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:

студент 3-го курсу, групи КП-82,

спеціальності 121 – Інженерія

програмного забезпечення

Мельничук Олексій Геннадійович

Перевірив:

к. т. н, старший викладач

Радченко Костянтин Олександрович

Київ – 2020

Мета роботи

Здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

Завдання

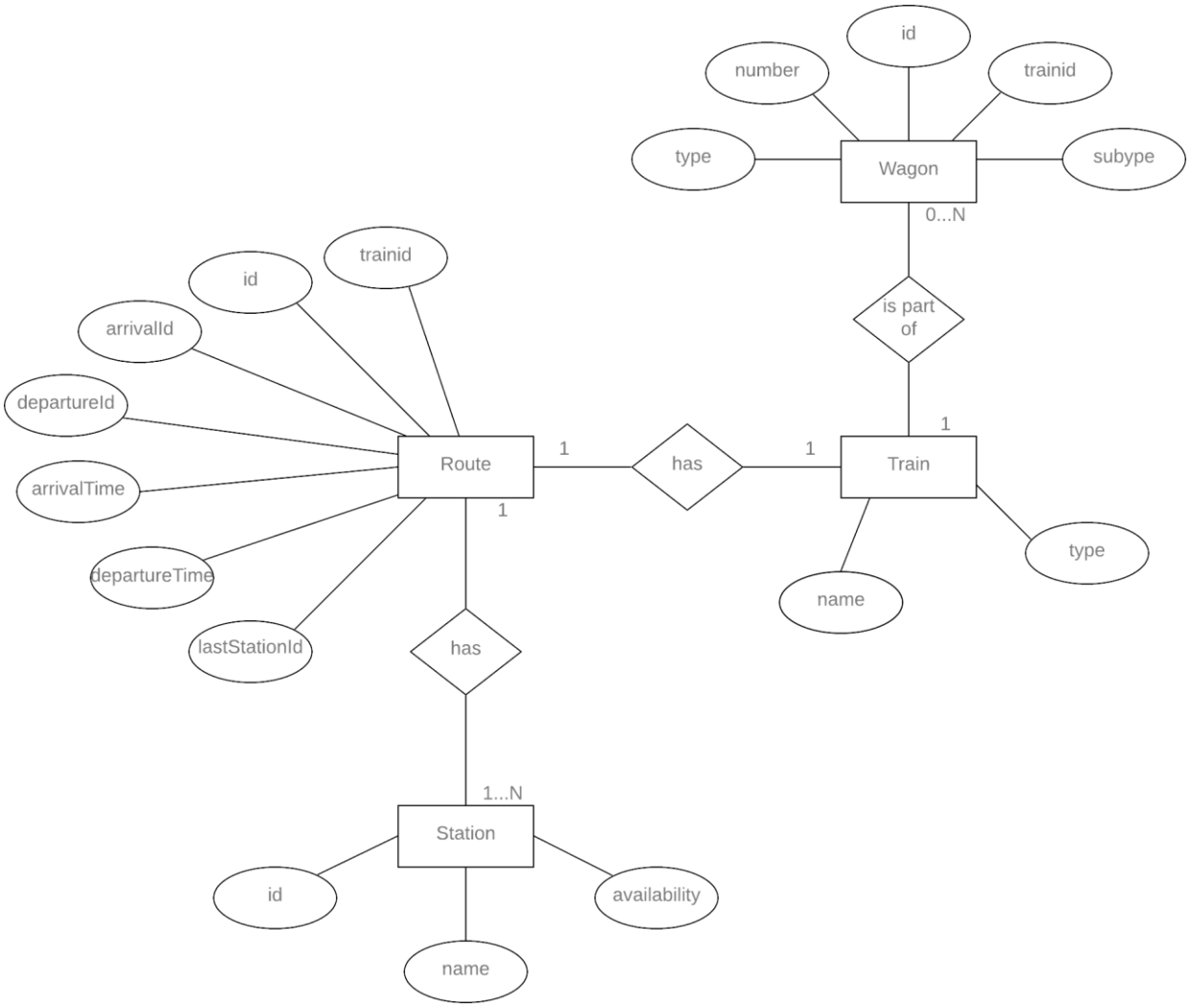
Варіант 12

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.

