

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 III курсу
ФПМ групи КВ-82
Любич І. Д.
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 проекції (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.

Корисні посилання: [тут](#) і [тут](#).

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).

Корисні посилання: [Hash](#), [B-tree](#), [GIN](#), [BRIN](#).

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.

Корисні посилання: [тут](#), [тут](#).

Вимоги до інструментарію

1. Бібліотека для реалізації ORM - [SQLAlchemy для Python](#) або інша з подібною функціональністю.
2. Середовище для відлагодження SQL-запитів до бази даних – pgAdmin 4.
3. СУБД - PostgreSQL 11-12.

Варіант:

<i>№ варіанта</i>	<i>Види індексів</i>	<i>Умови для тригера</i>
16	GIN, Hash	after delete, insert

Меню для навігації

1. [Завдання 1](#)
2. [Завдання 2](#)
3. [Завдання 3](#)

Завдання 1

На рис. 1 наведено логічну схему бази даних “Командні спортивні змагання”.

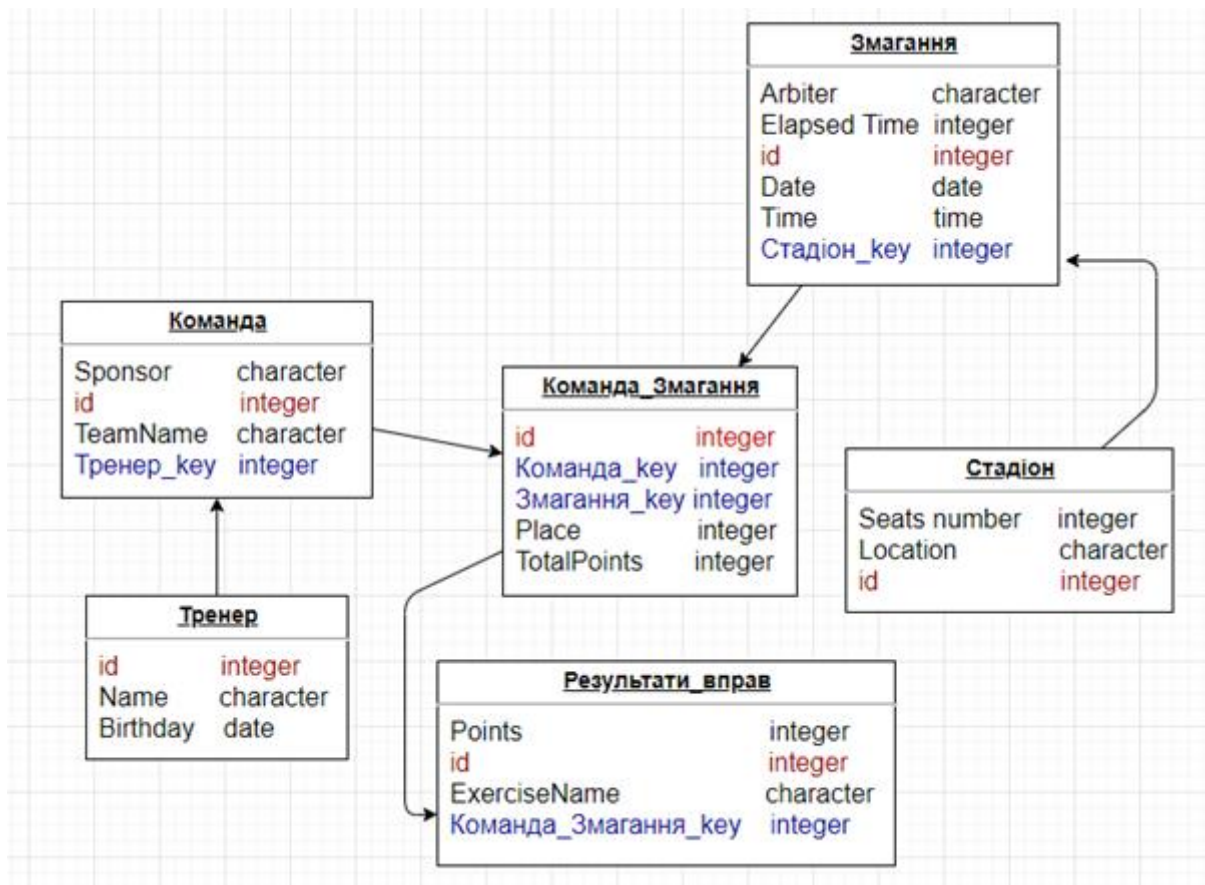


Рис. 1 – Логічна схема БД

Для вирішення задачі перетворення моделі з MVC у вигляд об’єктно-реляційної проєкції (ORM) використовується бібліотека .Net – Fluent NHibernate, яка є однією з найпопулярніших для вирішення задач об’єктно-реляційного відображення.

Представлення таблиць у класах полягає у представленні полів класів як колон таблиці. Для цього потрібно врахувати відповідність типів даних між сервером PostgreSQL і відповідною мовою програмування. Наприклад, типам даних *date* та *time without time zone* PostgreSQL відповідає один тип даних .NET – *DateTime*.

