

Міністерство освіти та 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 №2
з дисципліни
Бази даних та засоби управління

Виконав:
студент 3-го курсу,
групи КВ-23
Литвин Максим
Телеграм: @desp1c

Київ 2024

Постановка задачі:

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.

[Посилання на github](#)

Варіант 14:

14	<i>Btree, Hash</i>	<i>after insert, update</i>
----	--------------------	-----------------------------

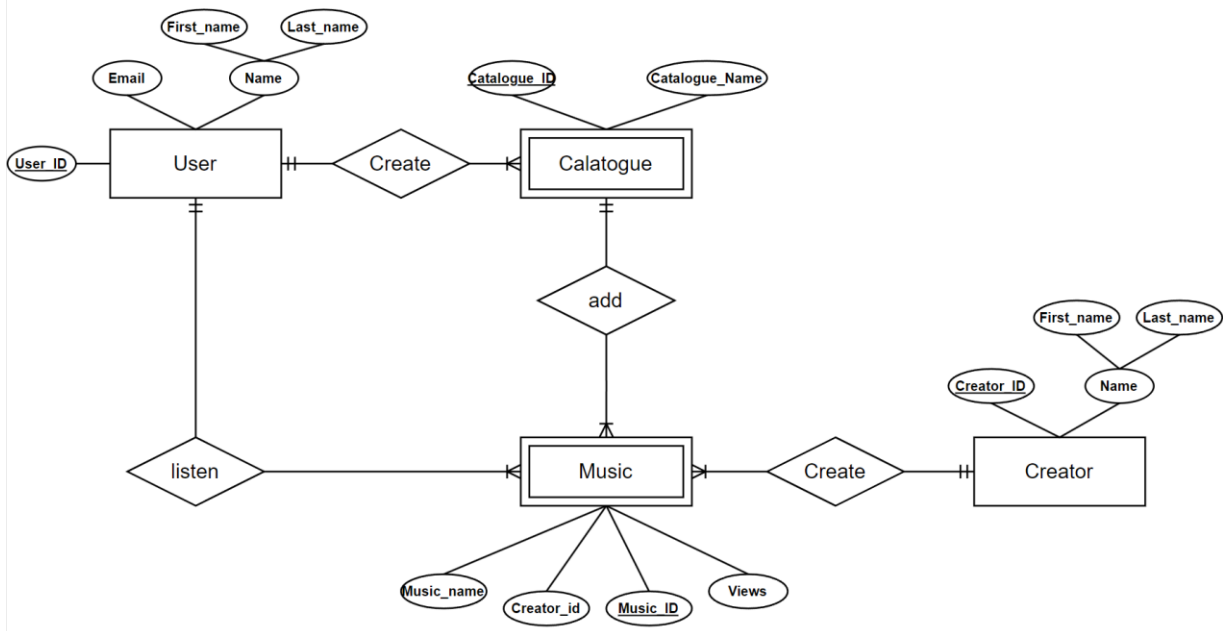
Структура бази даних:

User – користувач, який може створювати безліч каталогів, та має змогу додавати до кожного з них необхідну йому кількість музики до кожного каталогу.

Catalogue – каталог, який зберігає в собі посилання на довільну кількість музики, до якої через каталог має доступ користувач до прослуховування.

Music – музика, яка зберігає зв’язок між автором (творцем) та каталогом за яким вона закріплена в нашого користувача.

Creator – автор довільної кількості музики, яку може слухати наш користувач.



ER-діаграма в стилі «нотації Чена»

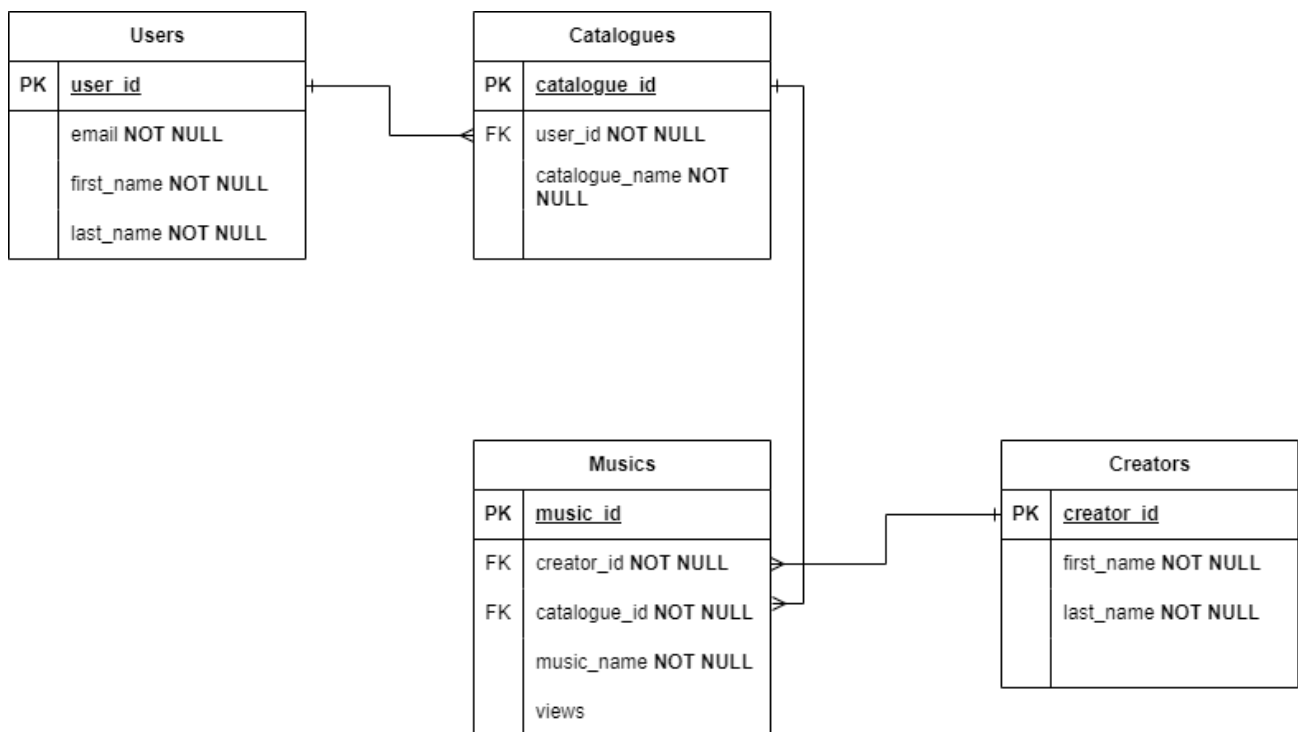
Зв'язки:

1:N – між user та catalogue (один користувач може створити безліч каталогів) – create(1)