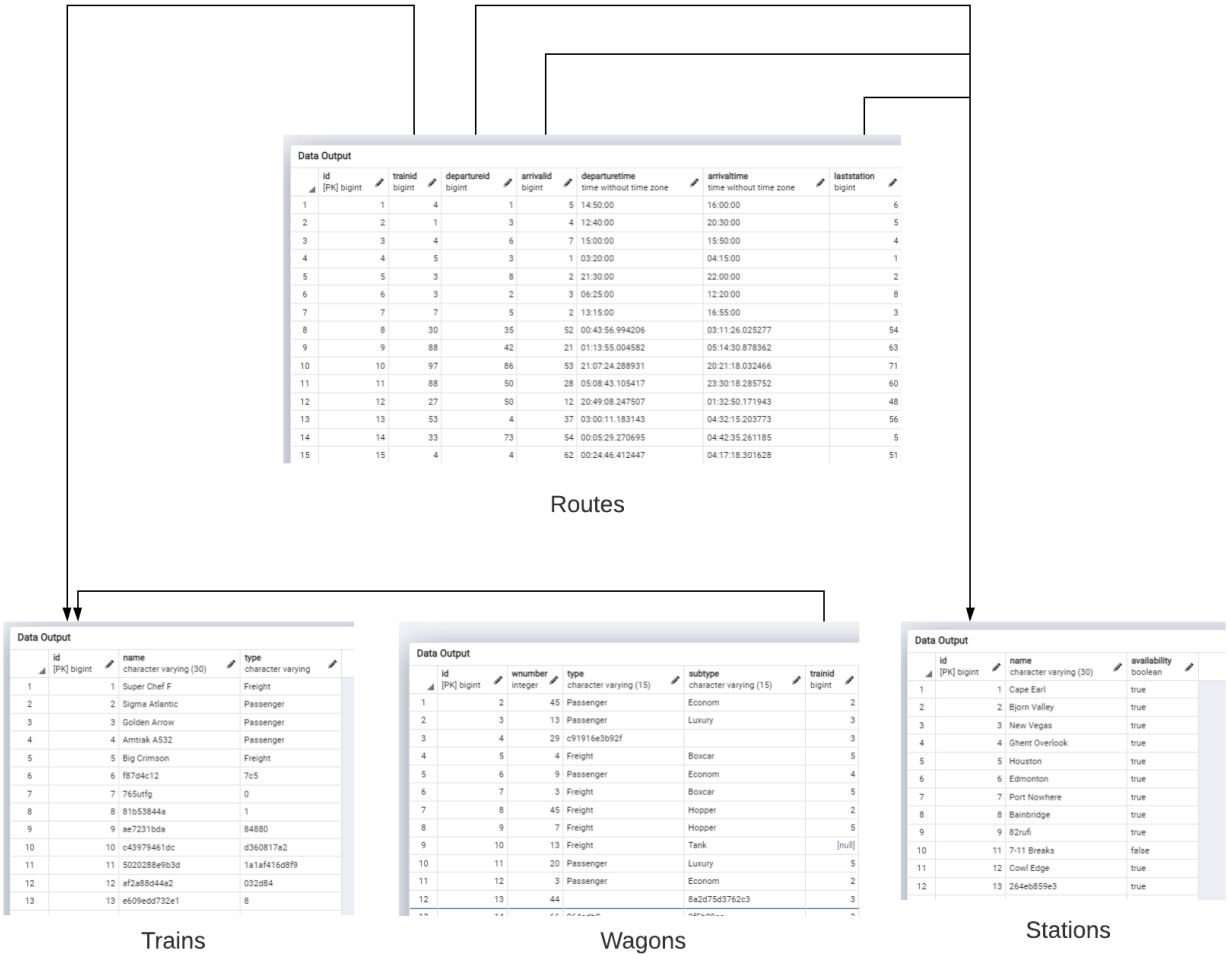
12	*BTree, GIN*	*after update, after insert*

Завдання 1

Графічна модель:



Зв’язки між таблицями:



Data Output

	id [PK] bigint	trainid bigint	departureid bigint	arrivalid bigint	departuretime time without time zone	arrivaltime time without time zone	laststation bigint
1	1	4	1	5	14:50:00	16:00:00	6
2	2	1	3	4	12:40:00	20:30:00	5
3	3	4	6	7	15:00:00	15:50:00	4
4	4	5	3	1	03:20:00	04:15:00	1
5	5	3	8	2	21:30:00	22:00:00	2
6	6	3	2	3	06:25:00	12:20:00	8
7	7	7	5	2	13:15:00	16:55:00	3
8	8	30	35	52	00:43:56.994206	03:11:26.025277	54
9	9	88	42	21	01:13:55.004582	05:14:30.878362	63
10	10	97	86	53	21:07:24.288931	20:21:18.032466	71
11	11	88	50	28	05:08:43.105417	23:30:18.285752	60
12	12	27	50	12	20:49:08.247507	01:32:50.171943	48
13	13	53	4	37	03:00:11.183143	04:32:15.203773	56
14	14	33	73	54	00:05:29.270695	04:42:35.261185	5
15	15	4	4	62	00:24:46.412447	04:17:18.301628	51

Routes

Data Output

	id [PK] bigint	name character varying (30)	type character varying
1	1	Super Chef F	Freight
2	2	Sigma Atlantic	Passenger
3	3	Golden Arrow	Passenger
4	4	Amtrak A532	Passenger
5	5	Big Crimson	Freight
6	6	f87d4c12	7c5
7	7	765utfg	0
8	8	81b53844a	1
9	9	ae7231bda	84880
10	10	c43979461dc	d360817a2
11	11	5020288e9b3d	1a1af416d8f9
12	12	af2a88d44a2	032d84
13	13	e609edd732e1	8

Trains

Data Output

	id [PK] bigint	wnumber integer	type character varying (15)	subtype character varying (15)	trainid bigint
1	2	45	Passenger	Econom	2
2	3	13	Passenger	Luxury	3
3	4	29	c91916e3b92f		3
4	5	4	Freight	Boxcar	5
5	6	9	Passenger	Econom	4
6	7	3	Freight	Boxcar	5
7	8	45	Freight	Hopper	2
8	9	7	Freight	Hopper	5
9	10	13	Freight	Tank	[null]
10	11	20	Passenger	Luxury	5
11	12	3	Passenger	Econom	2
12	13	44		8a2d75d3762c3	3

Wagons

Data Output

	id [PK] bigint	name character varying (30)	availability boolean
1	1	Cape Earl	true
2	2	Bjorn Valley	true
3	3	New Vegas	true
4	4	Ghent Overlook	true
5	5	Houston	true
6	6	Edmonton	true
7	7	Port Nowhere	true
8	8	Bainbridge	true
9	9	82rufi	true
10	11	7-11 Breaks	false
11	12	Cowl Edge	true
12	13	264eb859e3	true

Stations

Класи моделі:

```python
class Route(Base, Repr):
    __tablename__ = 'routes'

    id = Column(Integer, primary_key=True)
    departId = Column(Integer, ForeignKey('station.id'))
    arriveId = Column(Integer, ForeignKey('station.id'))
    lastId = Column(Integer, ForeignKey('station.id'))
    departTime = Column(Date)
    arriveTime = Column(Date)
    trainId = Column(Integer, ForeignKey('train.id'))

    def __init__(self, departId=None, arriveId=None, lastId=None, departTime=None, arriveTime=None, trainId=None ):
        self.departId = departId
        self.arriveId = arriveId
        self.lastId = lastId
        self.departTime = departTime
        self.arriveTime = arriveTime
        self.trainId = trainId
```

```python
class Station(Base, Repr):
    __tablename__ = 'routes'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    availability = Column(Boolean)

    def __init__(self, name=None, availability=None ):
        self.name = name
        self.availability = availability
```

```python
class Train(Base, Repr):
    __tablename__ = 'routes'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    tType = Column(String)

    def __init__(self, name=None, tType=None ):
        self.name = name
        self.tType = tType
```

```python
class Wagon(Base, Repr):
    __tablename__ = 'routes'

    id = Column(Integer, primary_key=True)
    trainId = Column(Integer, ForeignKey('train.id'))
    subType = Column(String)
    number = Column(Integer)

    def __init__(self, trainId=None, subType=None, number=None):
        self.trainId = trainId
        self.subType = subType
        self.number = number
```

Завдання 2: Індекси

Порядок звертання до таблиці без використання фільтру та індексу по колонці з доданим індексом:

Query Editor Query History

```
1  EXPLAIN ANALYZE SELECT * FROM table_trains
```

