

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное  
учреждение высшего образования  
«Самарский национальный исследовательский университет  
имени академика С. П. Королева»

Институт информатики и кибернетики  
Кафедра технической кибернетики

Лабораторная работа №1  
по курсу «Корпоративные базы данных»

Выполнил студент  
группы 6133-010402D  
Мишагина В.Ю.  
Преподаватель:  
Минаев Е.Ю.

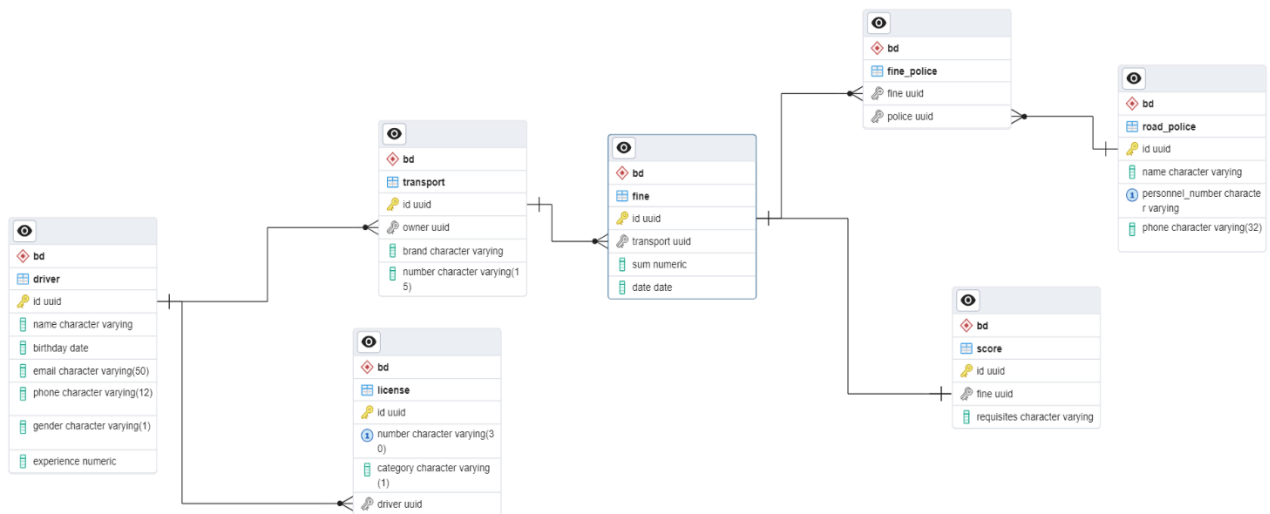
Самара 2024

## **Задание на лабораторную работу №1**

1. Выбрать предметную область
  2. Разработать ER модель, включающую минимум 5-6 сущностей и типы связей: 1-N, N:M, 1-1.
  3. Создать базу данных по модели в СУБД PostgreSQL.
  4. Определить индексы, уникальные индексы.
  5. Разработать типовые запросы к СУБД на языке SQL. Получение списков данных. Агрегация. Поиск.
  6. Разработайте хранимые процедуры на языке PL/pgSQL для генерации случайных данных для базы данных.
  7. Сгенерируйте тестовые данные при помощи разработанных процедур.
  8. Протестируйте работу запросов на больших объёмах данных (Порядка 1 миллиона записей в основных таблицах).
  9. Измените конфигурацию сервера PostgreSQL для достижения лучшей производительности на самых медленных запросах. Оптимизируйте схему БД и запросы для достижения лучшей производительности.
- Пункты 6-7 допустимо реализовывать другими способами без PL/pgSQL

## Ход работы

В качестве предметной области была выбрана система регистрации уплаты штрафов за нарушение ПДД. ER-модель приведена на рисунке.



В качестве сущностей выбраны: сотрудники ГАИ, зарегистрировавшие нарушения (road\_police), водитель (driver), транспортное средство (transport), лицензия на вождение (license), штраф (fine), счет по погашению штрафа (score). Для данной модели представлены виды связи: один ко многим (например, один водитель может управлять разными транспортными средствами или иметь разрешение на вождение разных категорий), многие ко многим (реализована через вспомогательную таблицу) (одно и то же нарушение может быть зарегистрировано несколькими сотрудниками, также как и сотрудник ГАИ может регистрировать множество нарушений) и один к одному (один штраф может быть уплачен только 1 раз, так же как и в 1 счете можно уплатить только за 1 штраф).

