



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики
Кафедра системного програмування і спеціалізованих комп’ютерних систем

Лабораторна робота № 3
з дисципліни «Бази даних і засоби управління»
«Засоби оптимізації роботи СУБД PostgreSQL»

Виконав: Орлов Євгеній
Студент групи KB-92
Перевірив: Петрашенко А.В.

Київ 2021

Лабораторна робота №3

Метою роботи є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

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 12

<i>№ варіанта</i>	<i>Види індексів</i>	<i>Умови для тригера</i>
<i>12</i>	<i>Btree, Gin</i>	<i>After update, insert</i>

Посилання на Github: <https://github.com/CoachJohnsy/PostgreSQL-basics>

Завдання 1

Логічна модель предметної області «Школа»

Обрана предметна галузь передбачає обробку даних про учнів, їхні номери телефонів, вчителів, предмети та розклад школи.

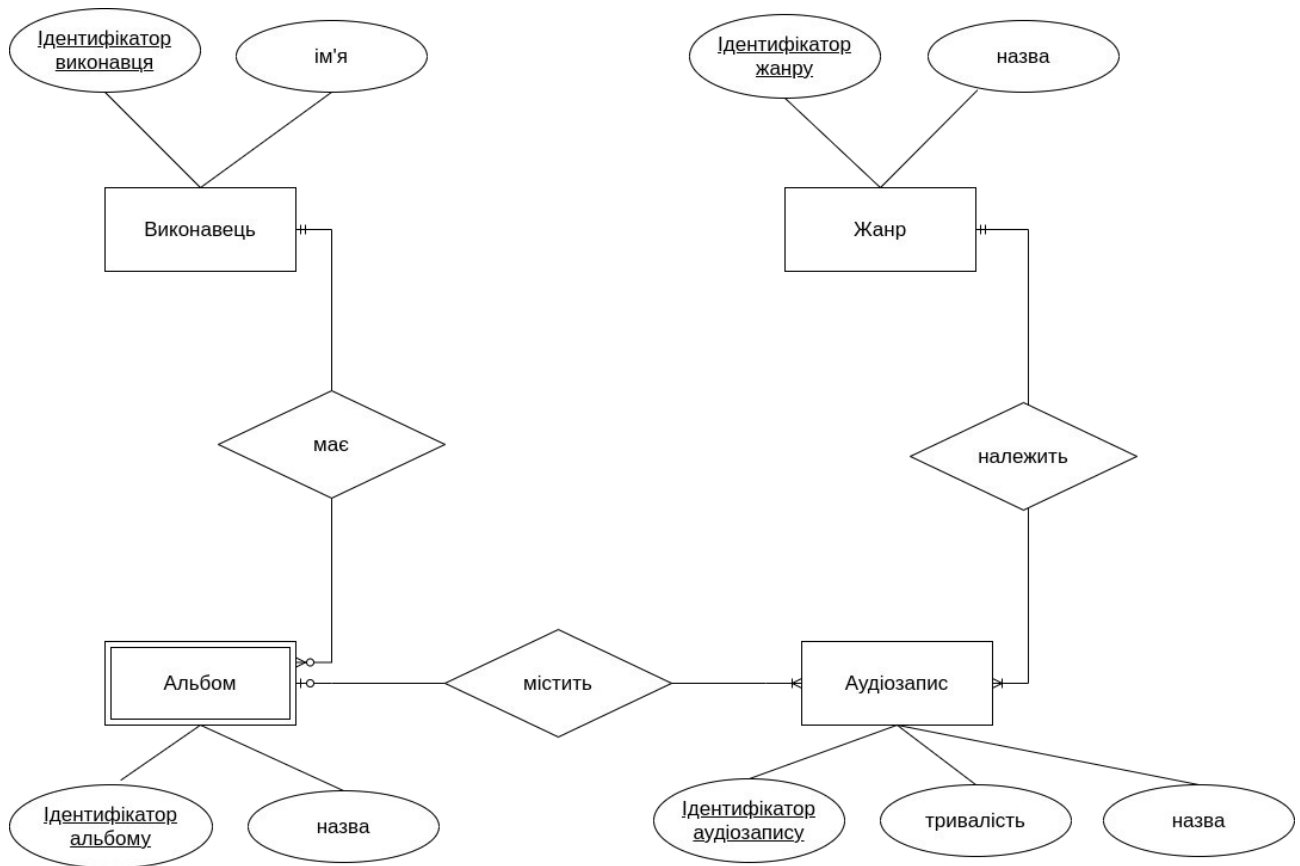
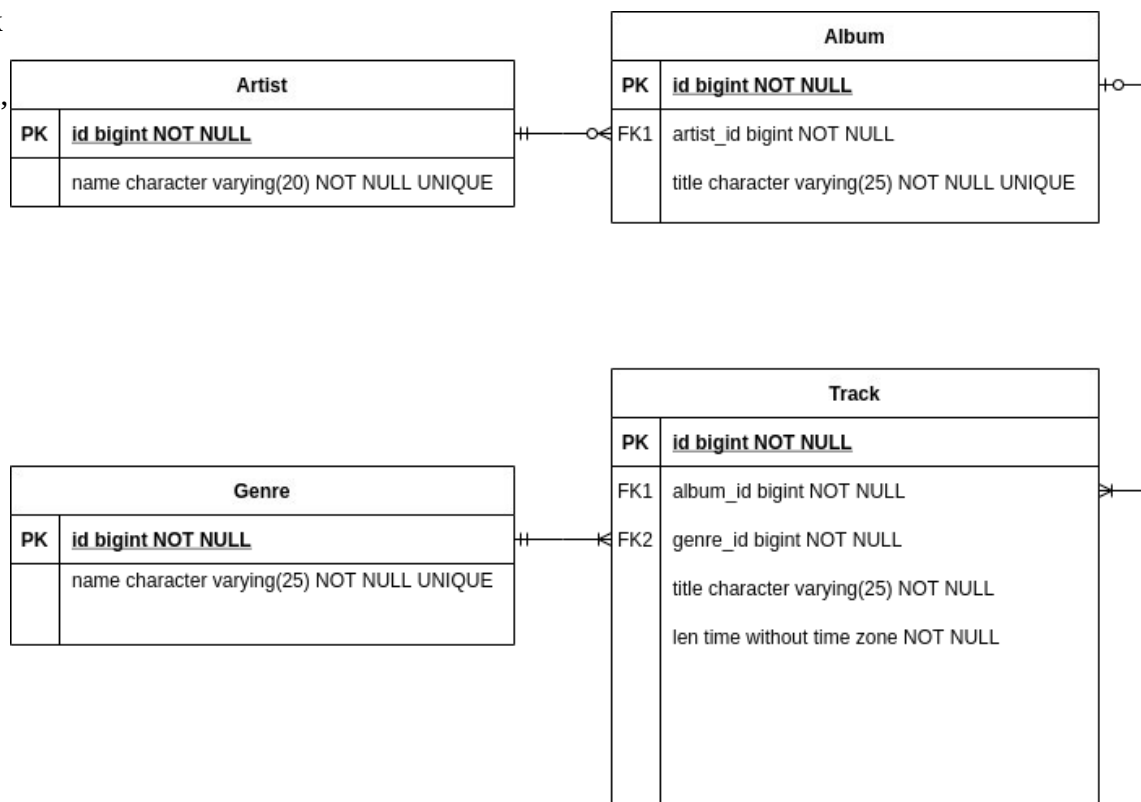


Рисунок 1. ER-діаграма

Рисунок
2 - ER-
діаграма,



переведена у таблиці БД

Середовище розробки та налаштування підключення до бази даних

Для виконання лабораторної роботи використовувалась мова програмування Python та середовище розробки PyCharm Professional 2021.3.

Для підключення до серверу бази даних PostgreSQL використано сторонню бібліотеку psycorg2, для реалізації моделі ORM використовувалася стороння бібліотека SQLAlchemy, середовище для відлагодження SQL-запитів до бази даних – psql.

Класи ORM

У даній лабораторній роботі було реалізовано 4 класи відповідно до 4 існуючих таблиць: Artist, Genre, Album, Track.

Таблиця Artist має стовпчики id (ідентифікатор) та name (ім'я), а також зв'язок 1:N із таблицею Album, тому в класі Artist встановлений зв'язок relationship('Album').

Таблиця Genre має стовпчики id (ідентифікатор), name (назва), а також зв'язок 1:N із таблицею Track, тому в класі Genre встановлений зв'язок relationship('Track').

Таблиця Album має стовпчики id (ідентифікатор), title (назва), tracks_number (розмір альбому), artist_id (зовнішній ключ з таблицею Artist), а також зв'язок 1:N із таблицею Track, тому в класі Album встановлений зв'язок relationship('Track').

