# **Лабораторна робота 2. Ознайомлення з базовими операціями СУБД PostgreSQL**

## **Виконав**

## **студент групи КП-73**

## **Гулько Данило Тарасович**

## **Постановка Завдання**

[Посилання](http://scs.kpi.ua/sites/default/files/lab2_bd_part1_2019_2020.pdf)

## **Предметна галузь**

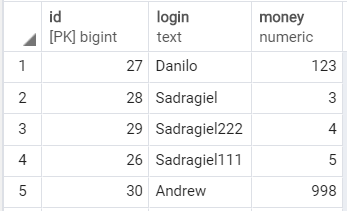
Карткова колекційна гра

## **Модель Бази Даних**

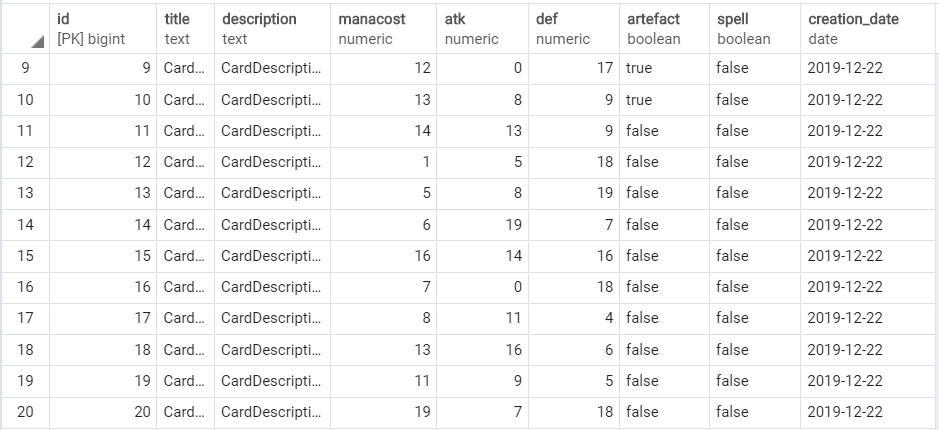
|  |  |  |
| --- | --- | --- |
| **Відношення** | **Атрибут** | **Тип Даних** |
| **Player**  Інформація про зареєстрованих гравців | **id** | Числовий  (первинний ключ) |
| login | Текстовий |
| money | Числовий |
| **Card**  Інформація про існуючі у грі карти | **id** | Числовий  (первинний ключ) |
| title | Текстовий |
| description | Текстовий |
| manacost | Числовий |
| atk | Числовий |
| def | Числовий |
| artefact | Булевий |
| spell | Булевий |
| creation\_date | Дата |
| **Deck**  Інформація про колоди карт гравців | **id** | Числовий  (первинний ключ) |
| title | Текстовий |
| owner | Числовий  (Зовнішній ключ - **Player.id**) |
| rotate | Булевий |
| **Effect**  Інформація про усі наявні у грі ефекти карт | **id** | Числовий  (первинний ключ) |
| aim | Текстовий |
| type | Текстовий |
| bonus | Числовий |
| activatable | Булевий |

