

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»  
(МОСКОВСКИЙ ПОЛИТЕХ)


## КУРСОВОЙ ПРОЕКТ

По курсу **Проектирование и администрирование баз данных**  
по направлению 09.03.01 – Информатика и вычислительная техника  
Образовательная программа (профиль) «Системная и программная  
инженерия»

### ТЕМА

**«ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА БАЗЫ ДАННЫХ «УЧЕТ  
ТЕЛЕФОННЫХ ПЕРЕГОВОРОВ» В СУБД POSTGRESQL»**

Студент:  / Медведев Клим Николаевич, 221-329/  
ФИО, группа

Проверил:  к.т.н., доцент Евдошенко О.И.

Москва, 2023

## СОДЕРЖАНИЕ

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ .....	3
ВВЕДЕНИЕ .....	4
1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ .....	6
2 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА БАЗЫ ДАННЫХ .....	8
3 ИСПОЛЬЗОВАНИЕ БАЗЫ ДАННЫХ .....	19
4 ЗАКЛЮЧЕНИЕ .....	31
5 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	32

Федеральное государственное автономное  
образовательное учреждение высшего образования  
«Московский политехнический университет»

**Кафедра:** инфокогнитивных технологий

**Направление подготовки:** 09.03.01 Информатика и вычислительная техника

**Дисциплина:** Проектирование и администрирование баз данных

**Тема:** Проектирование и разработка базы данных «Учет телефонных переговоров» в СУБД PostgreSQL

### ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

1. Необходимо выбрать и проанализировать предметную область.
2. Определить сущности (>5), атрибуты (>3), связи между сущностями. Привести отношения к 3 НФ.
3. Разработать базу данных средствами СУБД на основе спроектированной модели. Установить связи для поддержки ссылочной целостности.
4. Разработать SQL-запросы (DML) (не менее 20) различной степени сложности (агрегатные функции, группировка (GROUP BY, HAVING), подзапросы, CASE, оконные функции)
5. Разработать функции для расчетов (3 функции, используя язык SQL и plpgsql)
6. Разработать триггеры (5 триггеров)

**Исполнитель:** студент гр.221-329 Медведев К.Н.

  
подпись

**Руководитель:** к.т.н., доцент кафедры ИКТ Евдошенко О.И.

  
подпись

## **ВВЕДЕНИЕ**

### **Актуальность выполнения работы**

Разработка базы данных для учета телефонных разговоров в PostgreSQL имеет большую актуальность по нескольким причинам.

1. PostgreSQL - мощная и надежная система управления базами данных (СУБД), широко используемая в различных отраслях и проектах. Её открытый исходный код позволяет гибко настраивать и расширять функциональность СУБД под конкретные потребности проекта.

2. Базы данных играют важную роль в современных информационных системах. Они обеспечивают хранение, организацию и быстрый доступ к данным. В случае учета телефонных разговоров, база данных PostgreSQL позволяет эффективно структурировать и хранить информацию о звонках, а также осуществлять поиск и анализ данных.

3. Учет телефонных разговоров является неотъемлемой частью многих предприятий и организаций, особенно в сфере телекоммуникаций. Разработка базы данных для такого учета помогает автоматизировать процессы записи и анализа звонков, оптимизировать работу операторов связи и повысить качество обслуживания клиентов.

4. PostgreSQL обладает множеством продвинутых функций и возможностей, таких как поддержка транзакций, масштабируемость, возможность создания пользовательских функций и хранимых процедур. Это делает его привлекательным выбором для разработки сложных баз данных, включая учет телефонных разговоров.

5. Разработка базы данных в PostgreSQL позволяет применять современные методы и практики управления данными, такие как использование индексов, оптимизация запросов, резервное копирование и восстановление данных. Это способствует повышению производительности, надежности и безопасности системы.

В целом, актуальность разработки базы данных для учета телефонных разговоров в PostgreSQL обусловлена его широким применением,

надежностью, гибкостью настройки, а также потребностью в эффективной организации и анализе данных в сфере телекоммуникаций.

