



НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
Кафедра системного програмування та спеціалізованих комп'ютерних
систем

Лабораторна робота №3

з дисципліни

«Бази даних і засоби управління»

Тема: «Засоби оптимізації роботи СУБД PostgreSQL»

Виконав: студент 3 курсу
ФПМ групи КВ-82
Ященко Іван Васильович
Перевірів: Павловський В.І.

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 проєкції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.

Вимоги до пункту завдання №1

Для перетворення функцій, що реалізують запити до об’єктної бази даних, необхідно встановити бібліотеку sqlalchemy, налаштувати програму на роботу з ORM, розробити класи-сутності для об’єктів-сутностей, представлених відповідними таблицями БД та пов’язаних зв’язками 1:М, М:М та 1:1 виконати опис схеми бази даних. Особливу увагу приділити контролю зовнішніх зв’язків між таблицями засобами ORM.

Замінити виклики запитів мовою SQL на відповідні запити засобами SQLAlchemy по роботі з об’єктами. Обов’язковим є реалізація вставки, вилучення та редагування екземплярів класів-сутностей. Розробка запитів на генерацію даних та пошук екземплярів класів-сутностей вітається, але не є обов’язковою.

Інтерфейси функцій (вхідні та вихідні аргументи функцій модуля “Модель”) мають залишитись без змін.

Вимоги до пункту завдання №2

Відповідно до варіанту індексування продемонструвати на прикладах запитів SQL SELECT підвищення швидкодії їх виконання з використанням індексів, а також пояснити чому для деяких випадків індексування використовувати недоцільно. При цьому для наочного представлення слід використати функцію генерування рандомізованих даних з лабораторної роботи №2, створивши необхідну кількість тестових даних. Навести 4-5 прикладів запитів SELECT (із виведенням результуючих даних), що містять фільтрацію, агрегатні функції, групування та сортування (у необхідних комбінаціях).

Вимоги до пункту завдання №3

Створити тригер бази даних PostgreSQL відповідно до варіанта. Тригерна функція має включати обробку запису, що модифікується (вставляється або вилучається), умовні оператори, курсорні цикли та обробку

виключних ситуацій. Виконати відлагодження тригера при різних вхідних даних, навівши 2-3 приклади його використання.

Варіант 27:

27	<i>GIN, BRIN</i>	<i>after update, insert</i>
----	------------------	-----------------------------

Меню для навігації

Оглавление

Завдання 15

Завдання 29

Завдання 319

Модель бази даних

На Рисунку 1 наведено структуру бази даних яка використовується в даній роботі. Суттєвих змін, в порівнянні з 1та 2 Лабораторною роботами не відбулося.