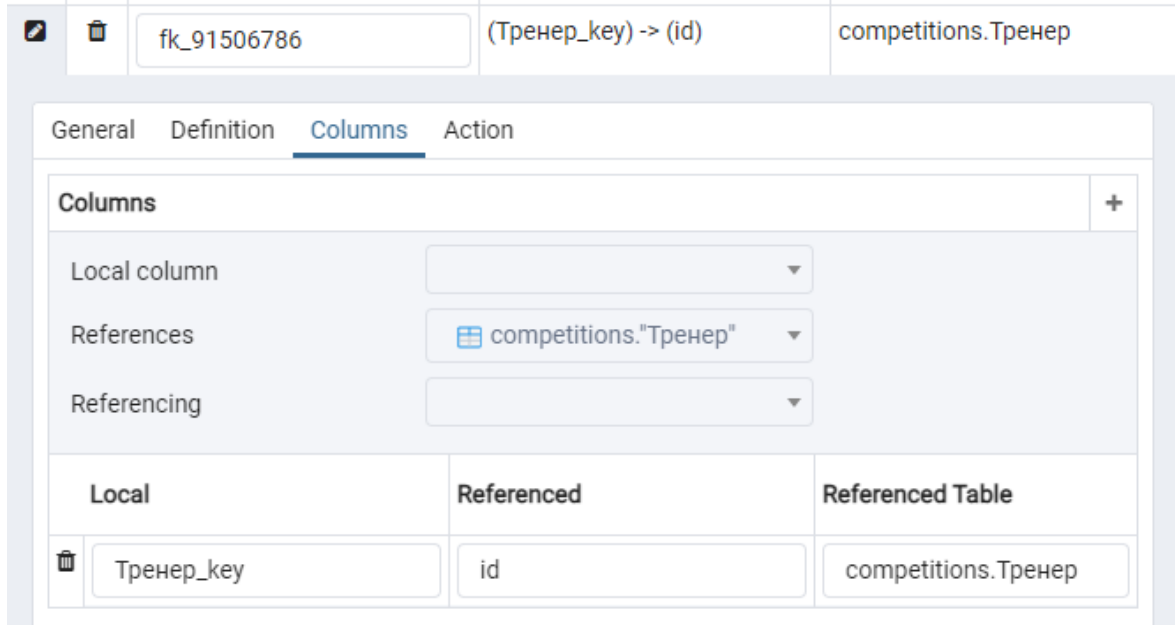
Для реалізації зовнішнього ключа у класі, який відповідає таблиці із зовнішнім ключем з відношенням до колонки таблиці А, потрібно додати поле типу класу, який відповідає таблиці А. Також потрібно написати відповідну команду для визначення цього поля як зовнішній ключ. Виглядає це так:

```

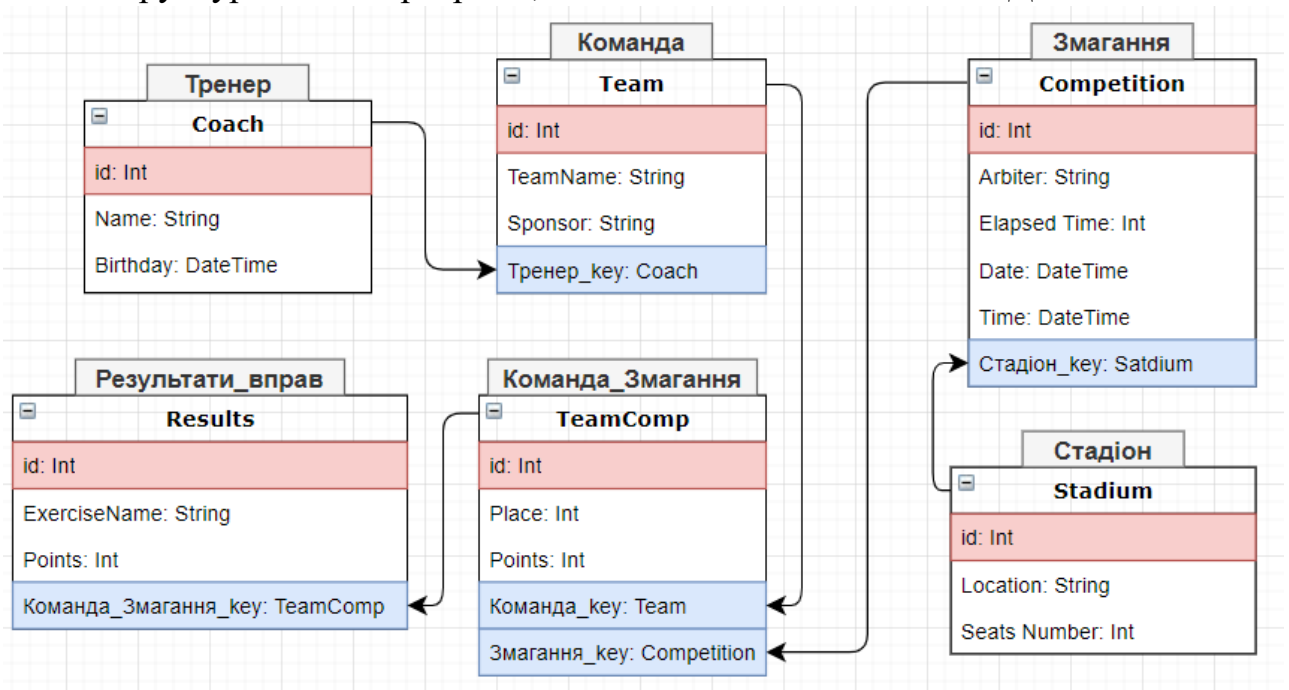
class TeamMap: ClassMap<Team>
{
    Ссылка: 0
    public TeamMap()
    {
        Table("\Команда\");
        References(x => x.CoachKey).Column("\Тренер_key\").Unique();
    }
}

```

Після виконання дій з класом, у БД в таблиці «Команда» буде згенеровано зовнішній ключ і обмеження для зв'язку 1:1. Скріншот pgAdmin:



Структура класів програми, які відповідають таблицям БД:



Продемонструємо код лише для одного класу – Team («Команда»):

```

public class Team
{
    Ссылка: 1
    public virtual int Id { get; set; }
    Ссылка: 3
    public virtual string TeamName { get; set; }
    Ссылка: 3
    public virtual string Sponsor { get; set; }
    Ссылка: 3
    public virtual Coach CoachKey { get; set; }
}