Курсовая работа сделана в рамках предмета «Проектирование и администрирование баз данных» для закрепления навыков работы с PostgreSQL.

**Цель работы:**

Систематизация и закрепление полученных теоретических и практических умений по разработке баз данных с использованием СУБД PostgreSQL.

**Задачи работы:**

1. Изучить литературу по проектированию и разработке базы данных в СУБД PostgreSQL.
2. Получить практический опыт разработки базы данных в СУБД PostgreSQL.

# **1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

## **Предметная область**

Предметная область данного задания - учет междугородних телефонных переговоров в коммерческой службе телефонной компании. Эта область включает следующие основные элементы:

1. Телефонные линии: Компания предоставляет абонентам телефонные линии, которые используются для совершения междугородних переговоров.

2. Абоненты: Юридические лица, которые являются клиентами компании и имеют свою телефонную точку. Для каждого абонента важны данные, такие как ИНН (Идентификационный номер налогоплательщика) и расчетный счет в банке.

3. Стоимость переговоров: Стоимость междугородних телефонных переговоров зависит от города, в который осуществляется звонок, и времени суток (день, ночь). В базе данных фиксируются эти параметры для последующего расчета стоимости разговора.

4. Запись звонков: Каждый звонок абонента автоматически фиксируется в базе данных. При этом сохраняются информация о городе, дате звонка, длительности разговора и времени суток. Эти данные используются для учета и анализа звонков.

Целью данной предметной области является отслеживание и учет стоимости междугородних телефонных переговоров для каждого абонента, а также анализ и оптимизация затрат на телефонные услуги. Разработка базы данных в PostgreSQL позволит эффективно хранить, обрабатывать и анализировать данные о звонках, облегчая работу коммерческой службы и обеспечивая точность и надежность учета.

### Основные бизнес-процессы

Номер бизнес-процесса	Название бизнес-процесса
1	Регистрация абонента.
2	Подключение телефонной линии.
3	Фиксация звонка.
4	Определение стоимости разговора.
5	Учет и анализ звонков.
6	Выставление счетов абонентам.
7	Проверка оплаты и контроль задолженностей.
8	Обработка жалоб и обращений абонентов.

## 2 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА БАЗЫ ДАННЫХ

### Концептуальное проектирование базы данных

Таблица 1 – сущность «cities»

Название параметра	Тип данных	Размер	Диапазон значений
Город	Числовой	1	Больше 0
Название города	Текстовый	255	

Таблица 2 – сущность «discount\_tariff»

Название параметра	Тип данных	Размер	Диапазон значений
Город	Числовой	1	Больше 0
Скидка	Числовой	1	Не меньше 0

Таблица 3 – сущность «tariffs»

Название параметра	Тип данных	Размер	Диапазон значений
Город	Числовой	1	Больше 0
Дневной тариф	Числовой	1	Не меньше 0
Ночной тариф	Числовой	1	Не меньше 0

Таблица 4 – сущность «total»

Название параметра	Тип данных	Размер	Диапазон значений
Разговор	Числовой	1	Больше 0
Скидка	Числовой	1	Не меньше 0
Общая сумма	Числовой	1	Не меньше 0

Таблица 5 – сущность «conversation»

Название параметра	Тип данных	Размер	Диапазон значений
Разговор	Числовой	1	Больше 0
Абонент	Числовой	1	Больше 0
Город	Числовой	1	Больше 0
Дата разговора	Дата	8	ДД.ММ.ГГГГ
Время разговора	Числовой	1	Больше 0
Время суток	type_of_day	1	'day', 'night'

Таблица 6 – сущность «subscribers»

Название параметра	Тип данных	Размер	Диапазон значений
Абонент	Числовой	1	Больше 0
ИНН	Текстовый	255	
Адрес	Текстовый	255	

Таблица 7 – сущность «phone\_numbers»

Название параметра	Тип данных	Размер	Диапазон значений
Абонент	Числовой	1	Больше 0
Номер телефона	Текстовый	255	