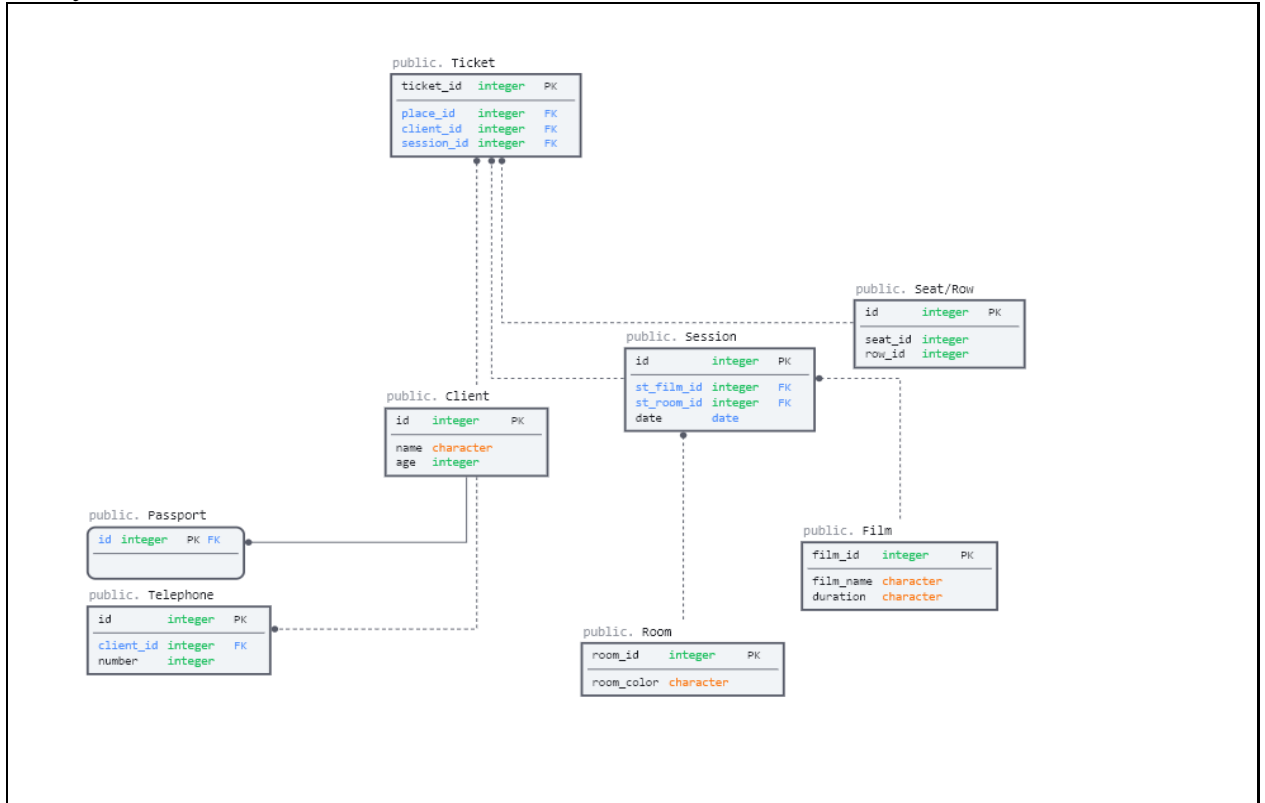


Рисунок 1 – Логічна модель (Структура) БД “ Сервіс продажу квитків кіно ” (засобами SqlDMB)

Завдання 1

Середовище розробки та налаштування підключення до бази даних

Для виконання лабораторної роботи використовувалась мова програмування C# та IDE Visual Studio 2019. Для підключення до серверу бази даних PostgreSQL використовувався пакет Npgsql. Разом з ORM Microsoft EntityFrameworkCore спеціально для пакету баз даних PostgreSQL, задля реалізації оптимізованих звернень до бази даних.