Таблиця Track має стовпчики id (ідентифікатор), title (назва), len (тривалість), year (рік), number_within_album(порядковий номер в альбомі), album_id (зовнішній ключ, що пов'язує пісню з альбомом), genre_id (зовнішній ключ, що пов'язує пісню з жанром).

Нижче наведена програмна реалізація класів ORM мовою Python (лістинги усіх модулів надані нижче):

```

class Artist(Base):
    __tablename__ = 'artist'

    artist_id = Column(Integer, nullable=False, primary_key=True)
    name = Column(String, nullable=False, unique=True)

    album = relationship("Album", cascade="all, delete")

    def __init__(self, artist_id: int, name: str):
        self.artist_id = artist_id
        self.name = name

    def __repr__(self):
        return "{:>10}{:>35}".format(self.artist_id, self.name)

class Genre(Base):
    __tablename__ = 'genre'

    genre_id = Column(Integer, primary_key=True)
    name = Column(String, unique=True, nullable=False)

    track = relationship("Track", cascade="all, delete")

    def __init__(self, genre_id: int, name: str):
        self.genre_id = genre_id
        self.name = name

    def __repr__(self):
        return "{:>10}{:>35}".format(self.genre_id, self.name)

class Album(Base):
    __tablename__ = 'album'

    album_id = Column(Integer, primary_key=True)
    title = Column(String, unique=True, nullable=False)
    tracks_number = Column(Integer, nullable=False)

    artist_id = Column(Integer, ForeignKey("artist.artist_id"))

    track = relationship('Track', cascade="all, delete")

    def __init__(self, album_id: int, title: str, tracks_number: int,
artist_fk: int):
        self.album_id = album_id
        self.title = title
        self.tracks_number = tracks_number
        self.artist_id = artist_fk

    def __repr__(self):
        return "{:>10}{:>35}".format(self.album_id, self.tracks_number)

class Track(Base):
    __tablename__ = "track"

```

```

track_id = Column(Integer, primary_key=True)
title = Column(String, nullable=False)
len = Column(String, nullable=False)
year = Column(Integer, nullable=False)
number_within_album = Column(Integer, nullable=False)

genre_id = Column(Integer, ForeignKey("genre.genre_id"))
album_id = Column(Integer, ForeignKey("album.album_id"))

def __init__(self, track_id: int, title: str, length: str, year:
int,
                num_in_album: int, genre_id: int, album_fk: int):
    self.track_id = track_id
    self.title = title
    self.len = length
    self.year = year
    self.number_within_album = num_in_album
    self.genre_id = genre_id
    self.album_id = album_fk

def __repr__(self):
    return "{:>10}{:>35}{:>35}{:>10}{:>10}{:>10}{:>10}" \
        .format(self.track_id, self.title, self.len, self.year,
                self.number_within_album,
                self.genre_id, self.album_fk)

```

Приклади запитів у вигляді ORM

Запит вставки

Цей запит реалізовано за допомогою функції insert. Спочатку у меню користувач обирає опцію вставки, далі обирає таблицю, до якої хоче додати запис і вводить необхідні дані. Є перевірка введених даних. У разі успішного додавання запису користувач бачить відповідне повідомлення. Реалізацію запиту вставки продемонструємо на прикладі таблиці subject.

Загальні режими меню

```
+-----+-----+
| Command | Action |
+-----+-----+
|      0 | Show one table |
|      1 | Show all tables |
|      2 | Insert data |
|      3 | Delete data |
|      4 | Update data |
|      5 | Exit |
+-----+-----+
Choose an option:
```

```
+-----+-----+
|      Command      | Action      |
+-----+-----+
|           0       | artist      |
|           1       | album       |
|           2       | track       |
|           3       | genre       |
+-----+-----+
Choose table number: █
```

```
+-----+-----+
|  Command  | Action  |
+-----+-----+
|         0 | artist  |
|         1 | album   |
|         2 | track   |
|         3 | genre   |
+-----+-----+
Choose table number: 3
Enter genre_id value: 75
Enter name value: new_genre
+-----+-----+-----+
| table  |   id | action  |
+-----+-----+-----+
| genre  |   75 | inserted |
+-----+-----+-----+
Continue working with insert? Enter Yes or No
```

Лістинг функції insert

```
def insert(table_num: int, params: typing.Dict) -> int:
    Session = sessionmaker(bind=engine)
    session = Session()
    id = None

    if table_num == 0:
        id, name = tuple(params.values())
        s = Artist(artist_id=id, name=name)
        session.add(s)

    elif table_num == 1:
        id, title, tracks_number, artist_fk =
tuple(params.values())
        s = Album(album_id=id, title=title,
                    tracks_number=tracks_number,
artist_fk=artist_fk)
        session.add(s)

    elif table_num == 2:
        id, title, length, year, al_num, genre_id, album_id =
tuple(params.values())
        s = Track(track_id=id, title=title, length=length,
year=year, num_in_album=al_num,
                    genre_id=genre_id, album_fk=album_id)
        session.add(s)

    elif table_num == 3:
        id, name = tuple(params.values())
        s = Genre(genre_id=id, name=name)
```

```
session.add(s)
```

```
session.commit()  
return id
```

Запит видалення

Цей запит реалізовано за допомогою функції `delete`. Спочатку користувач обирає таблицю, з якої потрібно видалити дані. Потім потрібно ввести номер ідентифікатора запису для видалення. Якщо такого ідентифікатора не існує, то користувач побачить повідомлення про неіснування. У разі успішного видалення запису користувач побачить відповідне повідомлення.

Функціонал видалення записів працює рекурсивно, тобто власноруч реалізовує `delete on cascade`.

```
music_test=# select * from genre where genre_id = 58;  
genre_id | name  
-----+-----  
58 | Jazz  
(1 row)  
  
music_test=# select * from track where genre_id = 58;  
track_id | title | len | year | number_within_album | album_id | genre_id  
-----+-----+-----+-----+-----+-----+-----  
2809 | Strange Fruit | 00:01:28 | 1965 | 8 | 1471 | 58  
2810 | Be My Husband | 00:12:22 | 1965 | 1 | 1471 | 58  
2812 | The Other Woman (Live In New York/1964) | 00:19:21 | 1966 | 2 | 1472 | 58  
3265 | The Secret Garden (Sweet Seduction Suite) [feat. Barry White, El DeBarge, Al B. Sure! & James Ingram] | 00:19:30 | 1989 | 14 | 1613 | 58  
3318 | Sinnerman | 00:16:13 | 1965 | 5 | 1655 | 58  
3344 | I Put A Spell On You | 00:02:53 | 2021 | 13 | 1678 | 58  
(6 rows)  
  
music_test=#
```


Каскадне видалення

+-----+		
Command	Action	
+-----+		
0	Show one table	
1	Show all tables	
2	Insert data	
3	Delete data	
4	Update data	
5	Exit	

+-----+		
Command	Action	
+-----+		
0	artist	
1	album	
2	track	
3	genre	

Choose table number: 3

Enter genre_id value: 58

TERM environment variable not set.

+-----+				
table	col. value	column	action	
+-----+				
track	58	genre_id	deleted	
track	58	genre_id	deleted	
track	58	genre_id	deleted	
track	58	genre_id	deleted	
track	58	genre_id	deleted	
track	58	genre_id	deleted	
genre	58	genre_id	deleted	

Continue working with delete? Enter Yes or No

```

music_test=# select * from track where genre_id = 58;
 track_id | title
-----+-----
    2809 | Strange Fruit
    2810 | Be My Husband
    2812 | The Other Woman (Live In New York/1964)
    3265 | The Secret Garden (Sweet Seduction Suite) [feat. Barry White, El DeBarge,
    3318 | Sinnerman
    3344 | I Put A Spell On You
(6 rows)

music_test=# select * from track where genre_id = 58;
 track_id | title | len | year | number_within_album | album_id | genre_id
-----+-----+-----+-----+-----+-----+-----
(0 rows)

music_test=#

```

Лістинг функції delete

```

def delete(table_num: int, params: typing.Dict) -> typing.List:
    Session = sessionmaker(bind=engine)
    session = Session()

    deletion_logs = []
    id = list(params.values())[0]

    if table_num == 0:
        records = session.query(Artist).get(id)
        if records is not None:
            records = session.query(Album).filter(Album.artist_id == id).all()

            if records is not None:
                for record in records:
                    album_id = record.album_id
                    record = session.query(Track).filter(Track.album_id ==
album_id).all()
                    if record is not None:
                        delete = session.query(Track).filter(Track.album_id ==
album_id)

                        for i in delete:
                            session.delete(i)
                            deletion_logs.append(["track", album_id, "album_id",
"deleted"])

                        delete = session.query(Album).filter(Album.artist_id == id)

                        for i in delete:
                            session.delete(i)
                            deletion_logs.append(["album", id, "artist_id", "deleted"])

                    session.delete(session.query(Artist).
                        filter(Artist.artist_id == id).one())

                deletion_logs.append(["artist", id, "artist_id", "deleted"])

