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 ”

Виконав

студент II курсу

групи КП-93

Катрич Владислав Сергійович

(*прізвище, ім’я, по батькові*)

Перевірив

“____” “_____” 20___ р.

викладач

Петрашенко Андрій Васильович

(*прізвище, ім’я, по батькові*)

Варіант № 1

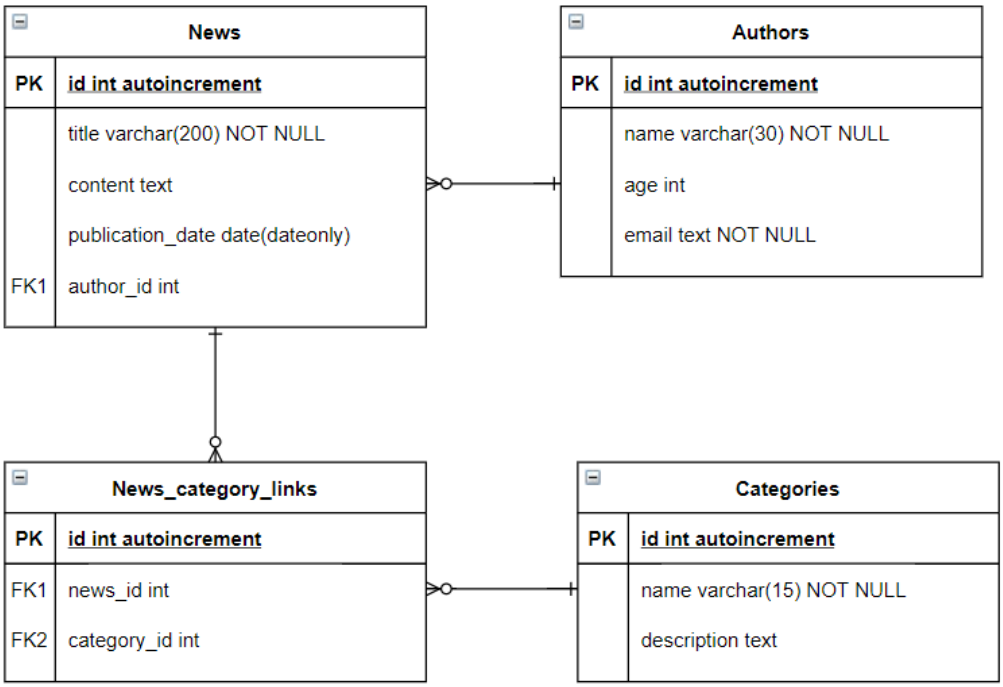
Київ 2020

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.

№1)

Схема бази даних у вигляді таблиць:



Відповідні класи ORM:

Author:

```
Author.init(
    {
        id: {
            type: DataTypes.INTEGER,
            primaryKey: true,
            autoIncrement: true,
        },
        name: {
            type: DataTypes.STRING(30),
            allowNull: false,
        },
        age: DataTypes.INTEGER,
        email: {
            type: DataTypes.STRING(50),
            allowNull: false,
        },
    },
    {
        modelName: 'Author',
        tableName: 'authors',
        timestamps: false,
        sequelize,
    }
);
```

News:

```
News.init(
    {
        id: {
            type: DataTypes.INTEGER,
            primaryKey: true,
            autoIncrement: true,
        },
        title: {
            type: DataTypes.STRING(200),
            allowNull: false,
        },
        content: DataTypes.TEXT,
        publication_date: DataTypes.DATEONLY,
        author_id: {
            type: DataTypes.INTEGER,

            references: {
                model: Author,
                key: 'id',
                deferrable: Deferrable.INITIALLY_IMMEDIAT,
            },
        },
    },
    {
        modelName: 'News',
        tableName: 'news',
        timestamps: false,
        sequelize,
    }
);
```