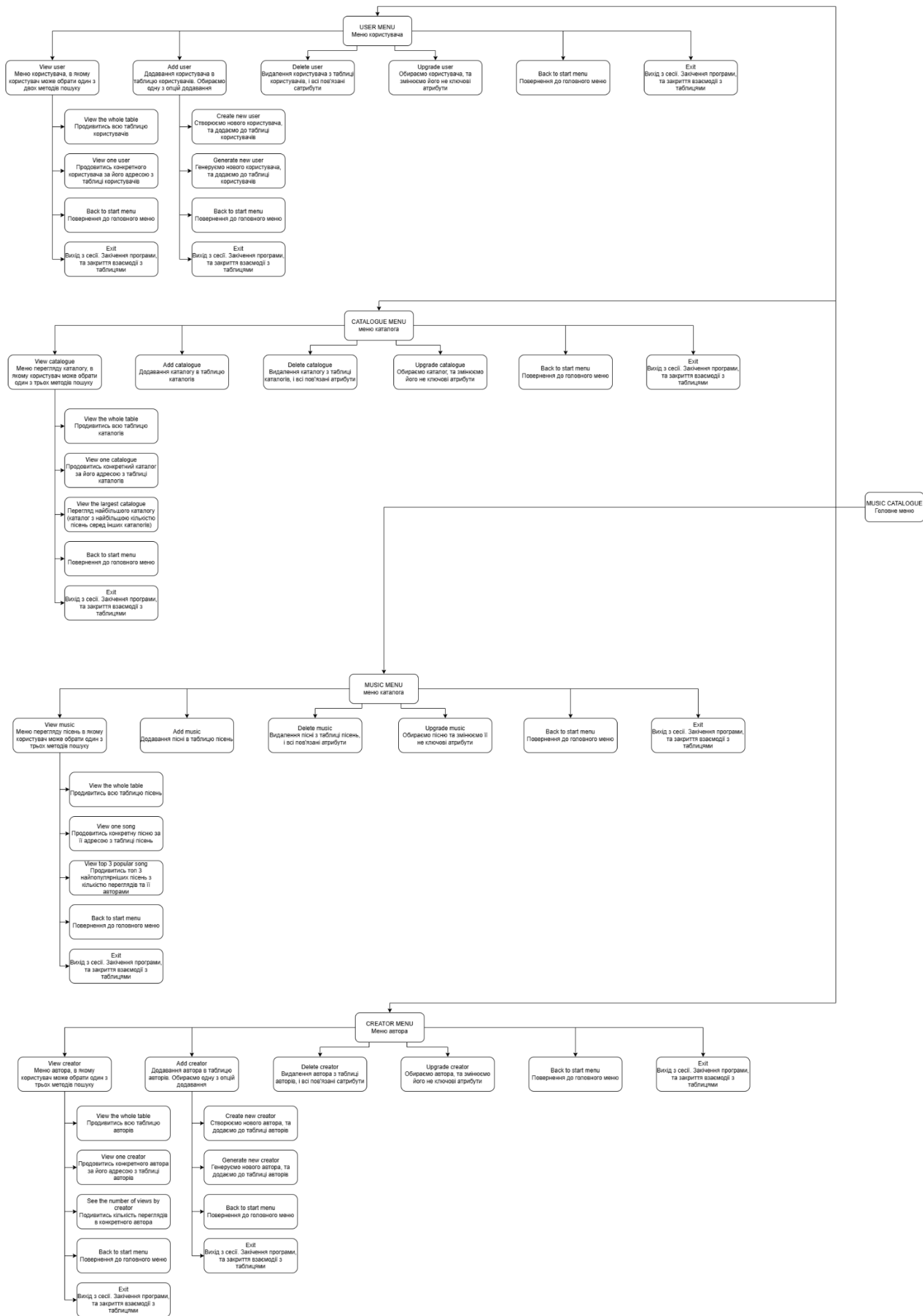
1:N – між user та music (один користувач може слухати безліч музики) – listen

1:N – між catalogue та music (один каталог може мати безліч музики) – add

1:N – між music та creator (один автор може створити безліч музики) – create(2)



Структура бази даних



Структура меню користувача

Мова програмування та додаткові бібліотеки:

Java, та базовий набір бібліотек.

Додаткові бібліотеки: [postgresql](#), [hibernate](#)

Етап №1:

Сутності:

User

```
@Entity
@Table(name = "users")
public class User {
    @Id
    @Column(name="user_id", unique = true, nullable = false)
    private int userId;

    @Column(name="email", length = 30)
    private String email;

    @Column(name="first_name", length = 20)
    private String firstName;

    @Column(name="last_name", length = 20)
    private String lastName;

    public User(){

    }

    public User(int userId, String email, String firstName, String
lastName){
        this.userId = userId;
        this.email = email;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public int getUserId() {
        return userId;
    }

    public String getEmail() {
        return email;
    }

    public String getFirstName() {
        return firstName;
    }
}
```

```

    }

    public String getLastName() {
        return lastName;
    }

```

Catalogue

```

@Entity
@Table(name="catalogues")
public class Catalogue {
    @Id
    @Column(name="catalogue_id", unique=true, nullable = false)
    int catalogueId;

    @Column(name="catalogue_name", length=20, nullable = false)
    String catalogueName;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "user_id", referencedColumnName = "user_id",
nullable = false)
    private User user;

    @OneToMany(mappedBy = "catalogue", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    private List<Music> musics;

    public Catalogue() {
    }

    public Catalogue(int catalogueId, String catalogueName, User user) {
        this.catalogueId = catalogueId;
        this.catalogueName = catalogueName;
        this.user = user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public int getCatalogueId() {
        return catalogueId;
    }

    public String getCatalogueName() {
        return catalogueName;
    }

    public User getUser() {
        return user;
    }

```

```
}  
}
```

Music

```
@Entity  
@Table(name="musics")  
public class Music {  
    @Id  
    @Column(name="music_id", length = 5, unique = true, nullable = false)  
    int musicId;  
  
    @Column(name="music_name", length = 20, nullable = false)  
    String musicName;  
  
    @Column(name="views", nullable = false)  
    int views;  
  
    @ManyToOne(fetch = FetchType.EAGER)  
    @JoinColumn(name = "catalogue_id", referencedColumnName =  
"catalogue_id", nullable = false)  
    private Catalogue catalogue;  
  
    @ManyToOne(fetch = FetchType.EAGER)  
    @JoinColumn(name = "creator_id", referencedColumnName = "creator_id",  
nullable = false)  
    private Creator creator;  
  
    public Music() {  
  
    }  
  
    public Music(int musicId, String musicName, Creator creator,  
Catalogue catalogue, int views) {  
        this.musicId = musicId;  
        this.musicName = musicName;  
        this.creator = creator;  
        this.catalogue = catalogue;  
        this.views = views;  
    }  
  
    public int getMusicId() {  
        return musicId;  
    }  
  
    public String getMusicName() {  
        return musicName;  
    }  
  
    public Creator getCreator() {  
        return creator;  
    }  
  
    public Catalogue getCatalogue() {
```



```

        return catalogue;
    }

    public int getViews() {
        return views;
    }

    public void setCatalogue(Catalogue catalogue){
        this.catalogue = catalogue;
    }

    public void setCreator(Creator creator) {
        this.creator = creator;
    }
}

```

Creator

```

@Entity
@Table(name="creators")
public class Creator {
    @Id
    @Column(name="creator_id", length=5, unique=true, nullable = false)
    int creatorId;

    @Column(name="first_name", length=20, nullable = false)
    String firstName;
    @Column(name="last_name", length=20, nullable = false)
    String lastName;

    public Creator(){
    }

    public Creator(int creatorId, String firstName, String lastName){
        this.creatorId = creatorId;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public int getCreatorId() {
        return creatorId;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }
}

```

Скріншоти:

Головне меню:

```
MUSIC CATALOGUE
Choose one option:
1. User table.
2. Catalogue table.
3. Music table.
4. Creator table.
9. Back to start menu.
0. Exit.
-> 1
```

Меню користувача:

```
USER MENU
Choose what you wish to do:
1. View user.
2. Add user.
3. Delete user.
4. Update user.
9. Back to start menu.
0. Exit
-> 1
```

Меню перегляду:

```
Select one of the methods for viewing the table:
1. View the whole table.
2. View one user.
9. Back to start menu.
0. Exit
-> 1
```

