МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

**ЗВІТ**

**з лабораторної роботи № 3**

**«Засоби оптимізації роботи СУБД PostgreSQL»**

**Виконав:**

Варіант: 21

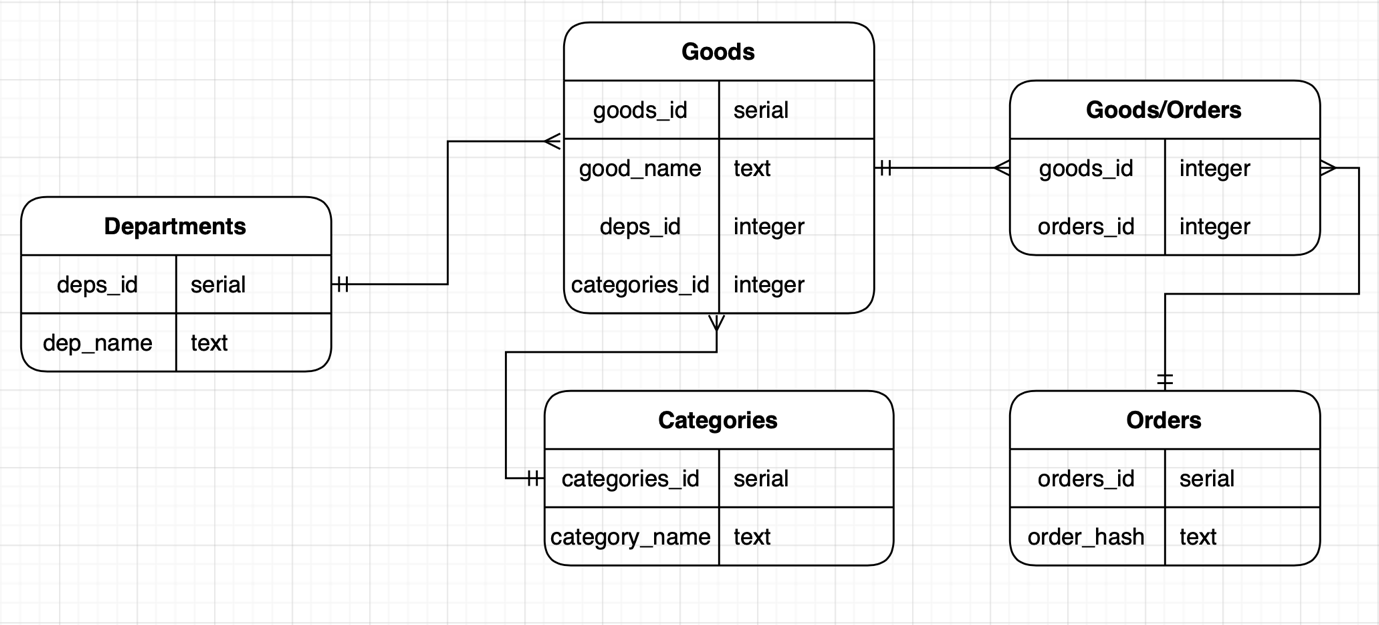
студент 3-го курсу, групи КП-83, спеціальності 121 – Інженерія програмного забезпечення

*Шаповалов Данило Данило*

Київ – 2020

**ТЕМА: «Магазин»**

**Розроблена модель «сутність-зв’язок»:**



**Завдання полягає у:**

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).

2. Створити та проаналізувати різні типи індексів у PostgreSQL.

3. Розробити тригер бази даних PostgreSQL.

4. Навести приклади та проаналізувати рівні ізоляції транзакцій уPostgreSQL.

**TASK 1**

Було використано бібліотеку “diesel” для надання лабораторній роботі вигляду об’єктно-реляційної проекції (ORM). Нижче наведено код, який використовує ORM.

Схеми для роботи з моделями:

|  |
| --- |
| schema.rs |
| table! {  use diesel::sql\_types::\*;  #[allow(non\_snake\_case)]  Categories (categories\_id) {  categories\_id -> Int4,  category\_name -> Text,  }  }  table! {  use diesel::sql\_types::\*;  #[allow(non\_snake\_case)]  Departments (departments\_id) {  departments\_id -> Int4,  department\_name -> Text,  }  }  table! {  use diesel::sql\_types::\*;  #[allow(non\_snake\_case)]  Goods (goods\_id) {  goods\_id -> Int4,  good\_name -> Text,  departments\_id -> Int4,  categories\_id -> Int4,  }  }  table! {  use diesel::sql\_types::\*;  #[allow(non\_snake\_case)]  Orders (orders\_id) {  orders\_id -> Int4,  orders\_name -> Text,  }  }  joinable!(Categories -> Goods (categories\_id));  joinable!(Departments -> Goods (departments\_id));  allow\_tables\_to\_appear\_in\_same\_query!(  Orders,  Goods,  Categories,  Departments,  ); |

