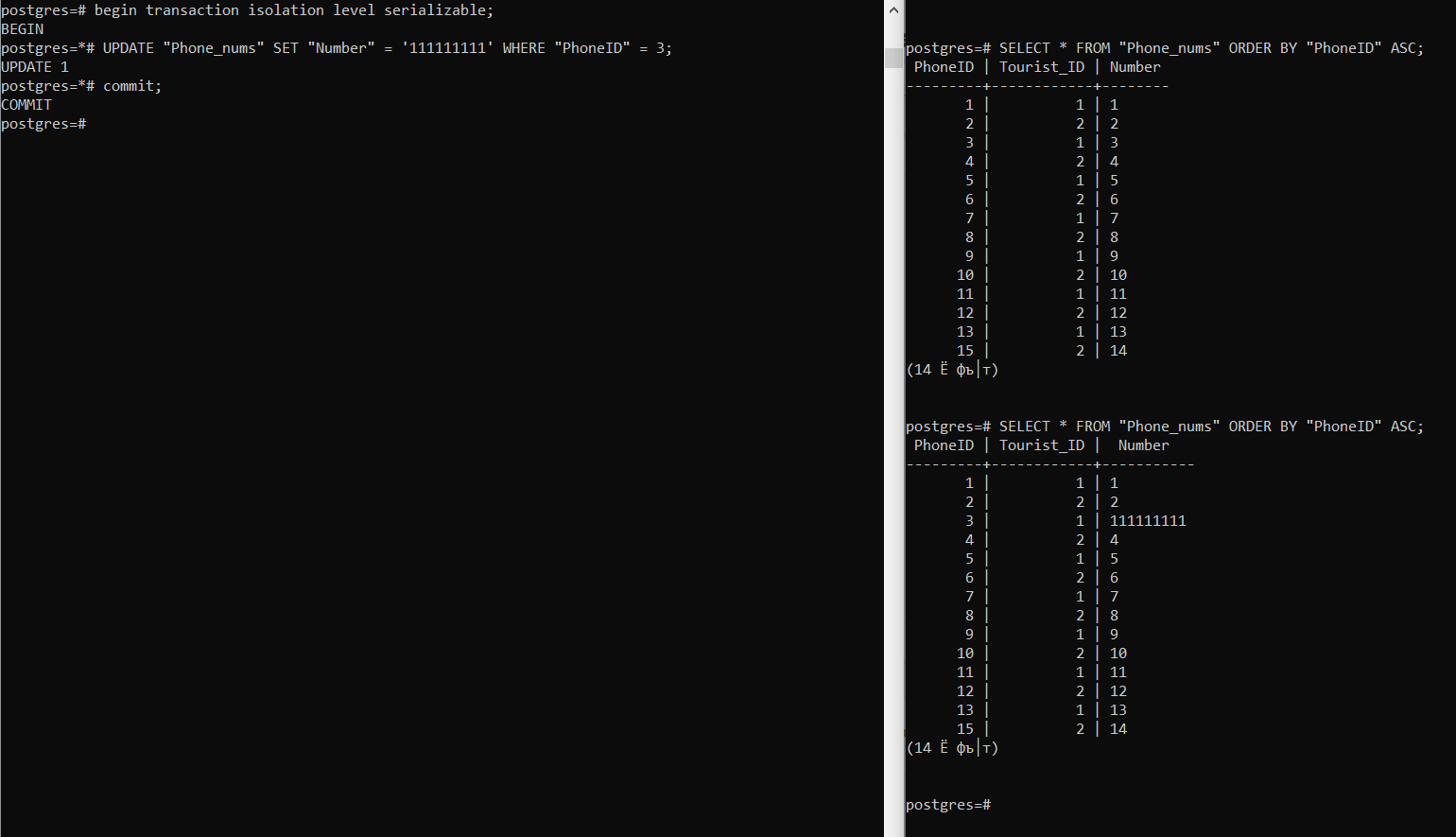


Феномен **Phantom reads** неможливий

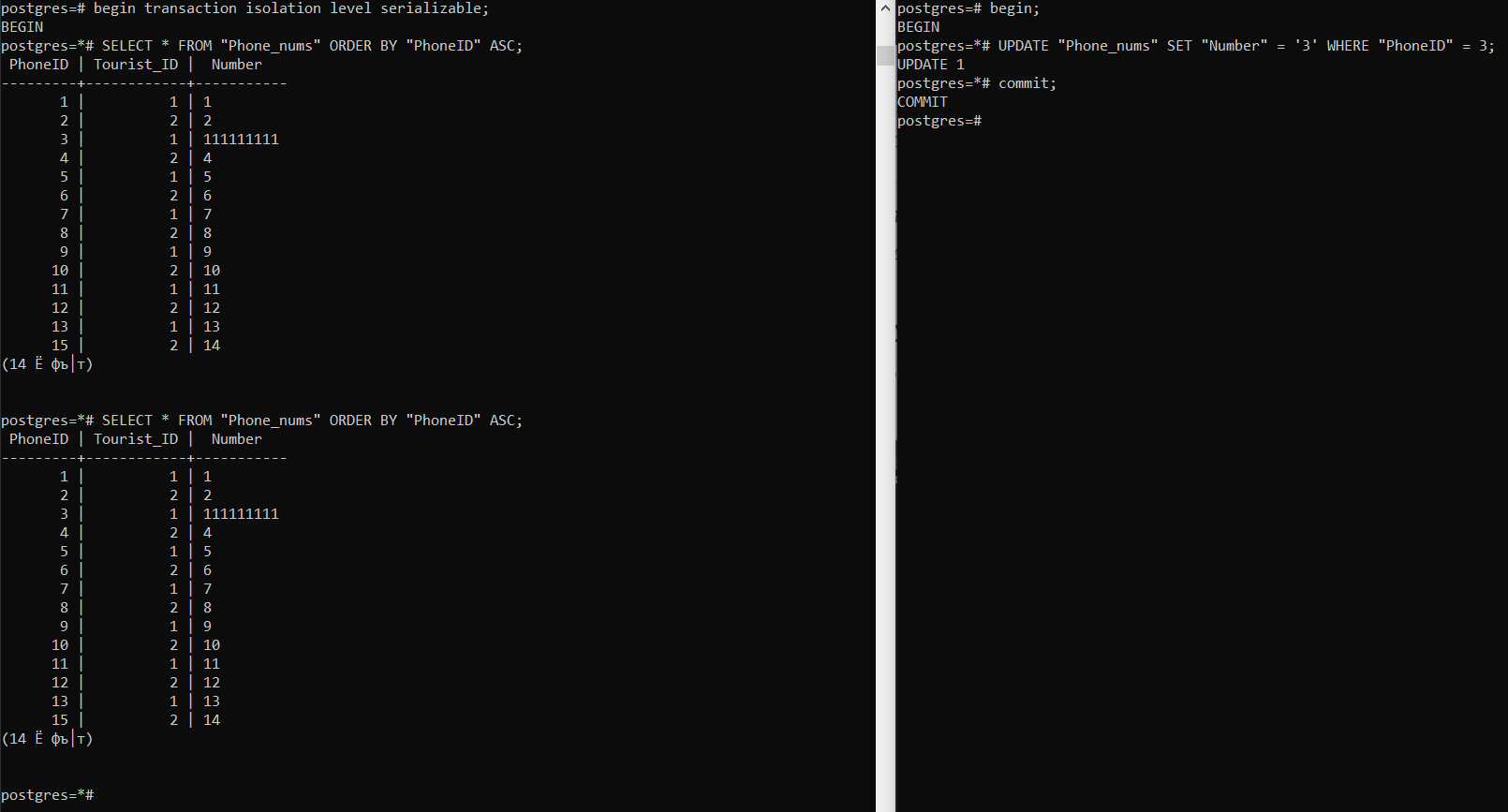


Феномен **Serialization Anomaly** можливий

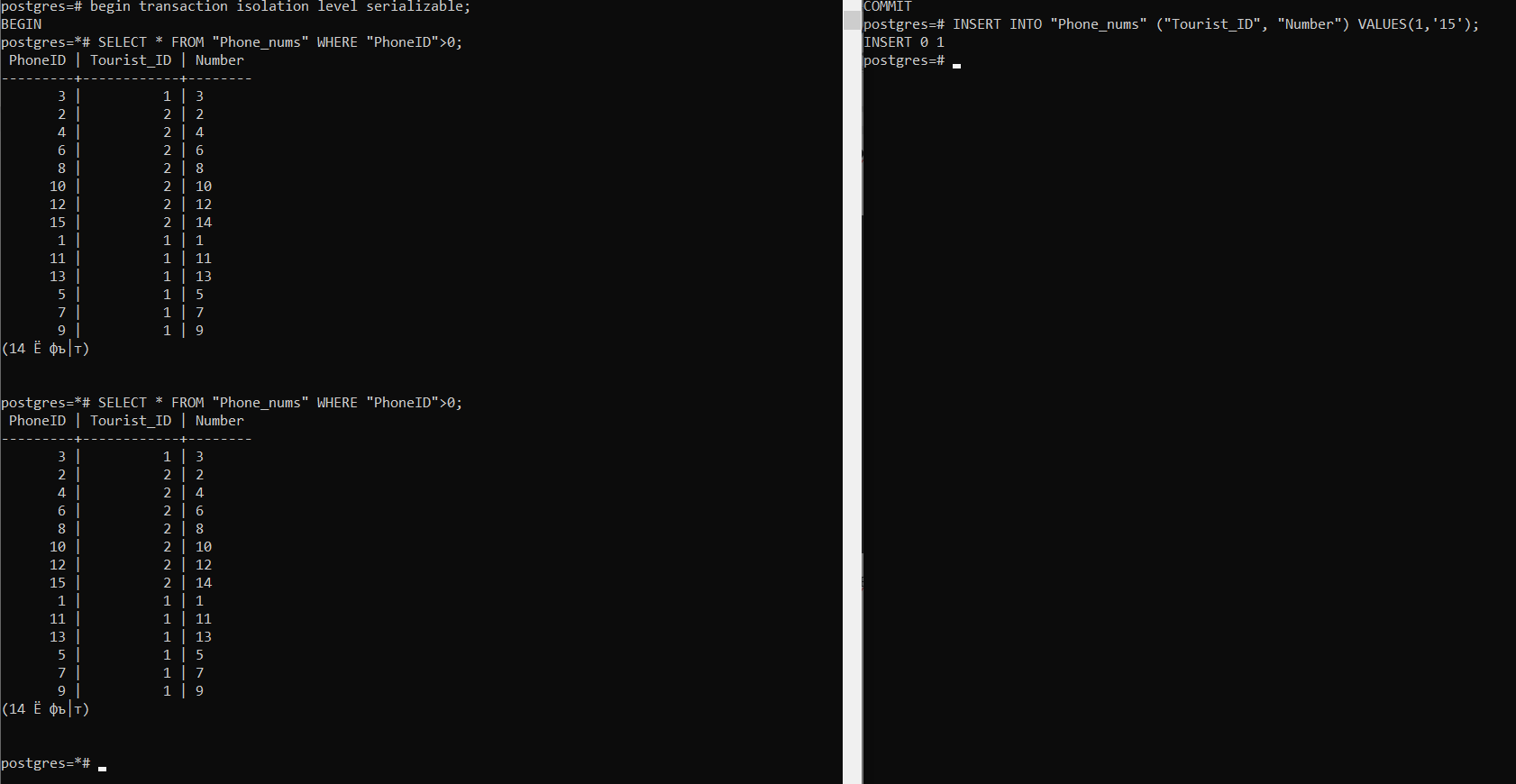