Додаткові пакети даних Visual Studio

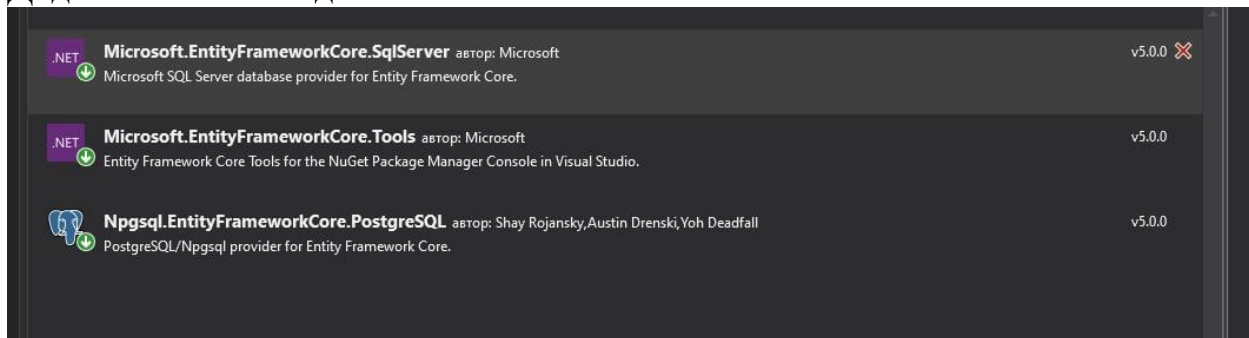


Рисунок 3 - Фотографія екрану, де описано підключення до БД засобами Npgsql

Підключення до БД



Рисунок 3 - Фотографія екрану, де описано підключення до БД засобами Npgsql

Структура програми

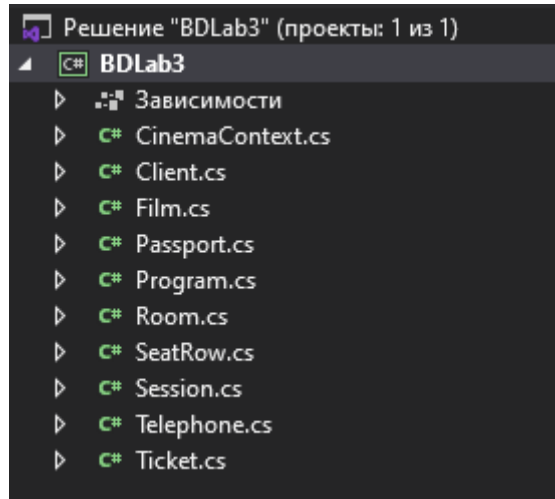


Рисунок 3 – Структура програми

Опис структури програми:

Програма містить 10 модулів. Кожна таблиця, що наявна в БД, була перероблена в окремий модуль-клас. Також тут ми маємо модуль CinemaContext, де описані взаємозв'язки БД, і модуль Program де реалізовані функції, додавання, вилучення та редагування даних, а також main функція програми.

Продемонструємо код лише для одного класу Client

```
5
6 namespace BDLab3
7 {
8     Ссылка: 6
9     public partial class Client
10    {
11        Ссылка: 0
12        public Client()
13        {
14            Tickets = new HashSet<Ticket>();
15        }
16
17        Ссылка: 1
18        public int Id { get; set; }
19        Ссылка: 1
20        public int Age { get; set; }
21        Ссылка: 1
22        public string Name { get; set; }
23
24        Ссылка: 1
25        public virtual Passport Passport { get; set; }
26        Ссылка: 1
27        public virtual Telephone Telephone { get; set; }
28        Ссылка: 2
29        public virtual ICollection<Ticket> Tickets { get; set; }
30    }
31 }
```

Рисунок 4 – Клас Client

Виконуємо синхронізацію з даними із БД

```
modelBuilder.Entity<Client>(entity =>
{
    entity.ToTable("Client");

    entity.Property(e => e.Id)
        .ValueGeneratedNever()
        .HasColumnName("id");

    entity.Property(e => e.Age).HasColumnName("age");

    entity.Property(e => e.Name)
        .HasColumnType("character varying")
        .HasColumnName("name");
});
```

Функція додавання даних класу з таблицею в БД:

```
public class Client_UDI
{
    Ссылка: 0
    public static void Insert(int id, int age, string name)
    {
        using(CinemaContext db = new CinemaContext())
        {
            Client newclient = new Client();

            newclient.Id = id;
            newclient.Age = age;
            newclient.Name = name;

            db.Clients.Add(newclient);
            db.SaveChanges();
        }
    }
}
```

Функція оновлення даних класу з таблицею в БД:

```
public static void Update(int id, int age, string name)
{
    using (CinemaContext db = new CinemaContext())
    {
        Client client = db.Clients.Find(id);
        client.Name = name;
        client.Age = age;
        db.Clients.Update(client);
        db.SaveChanges();
    }
}
```

Функція видалення даних класу з таблицею в БД:

```
public static void Delete(int id)
{
    using(CinemaContext db = new CinemaContext())
    {
        Client client = db.Clients.Find(id);
        db.Clients.Remove(client);
        db.SaveChanges();
    }
}
```

По аналогії створені інші функції додавання, оновлення та видалення інших класів.

Завдання 2

Створення та аналіз індекса GIN

GIN - це Generalized Inverted Index, або обернений індекс. Його основною задачею є прискорення повнотекстового пошуку і тому вивчати даний індекс будемо на цьому прикладі. Створимо вибірку згенерованих тестових рядочків у таблиці `inform` типу `tsvector`, і згенеруємо 500000 рядочків.




	 name tsvector	 doc text	
1	'hx':1	HX	
2	'ic':1	IC	
3	'km':1	KM	
4	'fd':1	FD	
5	'kf':1	KF	
6	'qv':1	QV	
7	'ty':1	TY	
8	'pn':1	PN	
9	'la':1	LA	
10	'mt':1	MT	
11	'ul':1	UL	
12	'ca':1	CA	
13	'xr':1	XR	
14	'ua':1	UA	
15	'vq':1	VQ	
16	'ol':1	OL	
17	'pp':1	PP	
18	'ai':1	AI	
19	'yx':1	YX	
20	'oo':1	OO	
21	'tr':1	TR	
22	'ol':1	OL	