```

```

        else:
            deletion_logs.append(["artist", id, "artist_id", "not found"])

    elif table_num == 1:
        records = session.query(Album).get(id)
        if records is not None:
            records = session.query(Track).filter(Track.album_id == id).all()
            if records is not None:
                delete = session.query(Track).filter(Track.album_id == id)
                for i in delete:
                    session.delete(i)
                deletion_logs.append(["track", id, "album_id", "deleted"])

            session.delete(session.query(Album).filter(Album.album_id ==
id).one())
            deletion_logs.append(["album", id, "album_id", "deleted"])
        else:
            deletion_logs.append(["artist", id, "artist_id", "not found"])

    elif table_num == 2:
        records = session.query(Track).get(id)

        if records is not None:
            session.delete(session.query(Track).filter(Track.track_id ==
id).one())

            deletion_logs.append(["track", id, "track_id", "deleted"])
        else:
            deletion_logs.append(["track", id, "track_id", "not found"])

    elif table_num == 3:
        records = session.query(Genre).get(id)
        if records is not None:
            records = session.query(Track).filter(Track.genre_id == id).all()
            if records is not None:
                delete = session.query(Track).filter(Track.genre_id == id)
                for i in delete:
                    session.delete(i)
                deletion_logs.append(["track", id, "genre_id", "deleted"])

            session.delete(session.query(Genre).filter(Genre.genre_id ==
id).one())
            deletion_logs.append(["genre", id, "genre_id", "deleted"])
        else:
            deletion_logs.append(["genre", id, "genre_id", "not found"])
    else:
        "Input correct number"

    session.commit()
    return deletion_logs

```

Запит редагування

Цей запит реалізовано за допомогою функції update. Спочатку користувач обирає, у якій таблиці потрібно змінити запис і за яким ідентифікатором. Також потрібно обрати атрибут(и), що редагує(ю)ться. У разі успішного редагування користувач побачить відповідне повідомлення. Редагування запиту продемонструємо на прикладі таблиці track.

```
music_test=# select * from album where artist_id = 7;
 album_id |          title          | tracks_number | artist_id
-----+-----+-----+-----
      15 | 25 Years of Disco-Pop |          17 |         7
(1 row)

music_test=# select * from track where album_id = 15;
 track_id |      title      |  len  | year | number_within_album | album_id | genre_id
-----+-----+-----+-----+-----+-----+-----
      93 | Cheri Cheri Lady | 00:03:27 | 1985 |                2 |        15 |         6
     101 | Brother Louie   | 00:02:57 | 1986 |                3 |        15 |         6
(2 rows)
```

Введення даних для редагування

```
Choose an option: 4
+-----+-----+
|  Command | Action      |
+-----+-----+
|      0 | Show one table |
|      1 | Show all tables |
|      2 | Insert data   |
|      3 | Delete data   |
|      4 | Update data   |
|      5 | Exit         |
+-----+-----+

+-----+-----+
|  Command | Action      |
+-----+-----+
|      0 | artist      |
|      1 | album       |
|      2 | track       |
|      3 | genre       |
+-----+-----+

Choose table number: 2
Enter track_id to start updating: 101
Choose updating column from ['track_id', 'title', 'len', 'year', 'number_within_album', 'album_id', 'genre_id']: year
Enter year value: 2033
Press 'Y' to enter more columns to update. Press 'n' otherwise.Y
Choose updating column from ['track_id', 'title', 'len', 'year', 'number_within_album', 'album_id', 'genre_id']: title
Enter title value: Father Johnsy
Press 'Y' to enter more columns to update. Press 'n' otherwise.Y
Choose updating column from ['track_id', 'title', 'len', 'year', 'number_within_album', 'album_id', 'genre_id']: number_within_album
Enter number_within_album value: 1
Press 'Y' to enter more columns to update. Press 'n' otherwise.n
```

```

+-----+-----+-----+
| table  |    id | action  |
+-----+-----+-----+
| track  |   101 | updated |
+-----+-----+-----+
Continue working with update? Enter Yes or No

```

Вплив на базу даних

```

music_test=# select * from track where album_id = 15;
 track_id |      title      | len  | year | number_within_album | album_id | genre_id
-----+-----+-----+-----+-----+-----+-----+
      93 | Cheri Cheri Lady | 00:03:27 | 1985 |          2 |      15 |        6
     101 | Brother Louie    | 00:02:57 | 1986 |          3 |      15 |        6
(2 rows)

music_test=# select * from track where album_id = 15;
 track_id |      title      | len  | year | number_within_album | album_id | genre_id
-----+-----+-----+-----+-----+-----+-----+
      93 | Cheri Cheri Lady | 00:03:27 | 1985 |          2 |      15 |        6
     101 | Father Johnsy    | 00:02:57 | 2033 |          1 |      15 |        6
(2 rows)

music_test=#

```

Лістинг функції update

```

def update(table_num: int, params: typing.Tuple[str, ...]) -> typing.List:
    Session = sessionmaker(bind=engine)
    session = Session()

    updating_logs = []

    if table_num == 0:
        id, name = params
        artist = session.query(Album).get(id)
        if artist is not None:
            artist.name = name if name is not None else artist.name
            session.add(artist)
            updating_logs.append(["artist", id, "updated"])
        else:
            updating_logs.append(["artist", id, "not found"])

    elif table_num == 1:
        id, title, artist_id = params

```

```

album = session.query(Album).get(id)
if album is not None:
    album.title = title if title is not None else album.title
    album.album_id = artist_id if artist_id is not None else
album.artist_id
    session.add(album)
    updating_logs.append(["album", id, "updated"])
else:
    updating_logs.append(["album", id, "not found"])

elif table_num == 2:
    id, title, len, year, number_within_album, album_id, genre_id = params
    track = session.query(Track).get(params[0])
    if track is not None:
        track.title = title if title is not None else track.title
        track.len = len if len is not None else track.len
        track.year = year if year is not None else track.year
        track.number_within_album = number_within_album \
            if number_within_album else track.number_within_album
        track.album_id = album_id if album_id is not None else
track.album_id
        track.genre_id = genre_id if genre_id is not None else
track.genre_id

        session.add(track)
        updating_logs.append(["track", id, "updated"])
    else:
        updating_logs.append(["track", id, "not found"])

elif table_num == 3:
    id, name = params
    genre = session.query(Genre).get(id)
    if genre is not None:
        genre.name = name if name is not None else genre.name
        session.add(genre)
        updating_logs.append(["dgenre", id, "updated"])
    else:
        updating_logs.append(["gtenre", id, "not found"])

session.commit()
return updating_logs

```

Завдання 2

Для тестування індексів було створено окремі таблиці у базі даних test з 1000000 записів.
GIN

GIN призначений для обробки випадків, коли елементи, що підлягають індексації, є складеними значеннями (наприклад - реченнями), а запити, які обробляються індексом, мають шукати значення елементів, які з'являються в складених елементах (повторювані частини слів або речень). Індекс GIN зберігає набір пар (ключ, список появи ключа), де список появи — це набір ідентифікаторів рядків, у яких міститься ключ. Один і той самий ідентифікатор рядка може знаходитись у кількох списках, оскільки елемент може містити більше одного ключа. Кожне значення ключа зберігається лише один раз, тому індекс GIN дуже швидкий для випадків, коли один і той же ключ з'являється багато разів. Цей індекс може взаємодіяти тільки з полем типу tsvector.