Category:

```
Category.init(
    {
        id: {
            type: DataTypes.INTEGER,
            primaryKey: true,
            autoIncrement: true,
        },
        name: {
            type: DataTypes.STRING(15),
            allowNull: false,
        },
        description: DataTypes.TEXT,
    },
    {
        modelName: 'Category',
        tableName: 'categories',
        timestamps: false,
        sequelize,
    }
);
```

CategoryNewsLinks:

```
CategoryNewsLinks.init(
    {
        id: {
            type: DataTypes.INTEGER,
            primaryKey: true,
            autoIncrement: true,
        },
        news_id: {
            type: DataTypes.INTEGER,

            references: {
                model: News,
                key: 'id',
                deferrable: Deferrable.INITIALLY_IMMEDIAT,
            },
        },
        category_id: {
            type: DataTypes.INTEGER,

            references: {
                model: Category,
                key: 'id',
                deferrable: Deferrable.INITIALLY_IMMEDIAT,
            },
        },
    },
    {
        modelName: 'CategoryNewsLinks',
        tableName: 'category_news_links',
        timestamps: false,
        sequelize,
    }
);
```

Приклади запитів у вигляді ORM:

```
class Author extends Model {
    async create(item) {
        await Author.create(item);
    }
    async delete(itemId) {
        await Author.destroy({ where: { id: itemId } });
    }
    async update(updatedItem, itemId) {
        await Author.update(updatedItem, { where: { id: itemId } });
    }
}
```

```
Enter command: create
Enter table name: categories
Enter name: Politics
Enter description: Very interestiing topic
Executing (default): INSERT INTO "categories" ("id","name","description") VALUES (DEFAULT,$1,$2) RETURNING "id","name","description";
```

```
async updateItem(tableName, updatedItem, itemId) {
    const classMap = new Map();
    classMap.set('authors', new Author().update);
    classMap.set('news', new News().update);
    classMap.set('categories', new Category().update);
    classMap.set('category_news_links', new CategoryNewsLinks().update);
    const updateFoo = classMap.get(tableName);

    try {
        updateFoo(updatedItem, itemId);
```

№2) *(Btree, Hash)*

Команди створення індексів:

```
CREATE INDEX emailIndex on authors using hash(email)
```

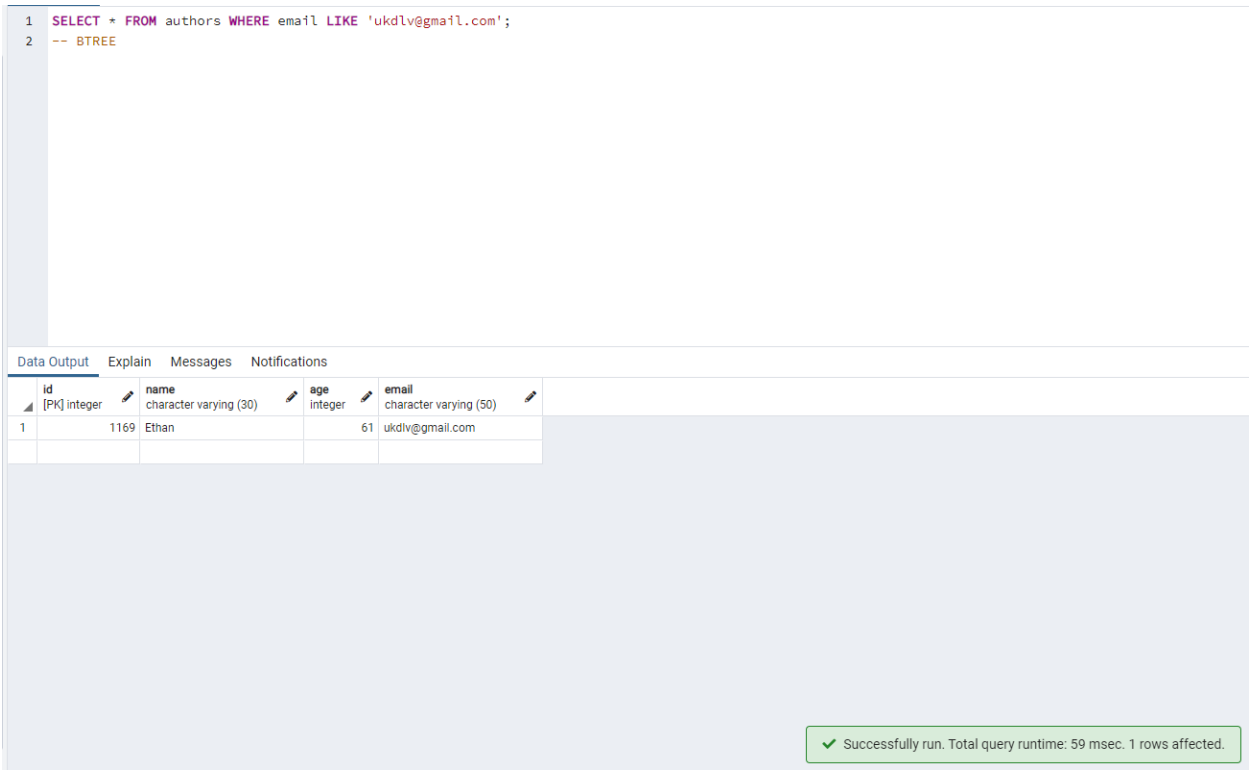
```
CREATE INDEX nameIndex on authors(name)
```