Data Output Explain Messages Notifications

	QUERY PLAN text
1	Seq Scan on table_trains (cost=0.00..1.05 rows=5 width=118) (actual time=0.015..0.018 rows=35 loops=1)
2	Planning Time: 0.058 ms
3	Execution Time: 0.029 ms

Пошук з фільтрацією(без індексів):

Query Editor Query History

```
1  EXPLAIN ANALYZE SELECT * FROM table_trains where id < 10
```

Data Output Explain Messages Notifications

	QUERY PLAN text
1	Seq Scan on table_trains (cost=0.00..1.06 rows=2 width=118) (actual time=0.018..0.022 rows=9 loops=1)
2	Filter: (id < 10)
3	Rows Removed by Filter: 26
4	Planning Time: 0.089 ms
5	Execution Time: 0.038 ms

Пошук з фільтрацією та сортуванням(без індексів):

```
1  EXPLAIN ANALYZE SELECT * FROM table_trains where id < 10 ORDER BY id
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=1.07..1.08 rows=2 width=118) (actual time=0.044..0.046 rows=9 loops=1)
2	Sort Key: id
3	Sort Method: quicksort Memory: 25kB
4	-> Seq Scan on table_trains (cost=0.00..1.06 rows=2 width=118) (actual time=0.025..0.029 rows=9 loops=1)
5	Filter: (id < 10)
6	Rows Removed by Filter: 26
7	Planning Time: 0.137 ms
8	Execution Time: 0.075 ms

BTree індекс:

Query Editor | Query History

```
1  CREATE INDEX treeIndex ON table_trains USING btree (id)
```

Data Output | Explain | Messages | Notifications

```
CREATE INDEX

Query returned successfully in 197 msec.
```

Порядок звертання до таблиці з використанням фільтру по колонці, на яку додано індекс (пошук відбувається за допомогою створеного індексу):

Query Editor Query History

```
1  EXPLAIN ANALYZE SELECT * FROM table_trains WHERE id<300
```

Data Output Explain Messages Notifications

	QUERY PLAN text
1	Index Scan using treeindex on table_trains (cost=0.28..19.57 rows=299 width=22) (actual time=0.150..0.258 rows=299 loops=1)
2	Index Cond: (id < 300)
3	Planning Time: 0.540 ms
4	Execution Time: 0.307 ms

GIN індекс:

Query Editor Query History

```
1  ALTER TABLE table_trains ADD COLUMN ts_vector tsvector;
2  UPDATE table_trains
3  SET ts_vector = to_tsvector(name)
4  WHERE true;
5
6  CREATE INDEX ginIndex ON table_trains USING GIN (ts_vector);
```

Data Output Explain Messages Notifications

```
CREATE INDEX

Query returned successfully in 156 msec.
```

Порядок звертання до таблиці з використанням фільтру по колонці, на яку додано індекс (пошук відбувається за допомогою створеного індексу):

Query Editor Query History

```
1  EXPLAIN ANALYZE SELECT * FROM table_trains WHERE to_tsquery('a') @@ ts_vector;
```

Data Output Explain Messages Notifications

	QUERY PLAN text
1	Bitmap Heap Scan on table_trains (cost=0.00..4.26 rows=1 width=49) (actual time=0.061..0.061 rows=0 loops=1)
2	Filter: (to_tsquery('a'::text) @@ ts_vector)
3	-> Bitmap Index Scan on ginindex (cost=0.00..0.00 rows=1 width=0) (actual time=0.058..0.059 rows=0 loops=1)
4	Index Cond: (ts_vector @@ to_tsquery('a'::text))
5	Planning Time: 0.337 ms
6	Execution Time: 0.119 ms

Індекси BTree та GIN працюють порівняно повільніше на малому наборі даних ніж пошук без індексів. Зазвичай при пошуку Pgadmin сам використовує найбільш оптимальний алгоритм.

Використання індексів:

- BTree є найпоширенішим і його використовують для більшості типів даних та запитів
- GIN для масивів, hstore та JSONB
- BRIN корисний для великих послідовних наборів даних
- Hash для операторів рівності в спеціальних випадках
- GiST корисні при повторюваних даних в окремих колонках, для геометричних типів та повнотекстовому пошуку
- SP-GiST для несбалансованих структур даних

Завдання 3: Тригери

after UPDATE:
Тригер буде змінювати *trainid* вагона на NULL якщо знайдеться вагон з однаковими значення полів *wnumber* та *trainid*.

Створення:

Query Editor Query History