## 

## **Структура БД**

**

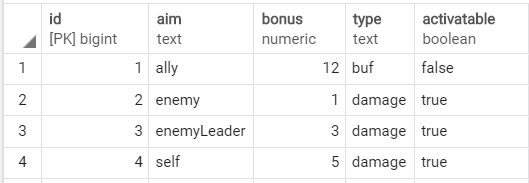
*Таблиця Player*

**

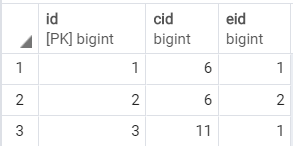
*Таблиця Card*

**

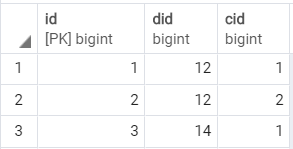
*Таблиця Deck*

**

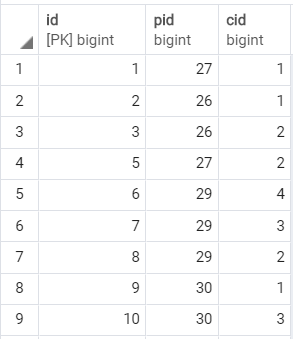
*Таблиця Effect*

**

*Таблиця card\_effect*

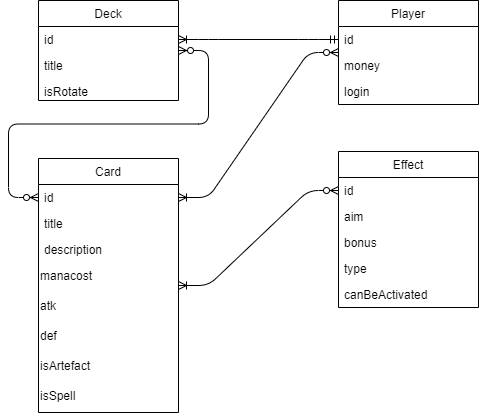
**

*Таблиця deck\_card*

**

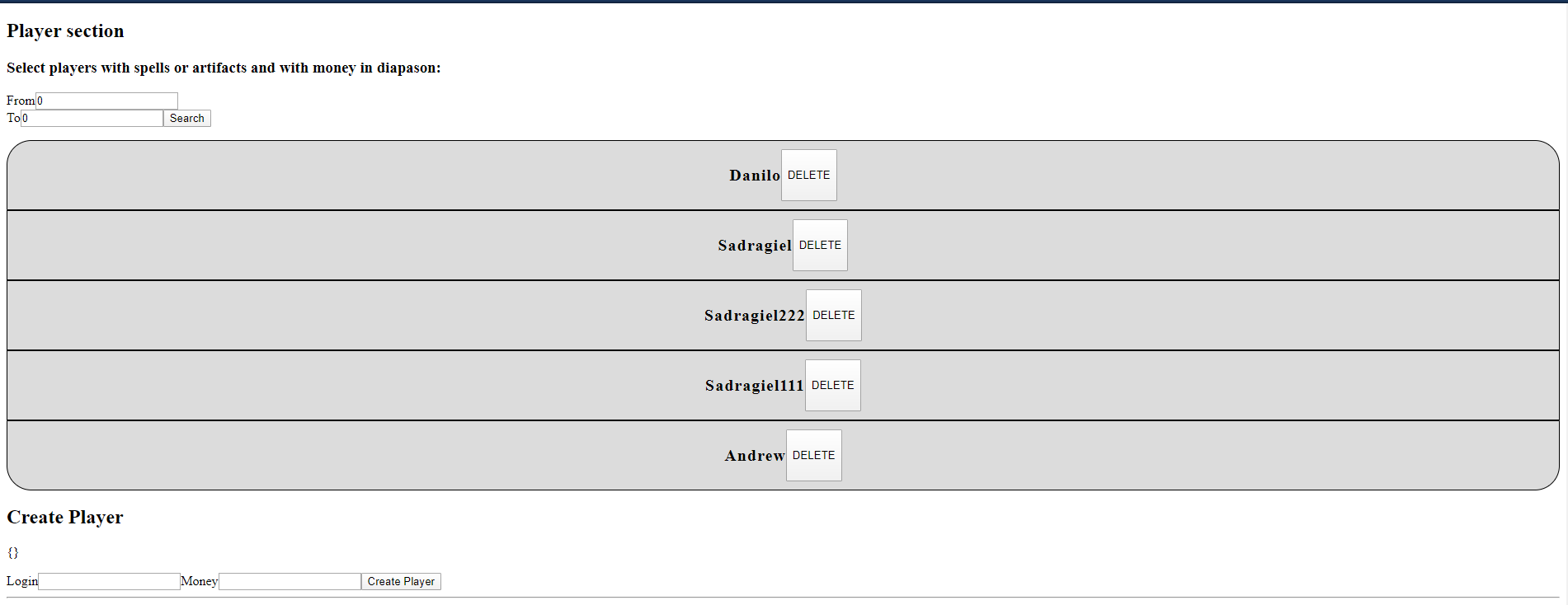
*Таблиця player\_card*