Рисунок 5 – Згенерована таблиця

Тепер проаналізуємо частоту повтору

	name tsvector	counter bigint
1		7114
2	'iu':1	368
3	'df':1	363
4	'ih':1	360
5	'yn':1	359
6	'xj':1	359
7	'hc':1	359
8	'qt':1	359
9	'id':1	357
10	'ks':1	356
11	'uq':1	356
12	'ee':1	356
13	'rt':1	356
14	'er':1	355
15	'qw':1	355
16	'ju':1	354
17	'ml':1	354
18	'vp':1	353

Рисунок 6 – Частота повтору даних згенерованої таблиці

Найчастіше тут зустрічається комбінація ""(пустий рядок), а рідше за все буде зустрічатись рядок name1, бо був заданий вручну.

Тепер перевіримо пошук цих рядків без використання індексів:

```
1 select name from info where name = ''
```

✓ Successfully run. Total query runtime: 241 msec. 7114 rows affected.

✓ Successfully run. Total query runtime: 234 msec. 7114 rows affected.

✓ Successfully run. Total query runtime: 274 msec. 7114 rows affected.

Середній час =249 msec

```
1 select name from info where name @@ to_tsquery('iu')
```

✓ Successfully run. Total query runtime: 314 msec. 368 rows affected.

✓ Successfully run. Total query runtime: 312 msec. 368 rows affected.

✓ Successfully run. Total query runtime: 437 msec. 368 rows affected.

Середній час =354 msec

```
1 select name from info where name @@ to_tsquery('name1')
```

✓ Successfully run. Total query runtime: 343 msec. 2 rows affected.

✓ Successfully run. Total query runtime: 346 msec. 2 rows affected.

✓ Successfully run. Total query runtime: 323 msec. 2 rows affected.

Середній час =337 msec

Тепер створимо GIN індекс:

```
1 create index on info using gin(name)
```

Data Output Explain Messages Notifications

CREATE INDEX

Query returned successfully in 336 msec.

Виконаємо ті ж самі запити:

```
1 select name from info where name = ''
```

✓ Successfully run. Total query runtime: 265 msec. 7114 rows affected.

✓ Successfully run. Total query runtime: 256 msec. 7114 rows affected.

✓ Successfully run. Total query runtime: 238 msec. 7114 rows affected.

Середній час =253 msec

```
1 select name from info where name @@ to_tsquery('iu')
```

✓ Successfully run. Total query runtime: 85 msec. 368 rows affected.

✓ Successfully run. Total query runtime: 78 msec. 368 rows affected.

✓ Successfully run. Total query runtime: 79 msec. 368 rows affected.

✓ Successfully run. Total query runtime: 76 msec. 368 rows affected.

Середній час =79.5 msec

```
1 select name from info where name @@ to_tsquery('name1')
```

✓ Successfully run. Total query runtime: 70 msec. 2 rows affected.

✓ Successfully run. Total query runtime: 73 msec. 2 rows affected.

✓ Successfully run. Total query runtime: 60 msec. 2 rows affected.

Середній час = 68 msec

Тепер бачимо, що після створення індекса, пошук та відображення даних для найчастіше зустрічаємих лексем не змінилося, але для рідко зустрічаємих лексем видно вже покращення результату приблизно в 4-5 разів. Тобто бачимо, що застосування індексу GIN покращило роботу повнотекстового пошуку.

GIN добре підходить для даних, які не часто оновлюються. Якщо поміркувати то для таблиці, де зберігаються часто-змінні дані не рекомендовано використовувати індекс GIN, бо переіндексація може займати багато часу.

Створення та аналіз індекса BRIN

BRIN – це Block Range Index. Він працює добре для тих стовпчиків, де значення корелюють із їх фізичним положенням в таблиці. Тобто, якщо запит без ORDER BY видає значення стовпчика практично в порядку зростання чи спадання

Створимо таблицю numbers з полем date1 де згенеруємо 500000 міток часу.

