НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО"

Кафедра програмного забезпечення комп'ютерних систем

Лабораторна робота №2
із дисципліни «Бази даних»
на тему « Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»

Виконав:
студент 3 курсу ФПМ групи КП-83
Коваль Андрій Олександрович

Прийняв:
Радченко К.О.
“____” “_____” 20___р.

КИЇВ — 2020

Мета: здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Завдання

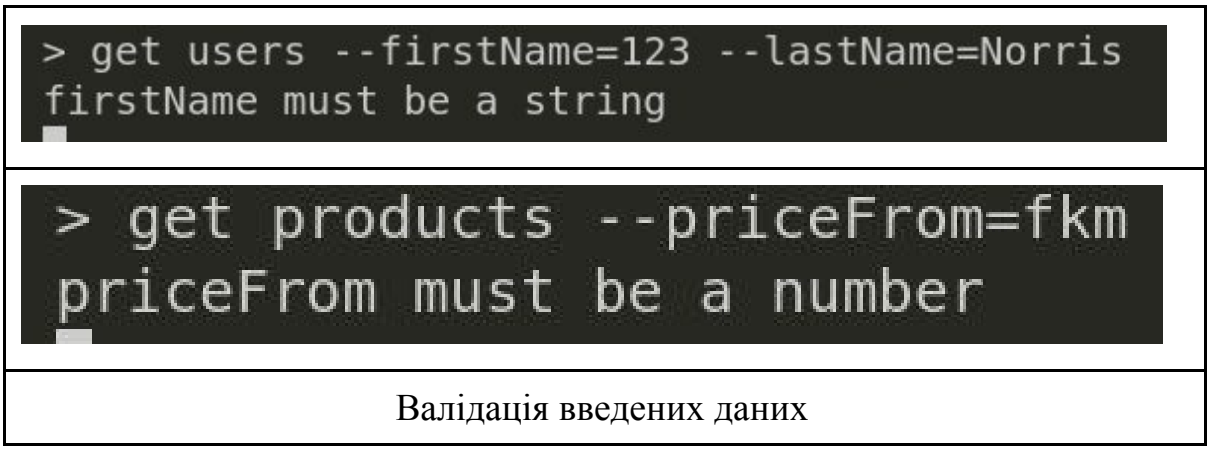
1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі No1, засобами консольного інтерфейсу.

2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.

3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.