Для создания базы данных по модели использовались следующие скрипты.

```
CREATE TABLE bd.road_police (
    id uuid NULL,
    "name" varchar NOT NULL,
    personnel_number varchar NOT NULL,
    phone character varying(32) NULL,
    CONSTRAINT road_police_pk PRIMARY KEY (id),
    CONSTRAINT road_police_un UNIQUE (personnel_number)
);
CREATE UNIQUE INDEX road_police_pk ON bd.road_police USING btree (id);
CREATE UNIQUE INDEX road_police_un ON bd.road_police USING btree (personnel_number);
```

```

CREATE TABLE bd.driver (
    id uuid NOT NULL,
    "name" varchar NOT NULL,
    birthday date NOT NULL,
    email varchar(50) NOT NULL,
    phone varchar(32) NOT NULL,
    gender character varying(1) NOT NULL,
    experience numeric NOT NULL,
    CONSTRAINT driver_pk PRIMARY KEY (id)
);
CREATE INDEX driver_gender_idx ON bd.driver USING btree (gender);
CREATE UNIQUE INDEX driver_id_idx ON bd.driver USING btree (id);
CREATE INDEX driver_phone_idx ON bd.driver USING btree (phone);

CREATE TABLE bd.transport (
    id uuid NOT NULL,
    "owner" uuid NOT NULL,
    brand varchar(255) NOT NULL,
    "number" varchar(15) NOT NULL,
    CONSTRAINT transport_pk PRIMARY KEY (id),
    CONSTRAINT transport_fk FOREIGN KEY ("owner") REFERENCES bd.driver(id)
ON DELETE CASCADE
);
CREATE INDEX transport_brand_idx ON bd.transport USING btree (brand);
CREATE INDEX transport_number_idx ON bd.transport USING btree (number);
CREATE INDEX transport_owner_idx ON bd.transport USING btree (owner);
CREATE UNIQUE INDEX transport_id_idx ON bd.transport USING btree (id);

CREATE TABLE bd.fine (
    id uuid NOT NULL,
    transport uuid NOT NULL,
    sum numeric NOT NULL,
    "date" date NOT NULL,
    score uuid NULL,
    CONSTRAINT fine_pk PRIMARY KEY (id),
    CONSTRAINT fine_un UNIQUE (score),
    CONSTRAINT fine_fk FOREIGN KEY (score) REFERENCES bd.score(id) ON DELETE
CASCADE ON UPDATE CASCADE,
    CONSTRAINT fine_fk_transport FOREIGN KEY (transport) REFERENCES
bd.transport(id) ON DELETE CASCADE
);
CREATE INDEX fine_date_idx ON bd.fine USING btree (date);
CREATE UNIQUE INDEX fine_id_idx ON bd.fine USING btree (id);
CREATE INDEX fine_transport_id_idx ON bd.fine USING btree (transport);
CREATE UNIQUE INDEX fine_un ON bd.fine USING btree (score);

CREATE TABLE bd.score (
    id uuid NOT NULL,
    fine uuid NOT NULL,
    requisites varchar NOT NULL,
    CONSTRAINT score_pk PRIMARY KEY (id),
    CONSTRAINT score_un UNIQUE (fine),
    CONSTRAINT score_fk FOREIGN KEY (fine) REFERENCES bd.fine(id) ON DELETE
CASCADE
);
CREATE UNIQUE INDEX score_fine_idx ON bd.score USING btree (fine);
CREATE UNIQUE INDEX score_id_idx ON bd.score USING btree (id);

CREATE TABLE bd.license (