**Serializable mode:**



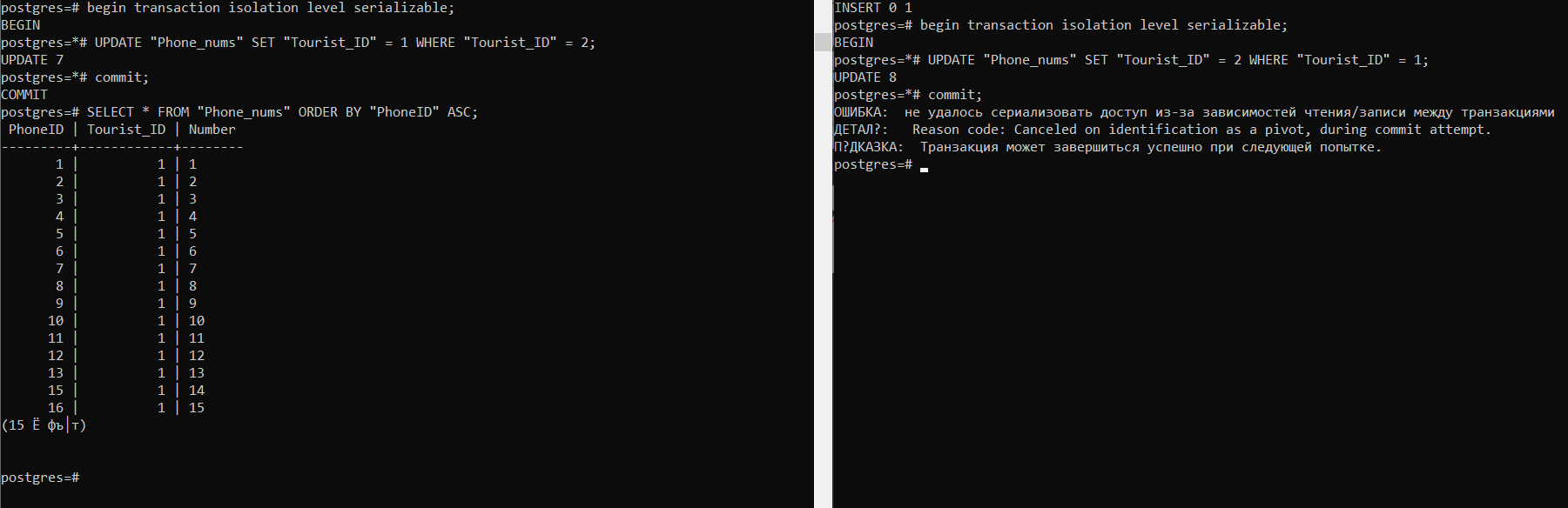
Феномен **Dirty Read** неможливий



Феномен **Nonrepeatable read** неможливий



Феномен **Phantom reads** неможливий



Феномен **Serialization Anomaly** неможливий

**Код model.py:**

import sqlalchemy  
from sqlalchemy import exc  
from sqlalchemy import func  
from sqlalchemy import create\_engine  
from sqlalchemy import Column, Integer, String,Date, ForeignKey  
from sqlalchemy.ext.declarative import declarative\_base  
from sqlalchemy.orm import relationship, backref  
from sqlalchemy.orm import sessionmaker  
import re  
from datetime import datetime  
  
