



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики  
Кафедра програмного забезпечення комп’ютерних систем

**Лабораторна робота №3**  
з дисципліни “Бази даних”  
тема “Засоби оптимізації роботи СУБД PostgreSQL ”

Виконав(ла)  
студент(ка) II курсу  
групи КП-03

Тимощук Роман Олександрович  
*(прізвище, ім'я, по батькові)*

Варіант №18

Перевірів  
“ \_\_\_\_ ” “ \_\_\_\_ ” 20\_\_ р.  
викладач

Радченко Константин  
Олександрович  
*(прізвище, ім'я, по батькові)*

Київ 2021

## Мета роботи

Здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

### Постановка завдання

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи No2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.

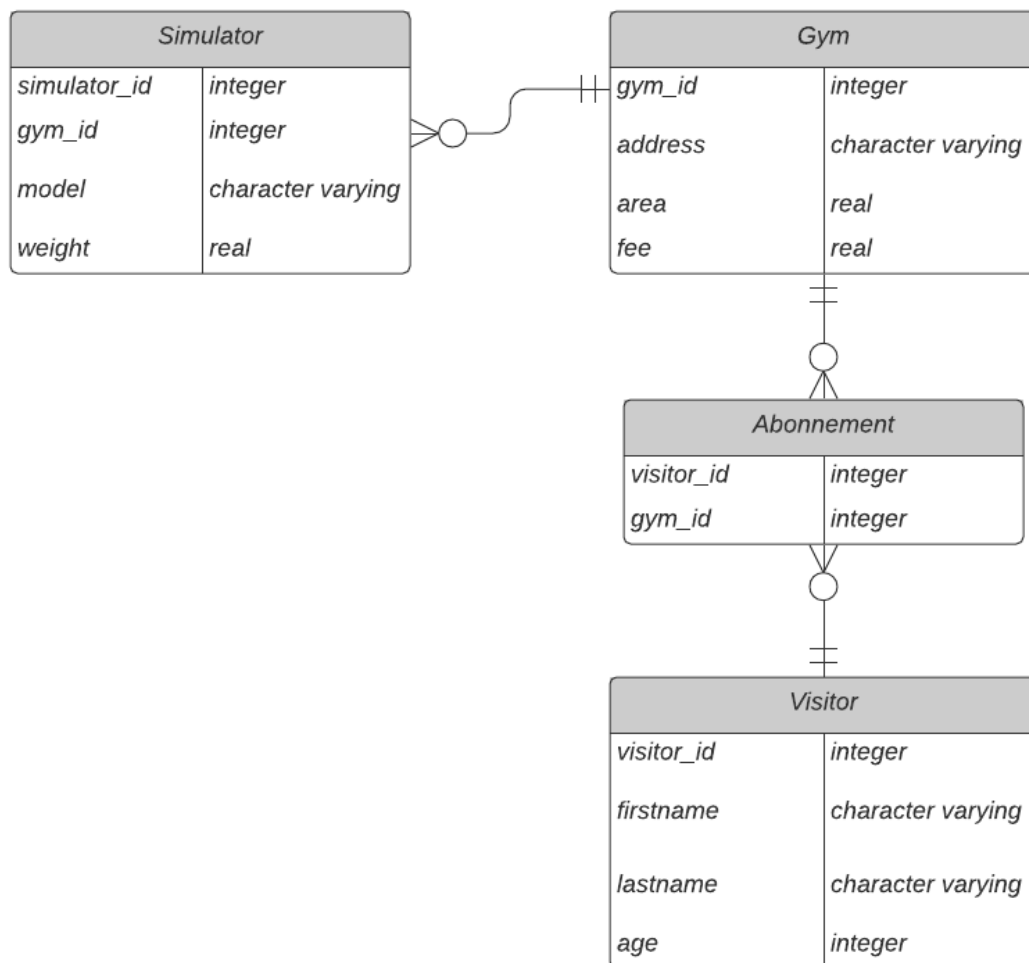
*BTree, GIN*

3. Розробити тригер бази даних PostgreSQL.