Сутність (наведено приклад сутності товару):

|  |
| --- |
| entities.rs |
| use diesel::{Queryable, Insertable};  use crate::{models::schema::Goods};  #[derive(Debug, Queryable, Insertable, AsChangeset)]  #[table\_name="Goods"]  pub struct Good {  pub goods\_id : i32,  pub good\_name : String,  pub departments\_id : i32,  pub categories\_id : i32,  }  #[derive(Debug, Queryable)]  pub struct GoodUpdate {  pub goods\_id: i32,  pub good\_name: Option<Option<String>>,  pub departments\_id : i32,  pub categories\_id : i32,  } |

Імплементація (на прикладі класу Товар):

|  |
| --- |
| From /traits |
| // create.rs  use crate::entities::\*;  use diesel::prelude::\*;  use crate::{database::PgConnPool};  pub trait Create<T> {  fn create(new : &T, connection : &mut PgConnPool) -> bool;  }  #[allow(unused)]  impl Create<Good> for Good {  fn create(new : &Good, connection : &mut PgConnPool) -> bool {  diesel::insert\_into(crate::schema::Goods::table)  .values(new)  .get\_result::<Good>(&(connection.get().unwrap()));  true  }  // update.rs  pub trait Update<T> {  fn update(new : &T, connection : &mut PgConnPool) -> bool;  }  #[allow(unused)]  impl Update<Good> for Good {  fn update(new : &Good, connection : &mut PgConnPool) -> bool {  let tb = crate::schema::Goods::table;  let id = crate::schema::Goods::goods\_id;  let name = crate::schema::Goods::good\_name;  let dep\_id = crate::schema::Goods::departments\_id;  let cat\_id = crate::schema::Goods::categories\_id;  diesel::update(tb.filter(id.eq(new.goods\_id)))  .set((name.eq(new.good\_name.clone()), dep\_id.eq(new.departments\_id), cat\_id.eq(new.categories\_id)))  .get\_result::<Good>(&(connection.get().unwrap()));  true  }  }  // delete.rs  pub trait Delete {  fn delete\_by\_id(index : i32, connection : &mut PgConnPool) -> bool;  }  #[allow(unused)]  impl Delete for Good {  fn delete\_by\_id(index : i32, connection : &mut PgConnPool) -> bool {  let tb = crate::schema::Goods::table;  let id = crate::schema::Goods::goods\_id;  diesel::delete(tb)  .filter(id.eq(index))  .execute(&(connection.get().unwrap()))  .map(|rows\_affected| rows\_affected > 0);  true  }  } |

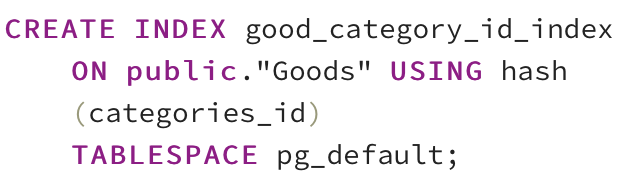
Функціонал аналогічний до функціонала попередньої лабораторної та візуально для користувача залишився незмінним.

**TASK 2**

Індексація за варіантом: btree та hash.

**BTREE**

Код створення:

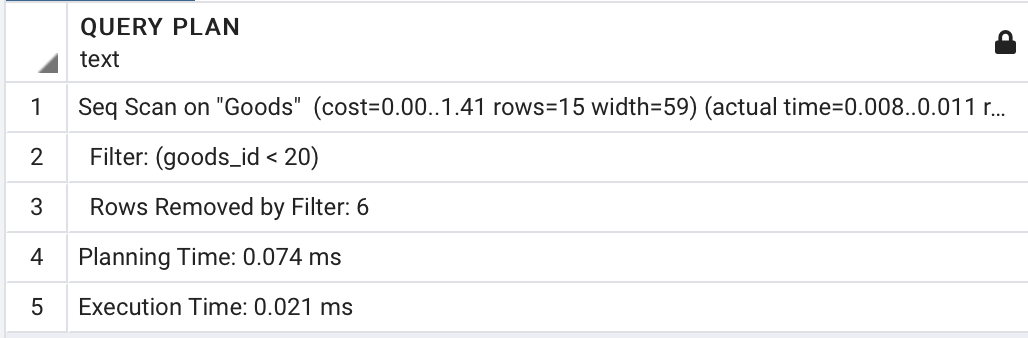


Використання:

Приклад 1:

|  |
| --- |
| EXPLAIN  SELECT \*  FROM public."Goods";  EXPLAIN ANALYSE  SELECT \*  FROM public."Goods"  where goods\_id < 20; |

Результат 1:

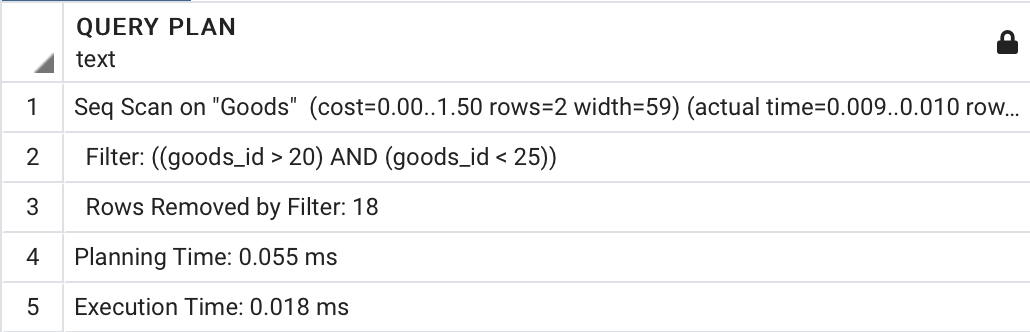


Як бачимо, швидкість виконання суттєво більша, ніж планований час завершення.

Приклад 2:

|  |
| --- |
| EXPLAIN  SELECT \*  FROM public."Goods";  EXPLAIN ANALYSE  SELECT \*  FROM public."Goods"  where goods\_id > 20 and goods\_id < 25; |

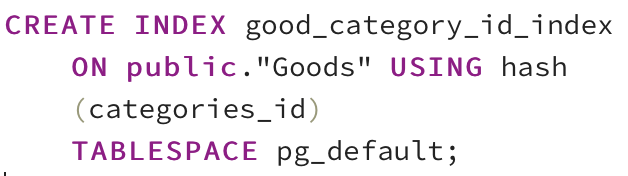
Результат 2:



І вдруге індексація засобом BTREE показує свою ефективність.

**HASH**

Код створення:

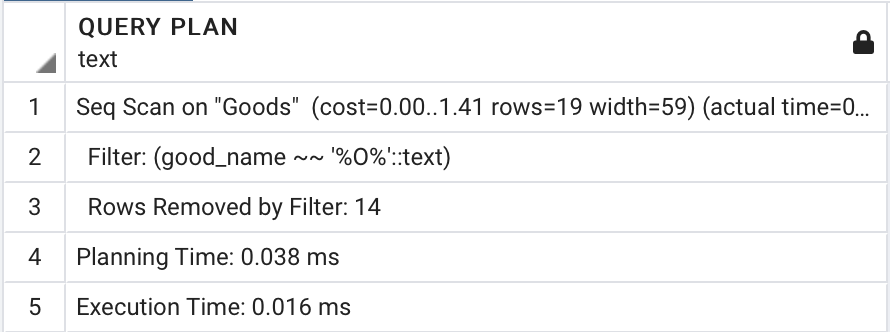


Використання:

Приклад 1:

|  |
| --- |
| EXPLAIN  SELECT \*  FROM public."Goods";  EXPLAIN ANALYSE  SELECT \*  FROM public."Goods"  where good\_name LIKE '%O%'; |

Результат 1:

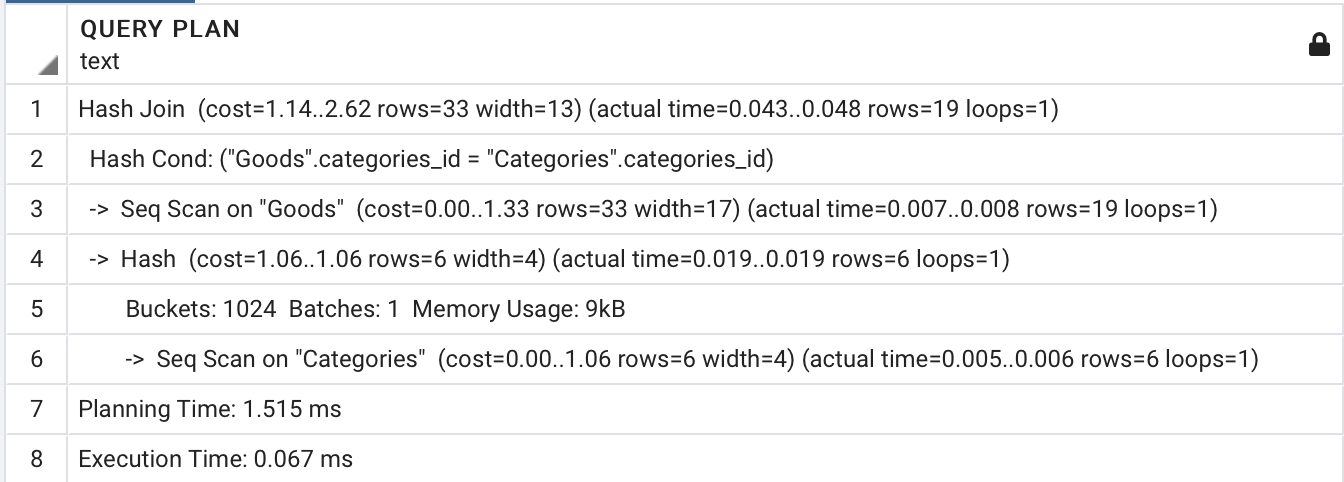


Як бачимо, швидкість виконання суттєво більша, ніж планований час завершення.

Приклад 2:

|  |
| --- |
| EXPLAIN  SELECT \*  FROM public."Goods";  EXPLAIN ANALYSE  SELECT public."Goods".good\_name  FROM public."Goods" join public."Categories"  ON public."Goods".categories\_id = public."Categories".categories\_id |

Результат 2:



Виконуючи складну операцію ми бачимо, наскільки суттєвий приріст у швидкості дає HASH індексація при роботі зі строками.

**TASK 3:**

Тригери за варіантом: before delete, update

Код створення *before delete*:

|  |
| --- |
| CREATE OR REPLACE FUNCTION beforeDeleteGood()  returns trigger  language plpgsql  AS  $$  BEGIN  DELETE FROM public."Orders\_Goods" WHERE goods\_id = OLD.goods\_id;  RETURN OLD;  END;  $$;  CREATE TRIGGER setDelName  BEFORE DELETE  ON public."Goods"  FOR EACH ROW  EXECUTE PROCEDURE beforeDeleteGood(); |

Код створення *before update*:

|  |
| --- |
| REATE OR REPLACE FUNCTION beforeUpdateGood()  returns trigger  language plpgsql  AS $$  DECLARE  departments cursor is select \* from public."Departments";  c BOOLEAN := FALSE;  BEGIN  FOR department IN departments LOOP  IF department.departments\_id = NEW.departments\_id THEN  c :=TRUE;  END IF;  end loop;  IF c = FALSE THEN  NEW.departments\_id = 1;  END IF;  return NEW;  END;  $$;  CREATE TRIGGER setUpdName  BEFORE UPDATE  ON public."Goods"  FOR EACH ROW  EXECUTE PROCEDURE beforeUpdateGood(); |

**Перевірки на коректність роботи**

**Read commited**

|  |  |
| --- | --- |
| Delete transaction 1 | Update transaction 1 |
| begin transaction isolation level read committed;  [TRIGGER CREATING]  DELETE FROM public."Goods" WHERE goods\_id = 28;    DELETE FROM public."Goods" WHERE goods\_id = 32;    DELETE FROM public."Goods" WHERE goods\_id = 33;  commit;  DROP TRIGGER setDelName on public."Goods"; | begin transaction isolation level read committed;  [TRIGGER CREATING]  UPDATE public."Goods" SET good\_name='Bottle', departments\_id=8, categories\_id=1, goods\_id=10 WHERE goods\_id=10;  UPDATE public."Goods" SET good\_name='Bottle', departments\_id=6, categories\_id=1, goods\_id=10 WHERE goods\_id=10;    DROP TRIGGER setUpdName on public."Goods";  commit; |

#### Repeatable read

|  |  |
| --- | --- |
| Delete transaction 2 | Update transaction 2 |
| begin transaction isolation level repeatable read;  [TRIGGER CREATING]  DELETE FROM public."Goods" WHERE goods\_id = 28;    DELETE FROM public."Goods" WHERE goods\_id = 32;  commit;    DELETE FROM public."Goods" WHERE goods\_id = 33;  DROP TRIGGER setDelName on public."Goods"; | begin transaction isolation level repeatable read;  [TRIGGER CREATING]  UPDATE public."Goods" SET good\_name='Bottle', departments\_id=8, categories\_id=1, goods\_id=10 WHERE goods\_id=10;  UPDATE public."Goods" SET good\_name='Bottle', departments\_id=6, categories\_id=1, goods\_id=10 WHERE goods\_id=10;    DROP TRIGGER setUpdName on public."Goods";  commit; |