Результат перегляду всієї таблиці:

```
user id = 1      name = Maksym Lytvyn      email = maxtenday6785@gmail.com
user id = 2      name = Gregory Skovoroda  email = test@fakemail.com
user id = 3      name = Bogdan Sopilnyak   email = bogdan.sopilnyak@gmail.com
user id = 6      name = fff lll            email = lllfff@gmail.com
user id = 7      name = ivan nikolaev      email = ivan.nikolaev@gmail.com
user id = 8      name = Arthur Stone       email = arthur.stone398@gmail.com
user id = 10     name = Nine Eleven        email = elevennine@gmail.com
```

Результат перегляду одного користувача:

```
Select one of the methods for viewing the table:
1. View the whole table.
2. View one user.
9. Back to start menu.
0. Exit
-> 2
Enter the user id of the user you want to view:
-> 7
user id = 7      name = ivan nikolaev      email = ivan.nikolaev@gmail.com
```

Меню додавання користувача:

```
Select one of the methods for adding to the table:
1. Create new user.
2. Generate new user.
9. Back to start menu.
0. Exit
```

Додавання користувача власноруч:

```
Select one of the methods for adding to the table:
1. Create new user.
2. Generate new user.
9. Back to start menu.
0. Exit
-> 1
Enter user id:
-> 9

Enter user first name: |
-> Alexandr
Enter user last name:
-> Lopatyn
Enter user email:
-> alxlopatyn@test.com
```

Додавання користувачів генерацією:

```
Select one of the methods for adding to the table:
1. Create new user.
2. Generate new user.
9. Back to start menu.
0. Exit
-> 2
Enter how many users you want to generate:
-> 100000
100000 records inserted successfully.
```

Перевіряємо результат:

```
user id = 99980 name = Ava Davies email = ava.davies540@gmail.com
user id = 99981 name = Lily Lewis email = lily.lewis168@gmail.com
user id = 99982 name = Lily Williams email = lily.williams545@gmail.com
user id = 99983 name = Harry Lewis email = harry.lewis138@gmail.com
user id = 99984 name = Freya Lewis email = freya.lewis645@gmail.com
user id = 99985 name = Lily Stone email = lily.stone546@gmail.com
user id = 99986 name = John Morgan email = john.morgan21@gmail.com
user id = 99987 name = Florence Lebron email = florence.lebron70@gmail.com
user id = 99988 name = Noah Davies email = noah.davies12@gmail.com
user id = 99989 name = Leo Stone email = leo.stone385@gmail.com
user id = 99990 name = Arthur Morgan email = arthur.morgan27@gmail.com
user id = 99991 name = Amelia Williams email = amelia.williams640@gmail.com
user id = 99992 name = Noah Evans email = noah.evans356@gmail.com
user id = 99993 name = Mike Bell email = mike.bell537@gmail.com
user id = 99994 name = Florence Nelson email = florence.nelson736@gmail.com
user id = 99995 name = Freya Lewis email = freya.lewis990@gmail.com
user id = 99996 name = Richard Bell email = richard.bell857@gmail.com
user id = 99997 name = Leo Davis email = leo.davis767@gmail.com
user id = 99998 name = Laura Smith email = laura.smith953@gmail.com
user id = 99999 name = Leo Williams email = leo.williams514@gmail.com
user id = 100000 name = Bob Adams email = bob.adams805@gmail.com
user id = 100001 name = Ann Roberts email = ann.roberts285@gmail.com
user id = 100002 name = John Williams email = john.williams399@gmail.com
user id = 100003 name = Stephen Stone email = stephen.stone477@gmail.com
user id = 100004 name = Leo Roberts email = leo.roberts412@gmail.com
user id = 100005 name = Florence Roberts email = florence.roberts835@gmail.com
user id = 100006 name = Amelia Stone email = amelia.stone710@gmail.com
user id = 100007 name = Tomas Jones email = tomas.jones937@gmail.com
user id = 100008 name = Florence Adams email = florence.adams733@gmail.com
user id = 100009 name = John Taylor email = john.taylor202@gmail.com
user id = 100010 name = Harry Lebron email = harry.lebron427@gmail.com
```

Enter the user id of the user you want to view:

-> 9

```
user id = 9      name = Alexandr Lopatyn      email = alxlopatyn@test.com
```

Оновлюємо користувача:

```
USER MENU
Choose what you wish to do:
1. View user.
2. Add user.
3. Delete user.
4. Update user.
9. Back to start menu.
0. Exit
-> 4
Enter the user id of the user you want to change:
-> 9
Old first name: Alexandr.
Enter new first name:
-> Martin
Old last name: Lopatyn.
Enter new last name:
-> Luther
Old email email: alxlopatyn@test.com.
Enter new email:
-> mlking@gmail.com
```

```
Enter the user id of the user you want to view:
-> 9
user id = 9      name = Martin Luther      email = mlking@gmail.com
```

Видаляємо користувача:

```
0. Exit
-> 3
Enter the id of the user you want to delete:
-> 9
Enter the user id of the user you want to view:
-> 9
The user with this id does not exist.
```

Меню каталогу:

```
CATALOGUE MENU
Choose what you wish to do:
1. View catalogue.
2. Add catalogue.
3. Delete catalogue.
4. Update catalogue.
9. Back to start menu.
0. Exit
```

Меню перегляду:

```
Select one of the methods for viewing the table:
1. View the whole table.
2. View one catalogue.
3. View the largest catalogue.
9. Back to start menu.
0. Exit
```

Перегляд всієї таблиці:

```
-> 1
catalogue id = 1      catalogue name = Main      user id = 1
catalogue id = 2      catalogue name = Phonk      user id = 2
catalogue id = 3      catalogue name = test      user id = 1
MUSIC CATALOGUE
```

Перегляд конкретного каталогу:

```
Enter the catalogue id of the catalogue you want to view:
-> 2
catalogue id = 2      catalogue name = Phonk      user id = 2
MUSIC CATALOGUE
```

Перегляд найбільшого каталогу:

```
Owner: Maksym Lytvyn. Catalogue: Main. Total music: 3
Time spent: 69451000 nanoseconds
```

Додавання каталогу:

```
Enter the user id of the user to which this catalogue will apply:
-> 2
Enter catalogue id:
-> 4
Enter catalogue name:
-> relax
Enter the catalogue id of the catalogue you want to view:
-> 4
catalogue id = 4      catalogue name = relax      user id = 2
```

Оновлення каталогу:

```
-> 4
Enter the catalogue id of the catalogue you want to change:
-> 4
Old catalogue name: relax.
Enter new catalogue name:
rock
```

```
Enter the catalogue id of the catalogue you want to view:
-> 4
catalogue id = 4      catalogue name = rock      user id = 2
```

Видалення каталогу:

```
-> 3
Enter the id of the catalogue you want to delete:
-> 4
```

```
Enter the catalogue id of the catalogue you want to view:
-> 4
This catalogue id is not exist.
```


Меню музики:

```
MUSIC MENU
Choose what you wish to do:
1. View music.
2. Add music.
3. Delete music.
4. Update music.
9. Back to start menu.
0. Exit
->
```

Меню перегляду:

```
Select one of the methods for viewing the table:
1. View the whole table.
2. View one song.
3. View top 3 popular song.
9. Back to start menu.
0. Exit
->
```

Перегляд всієї таблиці:

```
-> 1
music id = 1      music name = Imagination      creator id = 1      catalogue id = 2      views = 1000000
music id = 3      music name = Let It Snow!      creator id = 3      catalogue id = 1      views = 19000000
music id = 4      music name = Fly Me to the Moon      creator id = 3      catalogue id = 1      views = 16000000
music id = 5      music name = Moon River      creator id = 3      catalogue id = 1      views = 6800000
MUSIC CATALOGUE
```

Перегляд конкретної пісні:

```
-> 2
Enter the music id of the music you want to view:
-> 3
music id = 3      music name = Let It Snow!      creator id = 3      catalogue id = 1      views = 19000000
MUSIC CATALOGUE
```