SQL запити

Створення таблиці БД:

```

DROP TABLE IF EXISTS "gin_test";
CREATE TABLE "gin_test"("id" bigserial PRIMARY KEY, "string" text, "gin_vector"
tsvector);
INSERT INTO "gin_test"("string") SELECT substr(characters,
(random()*length(characters)+1)::integer, 10) FROM
(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as
symbols(characters), generate_series(1, 1000000) as q;
UPDATE "gin_test" set "gin_vector" = to_tsvector("string");

```

Запити для тестування:

Було протестовано 4 запити: 1 - виведення записів, у яких ідентифікатор кратний числу 2; 2 - виведення записів, у яких наявне сполучення букв bnm; 3 - виведення суми ідентифікаторів записів, у яких наявні сполучення букв QWERTYUIO або bnm; 4 - виведення мінімального ідентифікатора та максимального ідентифікатора записів, де є сполучення букв bnm, сортування за кратними 2 ідентифікаторами.

```

SELECT COUNT(*) FROM "gin_test" WHERE "id" % 2 = 0;
SELECT COUNT(*) FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm'));
SELECT SUM("id") FROM "gin_test" WHERE ("gin_vector" @@
to_tsquery('QWERTYUIOP')) OR ("gin_vector" @@ to_tsquery('bnm'));
SELECT MIN("id"), MAX("id") FROM "gin_test" WHERE ("gin_vector" @@
to_tsquery('bnm')) GROUP BY "id" % 2;

```

Створення індексу:

```

DROP INDEX IF EXISTS "gin_index";
CREATE INDEX "gin_index" ON "gin_test" USING gin("gin_vector");

```

Результати виконання запитів

Запити без індексування:

```

music_test=# CREATE TABLE "gin_test"("id" bigserial PRIMARY KEY, "string" text, "gin_vector" tsvector);
CREATE TABLE
music_test=# INSERT INTO "gin_test"("string") SELECT substr(characters, (random()*length(characters)+1)::
integer, 10) FROM (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters)
, generate_series(1, 1000000) as q;

INSERT 0 1000000
music_test=# UPDATE "gin_test" set "gin_vector" = to_tsvector("string");
UPDATE 1000000
music_test=#
music_test=# \timing on
Timing is on.
music_test=# SELECT COUNT(*) FROM "gin_test" WHERE "id" % 2 = 0;
count
-----
500000
(1 row)

Time: 92,733 ms
music_test=# SELECT COUNT(*) FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm'));
count
-----
19016
(1 row)

Time: 415,615 ms
music_test=# SELECT SUM("id") FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('QWERTYUIOP')) OR ("gin_v
ector" @@ to_tsquery('bnm'));
sum
-----
23863327444
(1 row)

Time: 915,954 ms
music_test=# SELECT MIN("id"), MAX("id") FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm')) GROUP
BY "id" % 2;
min | max
-----+-----
146 | 999954
201 | 999905
(2 rows)

Time: 419,639 ms
music_test=#

```


Час виконання запитів

Операція 1	Операція 2	Операція 3	Операція 4
92.733 мс	415.615 мс	915.954 мс	419.639 мс

Табл.1 – Час виконання запитів без індексу GIN

```
music_test=# CREATE INDEX "gin_index" ON "gin_test" USING gin("gin_vector");
CREATE INDEX
Time: 459,115 ms
music_test=# SELECT COUNT(*) FROM "gin_test" WHERE "id" % 2 = 0;
count
-----
500000
(1 row)

Time: 91,471 ms
music_test=# SELECT COUNT(*) FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm'));
count
-----
19016
(1 row)

Time: 85,169 ms
music_test=# SELECT SUM("id") FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('QWERTYUIOP')) OR ("gin_v
ector" @@ to_tsquery('bnm'));
sum
-----
23863327444
(1 row)

Time: 60,856 ms
music_test=# SELECT MIN("id"), MAX("id") FROM "gin_test" WHERE ("gin_vector" @@ to_tsquery('bnm')) GROUP
BY "id" % 2;
min | max
-----+-----
146 | 999954
201 | 999905
(2 rows)

Time: 54,766 ms
music_test=#
```

Час виконання запитів

Операція 1	Операція 2	Операція 3	Операція 4
91.471 мс	85.169 мс	60.85 мс	54.766 мс

Табл.2 – Час виконання операції з індексом GIN

З отриманих результатів бачимо, що в усіх заданих випадках пошук з індексацією відбувається значно швидше, ніж пошук без індексації (окрім першого, оскільки на перший запит дана індексація не впливає). Це відбувається завдяки головній особливості індексування GIN: кожне значення шуканого ключа зберігається один раз і запит іде не по всій таблиці, а лише по тим даним, що містяться у списку появи цього ключа. Для даних типу numeric даний тип індексування використовувати недоцільно і неможливо.

BTree

Індекс BTree призначений для даних, які можна відсортувати. Іншими словами, для типу даних мають бути визначені оператори «більше», «більше або дорівнює», «менше», «менше або дорівнює» та «дорівнює». Пошук починається з кореня вузла, і потрібно визначити, по якому з дочірніх вузлів спускатися. Знаючи ключі в корені, можна зрозуміти діапазони значень в дочірніх вузлах. Процедура повторюється до тих пір, поки не буде знайдено вузол, з якого можна отримати необхідні дані.

SQL запити

Створення таблиці БД і внесення 1000000 записів:

```
DROP TABLE IF EXISTS "btree_test";
CREATE TABLE "btree_test"("id" bigserial PRIMARY KEY, "time" timestamp);
INSERT INTO "btree_test"("time") SELECT (timestamp '2021-01-01' +
random()*(timestamp '2020-01-01'-timestamp '2022-01-01')) FROM
(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as
symbols(characters), generate_series(1, 1000000) as q;
```

Запити для тестування:

Було протестовано 4 запити: 1 – виведення записів, ідентифікатор яких кратний 2 (рис.18 та рис.21); 2 - виведення записів, у яких час більше або дорівнює 2019-10-01 (рис.19 та рис.21); 3 - виведення середнього значення ідентифікаторів записів, у яких час знаходиться в проміжку між 2019-10-01 та 2021-12-7 (рис.19 і рис.21); 4 - виведення суми ідентифікаторів, а також максимального ідентифікатора записів, у яких час знаходиться в проміжку між 2020-05-05 та 2021-05-05, сортування за кратними 2 ідентифікаторами (рис.20 і рис.21).

```
SELECT COUNT(*) FROM "btree_test" WHERE "id" % 2 = 0;
SELECT COUNT(*) FROM "btree_test" WHERE "time" >= '20191001';
SELECT AVG("id") FROM "btree_test" WHERE "time" >= '20191001' AND "time" <=
'20211207';
SELECT SUM("id"), MAX("id") FROM "btree_test" WHERE "time" >= '20200505' AND
"time" <= '20210505' GROUP BY "id"%2;
```

Створення індексу:

```
DROP INDEX IF EXISTS "btree_index";
CREATE INDEX "btree_time_index" ON "btree_test" ("id");
```

Результати виконання запитів

Запити без індексування:

Час виконання запитів

Операція 1	Операція 2	Операція 3	Операція 4
102.622 мс	98.768 мс	2.2 мс	4.524 мс

Табл.3 – Час виконання запитів без індексу BTree

```
Time: 0,758 ms
music_test=# SELECT COUNT(*) FROM "btree_test" WHERE "id" % 2 = 0;
count
-----
500000
(1 row)

Time: 82,632 ms
music_test=# SELECT COUNT(*) FROM "btree_test" WHERE 'time' >= '20191001';
count
-----
1000000
(1 row)

Time: 75,768 ms
music_test=# SELECT AVG("id") FROM "btree_test" WHERE 'time' >= '20191001' AND 'time' <= '20211207';
avg
-----
(1 row)

Time: 0,700 ms
music_test=# SELECT SUM("id"), MAX("id") FROM "btree_test" WHERE 'time' >= '20200505' AND 'time' <= '2021
0505' GROUP BY "id" %2;
sum | max
-----+-----
(0 rows)

Time: 1,024 ms
music_test=#
```

Рис.21 – Запити з індексуванням

Час виконання запитів

Операція 1	Операція 2	Операція 3	Операція 4
82,848 мс	75 мс	0.7 мс	1.024 мс

Табл.4 – Час виконання запитів з індексом Btree

Як бачимо з результатів, за допомогою використання індексу BTree виконання операцій дещо пришвидшилося. Це пов'язано з тим, що дерево утворює багато гілок, і через це В-дерево виходить неглибоким навіть для дуже великих таблиць. Цей індекс також рекомендовано використовувати саме для операцій пошуку з порівнянням (нерівностями), що і було продемонстровано в запитах.

Завдання 3

Для тестування тригера було створено дві таблиці в базі даних test: таблиця trigger_test з атрибутами trigger_testID (ідентифікатор) trigger_testName (ім'я), trigger_test_log з атрибутами id (ідентифікатор), trigger_test_log_ID (зовнішній ключ для зв'язку з таблицею trigger_test), trigger_test_log_name (ім'я).