4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

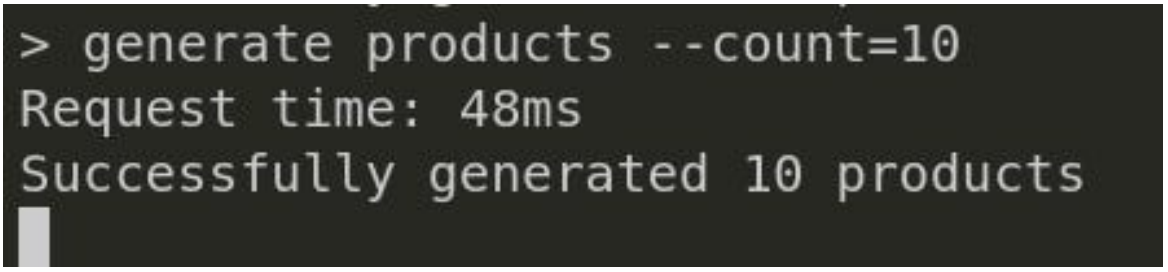
Завдання 1

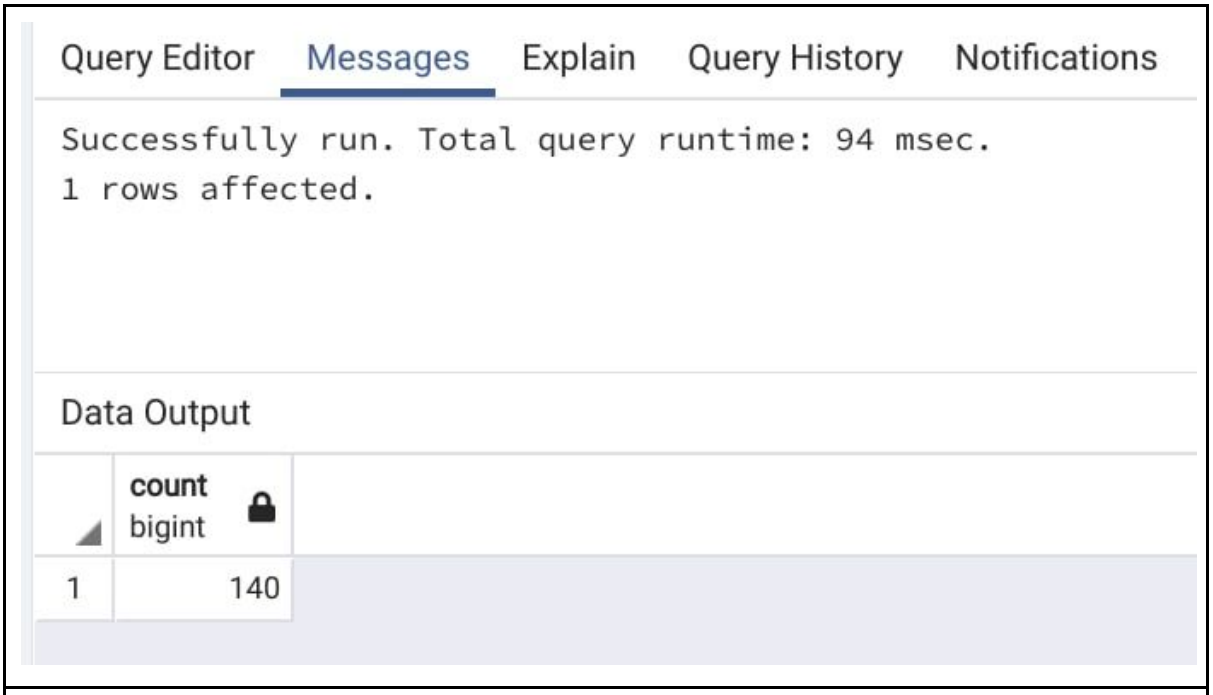


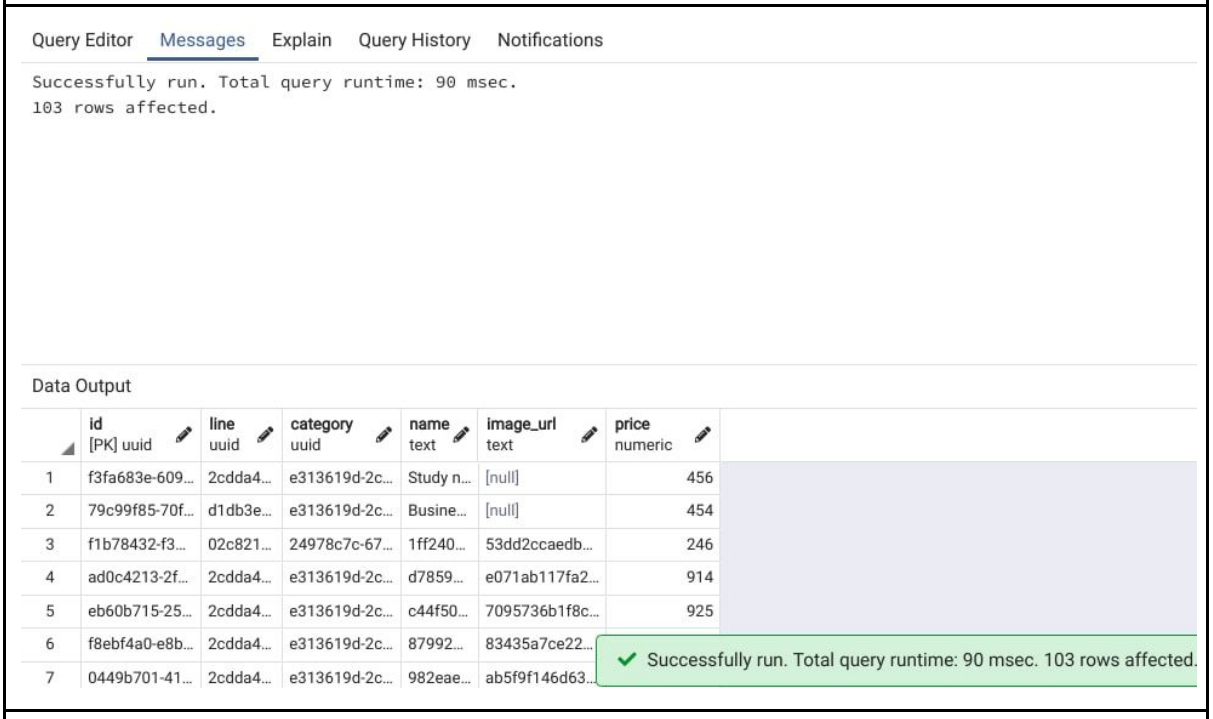
Валідація введених даних

Завдання 2

До генерації







Після генерації

Завдання 3

```
> get users --firstName=Cha
Request time: 44ms
[
  {
    "last_name": "Morris",
    "id": "9d54902b-b45e-4d0e-977f-1e0c8da298fa",
    "first_name": "Chack",
    "email": "mail@mail.com",
    "phone_number": "123123213123"
  }
]
```

```
> get orders --userId=9d54902b-b45e-4d0e-977f-1e0c8da298fa

Request time: 11ms
[
  {
    "id": "99bf0605-2732-4015-8961-e9d56b63b1ac",
    "user_id": "9d54902b-b45e-4d0e-977f-1e0c8da298fa",
    "total_price": "4547",
    "comment": "asd"
  }
]
```

```
> get users --firstName=Cha --lastName=Norris
Request time: 44ms
[]
```

```
> get users --firstName=Cha --lastName=Morris
Request time: 41ms
[
  {
    "last_name": "Morris",
    "id": "9d54902b-b45e-4d0e-977f-1e0c8da298fa",
    "first_name": "Chack",
    "email": "mail@mail.com",
    "phone_number": "123123213123"
  }
]
```

Завдання 4

Покажемо приклад програмного коду роботи з запитами на прикладі сутності “Продукт”

Сховище

```
class ProductsStorage extends BaseStorage {

  public get(filters: ProductFilters): Promise<Product[]> {
    const { sql, values } = filters.getSQLConditionsAndValues();
    const query = `SELECT * FROM products ${sql.length ? 'WHERE' : ''} ${sql}`;