1 `select date1, count(date1) as counter from numbers group by date1 order by counter DESC`

	Data Output	Explain	Messages	Notifications
	date1 date	counter bigint		
1	2014-03-01	1002		
2	2014-02-10	974		
3	2014-05-01	969		
4	2014-06-08	968		
5	2014-01-23	968		
6	2014-05-10	967		
7	2014-02-17	957		
8	2014-05-11	955		
9	2014-07-20	954		
10	2014-01-12	954		
11	2014-03-27	952		
12	2014-03-26	951		
13	2014-07-27	951		
14	2014-07-02	950		
15	2014-03-20	949		
16	2014-07-08	948		
17	2014-04-06	947		
18	2014-08-06	945		
...		

Рисунок 7– Частота повтору даних згенерованої таблиці

Дані у стовпчику date є не впорядкованими, тому робота індексу повинна бути не коректною, перевіримо це.

Тепер протестуємо дані без використання індексу:’

```
1 select date1 from numbers where date1 = '2014-03-01'
```

✓ Successfully run. Total query runtime: 119 msec. 1002 rows affected.

✓ Successfully run. Total query runtime: 95 msec. 1002 rows affected.

✓ Successfully run. Total query runtime: 112 msec. 1002 rows affected.

Середній час =109 msec

```
1 select date1 from numbers where date1 = '2014-07-20'
```

✓ Successfully run. Total query runtime: 130 msec. 954 rows affected.

✓ Successfully run. Total query runtime: 103 msec. 954 rows affected.

✓ Successfully run. Total query runtime: 162 msec. 954 rows affected.

Середній час =131 msec

```
1 select date1 from numbers where date1 = '2014-08-17'
```

✓ Successfully run. Total query runtime: 99 msec. 803 rows affected.

✓ Successfully run. Total query runtime: 107 msec. 803 rows affected.

✓ Successfully run. Total query runtime: 105 msec. 803 rows affected.

Середній час =104 msec

Тепер створимо індекс:

```
1 create index on numbers using brin(date1)
```

Data Output Explain Messages Notifications

CREATE INDEX

Query returned successfully in 260 msec.

Виконаємо ті ж самі запити:

```
1 select date1 from numbers where date1 = '2014-03-01'
```

✓ Successfully run. Total query runtime: 122 msec. 1002 rows affected.

✓ Successfully run. Total query runtime: 110 msec. 1002 rows affected.

✓ Successfully run. Total query runtime: 106 msec. 1002 rows affected.

Середній час = 113 msec

```
1 select date1 from numbers where date1 = '2014-07-20'
```

✓ Successfully run. Total query runtime: 111 msec. 954 rows affected.

✓ Successfully run. Total query runtime: 111 msec. 954 rows affected.

✓ Successfully run. Total query runtime: 107 msec. 954 rows affected.

Середній час = 109 msec


```
1 select date1 from numbers where date1 = '2014-08-17'
```

✓ Successfully run. Total query runtime: 109 msec. 803 rows affected.

✓ Successfully run. Total query runtime: 111 msec. 803 rows affected.

✓ Successfully run. Total query runtime: 109 msec. 803 rows affected.

Середній час =109 msec

Як ми можемо побачити із-за невідсортованості даних маємо незмінний результат, з-за внутрішньої будови індекса.

Тепер відсотуємо данні і перевіримо роботу індексу:

```
1 select date1 from numbers where date1 = '2014-03-01'
```

✓ Successfully run. Total query runtime: 22 msec. 1002 rows affected.

✓ Successfully run. Total query runtime: 21 msec. 1002 rows affected.

✓ Successfully run. Total query runtime: 26 msec. 1002 rows affected.

Середній час =23 msec

```
1 select date1 from numbers where date1 = '2014-07-20'
```

✓ Successfully run. Total query runtime: 11 msec. 954 rows affected.

✓ Successfully run. Total query runtime: 11 msec. 954 rows affected.

✓ Successfully run. Total query runtime: 17 msec. 954 rows affected.

Середній час =13 msec

```
1 select date1 from numbers where date1 = '2014-08-17'
```

✓ Successfully run. Total query runtime: 24 msec. 954 rows affected.