Тригер спрацьовує після операції вставки (after insert) та під час операції редагування (update). Серед усіх записів таблиці trigger_test у курсорному циклі обираються ті, що мають ідентифікатори кратні 2. Якщо цей ідентифікатор також кратний 3, то висвічується повідомлення, що число ділиться на 2 і 3. Також якщо ідентифікатор кратний 2 і 3, то в таблицю trigger_test_log вставляються рядки з цими ідентифікаторами та відповідними іменами. В іншому випадку (якщо число не ділиться на 3, але ділиться на 2), викликається повідомлення - «Число парне» і в таблицю trigger_test_log вставляються рядки з цими ідентифікаторами та відповідними іменами. Далі з атрибуту trigger_test_log_name видаляються набори символів 'log'. Якщо число не ділиться на 2, то висвічується повідомлення «Число непарне» і виконується редагування в курсорному циклі: для всіх записів таблиці trigger_test_log, що мають в назві сполучення букв '_id' потрібно замінити ім'я на '_' та trigger_test_log_name та '_log'.

Тригер спрацьовує, якщо викликати операцію редагування (update) або встави (insert). Нижче на знімках екрану продемонстровано коректну роботу тригера. Запити для створення таблиць:

```
DROP TABLE IF EXISTS "trigger_test";
CREATE TABLE "trigger_test"(
"trigger_testID" bigserial PRIMARY KEY,
"trigger_testName" text
);
```

```
DROP TABLE IF EXISTS "trigger_test_log";
CREATE TABLE "trigger_test_log"(
"id" bigserial PRIMARY KEY,
"trigger_test_log_ID" bigint,
"trigger_test_log_name" text
);
```

Команди, що ініціюють виконання тригера:

```
CREATE TRIGGER "after_update_insert_trigger"
AFTER INSERT OR UPDATE ON "trigger_test"
FOR EACH ROW
EXECUTE procedure after_update_func();
Початковий вміст таблиці trigger_test було задано запитом:
INSERT INTO trigger_test("trigger_testName") VALUES ('test1'), ('test2'),
('test3'), ('test4'), ('test5'), ('test6');
```

Текст тригера:

```
CREATE OR REPLACE FUNCTION after_update_func() RETURNS TRIGGER AS $trigger$
DECLARE
CURSOR LOG CURSOR FOR SELECT * FROM "trigger_test_log";
row_ "trigger_test_log"%ROWTYPE;
BEGIN
```

```

IF NEW."trigger_testID"%2=0 THEN
IF NEW."trigger_testID"%3=0 THEN
RAISE NOTICE 'trigger_testID is multiple of 2 and 3';
FOR row_ IN CURSOR_LOG LOOP
-- UPDATE "trigger_test_log" SET "trigger_test_log_name"='_' ||
row_."trigger_test_log_name" || '_log' WHERE "id"=row_."id";
INSERT INTO "trigger_test_log"("trigger_test_log_ID", "trigger_test_log_name")
VALUES (NEW."trigger_testID", NEW."trigger_testName");
END LOOP;
RETURN NEW;
ELSE
RAISE NOTICE 'trigger_testID is even';
INSERT INTO "trigger_test_log"("trigger_test_log_ID", "trigger_test_log_name")
VALUES (NEW."trigger_testID", NEW."trigger_testName");
UPDATE "trigger_test_log" SET "trigger_test_log_name" = trim(BOTH '_' FROM
"trigger_test_log_name");
RETURN NEW;
END IF;
ELSE
RAISE NOTICE 'trigger_testID is odd';
FOR row_ IN CURSOR_LOG LOOP
UPDATE "trigger_test_log" SET "trigger_test_log_name" = '_' ||
row_."trigger_test_log_name" || '_log' WHERE "id" = row_."id";
END LOOP;
RETURN NEW;
END IF;
END;
$trigger$ LANGUAGE plpgsql;

CREATE TRIGGER after_update_test
AFTER INSERT OR UPDATE ON "trigger_test"
FOR EACH ROW EXECUTE PROCEDURE after_update_func();

```

Результат виконання

Приклад виконання наведено для таблиць trigger_test та trigger_test_log.

	trigger_testID [PK] bigint	trigger_testName text

Початковий стан таблиці trigger_test

	id [PK] bigint	trigger_test_log_ID bigint	trigger_test_log_name text

Початковий стан таблиці trigger_test_log

Далі було виконано запит на вставку.

```
INSERT INTO trigger_test("trigger_testName") VALUES ('test7'), ('test8');
```

	trigger_testID [PK] bigint	trigger_testName text
1	1	test1
2	2	test2
3	3	test3
4	4	test4
5	5	test5
6	6	test6

Таблиця trigger_test після виконання операції вставки

	id [PK] bigint	trigger_test_log_ID bigint	trigger_test_log_name text
1	1	2	_test2_log
2	2	4	_test4_log
3	3	6	test6
4	4	6	test6

Таблиця trigger_test_log після виконання операції вставки

Запит на редагування (додає у всі записи із парними ідентифікаторами в імені сполучення символів '_2'):

```
UPDATE "trigger_test" SET "trigger_testName" = "trigger_testName" || '_2' WHERE "trigger_testID"%2=0
```

	trigger_testID [PK] bigint	trigger_testName text
1	1	test1
2	2	test2
3	3	test3
4	4	test4
5	5	test5
6	6	test6

Початковий стан таблиці trigger_test

	id [PK] bigint	trigger_test_log_ID bigint	trigger_test_log_name text
1	1	2	_test2_log
2	2	4	_test4_log
3	3	6	test6
4	4	6	test6

Початковий стан таблиці trigger_test_log

	trigger_testID [PK] bigint	trigger_testName text
1	1	test1
2	2	test2_2
3	3	test3
4	4	test4_2
5	5	test5
6	6	test6_2

Стан таблиці trigger_test після виконання операції редагування

	id [PK] bigint	trigger_test_log_ID bigint	trigger_test_log_name text
1	1	2	test2_2
2	2	4	test4_2
3	3	6	test6_2
4	4	6	test6_2

Стан таблиці trigger_test_log після виконання операції редагування та спрацювання тригера. Як видно на знімках, записи із парними ідентифікаторами записалися в таблицю trigger_test_log, аналогічно до першого випадку із запитом вставки. Алгоритмічно нічого не змінилося, оскільки всі дії виконуються в залежності від значення ідентифікатора, який не змінювався.

Якщо виконати запити вставки та редагування по черзі, то ситуація дещо зміниться.

Спочатку у таблицю trigger_test вставляються записи з ідентифікаторами від 1 до 8. Далі для парних ідентифікаторів записи копіюються в trigger_test_log, а запис з ідентифікатором 6 копіюється в таблицю тричі, оскільки він ділиться і на 2, і на 3.

	trigger_testID [PK] bigint	trigger_testName text

Стан таблиці trigger_test до почергової вставки та редагування

	id [PK] bigint	trigger_test_log_ID bigint	trigger_test_log_name text

Стан таблиці trigger_test_log до почергової вставки та редагування

	trigger_testID [PK] bigint	trigger_testName text
1	1	test1
2	2	test2_2
3	3	test3
4	4	test4_2
5	5	test5
6	6	test6_2
7	7	test7
8	8	test8_2

Стан таблиці trigger_test після вставки та редагування

	id [PK] bigint	trigger_test_log_ID bigint	trigger_test_log_name text
1	1	8	test8
2	2	2	test2_2
3	3	4	test4_2
4	4	6	test6_2
5	5	6	test6_2
6	6	6	test6_2
7	7	8	test8_2

Стан таблиці trigger_test_log після вставки та редагування

Завдання 4

Для цього завдання знадобилась окрема таблиця “transactions” з атрибутами id (ідентифікатор), numeric (число), text (текст). Також у таблицю було додано три записи за допомогою запиту вставки insert.

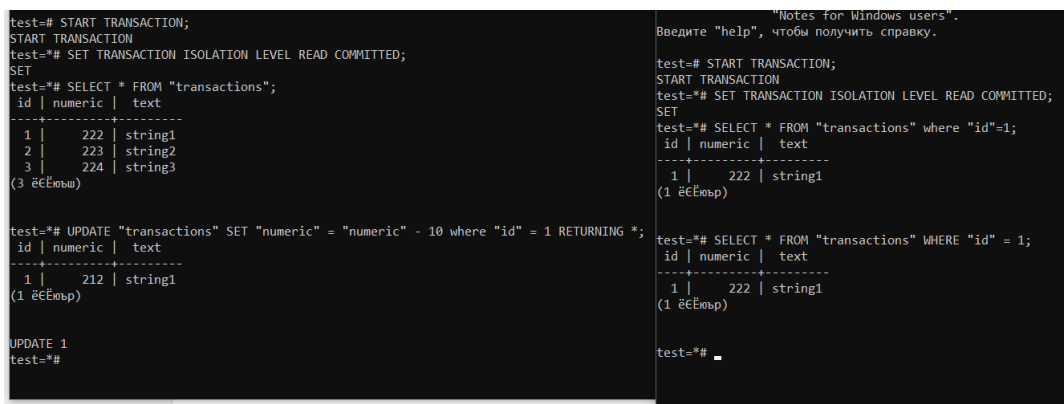
Запит на створення таблиці та вставку:

```
DROP TABLE IF EXISTS "transactions";
CREATE TABLE "transactions"(
  "id" bigserial PRIMARY KEY,
    "numeric" bigint,
    "text" text
);
INSERT INTO "transactions"("numeric", "text") VALUES (222, 'string1'), (223, 'string2'), (224, 'string3');
```

READ COMMITTED

На цьому рівні ізоляції одна транзакція не бачить змін у базі даних, викликаних іншою доки та не завершить своє виконання (командою COMMIT або ROLLBACK).

