

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИТМО»**

Отчет

По лабораторной работе №2

**«ЗАПРОСЫ НА ВЫБОРКУ И МОДИФИКАЦИЮ ДАННЫХ,
ПРЕДСТАВЛЕНИЯ И ИНДЕКСЫ В POSTGRESQL»**

Вариант 10. БД «Автовокзал»

Автор: Чан Дык Минь

Факультет: ИКТ

Группа: К32392

Преподаватель: Говорова М. М.

Санкт-Петербург, 2023

1 Описание работы

Цель работы: овладеть практическими навыками создания представлений и запросов на выборку данных к базе данных PostgreSQL, использования подзапросов при модификации данных и индексов.

Оборудование: компьютерный класс.

Программное обеспечение: СУБД PostgreSQL , pgAdmin 4.

Практическое задание:

1. Создать запросы и представления на выборку данных к базе данных PostgreSQL (согласно индивидуальному заданию, часть 2 и 3).
2. Составить 3 запроса на модификацию данных (INSERT, UPDATE, DELETE) с использованием подзапросов.
3. Изучить графическое представление запросов и просмотреть историю запросов.
4. Создать простой и составной индексы для двух произвольных запросов и сравнить время выполнения запросов без индексов и с индексами. Для получения плана запроса использовать команду EXPLAIN.

2 Описание предметной области

Вариант 10. БД «Автовокзал»

Описание предметной области: С автовокзала ежедневно отправляется несколько междугородных/международных автобусных рейсов. Номер рейса определяется маршрутом и временем отправления. По всем промежуточным остановкам на маршруте известны название, тип населенного пункта, время прибытия, отправления, время стоянки. Автобусы курсируют по расписанию, но могут назначаться дополнительные рейсы на заданный период или определенные даты. Билеты могут продаваться предварительно, но не ранее чем за 10 суток. В билете указывается номер места в автобусе. На каждый рейс может продаваться не более 10 билетов без места, цена на которые снижается на 10%. Пунктами отправления и назначения, согласно билету, могут быть промежуточные остановки. Билеты могут продаваться в кассе автовокзала или онлайн. На каждый рейс формируется экипаж из двух водителей. БД должна содержать следующий минимальный набор сведений: Номер рейса. Номер водителя. Номер автобуса. Паспортные данные водителя. Пункт отправления. Пункт назначения. Промежуточные остановки. Дата отправления. Время отправления. Время в пути. Тип автобуса. Количество мест в автобусе. Страна. Производитель. Год выпуска. Номер билета. Номер места в автобусе (при наличии). Цена билета. ФИО пассажира. Паспортные данные пассажира.

3 Выполнение работы

Схема базы данных:

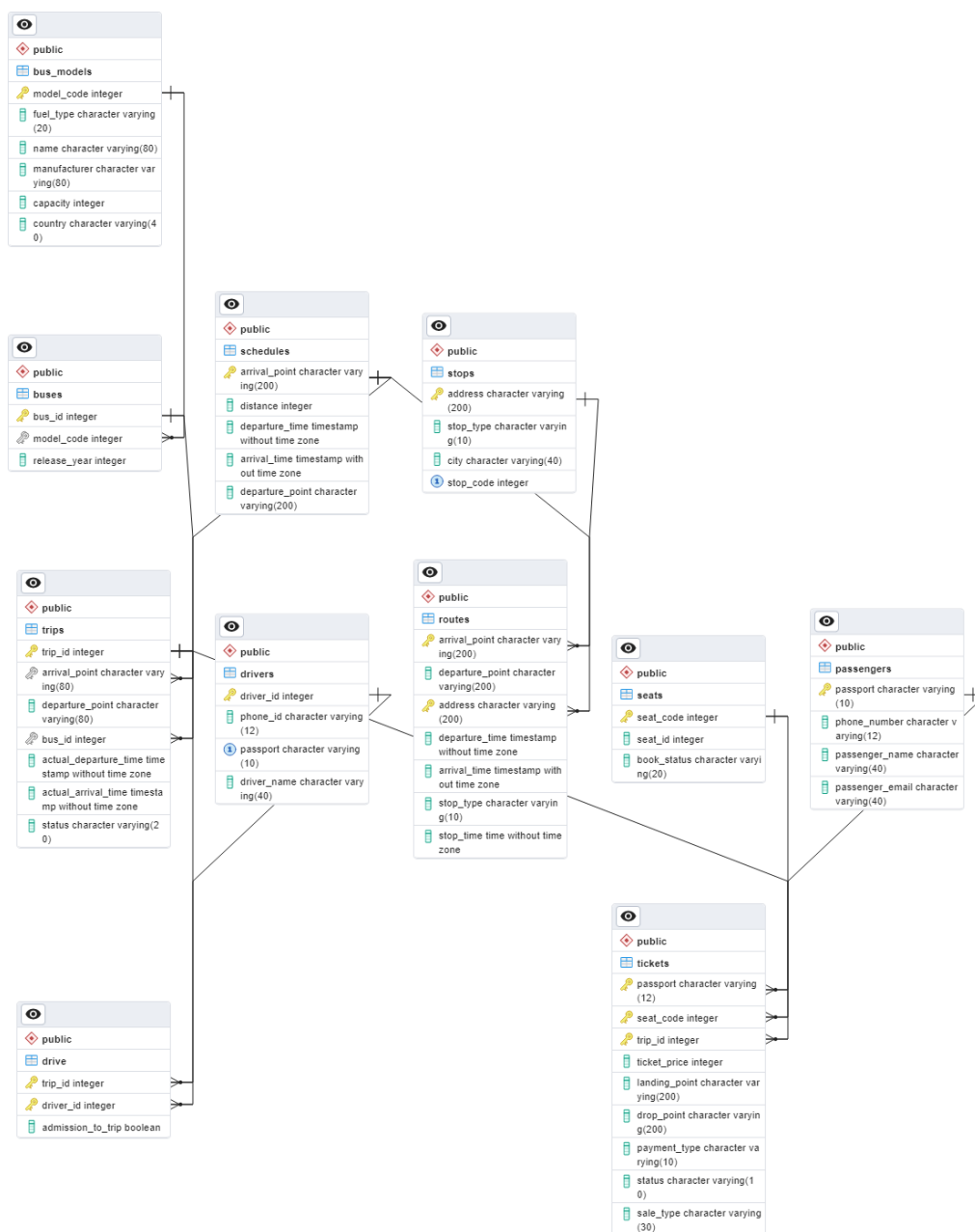


Рис. 1 – ERD диаграмм

Запросы к базе данных:

1. Вывести фамилии водителей и номера автобусов, отправившиеся в рейсы до 12 часов текущего дня.

```
SELECT d.driver_name,  
       b.bus_id  
FROM drivers d  
JOIN drive dr ON d.driver_id = dr.driver_id  
JOIN trips t ON dr.trip_id = t.trip_id  
JOIN buses b ON t.bus_id = b.bus_id  
WHERE t.actual_departure_time < CURRENT_DATE + INTERVAL '12 hours';
```

	driver_name character varying (40)	bus_id integer
1	Driver1	1

2. Рассчитать выручку от продажи билетов за прошедший день.

```
SELECT SUM(ticket_price) AS revenue  
FROM tickets  
JOIN trips ON tickets.trip_id = trips.trip_id  
WHERE DATE_PART('day', NOW() - trips.actual_departure_time) = 0;
```

	revenue bigint
1	25

3. Вывести список водителей, которые не выполнили ни одного рейса за прошедший день.

```
SELECT d.driver_name  
FROM drivers d  
LEFT JOIN drive dr ON d.driver_id = dr.driver_id  
LEFT JOIN trips t ON dr.trip_id = t.trip_id  
WHERE DATE_PART('day', NOW() - t.actual_departure_time) != 1  
      OR t.trip_id IS NULL;
```

	driver_name character varying (40) 🔒
1	Driver1
2	Driver2
3	Driver3

4. Вывести сумму убытков из-за непроданных мест в автобусе за прошедшую неделю.

```
SELECT SUM(t.ticket_price) AS loss
FROM seats s
JOIN tickets t ON s.seat_code = t.seat_code
JOIN trips ON t.trip_id = trips.trip_id
WHERE s.book_status = 'canceled'
AND DATE_PART('day', NOW() - trips.actual_departure_time) < 7;
```