Base = declarative\_base()  
  
class Database:  
  
 def \_\_init\_\_(self):  
 engine = create\_engine("postgresql://postgres:55krotvkaske@localhost:5432/postgres")  
 self.session\_class = sessionmaker(bind=engine)  
  
 def get\_all\_address(self):  
 s = self.session\_class()  
 tmp = s.query(Address).order\_by(Address.AddressID.asc()).all()  
 s.close()  
 return tmp  
  
 def get\_all\_orgs(self):  
 s = self.session\_class()  
 tmp = s.query(Оrganizers).order\_by(Оrganizers.ОrganizerID.asc()).all()  
 s.close()  
 return tmp  
  
 def get\_all\_tours(self):  
 s = self.session\_class()  
 tmp = s.query(Tourists).order\_by(Tourists.TouristID.asc()).all()  
 s.close()  
 return tmp  
 def get\_all\_phones(self):  
 s = self.session\_class()  
 tmp = s.query(Phone).order\_by(Phone.PhoneID.asc()).all()  
 s.close()  
 return tmp  
 def get\_all\_events(self):  
 s = self.session\_class()  
 tmp = s.query(Events).order\_by(Events.EventID.asc()).all()  
 s.close()  
 return tmp  
  
 def is\_email(self, email):  
 pattern = r"^[-\w\.]+@([-\w]+\.)+[-\w]{2,4}$"  
 if re.match(pattern, email) is not None:  
 return True  
 else:  
 return False  
  
 def not\_null(self, str):  
 if len(str) > 0:  
 return True  
 else:  
 return False  
  
 def is\_date(self, input\_string):  
 try:  
 datetime.strptime(input\_string, '%Y-%m-%d %H:%M:%S')  
 return True  
 except ValueError:  
 return False  
  
 def add\_tourist(self,f\_name,l\_name,email):  
 s = self.session\_class()  
 b = True  
 if self.is\_email(email) and self.not\_null(f\_name) and self.not\_null(l\_name):  
 s.add(Tourists(F\_name=f\_name, L\_name=l\_name, email=email))  
 try:  
 s.commit()  
 except exc.SQLAlchemyError:  
 b = False  
 else:  
 b = False  
  
  
 s.close()  
 return b  
  
  
 def add\_organizer(self,f\_name,l\_name,email):  
 s = self.session\_class()  
 b = True  
 if self.is\_email(email) and self.not\_null(f\_name) and self.not\_null(l\_name):  
 s.add(Оrganizers(F\_name=f\_name, L\_name=l\_name, email=email))  
 try:  
 s.commit()  
 except exc.SQLAlchemyError:  
 b = False  
 else:  
 b = False  
  
  
 s.close()  
 return b  
  
 def add\_event(self,title, type\_, date, address\_id):  
 s = self.session\_class()  
 b = True  
 if self.not\_null(title) and self.not\_null(type\_) and self.is\_date(date):  
 s.add(Events(Title=title, Type=type\_, Date=date,Address\_ID=address\_id))  
 try:  
 s.commit()  
 except exc.SQLAlchemyError:  
 b = False  
 else:  
 b = False  
  
  
 s.close()  
 return b  
  
 def add\_phone\_num(self,tourist\_id, number):  
 s = self.session\_class()  
 b = True  
 if self.not\_null(number):  
 s.add(Phone(Tourist\_ID=tourist\_id,Number=number))  
 try:  
 s.commit()  
 except exc.SQLAlchemyError:  
 b = False  
 else:  
 b = False  
  
  
 s.close()  
 return b  
  
 def update\_tour(self, id, f\_name, l\_name, email):  
 s = self.session\_class()  
 b = True  
 t = s.query(Tourists).get(id)  
 if t == None:  
 b = False  
 if b:  
 if self.not\_null(f\_name):  
 t.F\_name = f\_name  
 if self.not\_null(l\_name):  
 t.L\_name = l\_name  
 if self.not\_null(email):  
 if self.is\_email(email):  
 t.email = email  
 else:  
 b = False  
 if b:  
 try:  
 s.commit()  
 except exc.SQLAlchemyError:  
 b = False  
 s.close()  
 return b  
  