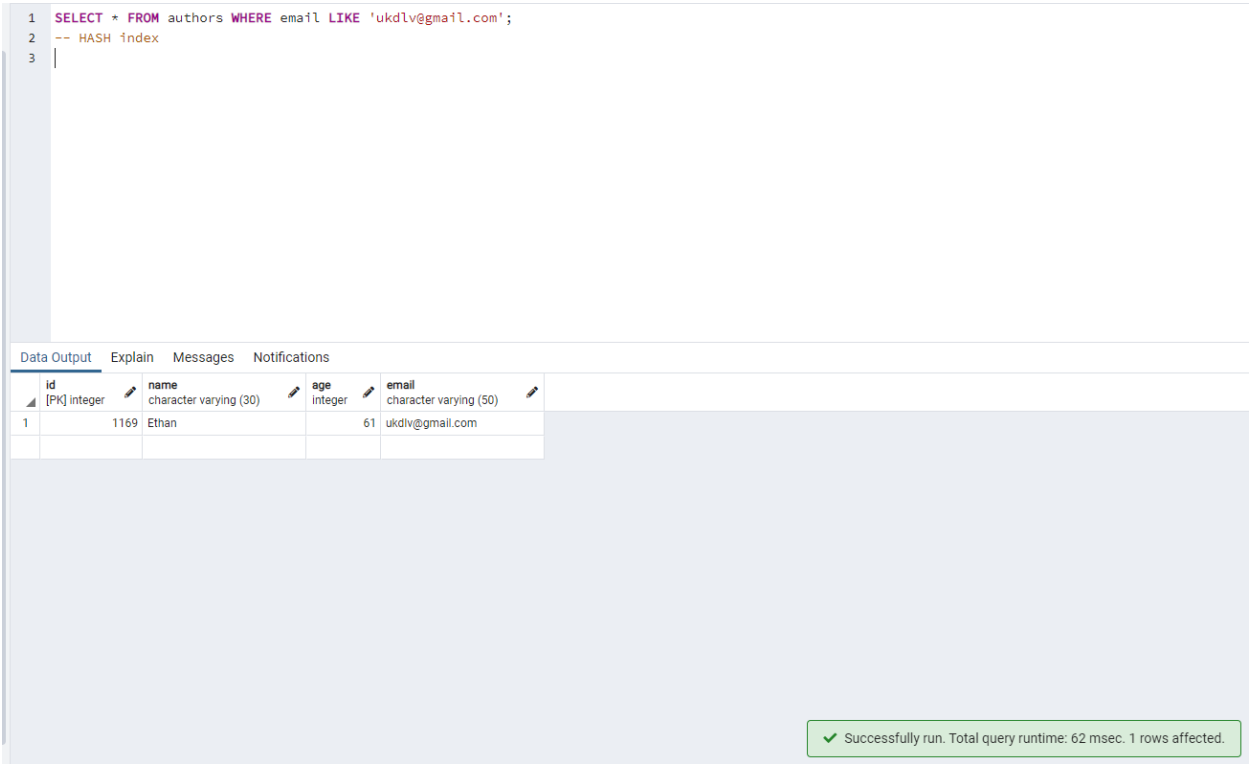
SQL тексти:

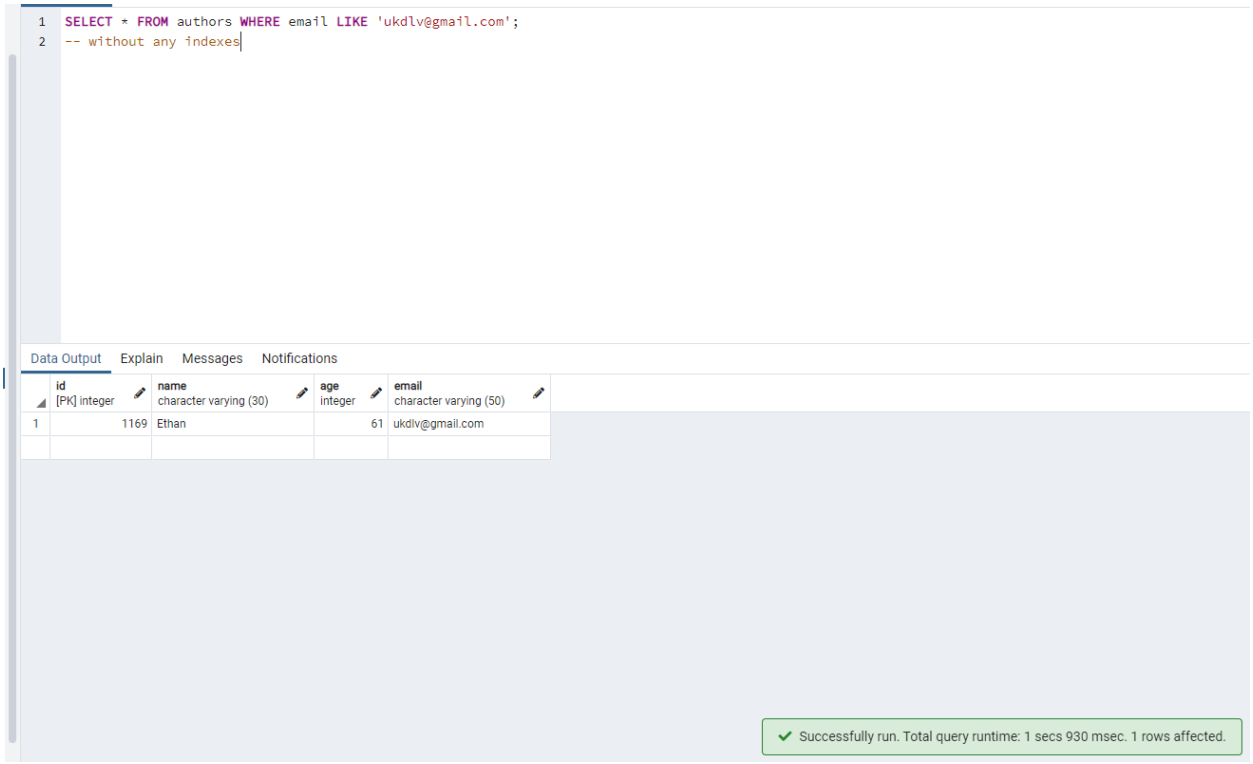
```
CREATE INDEX emailindex
    ON public.authors USING hash
    (email COLLATE pg_catalog."default")
    TABLESPACE pg_default;
```

```
CREATE INDEX nameindex
    ON public.authors USING btree
    (name COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```

Результати, час, аргументування:







Даний запит відбувався при об’ємі записів розміром в 11 мільйонів. Добре видно, який результат ми отримуємо від пошуку використовуючи індекси.

Це пришвидшення відбувається за рахунок того, що ми не проходимося по всім записам, а використовуємо певні алгоритми, які залежать від того, який саме вид індексів ми обрали.

Але також бувають ситуації, коли індекси не прискорюють пошук. Зазвичай таке трапляється, коли дані або однакові, або їх занадто мало, щоб розкрити потенціал роботи з індексами.

Далі для тестування буде взято лише 300.000 записів

Без використання індексів