	losses bigint 🔒
1	0

5. Сколько рейсов выполнил каждый водитель за последний месяц.

```
SELECT d.driver_name,
       COUNT(t.trip_id) AS trip_count
FROM drivers d
LEFT JOIN drive dr ON d.driver_id = dr.driver_id
LEFT JOIN trips t ON dr.trip_id = t.trip_id
WHERE DATE_PART('month', NOW() - t.actual_departure_time) = 1
GROUP BY d.driver_name;
```

	driver_name character varying (40) 🔒	trip_count bigint 🔒
1	Driver1	1
2	Driver2	1

6. Вывести тип автобуса, который используется на всех рейсах.

```
SELECT m.name
FROM trips t
JOIN buses b ON t.bus_id = b.bus_id
JOIN bus_models m ON b.model_code = m.model_code
```

```
GROUP BY m.name
HAVING COUNT(DISTINCT t.trip_id) = COUNT(*);
```

	name character varying (80) 🔒
1	BusModel1
2	BusModel2

7. Вывести данные водителя, который провел максимальное время в пути за прошедшую неделю.

```
SELECT d.driver_name,
       d.phone_id,
       d.passport,
       SUM(EXTRACT(EPOCH
                   FROM (t.actual_arrival_time -
                        t.actual_departure_time))) AS total_time
FROM drivers d
JOIN drive dr ON d.driver_id = dr.driver_id
JOIN trips t ON dr.trip_id = t.trip_id
WHERE DATE_PART('week', NOW() - t.actual_departure_time) = 1
GROUP BY d.driver_name,
         d.phone_id,
         d.passport
ORDER BY total_time DESC
LIMIT 1;
```

	driver_name character varying (40) 🔒	phone_id character varying (12) 🔒	passport character varying (10) 🔒	total_time double precision 🔒
1	Driver2	2222222222	A2222222	14400

Представления

Создать представление для пассажиров:

1. количество свободных мест на все рейсы на завтра;

```
CREATE OR REPLACE VIEW available_seats AS
SELECT t.trip_id, b.model_code, COUNT(s.seat_id) AS free_seats
FROM trips t
INNER JOIN buses b ON t.bus_id = b.bus_id
INNER JOIN seats s ON b.bus_id = s.seat_code
LEFT JOIN tickets ti ON ti.trip_id = t.trip_id AND ti.seat_code
= s.seat_id AND ti.status != 'empty'
WHERE t.actual_departure_time >= CURRENT_DATE + INTERVAL '1
day'
AND t.actual_departure_time < CURRENT_DATE + INTERVAL '2 day'
GROUP BY t.trip_id, b.model_code;
```

```
SELECT * FROM available_seats;
```

	trip_id integer	model_code integer	free_seats bigint
1	4	1	1

2. самый популярный маршрут этой зимой.

```
CREATE OR REPLACE VIEW popular_routes AS
SELECT r.departure_point,
       r.arrival_point,
       COUNT(t.trip_id) AS num_trips
FROM routes r
INNER JOIN trips t ON r.departure_point = t.departure_point
AND r.arrival_point = t.arrival_point
WHERE t.actual_departure_time >= '2023-12-01'
AND t.actual_departure_time < '2024-03-01'
GROUP BY r.departure_point,
         r.arrival_point
ORDER BY num_trips DESC
LIMIT 1;
```



```
SELECT * FROM popular_routes;
```

	departure_point character varying (200)	arrival_point character varying (200)	num_trips bigint
1	Address3	Address2	1

INSERT

Запрос добавляет новую запись в таблицу "schedules". Он выбирает данные из таблицы "schedules", где "arrival_point" равен 'Address3' и "distance" равен 100, и присваивает новые значения полям "departure_point", "distance", "departure_time", "arrival_time" и "arrival_point". В частности, поле "departure_point" принимает значение поля "arrival_point" из исходной записи, а поле "arrival_point" принимает значение 'NewAddress'. Поле "distance" увеличивается на 50, а поля "departure_time" и "arrival_time" устанавливаются на 1 и 3 часа 30 минут позже времени прибытия в исходной записи соответственно.

```
INSERT INTO schedules (departure_point, distance,  
departure_time, arrival_time, arrival_point)
```

```
SELECT  
    arrival_point,  
    distance + 50,  
    arrival_time + interval '1 hour',  
    arrival_time + interval '3 hour 30 minutes',  
    'NewAddress'
```

```
FROM  
    schedules
```

```
WHERE  
    arrival_point = 'Address3'  
    AND distance = 100;
```