 def update\_org(self, id, f\_name, l\_name, email):  
 s = self.session\_class()  
 b = True  
 t = s.query(Оrganizers).get(id)  
 if t == None:  
 b = False  
 if b:  
 if self.not\_null(f\_name):  
 t.F\_name = f\_name  
 if self.not\_null(l\_name):  
 t.L\_name = l\_name  
 if self.not\_null(email):  
 if self.is\_email(email):  
 t.email = email  
 else:  
 b = False  
 if b:  
 try:  
 s.commit()  
 except exc.SQLAlchemyError:  
 b = False  
 s.close()  
 return b  
  
 def update\_event(self, id, title, type\_, date, address\_id):  
 s = self.session\_class()  
 b = True  
 t = s.query(Events).get(id)  
 if t == None:  
 b = False  
 if b:  
 if self.not\_null(title):  
 t.Title = title  
 if self.not\_null(type\_):  
 t.Type = type\_  
 if self.not\_null(date):  
 if self.is\_date(date):  
 t.Date = date  
 else:  
 b = False  
 if self.not\_null(address\_id):  
 c = s.query(Address).get(address\_id)  
 if c is not None:  
 t.Address\_ID = address\_id  
 else:  
 b = False  
 if b:  
 try:  
 s.commit()  
 except exc.SQLAlchemyError:  
 b = False  
 s.close()  
 return b  
  
  
 def update\_phone(self, id, tourist\_id, number):  
 s = self.session\_class()  
 b = True  
 t = s.query(Phone).get(id)  
 if t == None:  
 b = False  
 if b:  
 if self.not\_null(number):  
 t.Number = number  
 if self.not\_null(tourist\_id):  
 c = s.query(Tourists).get(tourist\_id)  
 if c is not None:  
 t.Tourist\_ID = tourist\_id  
 else:  
 b = False  
 if b:  
 try:  
 s.commit()  
 except exc.SQLAlchemyError:  
 b = False  
 s.close()  
 return b  
  
 def delete\_p(self,id):  
 s = self.session\_class()  
 t = s.query(Phone).get(id)  
 if t == None:  
 s.close()  
 return False  
 else:  
 s.delete(t)  
 s.commit()  
 s.close()  
 return True  
  
 def cnt\_t(self, id):  
 s = self.session\_class()  
 cnt\_t = s.query(TouristsEventsAssoc).filter(TouristsEventsAssoc.Tourist == id).count()  
 t = s.query(TouristsEventsAssoc).filter(TouristsEventsAssoc.Tourist == id)  
 cnt\_p = s.query(Phone).filter(Phone.Tourist\_ID == id).count()  
 p = s.query(Phone).filter(Phone.Tourist\_ID == id)  
 s.close()  
 return cnt\_t,t,cnt\_p,p  
  
 def cnt\_o(self, id):  
 s = self.session\_class()  
 cnt\_o = s.query(OrganizersEventsAssoc).filter(OrganizersEventsAssoc.Оrganizer == id).count()  
 o = s.query(OrganizersEventsAssoc).filter(OrganizersEventsAssoc.Оrganizer == id)  
 s.close()  
 return cnt\_o,o  
  
 def cnt\_e(self,id):  
 s = self.session\_class()  
 e1 = s.query(TouristsEventsAssoc).filter(TouristsEventsAssoc.Event\_t == id).count()  
 ee1 = s.query(TouristsEventsAssoc).filter(TouristsEventsAssoc.Event\_t == id)  
 e2 = s.query(OrganizersEventsAssoc).filter(OrganizersEventsAssoc.Event\_o == id).count()  
 ee2 = s.query(OrganizersEventsAssoc).filter(OrganizersEventsAssoc.Event\_o == id)  
 s.close()  
 return e1,ee1,e2,ee2  
  