*after update, insert*

### Хід виконання

#### Структура нормалізованої бази даних



*ER-діаграма моделі*

## Класи ORM

### Visitor.py

```
from sqlalchemy import Column, Integer, String, Sequence
from sqlalchemy.orm import declarative_base, relationship

Base = declarative_base()

class Visitor(Base):
    __tablename__ = 'visitors'
    visitor_id = Column(Integer, Sequence('visitors_visitor_id_seq', increment=1), primary_key = True)
    firstname = Column(String(50), nullable=False)
    lastname = Column(String(50), nullable=False)
    age = Column(Integer, nullable=False)

    gyms = relationship("Abonnement", back_populates="visitor")

    def __repr__(self):
        return "<Visitor(firstname = '%s', lastname = '%s', age = '%s')>" % (self.firstname, self.lastname, self.age)
```

### Simulator.py

```
from sqlalchemy import Column, Integer, String, Sequence, ForeignKey
from sqlalchemy.orm import declarative_base

Base = declarative_base()

class Simulator(Base):
    __tablename__ = 'simulators'
    simulator_id = Column(Integer, Sequence('simulators_simulator_id_seq', increment=1), primary_key=True)
    gym_id = Column(Integer, ForeignKey('gyms.gym_id', ondelete="CASCADE"), nullable=False)
    model = Column(String(50), nullable=False)
    weight = Column(Integer, nullable=False)

    def __repr__(self):
        return "<Simulator(gym_id = '%s', model = '%s', weight = '%s')>" % (self.gym_id, self.model, self.weight)
```

### Gym.py

```
from sqlalchemy import Column, Integer, String, Sequence
from sqlalchemy.orm import declarative_base, relationship

Base = declarative_base()

class Gym(Base):
    __tablename__ = 'gyms'
    gym_id = Column(Integer, Sequence('gyms_gym_id_seq', increment=1), primary_key=True)
    address = Column(String(100), nullable=False)
    area = Column(Integer, nullable=False)
    fee = Column(Integer, nullable=False)

    visitors = relationship("Abonnement", back_populates="gym")

    def __repr__(self):
```

```
        return "<Gym(address = '%s', area = '%s', fee = '%s')>" % (self.address, self.area, self.fee)
```

## Abonnement.py

```
from sqlalchemy import Column, Integer, String, Sequence, ForeignKey
from sqlalchemy.orm import declarative_base, relationship

Base = declarative_base()

class Abonnement(Base):
    __tablename__ = 'abonnements'
    visitor_id = Column(Integer, ForeignKey('visitors.visitor_id', ondelete="CASCADE"),
        nullable=False)
    gym_id = Column(Integer, ForeignKey('gyms.gym_id', ondelete="CASCADE"), nullable=False)

    gym = relationship("Gym", back_populates="visitors")
    visitor = relationship("Visitor", back_populates="gyms")

    def __repr__(self):
        return "<Abonnement(visitor_id = '%s', gym_id = '%s')>" % (self.visitor_id, self.gym_id)
```

## Запити у вигляді ORM на прикладі класу Gym

### Insert

```
def insert_gym(self, input):
    newGym = Gym(address = input[0], area = input[1], fee = input[2])
    self.session.add(newGym)
    self.session.commit()
```

### Delete

```
def delete_gym(self, id):
    self.session.delete(self.session.query(Gym).get(id))
    self.session.commit()
```

### Update

```
def update_fee(self, input):
    newData = self.session.query(Gym).get(input[1])
    newData.fee = input[1]
    self.session.commit()
```

## Індекси та результат їх роботи

### Команди створення індексів

```
create index btree_visitors on visitors using btree(age);  
create index btree_simulators on simulators using btree(simulator_id);  
create index gin_visitors on visitors using gin(to_tsvector('english', lastname))
```

### Індекс B-tree

Результати запиту select \* from visitors where age > 18 and age <50 order by visitor\_id

До створення індексів:

```
Successfully run. Total query runtime: 196 msec.  
183220 rows affected.
```

Після створення індексів:

```
Successfully run. Total query runtime: 162 msec.  
183220 rows affected.
```

Результати запиту select \* from simulators where simulator\_id > 500 order by weight

До створення індексів:

```
Successfully run. Total query runtime: 411 msec.  
555545 rows affected.
```

Після створення індексів:

```
Successfully run. Total query runtime: 399 msec.  
555545 rows affected.
```

**Отже**, індекс **B-tree** допомагає оптимізувати роботу з даними, які підлягають фільтрації. Як в прикладах, наведених вище, швидкість виконання запитів з операторами “<”, “>” або “=” зростає після створення індексів. Під час роботи з запитамі без фільтрації даний тип індексів використовувати недоцільно.