## **Концептуальна Модель**

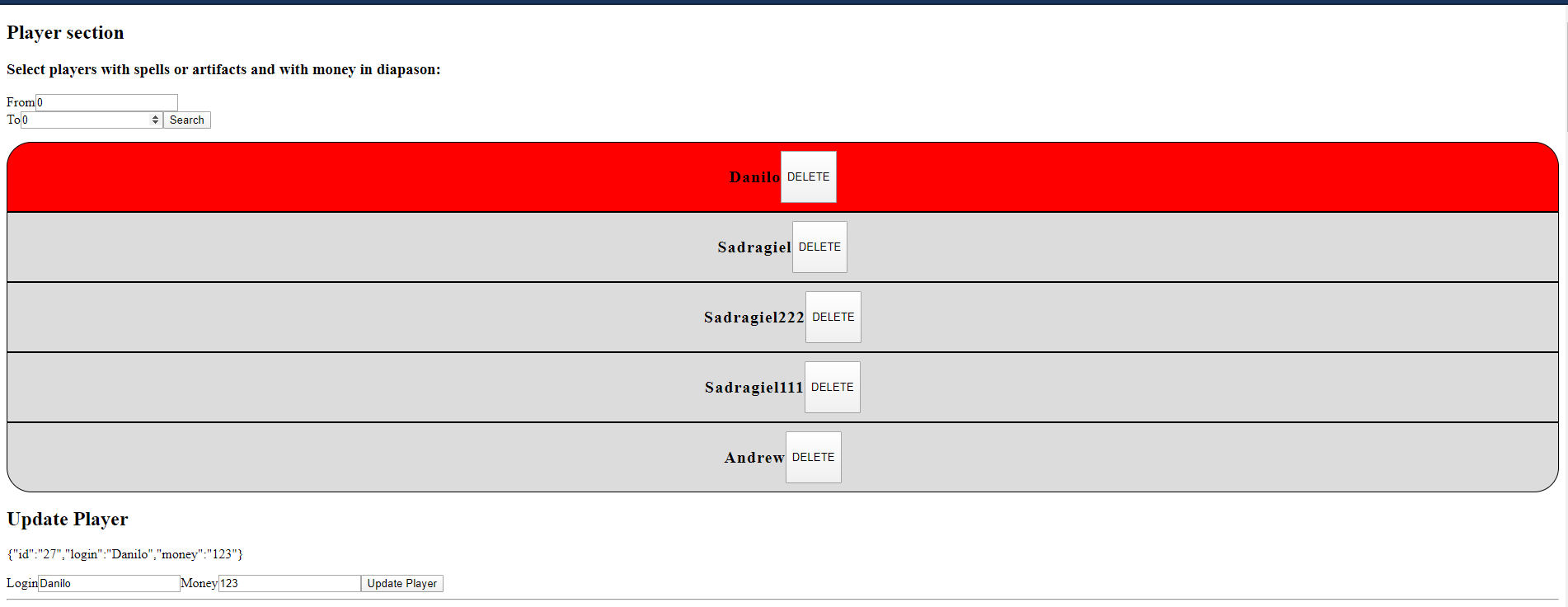
**

## **Приклади роботи програми**

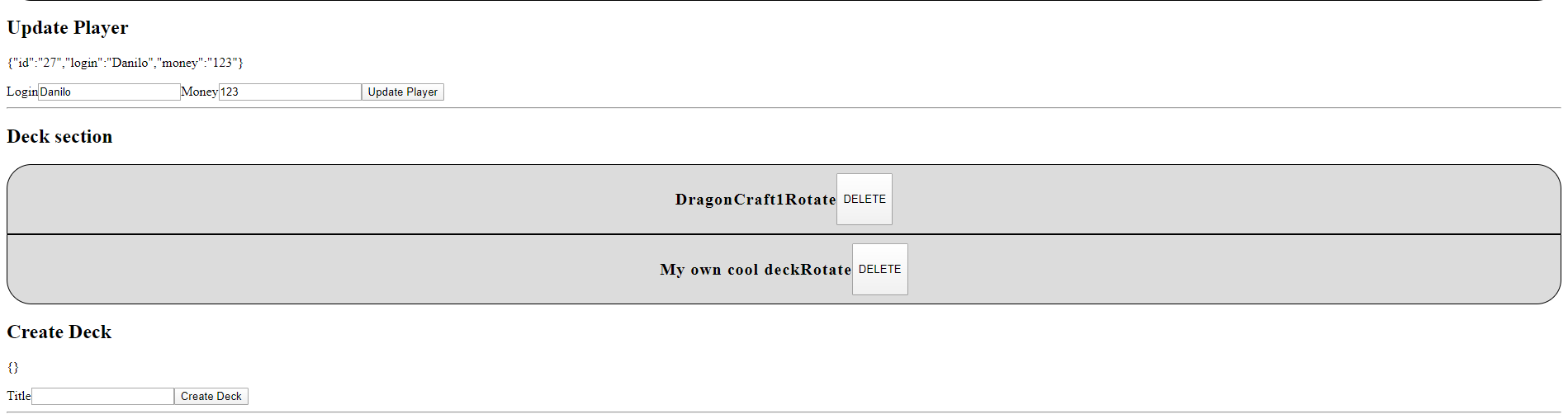
З програмним кодом можна ознайомитись у репозиторії [на гітхабі](https://github.com/Sadragiel/DB_2019)

**

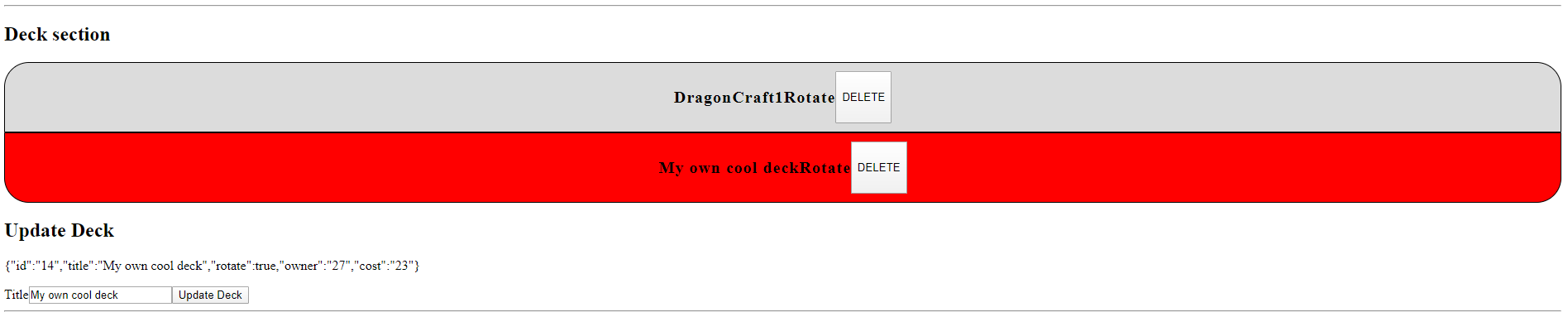
*Секція Гравця. Дві форми: для пошуку у списку сутностей та для створення нової сутності. Список надає базову інформацію про гравця та можливість його видалення з БД*