Перегляд топ 3 популярних пісень:

```
-> 3
Owner: Francis Sinatra. Music: Let It Snow!. Views: 19000000
Owner: Francis Sinatra. Music: Fly Me to the Moon. Views: 16000000
Owner: Francis Sinatra. Music: Moon River. Views: 6800000
MUSIC CATALOGUE
```

Додавання пісні:

```
-> 2
Enter the catalogue id of the catalogue to which this music will apply:
-> 1
Enter the creator id of the creator to which this music will apply:
-> 1
Enter music id:
-> test
Your choice is unavailable. Please, try again
Enter music id:
-> Your choice is unavailable. Please, try again
Enter music id:
-> 6
Enter music name:
-> test
Enter number of views:
-> 100
```

Enter the music id of the music you want to view:

-> 6

music id = 6 music name = test creator id = 1 catalogue id = 1 views = 100

Оновлення пісні:

```
-> 4
Enter the music id of the music you want to change:
-> 6
Old music name: test.
Enter new music name:
newTest
Old views: 100.
Enter new views:
52365
```

Enter the music id of the music you want to view:

-> 6

music id = 6 music name = newTest creator id = 1 catalogue id = 1 views = 52365

Видалення пісні:

```
0. Exit  
-> 3  
Enter the id of the music you want to delete:  
-> 6
```

```
-> 2  
Enter the music id of the music you want to view:  
-> 6  
This music id is not exist.
```

Меню творця:

```
CREATOR MENU  
Choose what you wish to do:  
1. View creator.  
2. Add creator.  
3. Delete creator.  
4. Update creator.  
9. Back to start menu.  
0. Exit  
->
```

Меню перегляду:

```
Select one of the methods for viewing the table:  
1. View the whole table.  
2. View one creator.  
3. See the number of views by creator.  
9. Back to start menu.  
0. Exit  
->
```

Перегляд всієї таблиці творців:

```
-> 1  
creator id = 1      name = Ivan Nikolaev  
creator id = 2      name = Taylor Switch  
creator id = 3      name = Francis Sinatra  
creator id = 4      name = Noah Taylor  
MUSIC CATALOGUE
```

Перегляд конкретного творця:

```
Enter the creator id of the creator you want to view:
-> 2
Enter the creator id of the creator you want to view:
-> 3
creator id = 3      name = Francis Sinatra
```

Перегляд сумарної кількості переглядів одного творця:

```
-> 3
Enter the creator id of the creator you want to view:
-> 3
Creator: Francis Sinatra. Total views: 41800000
Time spent: 61841300 nanoseconds
```

Меню додавання користувача:

```
Select one of the methods for adding to the table:
1. Create new creator.
2. Generate new creator.
9. Back to start menu.
0. Exit
-> 1
```

Додавання вручну:

```
Enter creator id:
-> 5
Enter creator first name:
-> Miles
Enter creator last name:
-> Davis
```

```
Enter the creator id of the creator you want to view:
-> 5
creator id = 5      name = Miles Davis
```

Згенеровані творці:

```
-> 2
Enter how many creators you want to generate:
-> 10
10 records inserted successfully.
```

```
creator id = 1      name = Ivan Nikolaev
creator id = 2      name = Taylor Switch
creator id = 3      name = Francis Sinatra
creator id = 4      name = Noah Taylor
creator id = 5      name = Miles Davis
creator id = 6      name = Leo Stone
creator id = 7      name = Oliver Shelby
creator id = 8      name = Lily Taylor
creator id = 9      name = Ann Lebron
creator id = 10     name = Arthur Davies
creator id = 11     name = Max Adams
creator id = 12     name = Mike Smith
creator id = 13     name = Oliver Smith
creator id = 14     name = Stephen Williams
creator id = 15     name = Tomas Nelson
```

Оновлення творця:

```
Enter the creator id of the creator you want to change:
-> 8
Old first name: Lily.
Enter new first name:
Marta
Old last name: Taylor.
Enter new last name:
Swift
```

```
Enter the creator id of the creator you want to view:
-> 8
creator id = 8      name = Marta Swift
```

Видалення творця:

```
-> 3
Enter the id of the creator you want to delete:
-> 15
```

```
Enter the creator id of the creator you want to view:
-> 15
The creator with this id does not exist.
```

Етап №2:

Індекси в PostgreSQL - спеціальні об'єкти бази даних, призначені в основному для прискорення доступу до даних. Це допоміжні структури: будь-який індекс можна видалити і відновити заново за інформацією в таблиці.

B-Tree – це індекс, яке використовується для організації індексів у базах даних. Воно дозволяє швидко знаходити, вставляти або видаляти дані. Ключі в B-Tree розташовані у впорядкованому вигляді, що забезпечує ефективний доступ до даних.

Hash – індекс базується на використанні хеш-функцій. Він обчислює хеш-значення для кожного ключа та використовує це значення для швидкого доступу до рядків.

B-Tree.

Плюси:

- Має швидкий пошук, та складність $O(\log_2 n)$ для пошуку, вставки та видалення
- Підтримує діапазони з операціями, як: $<$, $>$, BETWEEN, LIKE..
- Підтримує впорядкованість навіть при частих змінах даних.

Мінуси:

- При правильному використанні, програє в швидкості Hash індексам, коли йде мова про пошук точних збігів
- Використовує багато пам'яті, особливо, коли йде мова про великі таблиці

Hash:

Плюси:

- Один з найшвидкіших індексів, коли йде мова про точний пошук
- Займає менше пам'яті, бо не зберігає порядок ключів

Мінуси:

- Не підтримує діапазони, як це робить умовний B-Tree
- Не рекомендується використовувати при великій кількості оновлень, оскільки хеш-функція може створювати більше колізій при великих обсягах даних
- Не всі системи баз даних повністю можуть підтримати Hash індекси.

Як висновок, B-Tree можна використовувати як універсальний індекс, оскільки всюди він показує непогані результати, при великих затратах пам'яті. Коли як Hash потрібен лише тоді, коли ми маємо знайти якісь конкретні рядки, не використовуючи діапазони.

Запит №1 (без використання індексів)

The screenshot shows a database query execution interface. At the top, a SQL query is entered: `1 EXPLAIN ANALYZE SELECT * FROM users WHERE email = 'freya.campbell875@gmail.com'`. Below the query, the 'Data Output' tab is active, displaying a 'QUERY PLAN' table. The table has 8 rows of execution details. At the bottom, a status bar indicates 'Total rows: 8 of 8', 'Query complete: 00:00:00.533', and '1 n 1. Col 80'. A green notification box at the bottom right states: '✓ Запрос выполнен успешно. Общее время выполнения: 533 msec. обработано строк: 8. ✕'.

	QUERY PLAN
1	Gather (cost=1000.00..158163.29 rows=26 width=40) (actual time=5.532..509.239 rows=23 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	-> Parallel Seq Scan on users (cost=0.00..157160.69 rows=11 width=40) (actual time=80.042..478.957 rows=8 loop...
5	Filter: ((email)::text = 'freya.campbell875@gmail.com'::text)
6	Rows Removed by Filter: 3666661
7	Planning Time: 0.082 ms
8	Execution Time: 509.263 ms

Total rows: 8 of 8 Query complete: 00:00:00.533 1 n 1. Col 80

✓ Запрос выполнен успешно. Общее время выполнения: 533 msec. обработано строк: 8. ✕

Тут можемо помітити, що при звичайному пошуку за email-ом час запиту займає пів секунди, що досить довго. Тепер, використаємо індекси.