```

Клас для співвідношення даних класу з таблицею в БД:

```

class TeamMap: ClassMap<Team>
{
    Ссылка: 0
    public TeamMap()
    {
        Schema("competitions");
        Table("\"Команда\"");
        Id(x => x.Id).GeneratedBy.Custom("trigger-identity");
        Map(x => x.TeamName, "\"TeamName\"").Unique();
        Map(x => x.Sponsor, "\"Sponsor\"");
        References(x => x.CoachKey).Column("\"Тренер_key\"").Unique();
    }
}

```

Функція для додавання запису до таблиці:

```

public static void Insert(string teamName, string sponsor, int coachID)
{
    using (var session = DBHelper.OpenSession())
    {
        var team = new Team { TeamName = teamName, Sponsor = sponsor, CoachKey = session.Get<Coach>(coachID) };
        session.Save(team);
        session.Flush();
        session.Close();
    }
}

```

Функція для редагування запису в таблиці:

```

public static void Update(int id, string newTeam, string newSponsor, int newCoachID)
{
    using (var session = DBHelper.OpenSession())
    {
        var persistent = session.Get<Team>(id);
        persistent.TeamName = newTeam;
        persistent.Sponsor = newSponsor;
        persistent.CoachKey = session.Get<Coach>(newCoachID);
        session.Update(persistent);
        session.Flush();
        session.Close();
    }
}

```

Функція для видалення запису з таблиці:

```
public static void Delete(int id)
{
    using (var session = DBHelper.OpenSession())
    {
        session.Delete(session.Get<Team>(id));
        session.Flush();
        session.Close();
    }
}
```

Інші класи та функції реалізовані аналогічно.




Завдання 2

Створення та аналіз індекса GIN

Оскільки GIN – індекс, який використовується для повнотекстового пошуку, аналіз проводиться на текстових даних. Індекс створюється на стовпцях типу `tsvector`. Для дослідження використаємо таблицю *info* з єдиною колонкою *name* типу `tsvector`. До таблиці було додано 1 000 000 рандомізованих даних та декілька рядків вручну. Для аналізу швидкодії запитів будуть використовуватись рядки, які зустрічаються найчастіше й найрідше.




Дізнаємось рядки для аналізу:

```
12 select name, count(name) as cnt from info group by name order by cnt DESC
```

Data Output		Messages	Explain	Notifications
	name tsvector 	cnt bigint 		
1	'blade':1	38834		
2	'north':1	38824		
3	'z':1	38683		
4	'tuvwxyz':1	38678		

Найчастіше: оберемо 'blade' та 'north'.

```
12 select name, count(name) as cnt from info group by name order by cnt ASC
```

Data Output		Messages	Explain	Notifications
	name tsvector		cnt bigint	
1	'array':1		2	
2	'thing':1		3	
3	'abcdefghij':1		19293	
4	'opqrstuvwxyz':1		38068	

Найрідше: оберемо 'array' і 'thing'.

Виконаємо пошук записів за цими словами без використання індексів.

```
select * from info where name @@ to_tsquery('north');
```

✓ Successfully run. Total query runtime: 679 msec. 38824 rows affected.

✓ Successfully run. Total query runtime: 625 msec. 38824 rows affected.

✓ Successfully run. Total query runtime: 592 msec. 38824 rows affected.

Середній час виконання запиту: 632 ms.


```
select * from info where name @@ to_tsquery('blade');
```

✓ Successfully run. Total query runtime: 722 msec. 38834 rows affected.

✓ Successfully run. Total query runtime: 688 msec. 38834 rows affected.

✓ Successfully run. Total query runtime: 653 msec. 38834 rows affected.

Середній час виконання запиту: 687 ms.

```
select * from info where name @@ to_tsquery('array');
```

✓ Successfully run. Total query runtime: 694 msec. 2 rows affected.

✓ Successfully run. Total query runtime: 644 msec. 2 rows affected.

✓ Successfully run. Total query runtime: 620 msec. 2 rows affected.

Середній час виконання запиту: 652 ms.

```
select * from info where name @@ to_tsquery('thing');
```

✓ Successfully run. Total query runtime: 684 msec. 3 rows affected.