 def delete\_t(self,id):  
 s = self.session\_class()  
 t = s.query(Tourists).get(id)  
 if t == None:  
 s.close()  
 return False  
 else:  
 s.delete(t)  
 s.commit()  
 s.close()  
 return True  
  
 def delete\_o(self,id):  
 s = self.session\_class()  
 t = s.query(Оrganizers).get(id)  
 if t == None:  
 s.close()  
 return False  
 else:  
 s.delete(t)  
 s.commit()  
 s.close()  
 return True  
  
 def delete\_e(self,id):  
 s = self.session\_class()  
 t = s.query(Events).get(id)  
 if t == None:  
 s.close()  
 return False  
 else:  
 s.delete(t)  
 s.commit()  
 s.close()  
 return True  
  
class OrganizersEventsAssoc(Base):  
 \_\_tablename\_\_ = 'Оrganizers\_Events'  
 Оrganizer = Column(Integer, ForeignKey('Оrganizers.ОrganizerID'), primary\_key=True)  
 Event\_o = Column(Integer, ForeignKey('Events.EventID'), primary\_key=True)  
  
  
class TouristsEventsAssoc(Base):  
 \_\_tablename\_\_ = 'Tourists\_Events'  
 Tourist = Column(Integer, ForeignKey('Tourists.TouristID'), primary\_key=True)  
 Event\_t = Column(Integer, ForeignKey('Events.EventID'), primary\_key=True)  
 booking\_time = Column(Date, primary\_key=True, unique=True)  
  
class Оrganizers(Base):  
 \_\_tablename\_\_ = 'Оrganizers'  
 ОrganizerID = Column(Integer, primary\_key=True, nullable=False, unique=True)  
 F\_name = Column(String(30), nullable=False)  
 L\_name = Column(String(30), nullable=False)  
 email = Column(String(30), nullable=False, unique=True)  
  
 events = relationship("Events", secondary='Оrganizers\_Events', back\_populates="organizers")  
  
class Events(Base):  
 \_\_tablename\_\_ = 'Events'  
 EventID = Column(Integer, primary\_key=True, nullable=False, unique=True)  
 Title = Column(String(100), nullable=False)  
 Type = Column(String(30), nullable=False)  
 Date = Column(Date, nullable=False)  
 Address\_ID = Column(Integer, ForeignKey('Address.AddressID'), nullable=False)  
  
 address = relationship("Address", back\_populates="events")  
 tourists = relationship("Tourists", secondary='Tourists\_Events', back\_populates="events")  
 organizers = relationship("Оrganizers", secondary='Оrganizers\_Events', back\_populates="events")  
  
class Tourists(Base):  
 \_\_tablename\_\_ = 'Tourists'  
 TouristID = Column(Integer, primary\_key=True, nullable=False, unique=True)  
 F\_name = Column(String(30), nullable=False)  
 L\_name = Column(String(30), nullable=False)  
 email = Column(String(30), nullable=False, unique=True)  
  
 phones = relationship("Phone", order\_by="Phone.PhoneID", back\_populates="tourist")  
 events = relationship("Events",secondary='Tourists\_Events', back\_populates="tourists")  
  
class Phone(Base):  
 \_\_tablename\_\_ = 'Phone\_nums'  
 PhoneID = Column(Integer, primary\_key=True, nullable=False, unique=True)  
 Tourist\_ID = Column(Integer, ForeignKey('Tourists.TouristID'), nullable=False)  
 Number = Column(String(16), nullable=False, unique=True)  
  
 tourist = relationship("Tourists", back\_populates="phones")  
  
class Address(Base):  
 \_\_tablename\_\_ = 'Address'  
 AddressID = Column(Integer, primary\_key=True, nullable=False, unique=True)  
 Country = Column(String(30), nullable=False)  
 City = Column(String(30), nullable=False)  
 Street = Column(String(30), nullable=False)  
 House = Column(String(30), nullable=False)  
  
 events = relationship("Events",order\_by="Events.EventID", back\_populates="address")