Запит №2 (з використанням індексу B-Tree)

Індекс створюємо завдяки спеціальній команді:

```
CREATE INDEX idx_users_email_btree ON users(email)
```

Тепер, виконаємо той самий запит, але вже з використанням індексу:

```
1 EXPLAIN ANALYZE SELECT * FROM users WHERE email = 'freya.campbell875@gmail.com'
```

Data Output Сообщения Notifications

QUERY PLAN

1	Bitmap Heap Scan on users (cost=4.64..107.70 rows=26 width=40) (actual time=0.032..0.048 rows=23 loops=1)
2	Recheck Cond: ((email)::text = 'freya.campbell875@gmail.com':text)
3	Heap Blocks: exact=23
4	-> Bitmap Index Scan on idx_users_email_btree (cost=0.00..4.63 rows=26 width=0) (actual time=0.023..0.023 rows=23 loop=1)
5	Index Cond: ((email)::text = 'freya.campbell875@gmail.com':text)
6	Planning Time: 0.057 ms
7	Execution Time: 0.058 ms

✓ Запрос выполнен успешно. Общее время выполнения: 54 msec. обработано строк: 7. ✕

Total rows: 7 of 7 Query complete 00:00:00.054 In 1 Col 80

Як бачимо, прискорення відбулося майже в цілих 10 разів, що демонструє високу ефективність індексу B-Tree.

Запит №3 (з використанням Hash індексу)

Створюємо індекс:

```
CREATE INDEX idx_users_email_hash ON users USING HASH(email)
```

Повторюємо наш запит:

```
1 EXPLAIN ANALYZE
2 SELECT *
3 FROM users
4 WHERE email = 'reya.campbell875@gmail.com';
```

	QUERY PLAN
1	Bitmap Heap Scan on users (cost=4.20..107.27 rows=26 width=40) (actual time=0.020..0.020 rows=0 loops=1)
2	Recheck Cond: ((email)::text = 'reya.campbell875@gmail.com'::text)
3	-> Bitmap Index Scan on idx_users_email_hash (cost=0.00..4.20 rows=26 width=0) (actual time=0.011..0.011 rows=0 loop...
4	Index Cond: ((email)::text = 'reya.campbell875@gmail.com'::text)
5	Planning Time: 0.098 ms
6	Execution Time: 0.037 ms

Total rows: 6 of 6 Query complete 00:00:00.045 Ln 4, Col 42

✓ Запрос выполнен успешно. Общее время выполнения: 45 мсек. обработано строк: 6. ✕

І, як можемо помітити, Hash індекс в цьому випадку спрацював навіть швидше, ніж В-Tree індекс (як це ми обговорювали раніше). А відносно звичайного виклику, то він швидше майже в 15 разів, що є неймовірно швидко!

Запит №4 (без використання індексів)

```
1  EXPLAIN ANALYZE
2  SELECT user_id, COUNT(*)
3  FROM catalogues
4  GROUP BY user_id
5  ORDER BY COUNT(*) DESC;
```

Data Output Сообщения Notifications

	QUERY PLAN text	
1	Sort (cost=10000019915.13..10000020163.04 rows=99166 width=12) (actual time=51.227..58.066 rows=99501 loops=1)	
2	Sort Key: (count(*)) DESC	
3	Sort Method: external merge Disk: 2544kB	
4	-> GroupAggregate (cost=10000009943.82..10000011685.56 rows=99166 width=12) (actual time=16.170..36.305 rows=99501 loops=1)	
5	Group Key: user_id	
6	-> Sort (cost=10000009943.82..10000010193.85 rows=100010 width=4) (actual time=16.158..19.087 rows=100010 loops=1)	
7	Sort Key: user_id	
8	Sort Method: quicksort Memory: 3073kB	
9	-> Seq Scan on catalogues (cost=10000000000.00..10000001638.10 rows=100010 width=4) (actual time=0.018..5.185 rows=100010 loop...)	
10	Planning Time: 0.079 ms	
11	Execution Time: 62.426 ms	

Тут ми використали суміш агрегатної та функції сортування. І як бачимо, звичайний пошук і так займає не так багато часу, всього лише 62 мс. Давайте аподивимось, як з цим впорається В-Tree індекс.

Запит №5 (з використанням індекса B-Tree)

```
1  EXPLAIN ANALYZE
2  SELECT user_id, COUNT(*)
3  FROM catalogues
4  GROUP BY user_id
5  ORDER BY COUNT(*) DESC;
```

Data Output Сообщения Notifications

QUERY PLAN

1	Sort (cost=12329.72..12577.63 rows=99166 width=12) (actual time=30.838..35.392 rows=99501 loops=1)
2	Sort Key: (count(*)) DESC
3	Sort Method: external merge Disk: 2544kB
4	-> GroupAggregate (cost=0.29..4100.15 rows=99166 width=12) (actual time=0.022..19.429 rows=99501 loops=1)
5	Group Key: user_id
6	-> Index Only Scan using idx_catalogues_user_id_btree on catalogues (cost=0.29..2608.44 rows=100010 width=4) (actual time=0.017..5.579 rows=100010 loops=1)
7	Heap Fetches: 126
8	Planning Time: 0.075 ms
9	Execution Time: 38.458 ms

І бачимо, що результат з B-Tree індексом навіть за цих умов видає результат майже вдвічі швидше, що демонструє нам високу ефективність використання індексів.

Запит №6 (без використання індекса)

```
1  EXPLAIN ANALYZE
2  SELECT *
3  FROM musics
4  WHERE views BETWEEN 10000 AND 155000
5  ORDER BY views DESC;
```

Data Output Сообщения Notifications

QUERY PLAN
text

1	Sort (cost=10000011671.77..10000011894.03 rows=88902 width=27) (actual time=36.415..41.860 rows=89948 loops=1)
2	Sort Key: views DESC
3	Sort Method: external merge Disk: 3352kB
4	-> Seq Scan on musics (cost=10000000000.00..10000002236.06 rows=88902 width=27) (actual time=0.012..9.314 rows=89948 loop...)
5	Filter: ((views >= 10000) AND (views <= 155000))
6	Rows Removed by Filter: 10056
7	Planning Time: 0.056 ms
8	Execution Time: 44.635 ms

Тут ми використовуємо пошук за діапазоном та сортуванням, а саме шукаємо поміж всіх пісень пісні, в яких переглядів від 10000 до 155000. Звичайним запитом без індексів вдалося знайти всі пісні всього лише за 45 мс. Подивимось, який результат буде з B-Tree індексом.

Запит №7(з B-Tree індексом)

```
1 EXPLAIN ANALYZE
2 SELECT *
3 FROM musics
4 WHERE views BETWEEN 10000 AND 155000
5 ORDER BY views DESC;
```

Data Output Сообщения Notifications

QUERY PLAN

1	Index Scan Backward using idx_musics_views_btree on musics (cost=0.29..5586.32 rows=88902 width=27) (actual time=0.009..25.200 rows=89948 loop...
2	Index Cond: ((views >= 10000) AND (views <= 155000))
3	Planning Time: 0.094 ms
4	Execution Time: 27.188 ms

І, як можемо помітити, з B-Tree індексом наш запит був у 1.5 рази швидше за звичайний. І хочу відмітити, що в цій таблиці було лише 100000 пісень. При більш високому обсязі таблиці (хоча б, як в users та creators, де були під 10 мільйонів даних), результати були б більш приємні!