Спочатку у транзакціях 1 і 2 таблиця має однаковий стан. Якщо у транзакції 1 виконати редагування одного рядка, то в транзакції 2 цих змін не буде помітно, поки в першій транзакції не буде команди commit. Таким чином, феномен «брудного читання» на цьому рівні ізоляції неможливий.



The screenshot shows two terminal windows side-by-side, illustrating a Read Committed transaction isolation level. The left window shows a transaction (test=#) that starts, sets the isolation level to READ COMMITTED, and then selects all rows from the 'transactions' table, showing three rows with IDs 1, 2, and 3. It then updates the 'numeric' column of the first row (ID 1) from 222 to 212. The right window shows another transaction (test=#) that also starts, sets the isolation level to READ COMMITTED, and then selects the row where ID=1. It shows only one row with ID 1 and numeric 222, indicating it did not see the update made in the first transaction. The terminal output for the left window is as follows:

```
test=# START TRANSACTION;
START TRANSACTION
test=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET
test=# SELECT * FROM "transactions";
 id | numeric | text
-----+-----+-----
  1 |    222 | string1
  2 |    223 | string2
  3 |    224 | string3
(3 @@@@)

test=# UPDATE "transactions" SET "numeric" = "numeric" - 10 where "id" = 1 RETURNING *;
 id | numeric | text
-----+-----+-----
  1 |    212 | string1
(1 @@@@)

UPDATE 1
test=#
```

The terminal output for the right window is as follows:

```
"Notes for Windows users".
Введите "help", чтобы получить справку.

test=# START TRANSACTION;
START TRANSACTION
test=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET
test=# SELECT * FROM "transactions" where "id"=1;
 id | numeric | text
-----+-----+-----
  1 |    222 | string1
(1 @@@@)

test=# SELECT * FROM "transactions" WHERE "id" = 1;
 id | numeric | text
-----+-----+-----
  1 |    222 | string1
(1 @@@@)

test=#
```

Виконання редагування в одній з одночасних транзакцій на рівні ізоляції Rad committed. Тепер дослідимо феномен «фантомного читання». Після команди commit у першій транзакції у другій ми побачимо, що умові numeric >= 220 відповідають лише 2 рядки, а не 3, як раніше, оскільки зміни були внесені і збережені в обох транзакціях.


```
UPDATE 1
test=# COMMIT;
ПРЕДУПРЕЖДЕНИЕ: нет незавершённой транзакции
COMMIT
test=#
```

```
test=# UPDATE "transactions" SET "numeric" = "numeric"-10 WHERE "id"=1 RETURNING *;
ОШИБКА: не удалось сериализовать доступ из-за параллельного изменения
test=# ROLLBACK;
ROLLBACK
test=#
```

Помилка через паралельні зміни на рівні ізоляції Repeatable read

Дослідимо аномалію серіалізації. На рівні ізоляції repeatable read запустимо дві транзакції. У першій виведемо всі рядки і порахуємо суму стовпчика numeric у всіх записах. Додаємо запис із цим значенням в таблицю. Якщо у другій транзакції повторити ті ж самі операції, то стан таблиці на початку ще не змінений, сума буде такою ж, як у першій транзакції. Таким чином, ми додамо до таблиці такий самий рядок, як і першій транзакції. Виконуючи commit в обох транзакціях, ми побачимо два однакових записи в таблиці. Це і є феномен «серіалізації», що пояснюється серійним виконанням двох транзакцій однієї за одною, причому порядок виконання транзакції неважливий.

```
test=# START TRANSACTION;
START TRANSACTION
test=# SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SET
test=# SELECT * FROM "transactions";
 id | numeric | text
-----+-----+-----
  2 |    223 | string2
  3 |    224 | string3
  1 |    202 | string1
(3 rows)

test=# SELECT SUM("numeric") FROM "transactions";
 sum
-----
 649
(1 row)

test=# INSERT INTO "transactions"("numeric", "text") VALUES (649, 'string4') RETURNING *;
 id | numeric | text
-----+-----+-----
  4 |    649 | string4
(1 row)

INSERT 0 1
test=# SELECT * FROM "transactions";
 id | numeric | text
-----+-----+-----
  2 |    223 | string2
  3 |    224 | string3
  1 |    202 | string1
  4 |    649 | string4
(4 rows)

test=# commit;
COMMIT
test=#
```

```
test=# START TRANSACTION;
START TRANSACTION
test=# SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SET
test=# SELECT * FROM "transactions";
 id | numeric | text
-----+-----+-----
  2 |    223 | string2
  3 |    224 | string3
  1 |    202 | string1
(3 rows)

test=# SELECT SUM("numeric") FROM "transactions";
 sum
-----
 649
(1 row)

test=# INSERT INTO "transactions"("numeric", "text") VALUES (649, 'string4') RETURNING *;
 id | numeric | text
-----+-----+-----
  5 |    649 | string4
(1 row)

INSERT 0 1
test=# commit;
COMMIT
test=# SELECT * FROM "transactions";
 id | numeric | text
-----+-----+-----
  2 |    223 | string2
  3 |    224 | string3
  1 |    202 | string1
  4 |    649 | string4
  5 |    649 | string4
(5 rows)
```

Аномалія серіалізації

SERIALIZABLE

Запустимо дві транзакції на рівні Serializable. Спочатку стан таблиці однаковий. У першій транзакції видалимо рядок з id = 5 та виконаємо редагування рядку з

id = 4. Якщо у другій транзакції спробувати зробити ті ж операції, то ми повинні будемо очікувати, доки перша транзакція не завершиться. Коли команда commit у першій транзакції виконана, у другій виникає помилка через паралельне видалення. Це неможливо, оскільки якщо запис уже видалений в першій транзакції, то видалити рядок з неіснуючим ідентифікатором неможливо. Ситуацію рятує команда rollback, і після цього бачимо, що зміни внесені і в другу транзакцію.

```
test=# START TRANSACTION;
START TRANSACTION
test=# SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET
test=# SELECT * FROM "transactions";
 id | numeric | text
-----+-----+-----
  2 |    223 | string2
  3 |    224 | string3
  1 |    202 | string1
  4 |    649 | string4
  5 |    649 | string4
(5 rows)

test=# DELETE FROM "transactions" WHERE "id"=5;
DELETE 1
test=# SELECT * FROM "transactions";
 id | numeric | text
-----+-----+-----
  2 |    223 | string2
  3 |    224 | string3
  1 |    202 | string1
  4 |    649 | string4
(4 rows)
```

```
test=# SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET
test=# SELECT * FROM "transactions";
 id | numeric | text
-----+-----+-----
  2 |    223 | string2
  3 |    224 | string3
  1 |    202 | string1
  4 |    649 | string4
  5 |    649 | string4
(5 rows)

test=# DELETE FROM "transactions" WHERE "id"=5;
```

Спроба одночасного виконання запитів на рівні ізоляції Serializable

```
test=# UPDATE "transactions" SET "text" = 'new_string' WHERE "id" = 4;
UPDATE 1
test=# SELECT * FROM "transactions";
 id | numeric | text
-----+-----+-----
  2 |    223 | string2
  3 |    224 | string3
  1 |    202 | string1
  4 |    649 | new_string
(4 rows)

test=# COMMIT;
COMMIT
test=#
```

```
test=# DELETE FROM "transactions" WHERE "id"=5;
ОШИБКА: не удалось сериализовать доступ из-за параллельного удаления
test=# SELECT * FROM "transactions";
ОШИБКА: текущая транзакция прервана, команды до конца блока транзакции игнорируются
test=# ROLLBACK;
ROLLBACK
test=# SELECT * FROM "transactions";
 id | numeric | text
-----+-----+-----
  2 |    223 | string2
  3 |    224 | string3
  1 |    202 | string1
  4 |    649 | new_string
(4 rows)

test=#
```