**

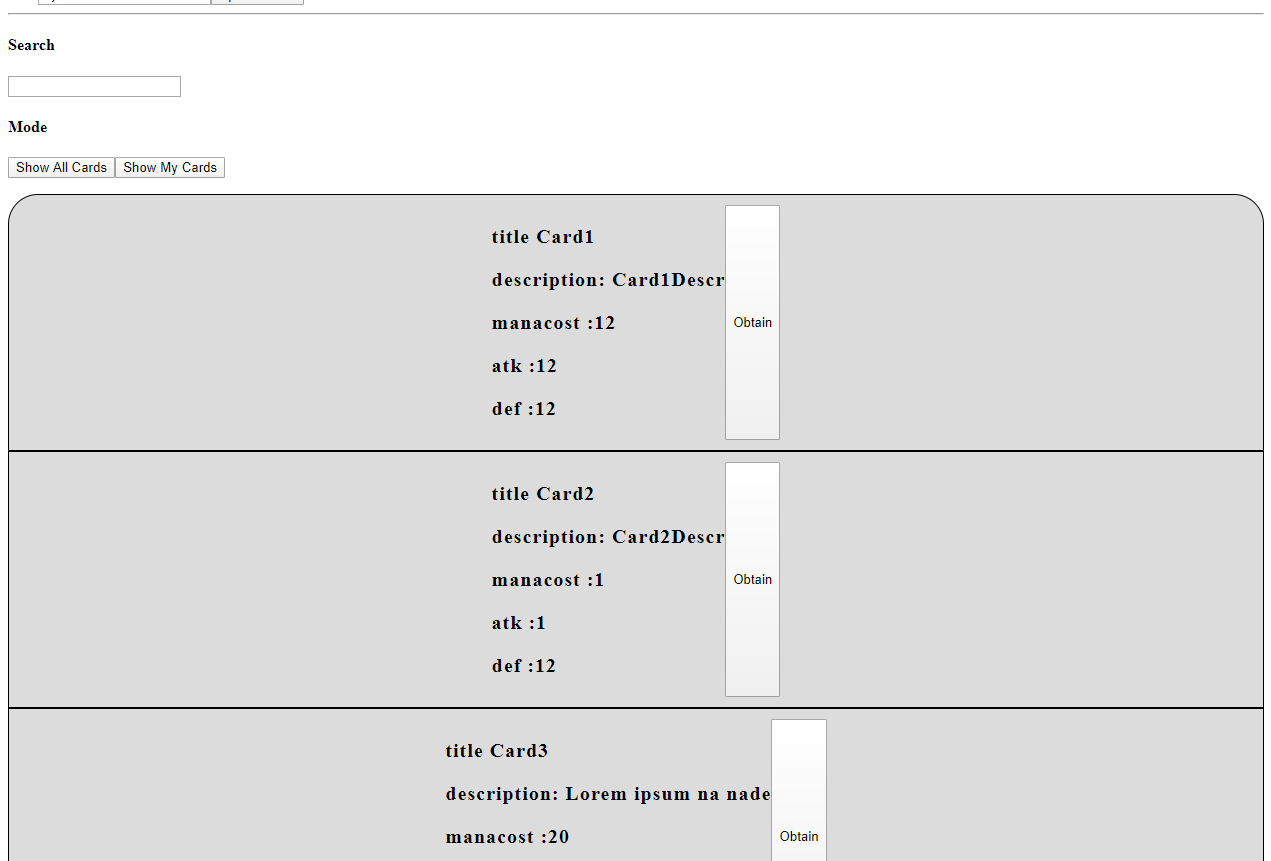
*Секція Гравця. Після обрання гравця для роботи форма Створення перетворюється на форму Оновлення сутності*

**

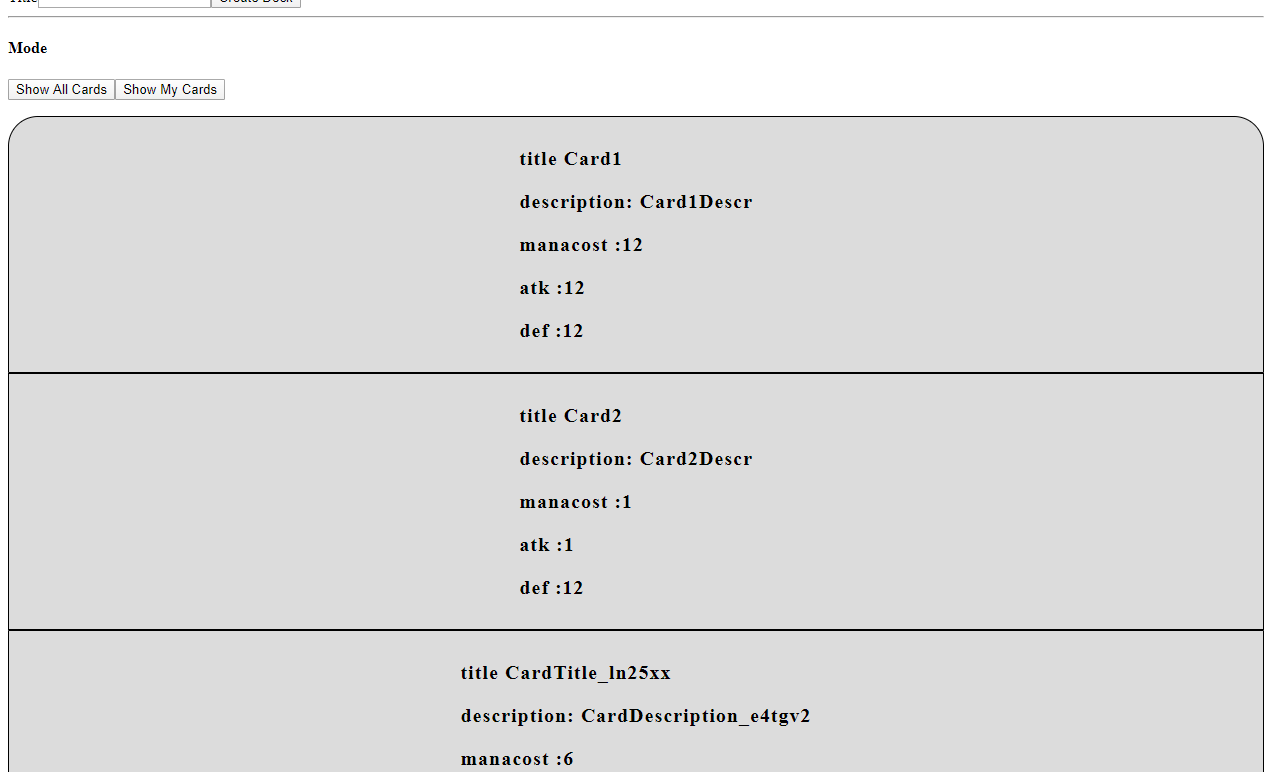
*Секція Колоди. Якщо обрано гравця, у секції виводиться список його колод з таким же функціоналом, як і список Гравців.*