    return db.any(query, values);
  }

  public delete(filters: ProductFilters): Promise<Product[]> {
    const { sql, values } = filters.getSQLConditionsAndValues();
    const query = `DELETE from products ${sql.length ? 'WHERE' : ''} ${sql}`;
    return db.any(query, values);
  }

  public async update(where: ProductFilters, what: ProductFilters): Promise<any> {
    const { sql: whatSql, values: whatValues } = what.getSQLSettingValues();
    const { sql: whereSql, values: whereValues } = where.getSQLConditionsAndValues({
startValueIndex: whatValues.length + 1 });
    const query = `UPDATE products SET ${whatSql} WHERE ${whereSql}`;
    return db.any(query, [ ...whatValues, ...whereValues ]);
  }

  public async generate(count: number): Promise<any> {
    const query = SQL`
    INSERT INTO products ("name", image_url, "line", "category", price)
  (SELECT
    md5(RANDOM() :: TEXT) as "name",
    md5(RANDOM() :: TEXT) as image_url,
       (SELECT id FROM
         product_lines OFFSET
         floor(RANDOM() * (
                SELECT COUNT(*) FROM product_lines)) LIMIT 1) as "line",
```

```
          (SELECT id FROM
              product_categories OFFSET
                floor(RANDOM() * (
               SELECT COUNT(*) FROM product_lines)) LIMIT 1) as "category",
  trunc((RANDOM() * 1000)) as price
  FROM
  generate_series(1, ${count}));`;
  return db.any(query.text, query.values);
  }
}
```