### Індекс GIN

Результати запиту select \* from visitors where lastname like '%M%' group by visitor\_id order by age desc

До створення індексів:

Successfully run. Total query runtime: 431 msec.  
43861 rows affected.

Після створення індексів:

Successfully run. Total query runtime: 222 msec.  
43861 rows affected.

**Отже**, індекс **GIN** допомагає оптимізувати запити з пошуком тексту, оскільки пошук відбувається за атомарними складовими, на які розбивається текст.

## Тригери та їх відлагодження







### Insert

Код тригера:

```
create function insert_trigger() returns trigger as $insert_trigger$  
  
begin  
  
    if(length(new.firstname) > 50) then  
        raise exception 'entered firstname is so long';  
    elseif(length(new.lastname) > 50) then  
        raise exception 'entered lastname is so long';  
    elseif(new.age < 0 or new.age > 120) then  
        raise exception 'incorrect age entered';  
    end if;  
  
    insert into operations(operation, visitor_id, firstname, lastname, age)  
    values('INSERT', new.visitor_id, new.firstname, new.lastname, new.age);  
  
    return new;  
  
end;  
$insert_trigger$ language plpgsql;  
  
CREATE TRIGGER insert_trigger  
BEFORE INSERT ON visitors  
FOR EACH ROW EXECUTE PROCEDURE insert_trigger();
```

Результат роботи тригера при коректно введених даних:

```
insert into visitors(firstname, lastname, age) VALUES ('Roman', 'Tymoshchuk', 19)
```

	operation 	visitor_id 	firstname 	lastname 	age 
	text	integer	text	text	integer
1	INSERT	555705	Roman	Tymoshchuk	19

Результат роботи тригера при некоректно введених даних:

```
insert into visitors(firstname, lastname, age) VALUES ('Maxym', 'Shevchenko', -4)
```

ERROR: ПОМИЛКА: incorrect age entered  
CONTEXT: Функція PL/pgSQL insert\_trigger() рядок 9 в RAISE

SQL state: P0001

## After update

### Код тригера:

```
create function after_update_trigger() returns trigger as $after_update_trigger$  
  
begin  
    if(length(new.lastname) > 50 or length(new.lastname) < 2) then  
        raise exception 'entered lastname is incorrect';  
    end if;  
    insert into operations(operation, visitor_id, firstname, lastname, age)  
    values('UPDATE', old.visitor_id, old.firstname, new.lastname, old.age);  
  
    return new;  
  
end;  
$after_update_trigger$ language plpgsql;  
  
CREATE TRIGGER after_update_trigger  
AFTER UPDATE ON visitors  
FOR EACH ROW EXECUTE PROCEDURE after_update_trigger();
```

### Результат роботи тригера при коректно введених даних:

**update** visitors **set** lastname = 'Moroz' **where** lastname = 'Tymoshchuk'

	<b>operation</b> text	<b>visitor_id</b> integer	<b>firstname</b> text	<b>lastname</b> text	<b>age</b> integer
1	INSERT	555705	Roman	Tymoshchuk	19
2	UPDATE	555705	Roman	Moroz	19

### Результат роботи тригера при некоректно введених даних:

**update** visitors **set** lastname = ' ' **where** lastname = 'Moroz'

ERROR: ПОМИЛКА: entered lastname is incorrect  
CONTEXT: Функція PL/pgSQL after\_update\_trigger() рядок 5 в RAISE

SQL state: P0001

## Висновки

Виконавши дану лабораторну роботу я **навчився** налаштовувати програму на роботу з ORM.

В ході роботи за допомогою ORM **було реалізовано** контроль зовнішніх зв'язків між таблицями за допомогою ORM. **Було створено** індекси в СУБД PostgreSQL та **досліджено** їх вплив на швидкодію запитів: при правильному використанні індексів запити виконуються швидше. Також **було розроблено** тригери бази даних insert та after update.