**

*Секція Колоди. Можливість редагування даних про колоду*

**

*Секція Карт. Можливість пошуку по фразі з опису карти шляхом вводу шуканої підстроки у текстове поле. Можливість переглядати всі існуючі карти та лише ті, що належать виділеному гравцю. У списку карт присутня можливість отримати карту, якщо є виділений гравець.*

**

*Секція Карт. Після натискання на відповідну кнопку выводиться список всіх крат, якими володіє гравець.*

Робота з сутністю Ефекту не проводилась, так як весь інтерфейс взаємодії з сутностями є однаковим, а надавати звичайному користувачу можливість редагування існуючих карт - не є доцільним з точки зору проекту.

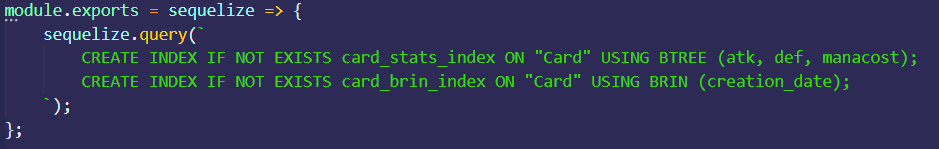
## **Приклади роботи програми**

З програмним кодом можна ознайомитись у репозиторії [на гітхабі](https://github.com/Sadragiel/DB_2019)

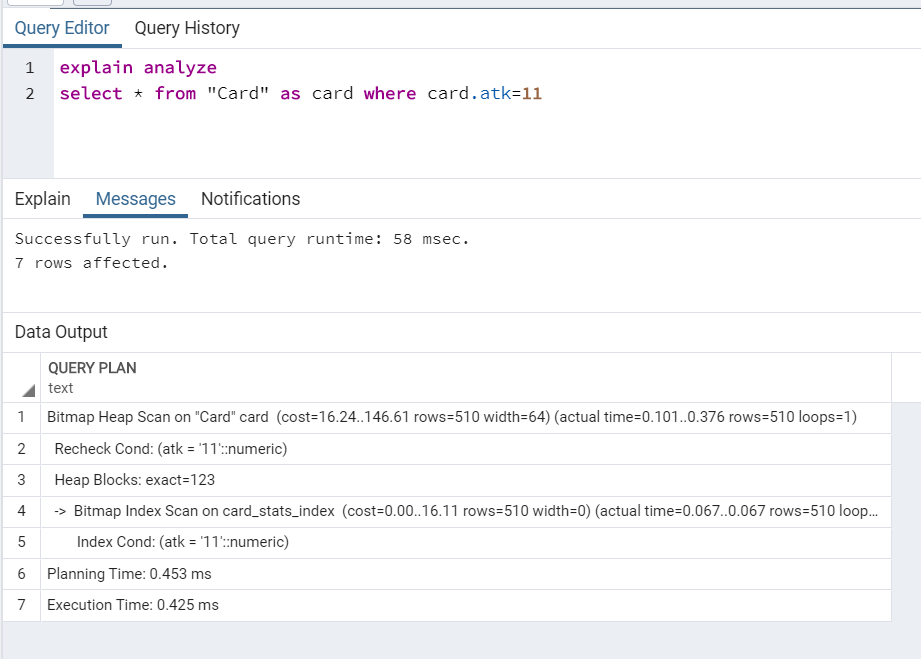
Було виконано наступні завдання:

1. Для роботи з базою даних було використано бібліотеку Sequelize.
2. Проведена робота з Індексами:

Програмна реалізація:



Використання при запитах:

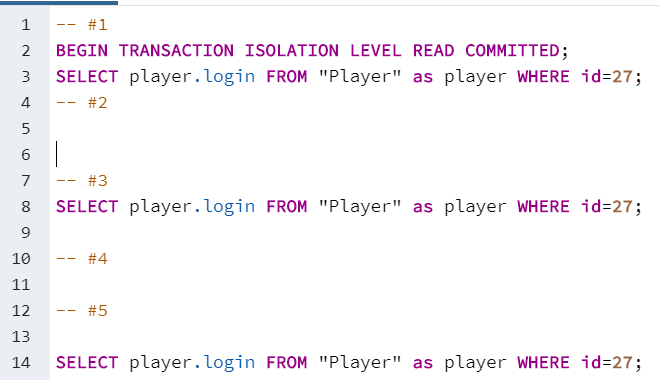


Використання індексу BRIN домогтися не вдалося. Серед можливих причин: відсутність упорядкованості у таблиці за полем, що було обрано для індексації, або занадто велика кількість однакових даних у цьому полі.