✓ Successfully run. Total query runtime: 21 msec. 954 rows affected.

✓ Successfully run. Total query runtime: 17 msec. 954 rows affected.

Середній час = 20 msec

Як можемо побачити, робота індекса оптимізувалась і працює коректно.

Для таблиць без кореляції певних стовпчиків із фізичним положенням даний індекс не буде добре працювати по причині внутрішньої реалізації даного індексу. Як ми і побачили на попередньому прикладі.

Завдання 3

Команда створення триггеру(функції тригера)

```
1 create or replace function my_trigger() returns trigger AS $$
2 Declare
3     curs CURSOR FOR SELECT * From test;
4     row test%ROWTYPE;
5 Begin
6     IF(TG_OP = 'UPDATE') THEN
7         IF NEW.number < 0 THEN
8             RAISE NOTICE 'Number can not be less than 0';
9             RETURN NULL;
10        END IF;
11        RAISE NOTICE 'Successfull update';
12        INSERT into logs VALUES ('UPDATE into table',NOW());
13        Return NEW;
14    ELSIF(TG_OP = 'DELETE')THEN
15        INSERT INTO logs VALUES ('DELETE from table', NOW());
16        RAISE NOTICE 'Successesfull delete';
17        RETURN OLD;
18    END IF;
19 END;
20
21 $$language plpgsql;
22
23
```

Рисунок 8– Функція триггеру

І підключимо тригер:

▼ ⚙ Triggers (1)
 → my_trigger

Тепер перевіримо працездатність тригера. Створимо таблицю test з полями id і number, куди додамо 3 рядочки .

▼ test

▼ Columns (2)

- id
- number

І створимо таблицю logs куди будуть записуватись усі успішно оброблені тригером записи.

▼ logs

▼ Columns (2)

- text
- time

Тепер оновимо ці рядочки і перевіримо роботу тригера.

```
1 Update test set number = 350 where id = 1
```

Explain Messages Notifications

ЗАМЕЧАНИЕ: Successfull update
UPDATE 1

Query returned successfully in 77 msec.

```
1 Update test set number = 430 where id = 2
```

Explain Messages Notifications

ЗАМЕЧАНИЕ: Successfull update
UPDATE 1

Query returned successfully in 78 msec.

```
1 Update test set number = 220 where id = 3
```

Explain Messages Notifications

ЗАМЕЧАНИЕ: Successfull update
UPDATE 1

Query returned successfully in 73 msec.

Як можемо побачити тригер спрацював на визовах UPDATE. Тепер спробуємо оновити рядочки значеннями num < 0, і перевіримо вставлену умову в тригері:

```
1 Update test set number = -100 where id = 1
```

Explain Messages Notifications

ЗАМЕЧАНИЕ: Number can not be less than 0
UPDATE 0

Query returned successfully in 76 msec.

```
1 Update test set number = -320 where id = 2
```

Explain Messages Notifications

ЗАМЕЧАНИЕ: Number can not be less than 0
UPDATE 0

Query returned successfully in 77 msec.

Як бачимо умова тригера спрацювала і рядочки не оновились, так як вони < 0 .

Тепер спробуємо видалити рядочки:

```
1 Delete from test where id =3
```

Explain Messages Notifications

ЗАМЕЧАНИЕ: Successesfull delete
DELETE 1

Query returned successfully in 73 msec.

```
1 Delete from test where id =2
```

Explain Messages Notifications

ЗАМЕЧАНИЕ: Successesfull delete
DELETE 1

Query returned successfully in 77 msec.

Як бачимо тригер спрацював.

Тепер подивимся таблицю logs:

	text text	time timestamp without time zone
1	UPDATE into table	2020-12-03 12:38:27.85039
2	UPDATE into table	2020-12-03 12:39:26.846865
3	UPDATE into table	2020-12-03 12:39:46.706341
4	DELETE from table	2020-12-03 12:41:14.638286
5	DELETE from table	2020-12-03 12:41:26.355806

Отже тригер працює на командах UPDATE і DELETE, а також я додав умову, що якщо оновлене значення < 0 тоді тригер не оброблює цю операцію і повертає NULL, і виводить відповідне сповіщення користувачу.