```
CREATE OR REPLACE FUNCTION afterUpdateWagon()
    RETURNS TRIGGER
    LANGUAGE 'plpgsql'
    AS $$
DECLARE
    ntrainId int;
    wagId int;
    wagnumber int;
BEGIN
    FOR ntrainId, wagId, wagnumber IN
        SELECT trainid, id, wnumber FROM table_wagons
    LOOP
        IF NEW.wnumber = wagnumber AND NEW.trainid = ntrainId THEN
            UPDATE table_wagons
            SET trainid = NULL
            WHERE id = NEW.id AND wnumber = NEW.wnumber;
            EXIT;
        END IF;
    END LOOP;
    RETURN NEW;
END;
$$;

CREATE TRIGGER afterUpdate
AFTER UPDATE
ON table_wagons
FOR EACH ROW
EXECUTE PROCEDURE afterUpdateWagon();
```

Без використання тригера:

```
UPDATE table_wagons
SET trainid = 3
WHERE id = 10 AND wnumber = 13;

SELECT id, wnumber, trainid
FROM table_wagons
WHERE wnumber = 13;
```

Explain **Messages** Notifications

```
Successfully run. Total query runtime: 141 msec.
2 rows affected.
```

Data Output

	id [PK] bigint	wnumber integer	trainid bigint
1	10	13	3
2	3	13	3

З тригером:

```
UPDATE table_wagons
SET trainid = 3
WHERE id = 10 AND wnumber = 13;

SELECT id, wnumber, trainid
FROM table_wagons
WHERE wnumber = 13;
```

Explain **Messages** Notifications

```
Successfully run. Total query runtime: 115 msec.
2 rows affected.
```

Data Output

	id [PK] bigint	wnumber integer	trainid bigint
1	10	13	[null]
2	3	13	3

after INSERT:

Тригер додає значення ID до назви потягу якщо в БД є потяг с такою ж назвою.

Створення:

postgres/postgres@PostgreSQL 13

Query Editor Query History

```
CREATE OR REPLACE FUNCTION afterInsertTrain()
    RETURNS TRIGGER
    LANGUAGE plpgsql
    AS $$
DECLARE
    newname text;
    newname_id int;
BEGIN
    FOR newname, newname_id IN
        SELECT name, id FROM table_trains
    LOOP
        IF NEW.name = newname AND NEW.id != newname_id THEN
            RAISE INFO 'Such train already exists';
            UPDATE table_trains SET name = NEW.name || '-' || NEW.id WHERE id = NEW.id;
            EXIT;
        END IF;
    END LOOP;
    RETURN NEW;
END;
$$;

CREATE TRIGGER afterInsert
AFTER INSERT
ON table_trains
FOR EACH ROW
EXECUTE PROCEDURE afterUpdateTrain();
```

Data Output Explain Messages Notifications

```
CREATE TRIGGER

Query returned successfully in 166 msec.
```

Без використання тригера:

postgres/postgres@PostgreSQL 13

Query Editor Query History

```
25      FOR EACH ROW
26  -- EXECUTE PROCEDURE afterUpdateTrain();
27
28
29  INSERT INTO table_trains (name, type)
30  VALUES ('Golden Arrow', 'Passenger');
31
32  SELECT id, name
33  FROM table_trains
34  WHERE name = 'Golden Arrow'
35
36
37
38
```

Data Output Explain Messages Notifications

	id [PK] bigint	name character varying (30)
1	3	Golden Arrow
2	1037	Golden Arrow

З тригером:

```
29
30  INSERT INTO table_trains (name, type)
31  VALUES ('Golden Arrow', 'Passenger');
32
33  SELECT id, name
34  FROM table_trains
35  WHERE name = 'Golden Arrow'
36
37
38
```

Data Output Explain Messages Notifications

	id [PK] bigint	name character varying (30)
1	3	Golden Arrow

1034	1034	e60868914f	eef82f7fed3
1035	1035	92c84e23a	aee9bd31
1036	1038	Golden Arrow-1038	Passenger

Висновок

Виконавши дану лабораторну роботу були здобуті практичні навички використання засобів оптимізації СУБД PostgreSQL, а саме проектування ORM, використання індексів пошуку та тригерів.