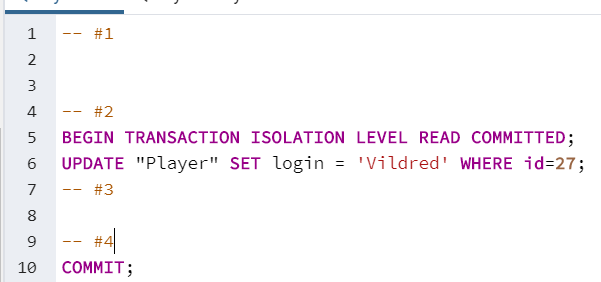
4) Проведено роботу з транзакціями

**Read Commited**

Транзакція 1:

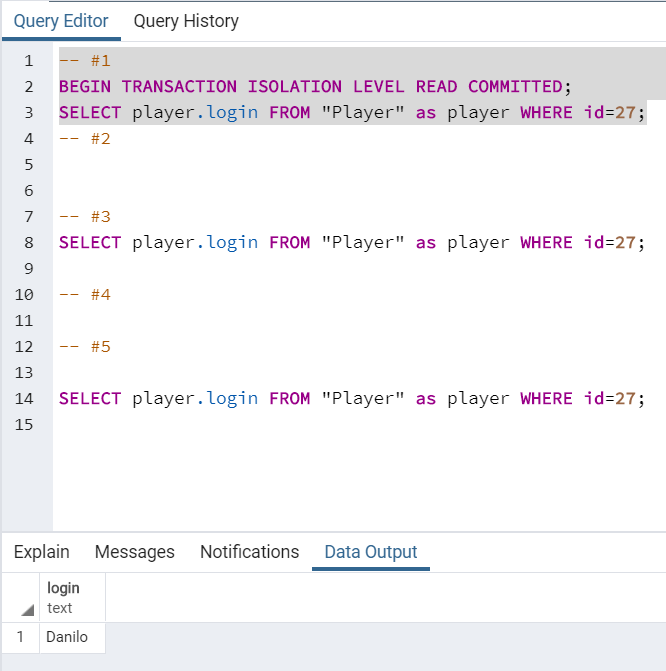


Транзакція 2:

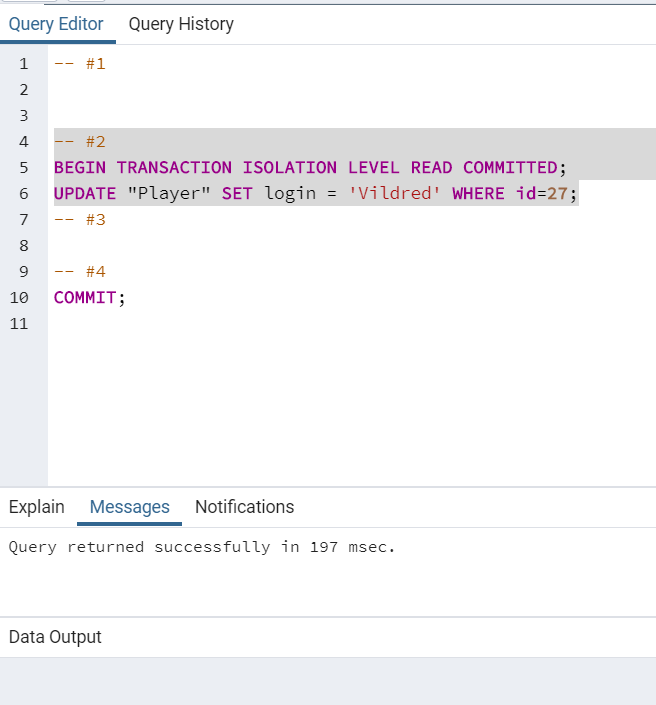


Виконуємо послідовно дії, розділені коментарями.

При читанні рядка першою Транзакцією, ми отримуємо дані про Гравця.



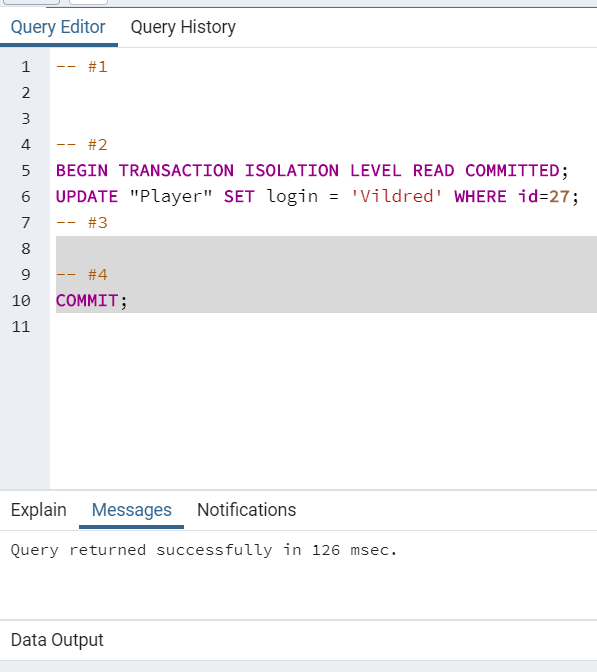
Друга транзакція виконує оновлення цього рядка, проте ще не робить Коміт