```
SELECT count(name) FROM authors WHERE name = 'Lucas'
GROUP BY name

-- DROP INDEX nameindex
```

Data Output | Explain | Messages | Notifications

	count bigint
1	100079

Successfully run. Total query runtime: 211 msec. 1 rows affected.

Successfully run. Total query runtime: 131 msec. 1 rows affected.

Successfully run. Total query runtime: 131 msec. 1 rows affected.

З використанням індексу HASH по name

```
SELECT count(name) FROM authors WHERE name = 'Lucas'
GROUP BY name;

-- DROP INDEX nameindex
-- CREATE INDEX nameIndex on authors using hash(name)
```

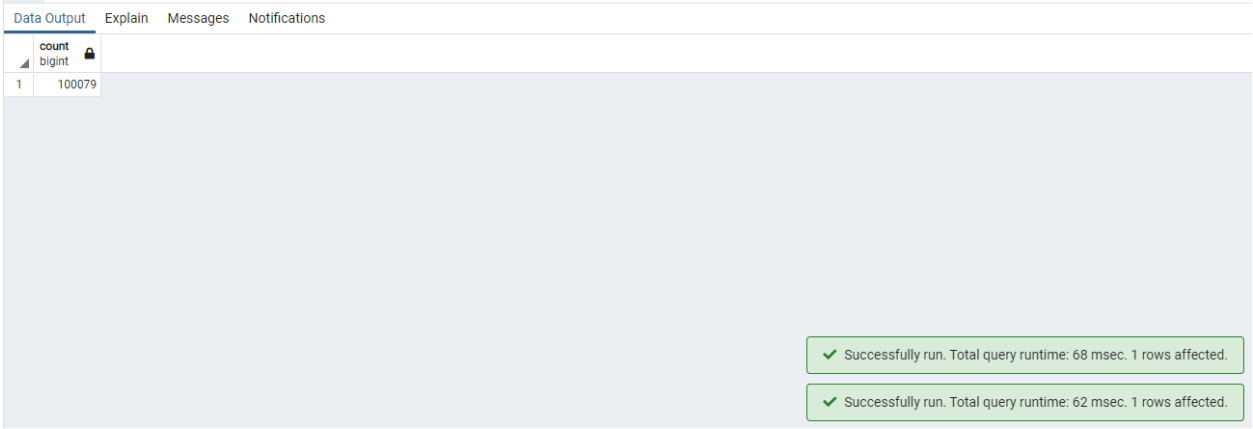
Data Output | Explain | Messages | Notifications

	count bigint
1	100079

Successfully run. Total query runtime: 118 msec. 1 rows affected.

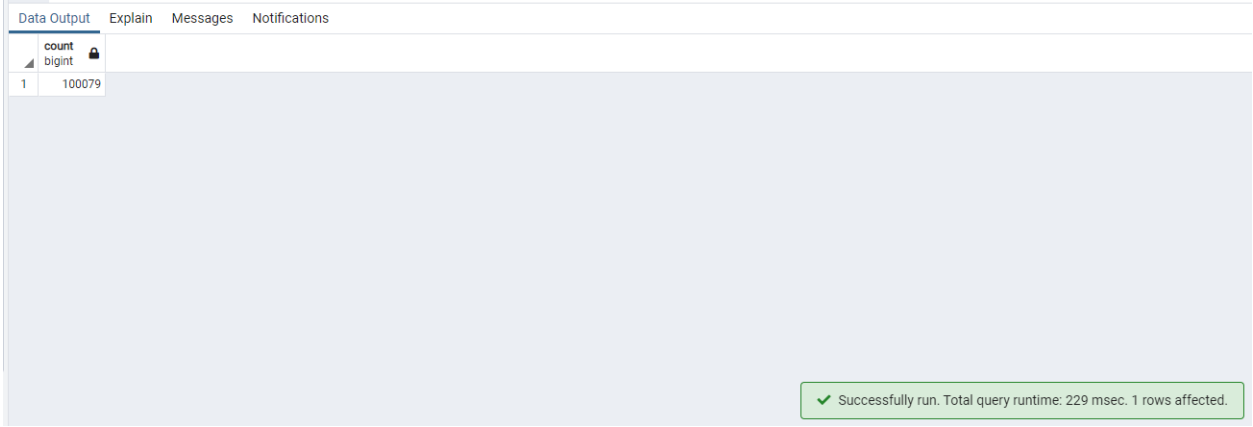
З використанням індексу BTREE по name