```

```

        id uuid NOT NULL,
        "number" varchar NOT NULL,
        category varchar(1) NOT NULL,
        driver uuid NOT NULL,
        CONSTRAINT license_pk PRIMARY KEY (id),
        CONSTRAINT license_un UNIQUE (number),
        CONSTRAINT license_fk FOREIGN KEY (driver) REFERENCES bd.driver(id)
    );
CREATE INDEX license_category_idx ON bd.license USING btree (category);
CREATE UNIQUE INDEX license_pk ON bd.license USING btree (id);
CREATE UNIQUE INDEX license_un ON bd.license USING btree (number);
CREATE INDEX license_driver_idx ON bd.license USING btree (driver);

CREATE TABLE bd.fine_police (
    fine uuid NOT NULL,
    police uuid NOT NULL,
    CONSTRAINT fine_police_fk FOREIGN KEY (fine) REFERENCES bd.fine(id) ON
DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fine_police_fk_1 FOREIGN KEY (police) REFERENCES
bd.road_police(id) ON DELETE CASCADE ON UPDATE CASCADE
);
CREATE INDEX fine_police_fine_idx ON bd.fine_police USING btree (fine);
CREATE INDEX fine_police_police_idx ON bd.fine_police USING btree (police);

```

Также в скриптах были созданы индексы по типу бинарного дерева для полей, по которым, предположительно, будет проводиться поиск (для его ускорения).

Для заполнения этих таблиц данными был написан код на python, где с помощью библиотек random, faker и mimesis были искусственно сгенерированы данные (файл прилагается). В основные таблицы (fine, score, fine\_police) было сгенерировано более 1млн записей. В программе выполнялась генерация данных и их запись в csv файл, далее с помощью процедуры данные из файла были импортированы в соответствующие таблицы базы данных.

Пример генерации для таблицы driver:

```

import faker
from datetime import datetime
from mimesis import Generic
import uuid
import random
import csv

# Создаем объект Generic для генерации данных
data = Generic('ru')

# Создание объекта Faker для конкретной локализации

```

```

fake = faker.Faker("ru_RU")
driver_file = open("driver.csv", "w+", newline='')

drivers = []
with driver_file:
    writer = csv.writer(driver_file)
    first = []
    first.append("id")
    first.append("name")
    first.append("birthday")
    first.append("email")
    first.append("phone")
    first.append("gender")
    first.append("experience")
    writer.writerow(first)
    for _ in range(200000):
        row = []
        id = str(uuid.uuid4())
        row.append(id)
        gender = data.person.gender()
        if (gender == 'Муж.'):
            row.append(fake.first_name_male() + ' ' + fake.last_name_male())
        else:
            row.append(fake.first_name_female() + ' ' +
fake.last_name_female())
        birthday = data.person.birthdate(min_year=1960, max_year=2005)
        row.append(str(birthday))
        row.append(data.person.email())
        row.append(data.person.phone_number())
        row.append(gender)
        now_date = datetime.now()
        max_exp = now_date.year - birthday.year - 18
        row.append(random.randint(0, max_exp))
        writer.writerow(row)
        drivers.append(id)
driver_file.close()

```

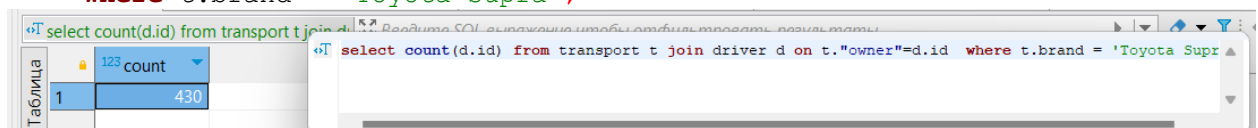
Далее после заполнения таблиц были разработаны типовые запросы для поиска и агрегации данных.

### 1. Подсчет количества водителей, которые управляют машиной марки Toyota Supra

```

select count(d.id) from transport t join driver d on t."owner"=d.id
where t.brand = 'Toyota Supra';

```



The screenshot shows a database query interface. The SQL query is: `select count(d.id) from transport t join driver d on t."owner"=d.id where t.brand = 'Toyota Supra';`. The results are displayed in a table with two columns: 'count' and '1'. The value in the 'count' column is 430.

count
430

## 2. Поиск сотрудника ГАИ, выписавшего штрафы на большую сумму начиная от 1 марта

```
select rp."name", rp.personnel_number
from road_police rp
where rp.id in (
    select id_police
    from (
        select fp.police as "id_police", sum(f.sum) as "sum_fine"
        from fine_police fp join fine f on fp.fine = f.id
        where f."date" > '2024-03-01'
        group by fp.police
        order by sum_fine
        desc limit 1
    )
)
```