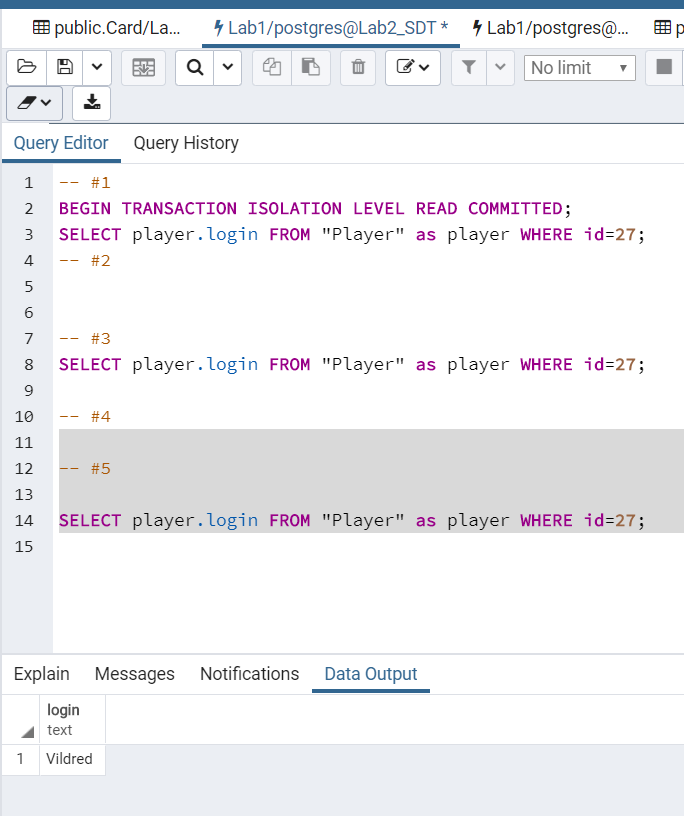
Перша транзакція повторно зчитує данні рядка, проте завдяки рівню ізоляції, що вирішує проблему Брудного Читання, ми отримуємо дані неоновленного рядка.



Виконуємо завершення другої транзакції, щоб перевірити, чи перестануть дані бути “Брудними”.

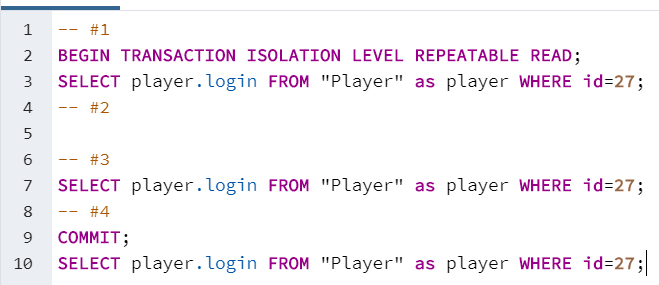


І, звісно, перевіряємо дані рядка. Вони були змінені завершеною транзакцією 2. Дані перестали бути брудними, тому даний рівень ізоляції дозволив зчитати оновлений рядок

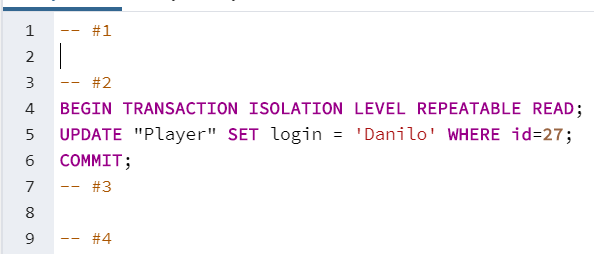


**Repeatable Read**

Транзакція 1:

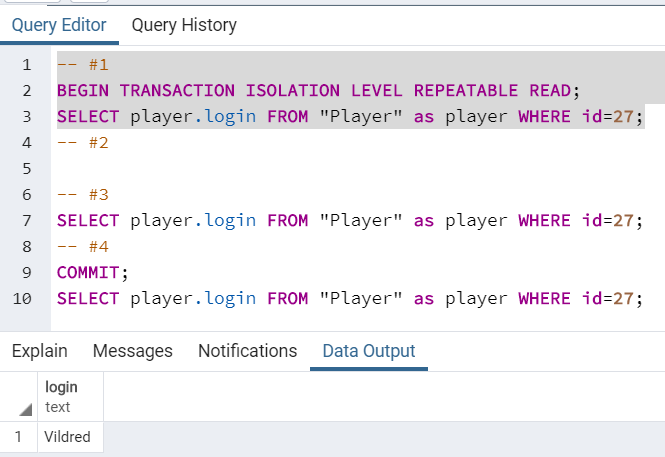


Транзакція 2:

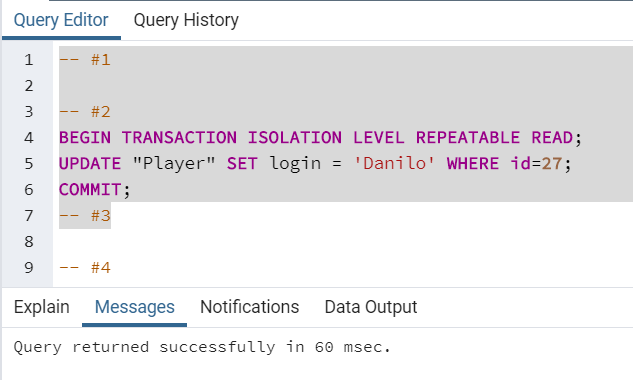


Виконуємо послідовно дії, розділені коментарями.

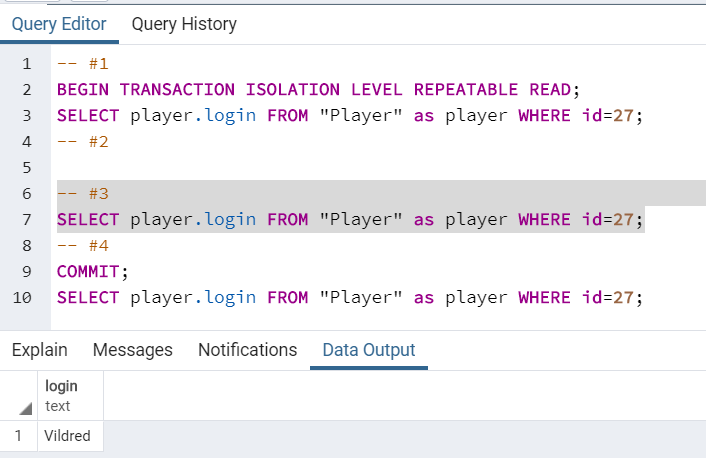
Першою транзакцією зчитуємо дані равця.