```
SELECT count(name) FROM authors WHERE name = 'Lucas'
GROUP BY name;

-- DROP INDEX nameindex;
-- CREATE INDEX nameIndex on authors(name)
```



Також можна помітити, шо при використанні LIKE ‘%VARIABLE%’ наші індекси не працюють. Наприклад:

```
SELECT count(name) FROM authors WHERE name LIKE '%Lucas%'
GROUP BY name;

-- DROP INDEX nameindex;
-- CREATE INDEX nameIndex on authors(name)
```



№3) *(before insert, delete)*

Команди, що ініціюють виконання тригера:

before insert

```
INSERT INTO authors(name, age, email) VALUES ('Lemp', 19, 'floadron');
```

```
40  INSERT INTO authors(name, age, email) VALUES ('Lemp', 19, 'fe');
41
42
```

Data Output | Explain | Messages | Notifications

```
ERROR: ПОМИЛКА:  Email length is too short
CONTEXT:  Функція PL/pgSQL author_trigger_func() рядок 7 в RAISE
```

```
INSERT INTO authors(name, age, email) VALUES ('Gabriel', 32, 'fweefe@gmail.com');
```

delete

```
43  DELETE FROM authors WHERE id = 300007;
44
45
```

Data Output | Explain | Messages | Notifications

```
ПОВІДОМЛЕННЯ:  OLD was: (300007,Gabriel,32,fweefe@gmail.com)
DELETE 1
```

Текст тригера:

before insert & delete

```
CREATE OR REPLACE FUNCTION author_trigger_func()
  RETURNS TRIGGER
  LANGUAGE PLPGSQL
  AS
$$
  DECLARE
    ch VARCHAR;
    cvar INTEGER;
BEGIN
    IF (TG_OP = 'INSERT') THEN
        IF LENGTH(NEW.email) < 5 THEN
            RAISE EXCEPTION 'Email length is too short';
        END IF;

        cvar = 0;
        FOREACH ch IN ARRAY regexp_split_to_array(NEW.email, '')
        LOOP
          IF ch = '@' THEN
            cvar = 1;
          END IF;
        END LOOP;

        IF cvar = 0 THEN
            NEW.email := NEW.email || '@gmail.com';
        END IF;
        RETURN NEW;
    ELSIF (TG_OP = 'DELETE') THEN
        RAISE NOTICE 'OLD was: %', OLD;
        INSERT INTO author_audit SELECT 'D', now(), OLD.id, OLD.name;
        RETURN OLD;
    END IF;
END;
$$
```