Контроллер та валідатор типів

```
const onGet = async (_filters: IProductFilters): Promise<string> => {
  const validationErrors = validateFilters(_filters);
  if (validationErrors.length) {
    return validationErrors.join('\n');
  }
  const filters = new ProductFilters(_filters);
  try {
    const products = await productsStorage.get(filters);
    return JSON.stringify(products, null, 2);
  } catch (err) {
    console.log(err);
    return err.message;
  }
};

const onDelete = async (_filters: IProductFilters): Promise<string> => {
  const validationErrors = validateFilters(_filters);
  if (validationErrors.length) {
    return validationErrors.join('\n');
  }
  const filters = new ProductFilters(_filters);
  if (filters.isEmpty()) {
    return 'Cannot delete without filters';
  }
  try {
    await productsStorage.delete(filters);
    return 'Successfully deleted';
  } catch (err) {
    return err.message;
  }
};

const onUpdate = async (id: string, updateFields: IProductFilters): Promise<string> => {
  const validationErrors = validateFilters(updateFields);
  if (validationErrors.length) {
    return validationErrors.join('\n');
  }
  delete updateFields['_'];
  const filters = new ProductFilters(updateFields);
  if (filters.isEmpty()) {
    return 'Cannot update without filters';
  }
  try {
    const idFilter = new ProductFilters({ id });
    await productsStorage.update(idFilter, filters);
    return 'Successfully update';
```

```
  } catch (err) {
    return err.message;
  }
};

const onGenerate = async (count) => {
  count = parseInt(count);
  if (isNaN(count) || count < 0) {
    return console.log('Count should be a number and more than 0');
  }
  try {
    await productsStorage.generate(count);
    return `Successfully generated ${count} products`;
  } catch (err) {
    return err.message;
  }
}

const validateFilters = (filters: IProductFilters): string[] => {
  const errors: string[] = [];
  if (filters.id && !validators.isString(filters.id)) {
    errors.push('id must be a string');
  }
  if (filters.category && !validators.isString(filters.category)) {
    errors.push('category must be a string');
  }
  if (filters.line && !validators.isString(filters.line)) {
    errors.push('line must be a string');
  }
  if (filters.name && !validators.isString(filters.name)) {
    errors.push('name must be a string');
  }
  if (filters.price && validators.isNumber(filters.price)) {
    errors.push('price must be a number');
  }
  if (filters.priceFrom && !validators.isNumber(filters.priceFrom)) {
    errors.push('priceFrom must be a number');
  }
  if (filters.priceTo && !validators.isNumber(filters.priceTo)) {
    errors.push('priceTo must be a number');
  }
  return errors;
};
```