Введите SQL выражение чтобы отфильтровать

Таблица	name	personnel_number
1	Пейтон Вавилов	84137028-0560/81

## 3. Вывод статистики подсчета количества штрафов для мужчин и женщин

```
select gender, sum(count_fines)
from (
    select d.id, d.gender as "gender", count(f.id) as "count_fines"
    from fine f full join transport t on f.transport = t.id full join
    driver d on t."owner" = d.id
    group by d.id
) group by gender;
```

Введите SQL выражение чтобы отфильтровать

Таблица	gender	sum
1	Жен.	601 953
2	Муж.	596 785

## 4. Поиск водителей, имеющих категорию вождения «D»

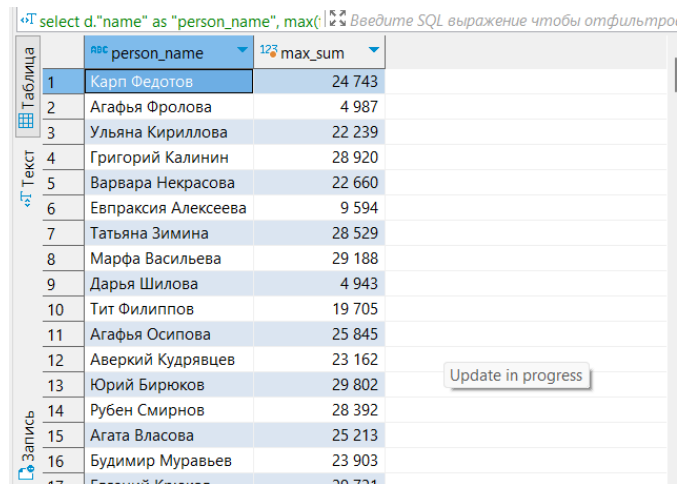
```
select d."name" from driver d join license l on l.driver = d.id where
l.category = 'D';
```

Введите SQL выражение чтобы отфильтровать

Таблица	name
1	Ангелика Медведева
2	Ксения Нестерова
3	Максим Родионов
4	Мargarита Дементьева
5	Агафья Юдина
6	Виссарион Артемьев
7	Зинаида Карпова
8	Фаина Сафонова
9	Раиса Чернова
10	Мир Симонов
11	Добромысл Сорокин
12	Тамара Турова
13	Лавр Сысоев
14	Ерофей Рябов
15	Агата Шестакова
16	Наталья Якушева
17	Фелоксий Елисеев

5. Поиск для каждого водителя максимальной суммы, на которую он был оштрафован до 1 мая 2024.

```
select d."name" as "person_name", max(f.sum) as "max_sum"
from fine_police fp join fine f on fp.fine = f.id
join transport t on f.transport = t.id
join driver d on t."owner" = d.id
where f."date" < '2024.05.01'
group by d.id;
```



	person_name	max_sum
1	Карп Федотов	24 743
2	Агафья Фролова	4 987
3	Ульяна Кириллова	22 239
4	Григорий Калинин	28 920
5	Варвара Некрасова	22 660
6	Евпраксия Алексеева	9 594
7	Татьяна Зимина	28 529
8	Марфа Васильева	29 188
9	Дарья Шилова	4 943
10	Тит Филиппов	19 705
11	Агафья Осипова	25 845
12	Аверкий Кудрявцев	23 162
13	Юрий Бирюков	29 802
14	Рубен Смирнов	28 392
15	Агата Власова	25 213
16	Будимир Муравьев	23 903
17	Евдоким Кудряков	20 721

Время выполнения этих запросов на заполненных таблицах:

№ запроса	1	2	3	4	5
Время, мс	259	4171	1916	318	3234

Поменяем конфигурации сервера для достижения большей производительности и ускорения запросов.

Были изменены следующие конфигурации сервера:

1. Shared\_buffers: Это память, которую PostgreSQL использует для хранения часто используемых данных. Увеличение этого параметра может значительно ускорить операции чтения, но также увеличивает риск потери данных при сбое сервера. Рекомендуемое значение составляет от 25% до 50% общего объема оперативной памяти сервера. Так как оперативная память сервера равно 16ГБ, было установлено значение 4ГБ, вместо стандартных 128Мб. Этот параметр значительно повлиял на скорость выполнения более долгих запросов (2, 3, 5), увеличив ее почти вдвое.