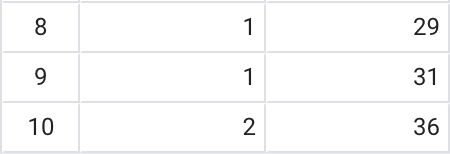
#### Serializable

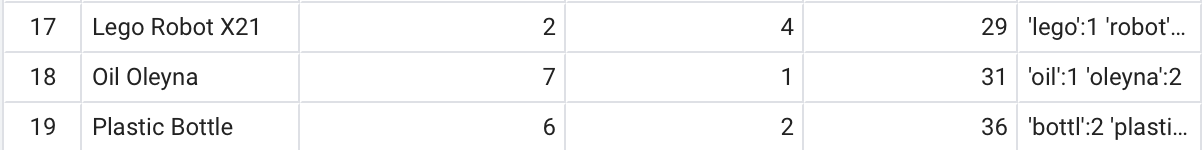
|  |  |
| --- | --- |
| Delete transaction 3 | Update transaction 3 |
| begin transaction isolation level serializable;  select count(\*) from public."Goods";  select count(\*) from public."Goods";  commit;  select count(\*) from public."Goods"; | begin transaction isolation level serializable;  [TRIGGER CREATING]  UPDATE public."Goods" SET good\_name='Bottle', departments\_id=8, categories\_id=1, goods\_id=10 WHERE goods\_id=10;  UPDATE public."Goods" SET good\_name='Bottle', departments\_id=6, categories\_id=1, goods\_id=10 WHERE goods\_id=10;    DROP TRIGGER setUpdName on public."Goods";  commit; |

Приклади виконання:

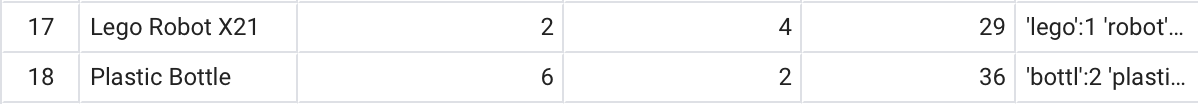
**Delete transaction 1:**

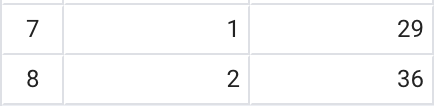
Відповідність замовлення і айді товара (числа 29, 31, 36):





Якщо видаляється товар, то видаляється і його айді з замовлення.

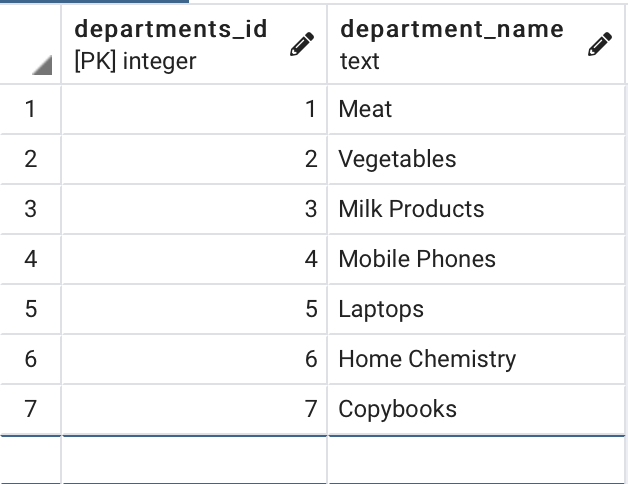




Бачимо, що елемент успішно видалений (айді 31).

**Update transaction 2:**

При оновленні товару перевіряється коректність вводу відділу (чи існує такий відділ). Якщо його не існує, виставляється значення 1 (хоча б один відділ має існувати).

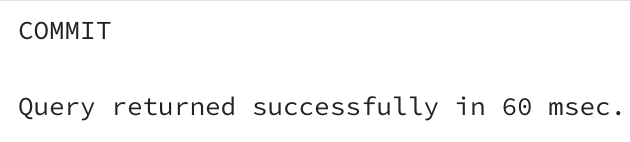


Останній елемент – 7. Якщо ми спробуємо оновити якийсь елемент і вставити айді більше за 7, то база не видасть помилок, а лише вставить елемент зі значенням айді відділу 1.

Було:



Транзакція успішна:



Отримали:



**ВИСНОВОК**

Здобули вміння щодо використання засобів оптимізації СУБД PostgreSQL.