View

```
class UI {

  private entityNames: string[] = [
    'users', 'products', 'orders', 'product_lines',
    'product_categories',
  ]

  public start() {
    const rl = readLine.createInterface({
      input: process.stdin,
      output: process.stdout
    });
    rl.on('line', (input: string) => {
        const [method, entity, ...filters] = input.split(' ');
        if (!method || !this.checkEntity(entity)) {
```

```
        return console.log('Invalid method or entity');
      }
      const parsedArgs = minimist(filters);
      const controllers = this.getControllers(entity);
      if (!controllers) {
        return console.log('Invalid method or entity');
      }
      switch (method.toUpperCase()) {
        case 'GET': this.onGet(controllers, parsedArgs); break;
        case 'DELETE': this.onDelete(controllers, parsedArgs); break;
        case 'UPDATE': this.onUpdate(controllers, parsedArgs._[0], parsedArgs); break;
        case 'GENERATE': this.onGenerate(controllers, parsedArgs.count); break;
        default: this.onInvalid();
      }
    }
  );
}

private onGet(controllers, filters) {
  controllers.onGet(filters).then(result => {
    console.log(result)
  });
}

private onDelete(controllers, filters) {
  controllers.onDelete(filters).then(result => {
    console.log(result)
  });
}

private onUpdate(controllers, id: string, filters) {
  if (!id) {
    return console.log('Should provide id to update');
  }
  controllers.onUpdate(id, filters).then(result => {
    console.log(result);
  });
}

private onGenerate(controllers, count: number) {
  if (!controllers.onGenerate) {
    return console.log('Cannot generate current entity');
  }
  controllers.onGenerate(count).then(result => {
    console.log(result);
  });
}

private onInvalid() {
  console.log('Invalid method');
}

private checkEntity(e: string): boolean {
  return this.entityNames.includes(e);
}

private getControllers(entityName: string) {
  switch (entityName) {
    case 'users': return userControllers;
    case 'orders': return orderControllers;
    case 'products': return productControllers;
    case 'product_lines': return productLineControllers;
    case 'product_categories': return productCategoriesControllers;
    default: return null;
  }
```

```
  }
}
```

Базовий абстрактний клас філтрів

```
export interface IFilters {
}

export interface SQLConditionsParams {
  startValueIndex: number;
  separator: string;
}

export default abstract class BaseFilters {
  abstract getSQLConditionsAndValues(opts: SQLConditionsParams): SQLParameters;
  abstract getSQLSettingValues(opts: SQLConditionsParams): SQLParameters;
  abstract isEmpty(): boolean;
}
```

ФІльтр продуктів

```
export default class ProductFilters extends BaseFilters {
  constructor(private filters: IProductFilters) {
    super();
  }

  isEmpty(): boolean {
    return areFiltersEmpty(this.filters);
  }

  getSQLConditionsAndValues({ startValueIndex = 1, separator = ' AND ' } = {}):
SQLParameters {
    const filters = this.filters;
    const fieldsWithOperators: [ string, string, string? ][] = [];
    const values: any[] = [];
    if (filters.id != null) {
      fieldsWithOperators.push(['id', '=']);
      values.push(filters.id);
    }
    if (filters.name != null) {
      fieldsWithOperators.push(['name', 'LIKE']);
      values.push('%' + filters.name + '%');
    }
    if (filters.category != null) {
      fieldsWithOperators.push(['category', '=']);
      values.push(filters.category);
    }
    if (filters.line != null) {
      fieldsWithOperators.push(['line', '=']);
      values.push(filters.line);
    }
    if (filters.priceFrom != null) {
      fieldsWithOperators.push(['price', '>=']);
      values.push(filters.priceFrom);
    }
    if (filters.priceTo != null) {
      fieldsWithOperators.push(['price', '<=']);
      values.push(filters.priceTo);
    }
```

```
    const sql: string = mapper.mapFieldsWithOperatorsToSQL(fieldsWithOperators,
startValueIndex, separator);
    return { sql, values };
  }

  getSQLSettingValues({ startValueIndex = 1, separator = ' , ' } = {}): SQLParameters {
    const filters = this.filters;
    const fieldsWithOperators: [ string, string, string? ][] = [];
    const values: any[] = [];
    if (filters.name != null) {
      fieldsWithOperators.push(['name', '=']);
      values.push(filters.name);
    }
    if (filters.category != null) {
      fieldsWithOperators.push(['category', '=']);
      values.push(filters.category);
    }
    if (filters.line != null) {
      fieldsWithOperators.push(['line', '=']);
      values.push(filters.line);
    }
    if (filters.price != null) {
      fieldsWithOperators.push(['price', '=']);
      values.push(filters.priceFrom);
    }

    const sql: string = mapper.mapFieldsWithOperatorsToSQL(fieldsWithOperators,
startValueIndex, separator);
    return { sql, values };
  }
}
```

Висновки

Я здобув вміння програмувати прикладні додатки з допомогою баз даних PostgreSQL, та обробляти введені дані користувача під час роботи з консольним інтерфейсом.