Помилка через паралельне вилучення запису на рівні ізоляції Serializable

main.py

```
from controller import Controller

if __name__ == '__main__':
    Controller.menu()
```

view.py

```
import typing

from tabulate import tabulate
import os

class View:
    def __init__(self, table, records):
        self.table = table
        self.records = records

    @staticmethod
    def cls():
        os.system('cls' if os.name == 'nt' else 'clear')

    @staticmethod
    def display_table_stdout(rows, headers):
        print(tabulate(rows, headers=headers, tablefmt="psql",
showindex=False))

    @staticmethod
    def display_attr_mistype_stdout(col: str, req_type: type,
entered_val: str):
        print(f"[ERROR] Entered value: ({entered_val}) of
attribute ({col}) "
            f"does not not match required type ({req_type})")

    @staticmethod
    def print_stdout(msg: str) -> None:
        print(msg)

    @staticmethod
    def get_stdin(msg: str) -> typing.Any:
        return input(msg)
```

controller.py

```

import collections
import typing

from view import View
from constants import MENU_ROWS, MENU_COLUMNS
import model

class Validator:

    @staticmethod
    def validate_table_num_range(num_of_tables: int, choice: int) -> bool:
        return 0 <= choice <= num_of_tables - 1

    @staticmethod
    def validate_table_num_type(choice: str) -> bool:
        return choice.isdigit()

    @staticmethod
    def validate_input_items(req_types: typing.Dict[str, type], column: str,
attr_val: str) -> bool:
        try:
            req_types[column](attr_val)
            return False
        except ValueError:
            View.display_attr_mistype_stdout(column, req_types[column],
attr_val)
            return True

class Controller:
    __TABLES = {num: table_name for num, table_name in
enumerate(model.get_table_names())}

    @staticmethod
    def table_num_input(num_of_tables: int) -> int:
        while True:
            View.display_table_stdout(
                rows=[
                    [command, action] for command, action in enumerate(
                        Controller.__TABLES.values())],
                headers=MENU_COLUMNS
            )
            num = View.get_stdin('Choose table number: ')
            if Validator.validate_table_num_type(num):
                table_num = int(num)
                if Validator.validate_table_num_range(num_of_tables, table_num):
                    return table_num
            else:
                View.cls()
                View.print_stdout('Incorrect input, try again.')
        else:
            View.cls()
            View.print_stdout('Incorrect input, try again.')

    @staticmethod
    def update_col_val_handle(table_name: str, table_columns: typing.List) ->
typing.Tuple[str, ...]:
        req_types = collections.OrderedDict({column: col_type
                                                for column, col_type in
model.get_table_attr_types(table_name)})

```

```

        tmp = dict()
        entered_all_columns = False

        id = View.get_stdin(f"Enter {table_columns[0]} to start updating: ")
        while Validator.validate_input_items(req_types, table_columns[0], id):
            id = View.get_stdin(f"Enter {table_columns[0]} to start updating: ")
    ")

    tmp[table_columns[0]] = id
    while not entered_all_columns:
        upd_col = View.get_stdin(f"Choose updating column from
{table_columns}: ")
        while upd_col not in table_columns:
            View.cls()
            View.print_stdout(f"column {upd_col} is not provided. Try
again")
            upd_col = View.get_stdin(f"Choose updating column from
{table_columns}: ")

        upd_val = View.get_stdin(f"Enter {upd_col} value: ")
        while Validator.validate_input_items(req_types, upd_col, upd_val):
            upd_val = View.get_stdin(f"Enter {upd_col} value: ")

        tmp[upd_col] = upd_val
        user_command = View.get_stdin(f"Press \'Y\' to enter more columns to
update. "
                                     f"Press \'N\' otherwise.")

        if user_command != 'Y':
            entered_all_columns = True
            View.cls()
            entered_values = tuple(tmp[col] if col in tmp.keys() else None for col
in table_columns)
            return entered_values

    @staticmethod
    def column_value_input(table_name: str, table_columns: typing.List,
                           only_id: bool = False) -> typing.OrderedDict |
typing.Dict:

        req_types = collections.OrderedDict({column: col_type
                                             for column, col_type in
model.get_table_attr_types(table_name)})
        entered_values = collections.OrderedDict()
        for column in table_columns:
            not_validated = True
            attr_val = ""
            while not_validated:
                attr_val = View.get_stdin(f"Enter {column} value: ")
                not_validated = Validator.validate_input_items(req_types,
column, attr_val)
                View.cls()
            else:
                entered_values[column] = attr_val
            if only_id:
                return {column: int(attr_val)}
        return entered_values

    @staticmethod
    def is_continue(mode: str, end_mode: bool) -> bool:
        incorrect = True
        while incorrect:
            answer = View.get_stdin(f'Continue working with {mode}? Enter Yes or

```

```

No ')
    if answer == 'No':
        end_mode = True
        incorrect = False
    elif answer == 'Yes':
        incorrect = False
        pass
    else:
        View.print_stdout('Please, enter Yes or No')

    return end_mode

@staticmethod
def menu():
    while True:

        View.display_table_stdout(rows=MENU_ROWS, headers=MENU_COLUMNS)
        choice = View.get_stdin("Choose an option: ")

        if choice == '0':
            View.cls()
            table_num =
Controller.table_num_input(len(Controller.__TABLES.values()))

View.display_table_stdout(model.get_rows(Controller.__TABLES[table_num]),
model.get_table_columns(Controller.__TABLES[table_num]))

            elif choice == '1':
                View.cls()
                for table_num in Controller.__TABLES.keys():
View.display_table_stdout(model.get_rows(Controller.__TABLES[table_num]),
model.get_table_columns(Controller.__TABLES[table_num]))
                    elif choice == '2':
                        View.cls()

                        end_insert = False
                        while not end_insert:
                            View.display_table_stdout(rows=MENU_ROWS,
headers=MENU_COLUMNS)
                            table_num =
Controller.table_num_input(len(Controller.__TABLES.values()))

                            entered_values = Controller.column_value_input(
                                table_name=Controller.__TABLES[table_num],
table_columns=model.get_table_columns(Controller.__TABLES[table_num]))

                            inserted_id = model.insert(table_num, entered_values)
                            View.display_table_stdout([[Controller.__TABLES[table_num],
inserted_id, "inserted"]],
                                                    headers=["table", "id", "action"])

                            end_insert = Controller.is_continue("insert", end_insert)

                        elif choice == '3':
                            end_delete = False
                            while not end_delete:
                                View.display_table stdout(rows=MENU_ROWS,

```

```

headers=MENU_COLUMNS)
        table_num =
Controller.table_num_input(len(Controller.__TABLES.values()))

        entered_values = Controller.column_value_input(
            table_name=Controller.__TABLES[table_num],

table_columns=model.get_table_columns(Controller.__TABLES[table_num]),
            only_id=True)

        deletion_logs = model.delete(table_num, entered_values)

        View.display_table_stdout(deletion_logs, headers=["table",
"col. value", "column", "action"])

        end_delete = Controller.is_continue("delete", end_delete)

    elif choice == '4':
        end_update = False
        while not end_update:
            View.display_table_stdout(rows=MENU_ROWS,
headers=MENU_COLUMNS)
            table_num =
Controller.table_num_input(len(Controller.__TABLES.values()))

            entered_values = Controller.update_col_val_handle(
                table_name=Controller.__TABLES[table_num],

table_columns=model.get_table_columns(Controller.__TABLES[table_num]))
            # )

            updating_logs = model.update(table_num, entered_values)
            View.display_table_stdout(updating_logs, headers=["table",
"id", "action"])

            end_update = Controller.is_continue("update", end_update)
    elif choice == '5':
        break
    else:
        View.print_stdout('Please, enter valid number')

```

model.py

```

import typing

import sqlalchemy
from sqlalchemy import Column, Integer, String, create_engine, ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
from sqlalchemy.orm import sessionmaker

Base = declarative_base()
engine =
create_engine('postgresql+psycopg2://postgres:20071944__@localhost:5432/
music_test')

class Artist(Base):