до:

	arrival_point [PK] character varying (200)	distance integer	departure_time timestamp without time zone	arrival_time timestamp without time zone	departure_point character varying (200)
1	Address1	100	2023-05-03 08:00:00	2023-05-03 10:00:00	Address2
2	Address2	200	2023-05-03 12:00:00	2023-05-03 16:00:00	Address3
3	Address3	100	2023-05-04 08:00:00	2023-05-04 10:15:00	Address3

после:

	arrival_point [PK] character varying (200)	distance integer	departure_time timestamp without time zone	arrival_time timestamp without time zone	departure_point character varying (200)
1	Address1	100	2023-05-03 08:00:00	2023-05-03 10:00:00	Address2
2	Address2	200	2023-05-03 12:00:00	2023-05-03 16:00:00	Address3
3	Address3	100	2023-05-04 08:00:00	2023-05-04 10:15:00	Address3
4	NewAddress	150	2023-05-04 11:15:00	2023-05-04 13:45:00	Address3

UPDATE:

Запрос обновляет номер телефона пассажира с именем "Passenger1".

Подзапрос находит номер телефона водителя с именем "Driver1" из таблицы "drivers". Затем оператор SET присваивает этот номер телефона полю "phone_number" в таблице "passengers". Оператор WHERE фильтрует только записи для пассажира с именем "Passenger1".

до:

```
UPDATE passengers
SET phone_number =
    (SELECT phone_id
     FROM drivers
     WHERE driver_name = 'Driver1' )
WHERE passenger_name = 'Passenger1';
```

	passport [PK] character varying (10)	phone_number character varying (12)	passenger_name character varying (40)	passenger_email character varying (40)
1	P1111111	4444444444	Passenger1	passenger1@example.com
2	P2222222	5555555555	Passenger2	passenger2@example.com

после:

	passport [PK] character varying (10)	phone_number character varying (12)	passenger_name character varying (40)	passenger_email character varying (40)
1	P2222222	5555555555	Passenger2	passenger2@example.com
2	P1111111	1111111111	Passenger1	passenger1@example.com

DELETE:

Запрос удаляет все билеты из таблицы "tickets", связанные с пассажиром, имя которого равно 'John Doe'. Подзапрос находит все паспортные данные для пассажира с именем 'John Doe'. Оператор IN в основном запросе выбирает все билеты, связанные с этими паспортными данными.

```
DELETE
FROM tickets
WHERE passport IN
    (SELECT passport
     FROM passengers
     WHERE passenger_name = 'Passenger1' )
```

до:

	passport [PK] character varying (12) ↗	seat_code [PK] integer ↗	trip_id [PK] integer ↗	ticket_price integer ↗	landing_point character varying (200) ↗	drop_point character varying (200) ↗	payment_type character varying (10) ↗	status character varying (10) ↗	sale_type character varying (30) ↗
1	P1111111	1	1	10	Address1	Address2	card	payed	sell directly
2	P2222222	2	1	15	Address1	Address2	cash	payed	sell directly

после:

	passport [PK] character varying (12) ↗	seat_code [PK] integer ↗	trip_id [PK] integer ↗	ticket_price integer ↗	landing_point character varying (200) ↗	drop_point character varying (200) ↗	payment_type character varying (10) ↗	status character varying (10) ↗	sale_type character varying (30) ↗
1	P2222222	2	1	15	Address1	Address2	cash	payed	sell directly

Создание индексов

Без индексов:

```
EXPLAIN ANALYZE
SELECT d.driver_name,
       b.bus_id
FROM drivers d
JOIN drive dr ON d.driver_id = dr.driver_id
JOIN trips t ON dr.trip_id = t.trip_id
JOIN buses b ON t.bus_id = b.bus_id
WHERE t.actual_departure_time < '2023-05-03 12:00:00';
```