✓ Successfully run. Total query runtime: 690 msec. 3 rows affected.

✓ Successfully run. Total query runtime: 644 msec. 3 rows affected.

Середній час виконання запиту: 672 ms.

Тепер створимо індекс GIN на колонці name:

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

Data Output Messages Explain Notifications

CREATE INDEX

Query returned successfully in 308 msec.

Виконаємо попередні команди знову.

```
select * from info where name @@ to_tsquery('north');
```

✓ Successfully run. Total query runtime: 111 msec. 38824 rows affected.

✓ Successfully run. Total query runtime: 104 msec. 38824 rows affected.

✓ Successfully run. Total query runtime: 107 msec. 38824 rows affected.

Середній час виконання запиту: 107 ms.

```
select * from info where name @@ to_tsquery('blade');
```

✓ Successfully run. Total query runtime: 105 msec. 38834 rows affected.

✓ Successfully run. Total query runtime: 115 msec. 38834 rows affected.

✓ Successfully run. Total query runtime: 117 msec. 38834 rows affected.

Середній час виконання запиту: 112 ms.

```
select * from info where name @@ to_tsquery('array');
```

✓ Successfully run. Total query runtime: 48 msec. 2 rows affected.

✓ Successfully run. Total query runtime: 47 msec. 2 rows affected.

✓ Successfully run. Total query runtime: 42 msec. 2 rows affected.

Середній час виконання запиту: 46 ms.

```
select * from info where name @@ to_tsquery('thing');
```

✓ Successfully run. Total query runtime: 43 msec. 3 rows affected.

✓ Successfully run. Total query runtime: 42 msec. 3 rows affected.

✓ Successfully run. Total query runtime: 44 msec. 3 rows affected.

Середній час виконання запиту: 43 ms.

Як бачимо, час виконання запитів для даних, які найчастіше зустрічаються, зменшився майже в 6 разів, для тексту, який зустрічається найрідше – приблизно в 14.

Отже, індексування GIN значно пришвидшило пошук і за текстом, який часто зустрічається, і, у більшій мірі, за текстом, який зустрічається найрідше.

Створення та аналіз індекса HASH

HASH-індекси працюють тільки з простими умовами рівності. Хеш-індекс може бути використаний лише за умови, що стовпець, який індексується, бере участь у порівнянні з оператором «=». Для дослідження використаємо таблицю *info* з єдиною колонкою *value* типу *integer*. До таблиці було додано 1 000 000 рандомізованих чисел. Для аналізу швидкодії запитів будуть використовуватись рядки, які зустрічаються найчастіше й найрідше.

Дізнаємось рядки для аналізу:

```
15 select value, count(value) as cnt from info group by value order by cnt DESC
```

	value integer	cnt bigint
1	19	7280
2	78	7270
3	48	7267
4	47	7248

Найчастіше: оберемо числа 19 та 48.

```
15 select value, count(value) as cnt from info group by value order by cnt ASC
```

	value integer	cnt bigint
1	10000	1
2	9859	1
3	9860	1
4	9861	1
5	9862	1

Найрідше: оберемо числа 10000 та 9860.

Виконаємо пошук записів за цими рядками без використання індексів.

```
select * from info where value = 19
```

✓ Successfully run. Total query runtime: 121 msec. 7280 rows affected.

✓ Successfully run. Total query runtime: 119 msec. 7280 rows affected.

✓ Successfully run. Total query runtime: 123 msec. 7280 rows affected.

Середній час виконання запиту: 121 ms.

```
select * from info where value = 48
```

✓ Successfully run. Total query runtime: 128 msec. 7267 rows affected.

✓ Successfully run. Total query runtime: 132 msec. 7267 rows affected.

✓ Successfully run. Total query runtime: 115 msec. 7267 rows affected.

Середній час виконання запиту: 125 ms.

```
select * from info where value = 10000
```

✓ Successfully run. Total query runtime: 117 msec. 1 rows affected.