Запит №8 (без індексування)

```
1  EXPLAIN ANALYZE SELECT last_name, COUNT(*) FROM creators GROUP BY last_name;
```

Data Output Сообщения Notifications

QUERY PLAN

1	GroupAggregate (cost=10001595034.70..10001670035.01 rows=20 width=14) (actual time=5653.027..6960.195 rows=24 loops=1)
2	Group Key: last_name
3	-> Sort (cost=10001595034.70..10001620034.74 rows=10000014 width=6) (actual time=5577.767..6354.027 rows=10000014 loops=1)
4	Sort Key: last_name
5	Sort Method: external merge Disk: 106320kB
6	-> Seq Scan on creators (cost=10000000000.00..10000158917.14 rows=10000014 width=6) (actual time=0.033..618.086 rows=10000014 loop...
7	Planning Time: 0.440 ms
8	Execution Time: 6968.055 ms

✓ Запрос выполнен успешно. Общее время выполнения: 6 secs 985 msec. обработано строк: 8. ✕

Цей запит рахує кількість прізвищ, і виводить самі прізвища, та їх кількість у таблиці з групуванням. В-Tree індекс не дуже любить працювати з купою однакових клітинок. Але, подивимось на результат.

Запит №9 (з B-Tree індексом)

```
CREATE INDEX idx_creators_last_name_btree ON creators(last_name)
```

```
1 EXPLAIN ANALYZE SELECT last_name, COUNT(*) FROM creators
2 GROUP BY last_name
```

Data Output Сообщения Notifications

QUERY PLAN

1	Finalize GroupAggregate (cost=1000.46..382700.36 rows=20 width=14) (actual time=3681.580..3689.128 rows=24 loops=1)
2	Group Key: last_name
3	-> Gather Merge (cost=1000.46..382699.96 rows=40 width=14) (actual time=3680.465..3689.111 rows=46 loops=1)
4	Workers Planned: 2
5	Workers Launched: 2
6	-> Partial GroupAggregate (cost=0.43..381695.32 rows=20 width=14) (actual time=166.330..2492.686 rows=15 loops=3)
7	Group Key: last_name
8	-> Parallel Index Only Scan using idx_creators_last_name_btree on creators (cost=0.43..360861.76 rows=4166672 width=6) (actual time=0.425..2252.610 rows=3333338 loops=1)
9	Heap Fetches: 10000014
10	Planning Time: 0.683 ms
11	Execution Time: 3689.184 ms

✓ Запрос выполнен успешно. Общее время выполнения: 3 secs 714 msec. обработано строк: 11. ✕

І як бачимо, B-Tree спрацював у двічі швидше, навіть не з його «ідеальним» варіантом таблиці. Тим самим, можемо побачити, наскільки ефективно можна використовувати індекси.

Етап №3:

Тригери в PostgreSQL — це спеціальні механізми, які дозволяють автоматично виконувати деякі функції у відповідь на зміни, що відбулися в таблицях бази даних. Тобто, це свого роду підключення до певних подій (вставлення, оновлення, видалення даних), які запускають наш код.

Функція тригера:

```
CREATE OR REPLACE FUNCTION process_music_changes()
RETURNS TRIGGER AS $$
DECLARE
    catalogue_exists BOOLEAN;
    creator_exists BOOLEAN;
    music_cursor REFCURSOR;
    music_record RECORD;
BEGIN
    SELECT EXISTS (
        SELECT 1 FROM catalogues WHERE catalogue_id = NEW.catalogue_id
    ) INTO catalogue_exists;

    IF NOT catalogue_exists THEN
        RAISE EXCEPTION 'Catalogue with ID % does not exist', NEW.catalogue_id;
    END IF;

    SELECT EXISTS (
        SELECT 1 FROM creators WHERE creator_id = NEW.creator_id
    ) INTO creator_exists;

    IF NOT creator_exists THEN
        RAISE EXCEPTION 'Creator with ID % does not exist', NEW.creator_id;
    END IF;

    OPEN music_cursor FOR
        SELECT * FROM musics WHERE catalogue_id = NEW.catalogue_id;

    LOOP
        FETCH music_cursor INTO music_record;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'Music ID: %, Name: %', music_record.music_id, music_record.music_name;
    END LOOP;

    CLOSE music_cursor;

    RETURN NEW;
EXCEPTION
    WHEN OTHERS THEN
        RAISE WARNING 'Error related to: %', SQLERRM;
        RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```


Створення тригера:

```
CREATE TRIGGER music_changes_trigger
AFTER INSERT OR UPDATE ON musics
FOR EACH ROW
EXECUTE FUNCTION process_music_changes();
```

Використання тригера:

Вставка з вже існуючим каталогом та виконавцем:

```
1 INSERT INTO musics (music_id, music_name, views, creator_id, catalogue_id)
2 VALUES (2, 'Unforgettable', 100000, 1, 1);
```

Data Output	Сообщения	Notifications
-------------	-----------	---------------

ПОВІДОМЛЕННЯ:	Music ID: 99887, Name: Music 99887	
ПОВІДОМЛЕННЯ:	Music ID: 99912, Name: Music 99912	
ПОВІДОМЛЕННЯ:	Music ID: 99913, Name: Music 99913	
ПОВІДОМЛЕННЯ:	Music ID: 99925, Name: Music 99925	
ПОВІДОМЛЕННЯ:	Music ID: 99932, Name: Music 99932	
ПОВІДОМЛЕННЯ:	Music ID: 99934, Name: Music 99934	
ПОВІДОМЛЕННЯ:	Music ID: 99961, Name: Music 99961	
ПОВІДОМЛЕННЯ:	Music ID: 99962, Name: Music 99962	
ПОВІДОМЛЕННЯ:	Music ID: 99975, Name: Music 99975	
ПОВІДОМЛЕННЯ:	Music ID: 99978, Name: Music 99978	
ПОВІДОМЛЕННЯ:	Music ID: 99990, Name: Music 99990	
ПОВІДОМЛЕННЯ:	Music ID: 100004, Name: Music 100004	
ПОВІДОМЛЕННЯ:	Music ID: 100005, Name: Music 100005	
ПОВІДОМЛЕННЯ:	Music ID: 2, Name: Unforgettable	
INSERT	0 1	

Запрос завершён успешно, время выполнения: 291 msec.

Вставка з неіснуючим каталогом:

```
1 ▾ INSERT INTO musics (music_id, music_name, views, creator_id, catalogue_id)
2   VALUES (100007, 'Unknown', 50, 1, 1000000);
```

Data Output [Сообщения](#) Notifications

ERROR: Ключ (catalogue_id)=(1000000) не присутній в таблиці "catalogues".insert або update в таблиці "musics" порушує обмеження зовнішнього ключа "catalogue_fk"

Оновлення пісні:

```
1 UPDATE musics SET music_name = 'Changed' WHERE music_id = 2;
```

Data Output	Сообщения	Notifications
-------------	-----------	---------------

ПОВІДОМЛЕННЯ:	Music ID: 99887, Name: Music 99887	
ПОВІДОМЛЕННЯ:	Music ID: 99912, Name: Music 99912	
ПОВІДОМЛЕННЯ:	Music ID: 99913, Name: Music 99913	
ПОВІДОМЛЕННЯ:	Music ID: 99925, Name: Music 99925	
ПОВІДОМЛЕННЯ:	Music ID: 99932, Name: Music 99932	
ПОВІДОМЛЕННЯ:	Music ID: 99934, Name: Music 99934	
ПОВІДОМЛЕННЯ:	Music ID: 99961, Name: Music 99961	
ПОВІДОМЛЕННЯ:	Music ID: 99962, Name: Music 99962	
ПОВІДОМЛЕННЯ:	Music ID: 99975, Name: Music 99975	
ПОВІДОМЛЕННЯ:	Music ID: 99978, Name: Music 99978	
ПОВІДОМЛЕННЯ:	Music ID: 99990, Name: Music 99990	
ПОВІДОМЛЕННЯ:	Music ID: 100004, Name: Music 100004	
ПОВІДОМЛЕННЯ:	Music ID: 100005, Name: Music 100005	
ПОВІДОМЛЕННЯ:	Music ID: 2, Name: Changed	
UPDATE	1	

І дійсно, можемо побачити, що зміни спрацювали коректно:

```
1 SELECT * FROM musics WHERE music_id < 5
```

Data Output Сообщения Notifications

	music_id [PK] integer	music_name character varying (20)	views integer	creator_id integer	catalogue_id integer
1	1	Imagination	1000000	1	2
2	2	Changed	50	1	1
3	3	Let It Snow!	19000000	3	1
4	4	Fly Me to the Moon	16000000	3	1

Етап №4:

READ COMMITED – це найслабший рівень ізоляції, коли транзакція може бачити результати інших транзакцій, навіть, якщо вони ще не закомічені (не збережені)

Транзакція 1

```
1 BEGIN;  
2 SELECT * FROM musics WHERE music_id = 2  
3
```

Data Output Сообщения Notifications

	music_id [PK] integer	music_name character varying (20)	views integer	creator_id integer	catalogue_id integer
1	2	Changed	700	1	1

Транзакція 2

```
1 BEGIN;  
2  
3 UPDATE musics SET views = 350 WHERE music_id = 2;  
4
```

Data Output Сообщения Notifications

UPDATE 1

Запрос завершён успешно, время выполнения: 29 msec.

Транзакція 1

```
1 BEGIN;  
2 SELECT * FROM musics WHERE music_id = 2  
3
```

Data Output Сообщения Notifications

	music_id [PK] integer	music_name character varying (20)	views integer	creator_id integer	catalogue_id integer
1	2	Changed	700	1	1

Брудне читання (dirty read) – явище, коли дані, які вже були прочитані, хтось може відкотити ще до того, як ми завершимо нашу транзакцію.

Брудне читання погане тим, що в моменті, коли ми хочемо з таблиці взяти потрібні нам дані, та вже потім з ними працювати, через цей феномен ми можемо неправильно обробити дані, та видати не вірний результат, оскільки хтось іншим відкотив зміни в своїй транзакції, та ті дані, які ми «брудно» прочитали, вже не є актуальними.

В нашому випадку «dirty read» не відбувся. Але, подібне цілком можливо.

Транзакція №1

```
1 BEGIN;  
2 SELECT * FROM musics WHERE music_id = 2
```

Data Output Сообщения Notifications

	music_id [PK] integer	music_name character varying (20)	views integer	creator_id integer	catalogue_id integer
1	2	Changed	700	1	1

Ми викликали транзакцію, читаємо інформація з таблички, але не комітемо її.

Транзакція №2

```
1 BEGIN;  
2  
3 UPDATE musics SET views = 350 WHERE music_id = 2;  
4  
5  
6 COMMIT;
```

Data Output Сообщения Notifications

ПОПЕРЕДЖЕННЯ: транзакція вже виконується
COMMIT

Далі, змінюємо в другій транзакції views на 700, та комітемо. І зараз, в нас висить **транзакція 1**, яка ще не закомічена, але вже читає застарілі дані. І в разі коміту, вона може повторно прочитати таблицю, і ми матимемо в результаті два результати, які між собою є різними.

Транзакція 1

```
1 BEGIN;  
2 SELECT * FROM musics WHERE music_id = 2
```

Data Output Сообщения Notifications



	music_id [PK] integer	music_name character varying (20)	views integer	creator_id integer	catalogue_id integer
1	2	Changed	350	1	1

Ця проблема називається «Non-repeatable Read».

Неповторюване читання (Non-repeatable Read) – це коли дані, котрі ми прочитали, хтось може змінити ще до того, як ми завершимо нашу транзакцію. Тобто, значення одного й того ж рядка може змінюється не в межах транзакції між двома читаннями.

Транзакція 1

Ми читасмо дані з таблиці:

```
1 BEGIN;
2 SELECT * FROM musics
3 WHERE views = 600
4
```

Data Output Сообщения Notifications

	music_id [PK] integer	music_name character varying (20)	views integer	creator_id integer	catalogue_id integer
1	2	Changed	600	1	1

І наприклад, під час нашої транзакції, хтось у своїй (транзакція 2) транзакції поновлює дані таблиці:

Транзакція 2

```
1 BEGIN;  
2  
3 UPDATE musics SET views = 700 WHERE music_id = 2;  
4  
5 COMMIT;  
6
```

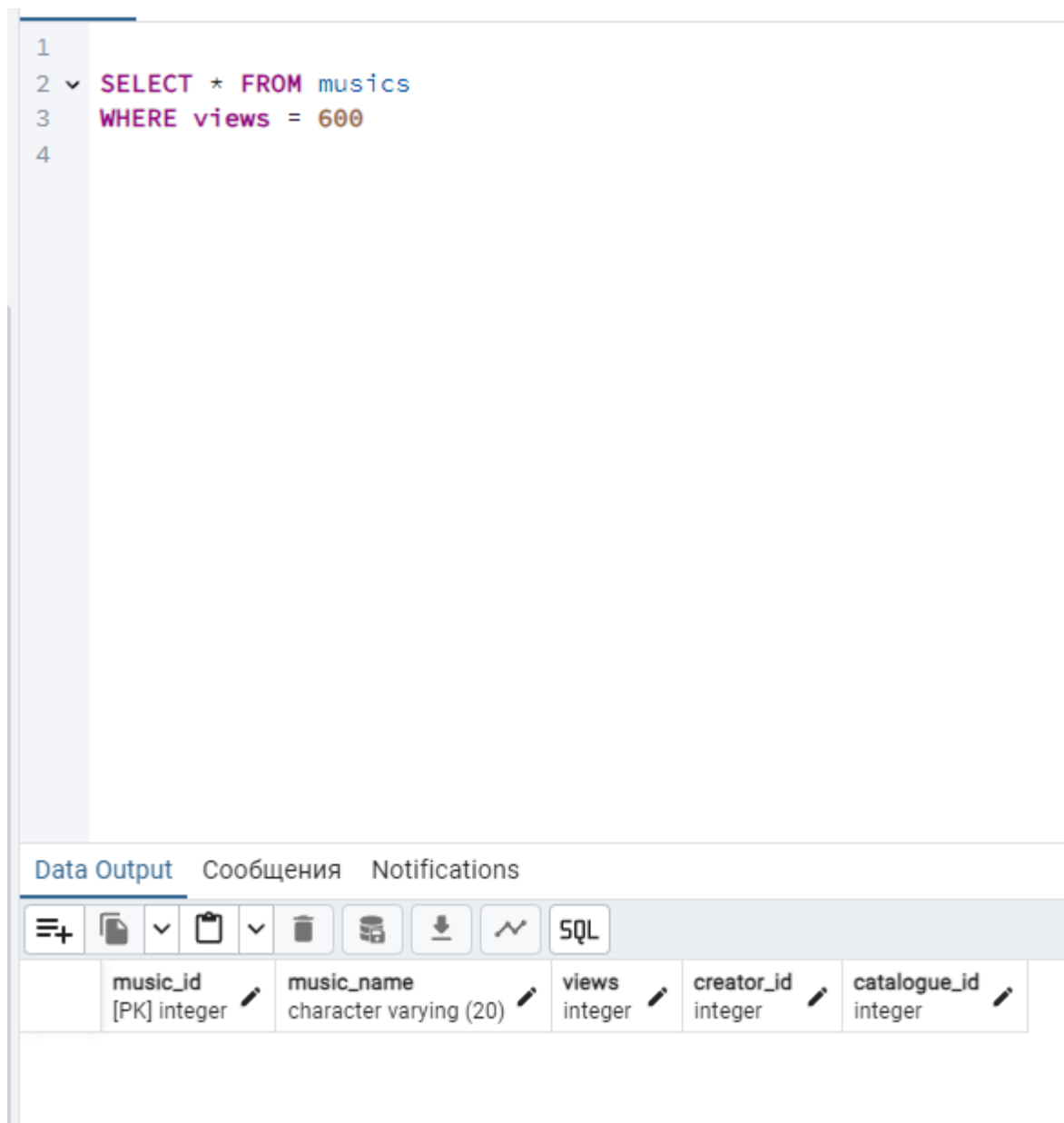
Data Output Сообщения Notifications

COMMIT

Запрос завершён успешно, время выполнения: 31 мсес.

І вже в транзакції один, коли ми ще не завершили нашу транзакцію виникає проблема, і помічаємо що на час коміту, таких пісень, де views = 600 вже нема.

Транзакція 1



Цей феномен називається «Phantom Read»

Фантомне читання (Phantom Read) – це коли певні дані, які ми прочитали, хтось може видалити або додати ще до того, як ми завершимо власну транзакцію. Тобто, під час повторного виконання запиту з’являються або зникають нові рядки, яких раніше не було

REPEATABLE READ – цей рівень означає, що поки транзакція не буде завершена, ніхто не зможе змінювати або видаляти рядки, які транзакція вже прочитала.

Цей рівень ізоляції рятує нас від таких проблем, як «dirty read» та «non-repeatable read». Але, все ще залишається проблема «Фантомного читання».

Причина? Тому що при даній ізоляції хоч нам і не дають змогу видаляти, або змінювати строки, але ми все ще можемо їх додавати паралельно.

І давайте перевіримо, чи це так:

Транзакція 1