	QUERY PLAN	
	text	
1	Hash Join (cost=70.72..113.48 rows=738 width=102) (actual time=0.061..0.062 rows=1 loops=1)	
2	Hash Cond: (dr.driver_id = d.driver_id)	
3	-> Hash Join (cost=52.17..92.99 rows=738 width=8) (actual time=0.035..0.036 rows=1 loops=1)	
4	Hash Cond: (dr.trip_id = t.trip_id)	
5	-> Seq Scan on drive dr (cost=0.00..32.00 rows=2200 width=8) (actual time=0.005..0.005 rows=2 loops=1)	
6	-> Hash (cost=51.46..51.46 rows=57 width=8) (actual time=0.022..0.023 rows=1 loops=1)	
7	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
8	-> Hash Join (cost=12.84..51.46 rows=57 width=8) (actual time=0.020..0.021 rows=1 loops=1)	
9	Hash Cond: (b.bus_id = t.bus_id)	
10	-> Seq Scan on buses b (cost=0.00..30.40 rows=2040 width=4) (actual time=0.005..0.005 rows=3 loops=1)	
11	-> Hash (cost=12.13..12.13 rows=57 width=8) (actual time=0.007..0.007 rows=1 loops=1)	
12	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
13	-> Seq Scan on trips t (cost=0.00..12.13 rows=57 width=8) (actual time=0.005..0.005 rows=1 loops=1)	
14	Filter: (actual_departure_time < '2023-05-03 12:00:00':timestamp without time zone)	
15	Rows Removed by Filter: 1	
16	-> Hash (cost=13.80..13.80 rows=380 width=102) (actual time=0.017..0.017 rows=3 loops=1)	
17	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
18	-> Seq Scan on drivers d (cost=0.00..13.80 rows=380 width=102) (actual time=0.010..0.011 rows=3 loops=1)	
19	Planning Time: 0.238 ms	
20	Execution Time: 0.096 ms	

Создание индексов:

```
CREATE INDEX driver_index ON drivers(driver_id);
CREATE INDEX trip_index ON trips(trip_id);
CREATE INDEX bus_index ON buses(bus_id);
```

С индексами:

	QUERY PLAN text	
1	Hash Join (cost=1.22..14.23 rows=1100 width=102) (actual time=0.037..0.038 rows=1 loops=1)	
2	Hash Cond: (dr.driver_id = d.driver_id)	
3	-> Nested Loop (cost=0.15..10.27 rows=1100 width=8) (actual time=0.023..0.024 rows=1 loops=1)	
4	-> Nested Loop (cost=0.00..2.09 rows=1 width=8) (actual time=0.017..0.017 rows=1 loops=1)	
5	Join Filter: (t.bus_id = b.bus_id)	
6	Rows Removed by Join Filter: 1	
7	-> Seq Scan on trips t (cost=0.00..1.02 rows=1 width=8) (actual time=0.010..0.011 rows=1 loops=1)	
8	Filter: (actual_departure_time < '2023-05-03 12:00:00':timestamp without time zone)	
9	Rows Removed by Filter: 1	
10	-> Seq Scan on buses b (cost=0.00..1.03 rows=3 width=4) (actual time=0.004..0.005 rows=2 loops=1)	
11	-> Index Scan using drive_trip_id_key on drive dr (cost=0.15..8.17 rows=1 width=8) (actual time=0.005..0.005 rows=1 loops=1)	
12	Index Cond: (trip_id = t.trip_id)	
13	-> Hash (cost=1.03..1.03 rows=3 width=102) (actual time=0.009..0.009 rows=3 loops=1)	
14	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
15	-> Seq Scan on drivers d (cost=0.00..1.03 rows=3 width=102) (actual time=0.005..0.005 rows=3 loops=1)	
16	Planning Time: 2.812 ms	
17	Execution Time: 0.066 ms	

Вот некоторые из преимуществ, получаемых после создания индекса для запроса:

- Увеличьте скорость запроса: индексы помогают базе данных быстрее находить данные за счет уменьшения объема данных, которые необходимо искать и извлекать.
- Оптимизация производительности: индексы помогают снизить нагрузку на сервер базы данных и сократить время выполнения запросов, помогая оптимизировать производительность системы.
- Повышенная точность: индексы помогают гарантировать, что запросы возвращают точные и полные данные.
- Улучшенная масштабируемость: индексы помогают улучшить масштабируемость базы данных за счет снижения нагрузки на сервер базы данных по мере увеличения размера базы данных.

4 Вывод

Во время лабораторной работы я научился создавать пользовательские представления, использовать запросы для извлечения данных. В базе данных PostgreSQL использовать подзапросы при изменении данных, а также узнать о преимуществах использования индексов.

На мой взгляд, эти знания очень полезны и часто применяются на практике.