✓ Successfully run. Total query runtime: 110 msec. 1 rows affected.

✓ Successfully run. Total query runtime: 115 msec. 1 rows affected.

Середній час виконання запиту: 114 ms.

```
select * from info where value = 9860
```

✓ Successfully run. Total query runtime: 112 msec. 1 rows affected.

✓ Successfully run. Total query runtime: 120 msec. 1 rows affected.

✓ Successfully run. Total query runtime: 114 msec. 1 rows affected.

Середній час виконання запиту: 115 ms.

Тепер створимо індекс Hash на колонці value

```
16 create index on info using hash(value)
```

Data Output Messages Explain Notifications

CREATE INDEX

Query returned successfully in 3 secs 39 msec.

Виконаємо попередні команди знову.

```
select * from info where value = 19
```

✓ Successfully run. Total query runtime: 50 msec. 7280 rows affected.

✓ Successfully run. Total query runtime: 53 msec. 7280 rows affected.

✓ Successfully run. Total query runtime: 51 msec. 7280 rows affected.

Середній час виконання запиту: 51 ms.

```
select * from info where value = 48
```

✓ Successfully run. Total query runtime: 53 msec. 7267 rows affected.

✓ Successfully run. Total query runtime: 51 msec. 7267 rows affected.

✓ Successfully run. Total query runtime: 56 msec. 7267 rows affected.

Середній час виконання запиту: 53 ms.

```
select * from info where value = 10000
```

✓ Successfully run. Total query runtime: 43 msec. 1 rows affected.

✓ Successfully run. Total query runtime: 41 msec. 1 rows affected.

✓ Successfully run. Total query runtime: 47 msec. 1 rows affected.

Середній час виконання запиту: 44 ms.

```
select * from info where value = 9860
```

✓ Successfully run. Total query runtime: 49 msec. 1 rows affected.

✓ Successfully run. Total query runtime: 45 msec. 1 rows affected.

✓ Successfully run. Total query runtime: 44 msec. 1 rows affected.

Середній час виконання запиту: 46 ms.

Як бачимо, час виконання запитів для даних, які найчастіше зустрічаються, зменшився приблизно в 2.5 рази. Для даних, які зустрічаються найрідше – приблизно в стільки ж.

Отже, індексування HASH значно пришвидшило пошук і за значеннями, які часто зустрічається, і за значеннями, які зустрічаються найрідше. Використання цього індексу має місце, якщо на сервер часто надходять запити типу «..where *param* = ...». Якщо ж на це поле також можуть надходити запити з порівнянням (<, >) – індексування BTREE виконуватиме свою роботу краще.

Завдання 3

Логіка тригера

Створений тригер спрацьовує після видалення або додавання запису. Тригер було побудовано для таблиці «Результати_вправ». Якщо маємо операцію видалення, виводимо повідомлення про успішне видалення. Якщо маємо операцію додавання запису, перевіряємо чи знаходиться значення поля «Points» в межах дозволу, якщо ні – видаляємо доданий запис, виводиться повідомлення про помилку. Також, якщо був доданий запис із результатом команди в змаганні, який вже записаний, повторний запис буде видалено, також виводиться повідомлення про помилку.

Команда створення тригера:

```
1 create or replace function Trigger01() returns trigger as $$
2 DECLARE
3     crs Cursor for select * from competitions."Результати_вправ";
4     row competitions."Результати_вправ"%rowtype;
5 BEGIN
6     IF(TG_OP = 'DELETE') THEN
7         raise notice 'record was successfully deleted';
8         return old;
9     ELSEIF(TG_OP = 'INSERT') THEN
10        IF new."Points" < 0 THEN
11            raise notice 'Points number cannot be less than zero';
12            delete from competitions."Результати_вправ" where id = new.id;
13            return NULL;
14        END IF;
15        FOR row in crs loop
16            IF new.id = row.id THEN CONTINUE;
17            END IF;
18            IF new."ExerciseName" = row."ExerciseName" AND
19                new."Команда_Змагання_key" = row."Команда_Змагання_key" THEN
20                raise notice 'Repeatable records are not allowed';
21                delete from competitions."Результати_вправ" where id = new.id;
22                return NULL;
23            END IF;
24        END loop;
25        raise notice 'Successful insertion';
26        return new;
27    END IF;
28 END;
29 $$ language plpgsql;
```

Data Output Messages Explain Notifications

CREATE FUNCTION

Query returned successfully in 58 msec.