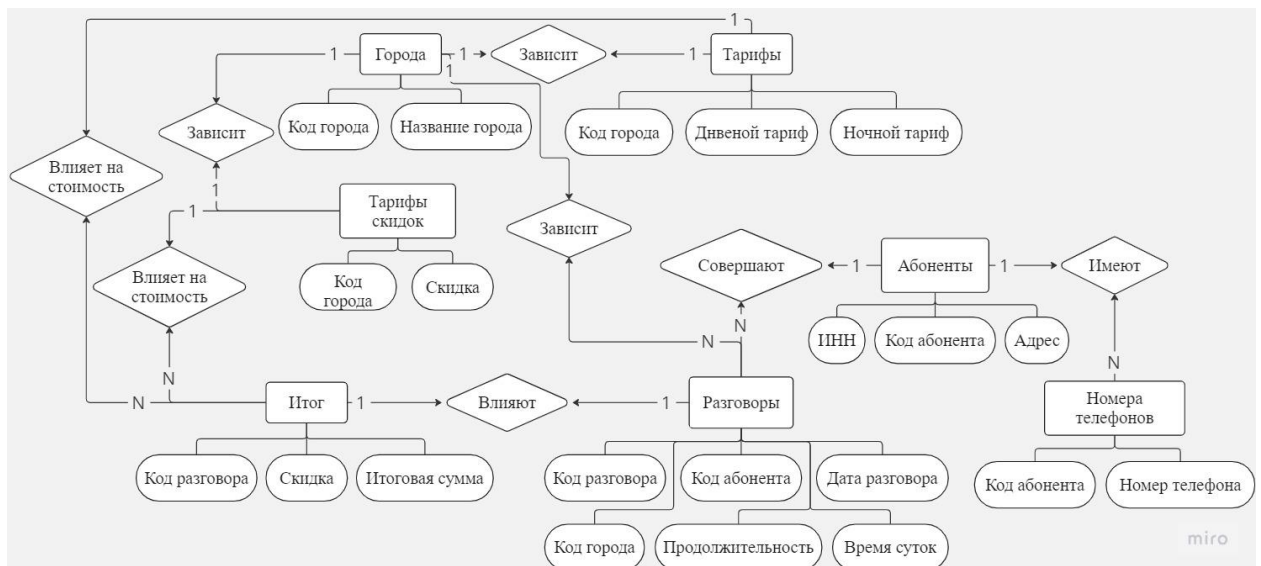


Рисунок 1 – нотация Питера Чена

## Логическое проектирование базы данных

Для выявленных сущностей была создана логическая схема базы данных, представленная на рисунке 2.