```
1 BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
2
3 SELECT *
4 FROM musics
5 WHERE views = 600
6
```

Data Output Сообщения Notifications

	music_id [PK] integer	music_name character varying (20)	views integer	creator_id integer	catalogue_id integer
1	2	Changed	600	1	1

Транзакція 2

```
1 BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;  
2  
3 UPDATE musics SET views = 1200 WHERE music_id = 2;  
4  
5 COMMIT;  
6
```

Data Output Сообщения Notifications

COMMIT

Запрос завершён успешно, время выполнения: 28 мсек.

Оновлюємо дані. Тепер, якщо «брудне читання» працює, ми маємо побачити результат 0:

Транзакція 1

```
1 SELECT *
2 FROM musics
3 WHERE views = 600
4
```

Data Output Сообщения Notifications

	music_id [PK] integer	music_name character varying (20)	views integer	creator_id integer	catalogue_id integer
1	2	Changed	600	1	1

✓ Запрос выполнен успешно. Общее время выполнения: 52 мсек. обработано строк

Але, як ми бачимо, все працює коректно.

Тепер, подивимось на феномен фантомного читання:

Транзакція 1

Запрос

История запросов

1

BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;

2

3

SELECT * FROM musics WHERE views = 600

Data Output

Сообщения

Notifications

≡+

▼

▼

SQL

	music_id [PK] integer	music_name character varying (20)	views integer	creator_id integer	catalogue_id integer
1	2	Changed	600	1	1

Як бачимо, лише одна пісня має 600 переглядів.

Тепер, додамо в іншій транзакції новий рядок (пісню), в якій буде 600 переглядів:

Транзакція 2

```
1 BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
2
3 ▾ INSERT INTO musics (music_id, music_name, views, catalogue_id, creator_id)
4   VALUES(1000006, 'SomeSong', 600, 1, 1);
5
6 COMMIT;
```

Data Output Сообщения Notifications

COMMIT

Запрос завершён успешно, время выполнения: 40 msec.

Транзакція 1

```
1 SELECT * FROM musics WHERE views = 600
```

Data Output Сообщения Notifications

	music_id [PK] integer	music_name character varying (20)	views integer	creator_id integer	catalogue_id integer
1	2	Changed	600	1	1
2	100006	SomeSong	600	1	1

Ось таким чином мало б спрацювати фантомне читання.

SERIALIZABLE – найбезпечніший, але найтяжчий для баз даних та повільний для обробки запитів. Він блокує будь-які дії, доки працює транзакція – транзакції йдуть одна за одною і максимально ізолюються одне від одного. Це трапляється завдяки блокуванню всієї таблиці від будь-яких взаємодій з нею. Але, деякі СУБД роблять її менш радикальною, а саме – блокують тільки ті строки, котрі задіюють нинішню транзакцію або діапазон рядків.

Перевіримо всі три випадки (dirty read, non-repeatable read, phantom read):

dirty read:

Транзакція 1

```
1 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
2
3 SELECT COUNT(*) FROM musics WHERE views < 1000
4
5
```

Data Output Сообщения Notifications

	count bigint
1	1031

Транзакція 2

```
1 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
2  
3 ✓ INSERT INTO musics (music_id, music_name, views, catalogue_id, creator_id)  
4 VALUES(100007, 'SomeSong', 600, 1, 1);  
5  
6 ROLLBACK;
```

Data Output Сообщения Notifications

ROLLBACK

Запрос завершён успешно, время выполнения: 43 msec.

Транзакція 1

```
1 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
2
3 SELECT COUNT(*) FROM musics WHERE views < 1000
4
5
```

	count	bigint
1	1031	






Як бачимо, ми не перетинались з транзакцією 2, і наші дані залишились не змінними.

non-repeatable read:

Транзакція 1

```
1 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
2  
3 SELECT * FROM music WHERE views = 600  
4  
5
```

Data Output Сообщения Notifications

	 music_id [PK] integer	 music_name character varying (20)	 views integer	 creator_id integer	 catalogue_id integer
1	2	Changed	600	1	1
2	100006	SomeSong	600	1	1

Транзакція 2

```
1 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
2  
3 UPDATE musics SET views = 700 WHERE music_id = 2;  
4  
5 COMMIT;
```

Data Output Сообщения Notifications

COMMIT

Запрос завершён успешно, время выполнения: 42 msec.

Транзакція 1

```
1 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
2  
3 SELECT * FROM music WHERE views = 600  
4  
5
```

Data Output Сообщения Notifications

	music_id [PK] integer	music_name character varying (20)	views integer	creator_id integer	catalogue_id integer
1	2	Changed	600	1	1
2	100006	SomeSong	600	1	1






Як бачимо, змін не відбулось, а отже в нашій транзакції під час роботи нічого не змінилось. А отже, все працює вірно.

Phantom read:

Транзакція 1

```
1 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
2  
3 SELECT * FROM musics WHERE views = 600  
4  
5
```

Data Output Сообщения Notifications

	 music_id [PK] integer	 music_name character varying (20)	 views integer	 creator_id integer	 catalogue_id integer
1	2	Changed	600	1	1
2	100006	SomeSong	600	1	1

Транзакція 2

```
1 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
2  
3 DELETE FROM musics WHERE music_id = 2;  
4  
5 COMMIT;
```

Data Output Сообщения Notifications






COMMIT

Запрос завершён успешно, время выполнения: 40 msec.

Транзакція 1

```
1 BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
2  
3 SELECT * FROM musics WHERE views = 600  
4  
5
```

Data Output Сообщения Notifications

	 music_id [PK] integer	 music_name character varying (20)	 views integer	 creator_id integer	 catalogue_id integer
1	2	Changed	600	1	1
2	100006	SomeSong	600	1	1

Як бачимо, видалення сюди не дійшло, і це єдиний рівень ізоляції, в якому уникається phantom read. А отже, все вийшло правильно!

ПРИМІТКА: Транзакції відбуваються послідовно, а саме так, як завантажені