Підключення триггеру до таблиці «Результати_вправ»:

```
1 create trigger trig01
2 after delete or insert
3 on competitions."Результати_вправ"
4 for each row execute procedure competitions.trigger01();
```

Data Output Messages Explain Notifications

CREATE TRIGGER

Query returned successfully in 50 msec.

Таблиця «Результати_вправ» до видалення запису:

```
1 SELECT * FROM competitions."Результати_вправ"
2 ORDER BY id ASC
```

Data Output Messages Explain Notifications

	Points integer	id [PK] integer	ExerciseName character varying	Команда_Змагання_key integer
1	56	24	Ex1	13
2	666	25	EXEX2	14
3	55	26	Ex1	15
4	55	27	Ex2	16
5	44	28	Ex2	17
6	45	29	Ex2	18
7	155	30	Ex1	13
8	143	31	Ex1	14
9	60	32	Push ups	14
10	23	33	Pull ups	15

Видалення запису:

```
1 delete from competitions."Результати_вправ"
2 where id = 24
```

Data Output Messages Explain Notifications

ЗАМЕЧАНИЕ: record was successfully deleted
DELETE 1

Query returned successfully in 53 msec.

Результат:

```
5 select * from competitions."Результати_вправ"
```

	Points integer	id [PK] integer	ExerciseName character varying	Команда_Змагання_key integer
1	55	26	Ex1	15
2	55	27	Ex2	16
3	44	28	Ex2	17
4	45	29	Ex2	18
5	155	30	Ex1	13
6	143	31	Ex1	14
7	666	25	EXEX2	14
8	23	33	Pull ups	15
9	60	32	Push ups	14

Додавання записів, які є недопустимі:

```
4 insert into competitions."Результати_вправ" ("Команда_Змагання_key", "ExerciseName", "Points") values (13, 'someName', -4);
```

Data Output	Messages	Explain	Notifications
	ЗАМЕЧАНИЕ: Points number cannot be less than zero ЗАМЕЧАНИЕ: record was successfully deleted INSERT 0 1 Query returned successfully in 44 msec.		

```
4 insert into competitions."Результати_вправ" ("Команда_Змагання_key", "ExerciseName", "Points") values (15, 'someName', 100);
```

Data Output	Messages	Explain	Notifications
	ЗАМЕЧАНИЕ: Repeatable records are not allowed ЗАМЕЧАНИЕ: record was successfully deleted INSERT 0 1 Query returned successfully in 41 msec.		

Додавання допустимого запису:

```
4 insert into competitions."Результати_вправ" ("Команда_Змагання_key", "ExerciseName", "Points") values (15, 'newName', 100);
```

Data Output	Messages	Explain	Notifications
	ЗАМЕЧАНИЕ: Successful insertion INSERT 0 1 Query returned successfully in 44 msec.		

Результат:

5

select * from competitions."Результати_вправ"

Data Output

Messages

Explain

Notifications

	Points integer	id [PK] integer	ExerciseName character varying	Команда_Змагання_key integer
1	55	26	Ex1	15
2	55	27	Ex2	16
3	44	28	Ex2	17
4	45	29	Ex2	18
5	155	30	Ex1	13
6	143	31	Ex1	14
7	666	25	EXEX2	14
8	23	33	Pull ups	15
9	60	32	Push ups	14
10	100	56	newName	15

Було додано додано тільки один запис. Недопустимі були відразу видалені функцією тригера. Отже, тригер працює правильно.