2. `Work_mem`: Это объем памяти, который PostgreSQL использует для временного хранения данных во время операций сортировки и объединения. Большее значение `work_mem` может ускорить некоторые операции, но может привести к проблемам с конкуренцией за память между процессами. Рекомендуемое значение - около 1/4 от размера `shared_buffers`. Была установлена в 1ГБ. Во время выполнения второго и пятого запроса отмечен прирост производительности.

4. `Effective_cache_size`: Это параметр, который помогает планировщику запросов оценить, сколько данных может быть кэшировано. Он не влияет напрямую на производительность, но может помочь планировщику выбрать более эффективный план выполнения запроса. Рекомендуемое значение - около 2/3 от общего объема оперативной памяти сервера. Этот параметр был установлен в 4ГБ. Особого прироста производительности замечено не было.

Так же был создан новый индекс.

```
CREATE INDEX fine_sum_idx ON bd.fine (sum);
```

Кроме того, известно, что для больших таблиц прироста производительности можно добиться ее партиционированием. Было решено партиционировать таблицу `fine` по дате штрафа. Скрипт для создания партиционированной таблицы и копирования в нее данных:

```
CREATE TABLE bd.fine_partitioned (  
    id uuid NOT NULL,  
    transport uuid NOT NULL,  
    sum numeric NOT NULL,  
    "date" date NOT NULL,  
    score uuid NULL,  
    CONSTRAINT fine_un_date UNIQUE (id, "date") DEFERRABLE INITIALLY IMMEDIATE,  
    CONSTRAINT fine_fk_transport FOREIGN KEY (transport) REFERENCES bd.transport(id)  
ON DELETE cascade DEFERRABLE INITIALLY IMMEDIATE  
) partition by range(date);  
ALTER TABLE bd.fine_partitioned ADD PRIMARY KEY (id, date) DEFERRABLE INITIALLY IMMEDIATE;  
CREATE INDEX fine_partitioned_date_idx ON bd.fine_partitioned USING btree (date);  
CREATE INDEX fine_partitioned_transport_id_idx ON bd.fine_partitioned USING btree (transport);  
  
CREATE TABLE bd.fine_1 PARTITION OF bd.fine_partitioned  
FOR VALUES FROM ('2024-01-01') TO ('2024-02-01');  
  
CREATE TABLE bd.fine_2 PARTITION OF bd.fine_partitioned  
FOR VALUES FROM ('2024-02-01') TO ('2024-03-01');
```

```

CREATE TABLE bd.fine_3 PARTITION OF bd.fine_partitioned
FOR VALUES FROM ('2024-03-01') TO ('2024-04-01');

CREATE TABLE bd.fine_4 PARTITION OF bd.fine_partitioned
FOR VALUES FROM ('2024-04-01') TO ('2024-05-01');

CREATE TABLE bd.fine_5 PARTITION OF bd.fine_partitioned
FOR VALUES FROM ('2024-05-01') TO ('2024-06-01');

CREATE TABLE bd.fine_6 PARTITION OF bd.fine_partitioned
FOR VALUES FROM ('2024-06-01') TO ('2024-07-01');

CREATE TABLE bd.fine_7 PARTITION OF bd.fine_partitioned
FOR VALUES FROM ('2024-07-01') TO ('2024-08-01');

CREATE TABLE bd.fine_8 PARTITION OF bd.fine_partitioned
FOR VALUES FROM ('2024-08-01') TO ('2024-09-01');

CREATE TABLE bd.fine_9 PARTITION OF bd.fine_partitioned
FOR VALUES FROM ('2024-09-01') TO ('2024-10-01');

CREATE TABLE bd.fine_10 PARTITION OF bd.fine_partitioned
FOR VALUES FROM ('2024-10-01') TO ('2024-11-01');

CREATE TABLE bd.fine_11 PARTITION OF bd.fine_partitioned
FOR VALUES FROM ('2024-11-01') TO ('2024-12-01');

CREATE TABLE bd.fine_12 PARTITION OF bd.fine_partitioned
FOR VALUES FROM ('2024-12-01') TO ('2025-01-01');

insert into fine_partitioned select * from fine;

```

Партиционирование ускорило 3 и 5 запросы. Кроме того, значительно быстрее стали происходить операции вставки, так как теперь индексы пересчитываются только для той партии, в которую происходит вставка.

Время выполнения запросов после всех оптимизаций:

№ запроса	1	2	3	4	5
Время, мс	86	2611	850	137	1345