```
CREATE TRIGGER author_trigger
  BEFORE INSERT OR DELETE
  ON "authors"
  FOR EACH ROW
  EXECUTE PROCEDURE author_trigger_func();
```

Скріншоти зі змінами у таблицях бази даних:

before insert

	id [PK] integer	name character varying (30)	age integer	email character varying (50)
1	300007	Gabriel	32	fweefe@gmail.com
2	300004	Lemp	19	ewefefel@gmail.com
3	300003	Lemp	19	floadron@gmail.com
4	300002	Kaban	23	nicemail@gmail.com
5	300001	Elon	21	fff@gmail.com
6	300000	Lucas	27	cqvudo@gmail.com

delete

	id [PK] integer	name character varying (30)	age integer	email character varying (50)
1	300003	Lemp	19	floadron@gmail.com
2	300002	Kaban	23	nicemail@gmail.com
3	300001	Elon	21	fff@gmail.com
4	300000	Lucas	27	cqvudo@gmail.com
5	299999	Ethan	34	jzxvbb@gmail.com
6	299998	Ethan	53	arkix@gmail.com

	operation character (1)	stamp timestamp without time zone	authorid integer	authorname text
1	D	2020-11-19 15:24:34.197268	300007	Gabriel

Контрольні запитання:

1. Сформулювати призначення та задачі об’єктно-реляційної проекції (ORM).
2. Проаналізувати основні види індексів у PostgreSQL (*BTree, BRIN, GIN, Hash*): призначення, сфера застосування, переваги та недоліки.
3. Пояснити призначення тригерів та функцій у базах даних.

Відповіді:

1. Призначення ORM полягає в тому, що завдяки йому ми можемо спростити процес збереження об’єктів в реляційну базу даних і їх вилучення. Більшість ORM об’єктів покладаються саме на метадані бази даних і об’єктів, тобто об’єктам нічого не потрібно знати про структуру бази даних, а базі даних нічого про те, як дані організовані в додатку. ORM забезпечує повне розділення завдань в добре

спроетованих додатках, при якому і база даних, і додаток можуть працювати з даними кожен у своїй вихідній формі.

2.

a. GIN – застосовується для обробки випадків, коли елементи, що підлягають індексації, є складеними значеннями, а запити, що обробляються індексом, повинні шукати значення елементів, які відображаються в складених елементах. Однією з переваг GIN є те, що він дозволяє розробляти власні типи даних із відповідними методами доступу. А також кожне значення ключа зберігається лише один раз, тому індекс GIN є дуже компактним для випадків, коли той самий ключ відображається багато разів.
b. BTree – Btree забезпечує ефективний спосіб вставки та читання даних. Дерево надає можливості послідовного пошуку в додаток до двійкового пошуку, що надає базі даних більший контроль для пошуку значень без індексу в базі даних.
c. Hash – перевага полягає в тому, що розмір індексів досить малий. А також в деяких випадках можуть бути навіть швидшими за BTree. А з недостатків є те, що хеш-функція не зберігається відношення порядку: з того, що значення хеш-функції одного ключа менше значення функції іншого ключа, можна зробити ніяких висновків про те, як впорядковані самі ключі. Тому хеш-індекс в принципі може підтримувати єдину операцію «дорівнює»:
d. BRIN - Індекси BRIN ефективні, якщо впорядкування значень ключів слідує за організацією блоків на рівні зберігання. У найпростішому випадку це може вимагати фізичного впорядкування таблиці, яка часто є порядком створення рядків у ній, щоб відповідати порядку ключа. Ключі від сформованих порядкових номерів або створених даних є найкращими кандидатами для індексу BRIN. Оскільки індекс BRIN дуже малий, сканування індексу додає мало накладних витрат порівняно з послідовним скануванням, але може уникнути сканування великих частин таблиці, які, як відомо, не містять відповідних кортежів.

3. Взагалі функції як і в будь-якій мові програмування слугують для того, щоб можна було зробити так звану обгортку для якогось блоку коду і визивати його, коли завгодно. В цей час тригери це просто доповнення до функцій завдяки яким нам не потрібно їх визивати власноруч, напроти, ми можемо прив’язати певну функцію до певної події і цим самим наша функція буде визиватись(виконуватись) автоматично при певних умовах.