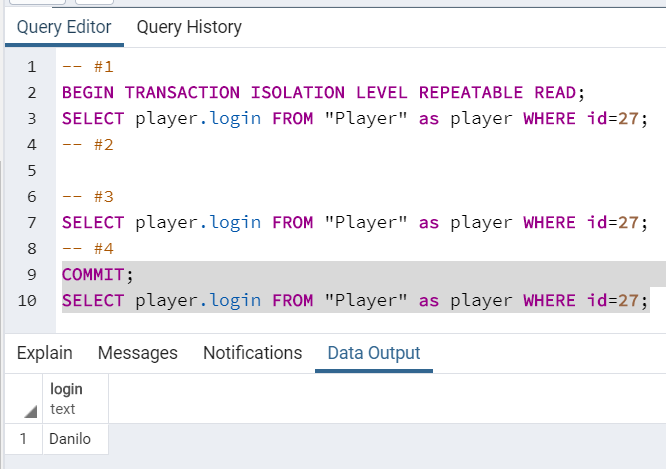
Друга транзакція вносить свої зміни до цього рядка та завершує виконання.



Перша транзакція повторно зчитує рядок таблиці, проте завдяки рівню ізоляції, на дані рядка, що зчитує транзакція, не можуть вплинути інші транзакції

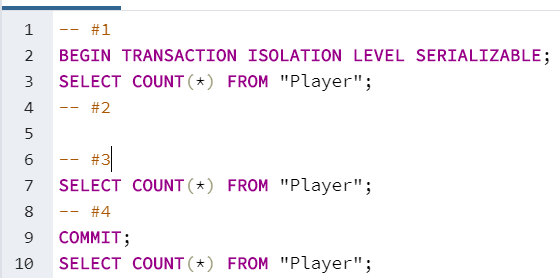


Завершуємо Транзакцію 1 та перевіряємо вміст рядку таблиці. Він оновився, так як ми завершили транзакцію і зробили незалежний запит.



**Serializable**

Транзакція 1:

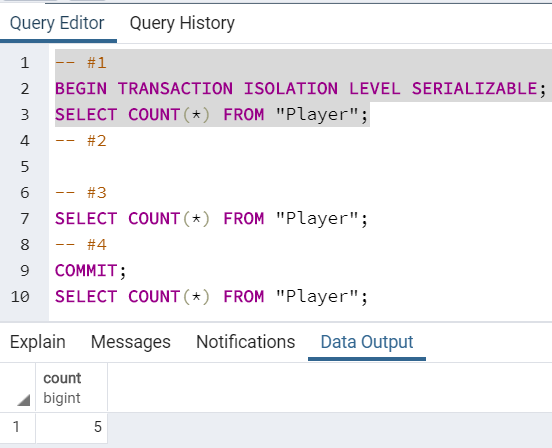


Транзакція 2:

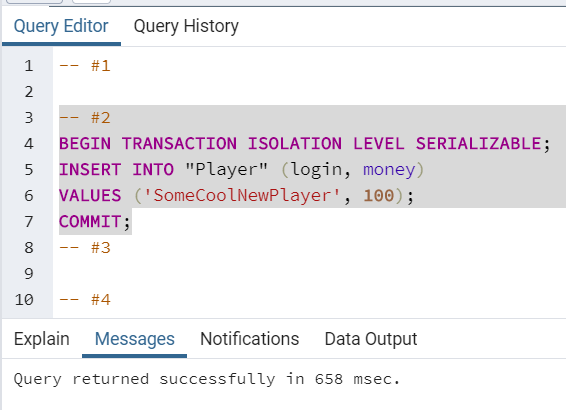


Виконуємо послідовно дії, розділені коментарями.

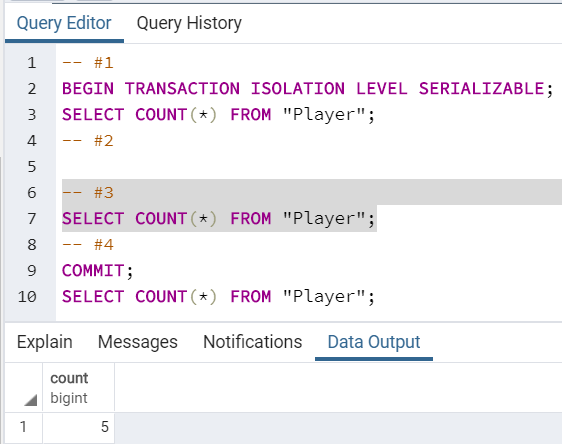
Починаємо транзакцію 1 та рахуємо кількість рядків у таблиці Гравців.



Починаємо другу транзакцію, що додає нового гравця до бази.



Знову рахуємо кількість гравців у базі через першу транзакцію, помічаємо, що їх кількість не змінилася, незважаючи на те, що ми додали новий рядок до таблиці. Це пояснюється рівнем ізоляції Серіалізації. База даних, з якою працює транзакція є повністю ізольованою (серіалізованою).



Завершуємо першу транзакцію та знову перевіряємо кількість рядків таблиці Гравця. Вона змінилася, так як ми завершили ізольовану транзакцію і зробили незалежний запит.