```



```

__tablename__ = 'artist'

artist_id = Column(Integer, nullable=False, primary_key=True)
name = Column(String, nullable=False, unique=True)

album = relationship("Album", cascade="all, delete")

def __init__(self, artist_id: int, name: str):
    self.artist_id = artist_id
    self.name = name

def __repr__(self):
    return "{:>10}{:>35}".format(self.artist_id, self.name)

class Genre(Base):
    __tablename__ = 'genre'

    genre_id = Column(Integer, primary_key=True)
    name = Column(String, unique=True, nullable=False)

    track = relationship("Track", cascade="all, delete")

    def __init__(self, genre_id: int, name: str):
        self.genre_id = genre_id
        self.name = name

    def __repr__(self):
        return "{:>10}{:>35}".format(self.genre_id, self.name)

class Album(Base):
    __tablename__ = 'album'

    album_id = Column(Integer, primary_key=True)
    title = Column(String, unique=True, nullable=False)
    tracks_number = Column(Integer, nullable=False)

    artist_id = Column(Integer, ForeignKey("artist.artist_id"))

    track = relationship('Track', cascade="all, delete")

    def __init__(self, album_id: int, title: str, tracks_number: int, artist_fk:
int):
        self.album_id = album_id
        self.title = title
        self.tracks_number = tracks_number
        self.artist_id = artist_fk

    def __repr__(self):
        return "{:>10}{:>35}".format(self.album_id, self.tracks_number)

class Track(Base):
    __tablename__ = "track"

    track_id = Column(Integer, primary_key=True)
    title = Column(String, nullable=False)
    len = Column(String, nullable=False)
    year = Column(Integer, nullable=False)
    number_within_album = Column(Integer, nullable=False)

```

```

genre_id = Column(Integer, ForeignKey("genre.genre_id"))
album_id = Column(Integer, ForeignKey("album.album_id"))

def __init__(self, track_id: int, title: str, length: str, year: int,
              num_in_album: int, genre_id: int, album_fk: int):
    self.track_id = track_id
    self.title = title
    self.len = length
    self.year = year
    self.number_within_album = num_in_album
    self.genre_id = genre_id
    self.album_id = album_fk

def __repr__(self):
    return "{:>10}{:>35}{:>35}{:>10}{:>10}{:>10}{:>10}" \
        .format(self.track_id, self.title, self.len, self.year,
                self.number_within_album,
                self.genre_id, self.album_fk)

def get_rows(table_name: str) -> typing.Set | None:
    Session = sessionmaker(bind=engine)
    session = Session()
    records = None

    if "artist" == table_name:
        records = session.query(Artist.artist_id, Artist.name).all()
    elif "album" == table_name:
        records = session.query(Album.album_id, Album.title,
                                Album.tracks_number, Album.artist_id).all()
    elif "track" == table_name:
        records = session.query(Track.track_id, Track.title,
                                Track.len, Track.year,
                                Track.number_within_album,
                                Track.album_id, Track.genre_id).all()
    elif "genre" == table_name:
        records = session.query(Genre.genre_id, Genre.name).all()

    return records

def get_table_names() -> typing.List:
    return sqlalchemy.inspect(engine).get_table_names(schema="public")

def get_table_columns(table_name: str) -> typing.List:
    inspector = sqlalchemy.inspect(engine)
    return [col_descriptor["name"]
            for col_descriptor in inspector.get_columns(table_name,
schema="public")]

def get_table_attr_types(table_name: str) -> typing.List:
    inspector = sqlalchemy.inspect(engine)
    return [(col_descriptor["name"], col_descriptor["type"].python_type)
            for col_descriptor in inspector.get_columns(table_name,
schema="public")]

def insert(table_num: int, params: typing.Dict) -> int:
    Session = sessionmaker(bind=engine)
    session = Session()

```

```

id = None

if table_num == 0:
    id, name = tuple(params.values())
    s = Artist(artist_id=id, name=name)
    session.add(s)

elif table_num == 1:
    id, title, tracks_number, artist_fk = tuple(params.values())
    s = Album(album_id=id, title=title,
              tracks_number=tracks_number, artist_fk=artist_fk)
    session.add(s)

elif table_num == 2:
    id, title, length, year, al_num, genre_id, album_id =
tuple(params.values())
    s = Track(track_id=id, title=title, length=length, year=year,
num_in_album=al_num,
              genre_id=genre_id, album_fk=album_id)
    session.add(s)

elif table_num == 3:
    id, name = tuple(params.values())
    s = Genre(genre_id=id, name=name)
    session.add(s)

session.commit()
return id

def delete(table_num: int, params: typing.Dict) -> typing.List:
    Session = sessionmaker(bind=engine)
    session = Session()

    deletion_logs = []
    id = list(params.values())[0]

    if table_num == 0:
        records = session.query(Artist).get(id)
        if records is not None:
            records = session.query(Album).filter(Album.artist_id == id).all()

            if records is not None:
                for record in records:
                    album_id = record.album_id
                    record = session.query(Track).filter(Track.album_id ==
album_id).all()
                    if record is not None:
                        delete = session.query(Track).filter(Track.album_id ==
album_id)

                        for i in delete:
                            session.delete(i)
                            deletion_logs.append(["track", album_id, "album_id",
"deleted"])

                        delete = session.query(Album).filter(Album.artist_id == id)

                        for i in delete:
                            session.delete(i)
                            deletion_logs.append(["album", id, "artist_id", "deleted"])

```

```

        session.delete(session.query(Artist).
                        filter(Artist.artist_id == id).one())

        deletion_logs.append(["artist", id, "artist_id", "deleted"])
    else:
        deletion_logs.append(["artist", id, "artist_id", "not found"])

elif table_num == 1:
    records = session.query(Album).get(id)
    if records is not None:
        records = session.query(Track).filter(Track.album_id == id).all()
        if records is not None:
            delete = session.query(Track).filter(Track.album_id == id)
            for i in delete:
                session.delete(i)
            deletion_logs.append(["track", id, "album_id", "deleted"])

        session.delete(session.query(Album).filter(Album.album_id ==
id).one())
        deletion_logs.append(["album", id, "album_id", "deleted"])
    else:
        deletion_logs.append(["artist", id, "artist_id", "not found"])

elif table_num == 2:
    records = session.query(Track).get(id)

    if records is not None:
        session.delete(session.query(Track).filter(Track.track_id ==
id).one())

        deletion_logs.append(["track", id, "track_id", "deleted"])
    else:
        deletion_logs.append(["track", id, "track_id", "not found"])

elif table_num == 3:
    records = session.query(Genre).get(id)
    if records is not None:
        records = session.query(Track).filter(Track.genre_id == id).all()
        if records is not None:
            delete = session.query(Track).filter(Track.genre_id == id)
            for i in delete:
                session.delete(i)
            deletion_logs.append(["track", id, "genre_id", "deleted"])

        session.delete(session.query(Genre).filter(Genre.genre_id ==
id).one())
        deletion_logs.append(["genre", id, "genre_id", "deleted"])
    else:
        deletion_logs.append(["genre", id, "genre_id", "not found"])
else:
    "Input correct number"

session.commit()
return deletion_logs

def update(table_num: int, params: typing.Tuple[str, ...]) -> typing.List:
    Session = sessionmaker(bind=engine)
    session = Session()

    updating_logs = []

```

```

if table_num == 0:
    id, name = params
    artist = session.query(Album).get(id)
    if artist is not None:
        artist.name = name if name is not None else artist.name
        session.add(artist)
        updating_logs.append(["artist", id, "updated"])
    else:
        updating_logs.append(["artist", id, "not found"])

elif table_num == 1:
    id, title, artist_id = params
    album = session.query(Album).get(id)
    if album is not None:
        album.title = title if title is not None else album.title
        album.album_id = artist_id if artist_id is not None else
album.artist_id
        session.add(album)
        updating_logs.append(["album", id, "updated"])
    else:
        updating_logs.append(["album", id, "not found"])

elif table_num == 2:
    id, title, len, year, number_within_album, album_id, genre_id = params
    track = session.query(Track).get(params[0])
    if track is not None:
        track.title = title if title is not None else track.title
        track.len = len if len is not None else track.len
        track.year = year if year is not None else track.year
        track.number_within_album = number_within_album \
            if number_within_album else track.number_within_album
        track.album_id = album_id if album_id is not None else
track.album_id
        track.genre_id = genre_id if genre_id is not None else
track.genre_id

        session.add(track)
        updating_logs.append(["track", id, "updated"])
    else:
        updating_logs.append(["track", id, "not found"])

elif table_num == 3:
    id, name = params
    genre = session.query(Genre).get(id)
    if genre is not None:
        genre.name = name if name is not None else genre.name
        session.add(genre)
        updating_logs.append(["dgenre", id, "updated"])
    else:
        updating_logs.append(["ggenre", id, "not found"])

session.commit()
return updating_logs

```

constants.py

```

MENU_ROWS = [(0, "Show one table"),
              (1, "Show all tables"),
              (2, "Insert data"),
              (3, "Delete data"),

```

```
(4, "Update data"),  
(5, "Exit")]
```

```
MENU_COLUMNS =\  
["Command",  
 "Action"]
```