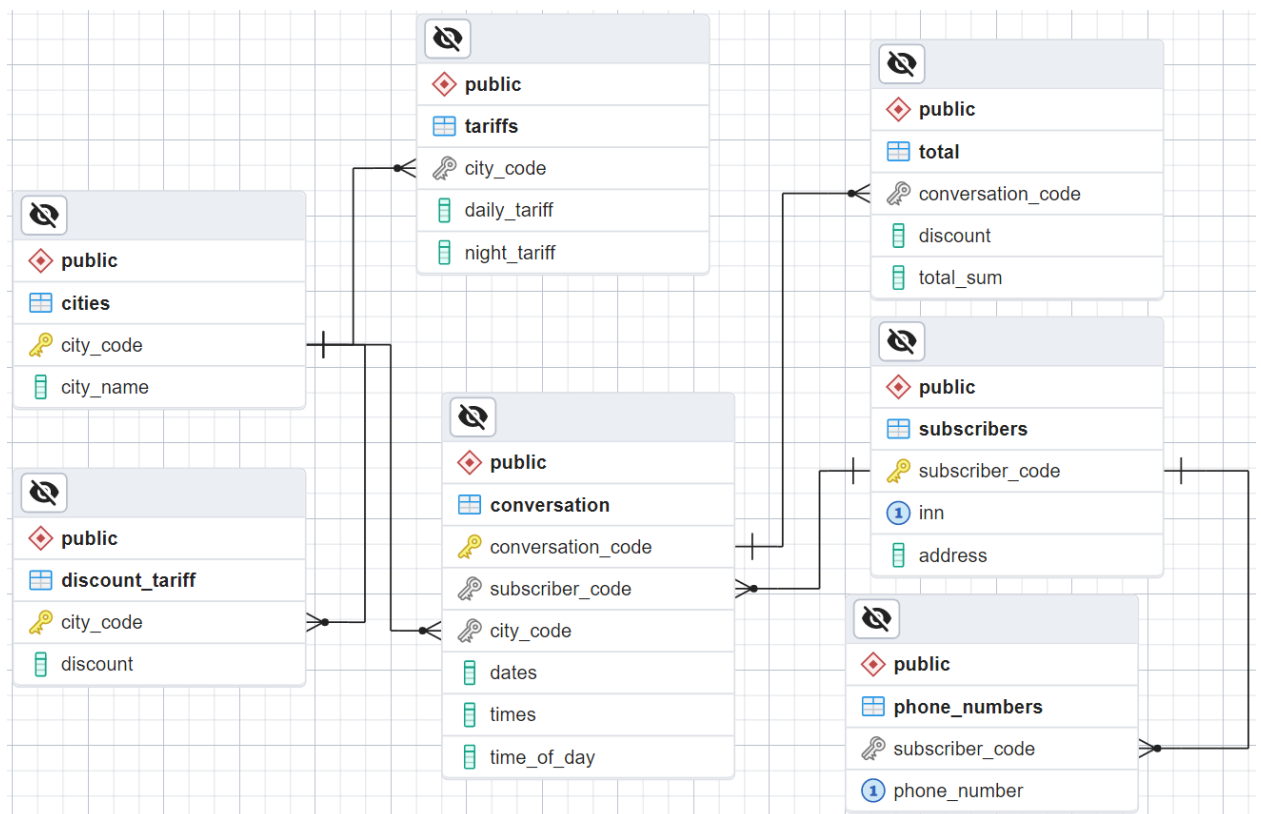


Рисунок 2 – логическая схема базы данных

## Выбор и описание СУБД

Проект учета телефонных разговоров выбрал PostgreSQL в качестве системы управления базами данных (СУБД) по следующим причинам:

1. Надежность и устойчивость: PostgreSQL известен своей стабильностью и надежностью. Он имеет множество механизмов обеспечения целостности данных, обработки сбоев и восстановления, что особенно важно в проекте учета телефонных разговоров, где точность и сохранность данных являются приоритетом.

2. Гибкость и расширяемость: PostgreSQL предлагает широкий набор возможностей и функций для разработки и оптимизации баз данных. Его открытый исходный код позволяет настраивать СУБД под специфические потребности проекта, создавать пользовательские функции и хранимые процедуры.

3. Производительность: PostgreSQL обладает эффективным механизмом оптимизации запросов и поддержкой индексов, что обеспечивает высокую производительность при обработке больших объемов данных, что может быть важным при учете большого количества телефонных разговоров.

4. Масштабируемость: PostgreSQL позволяет горизонтальное и вертикальное масштабирование, что означает возможность расширения и улучшения производительности базы данных по мере роста количества абонентов и объема данных в проекте.

5. Большое сообщество и поддержка: PostgreSQL имеет активное сообщество разработчиков и пользователей, что обеспечивает доступ к обширной документации, форумам поддержки и решению проблем. Это позволяет быстро решать возникающие вопросы и проблемы в процессе разработки и эксплуатации базы данных.

Учитывая эти преимущества, PostgreSQL является привлекательным выбором для проекта учета телефонных разговоров, обеспечивая надежность, гибкость, производительность и расширяемость базы данных.

## Физическое проектирование базы данных

### Инициализация таблиц

Используя функционал PostgreSQL была создана физическая схема базы данных, представленная на рисунке 3.

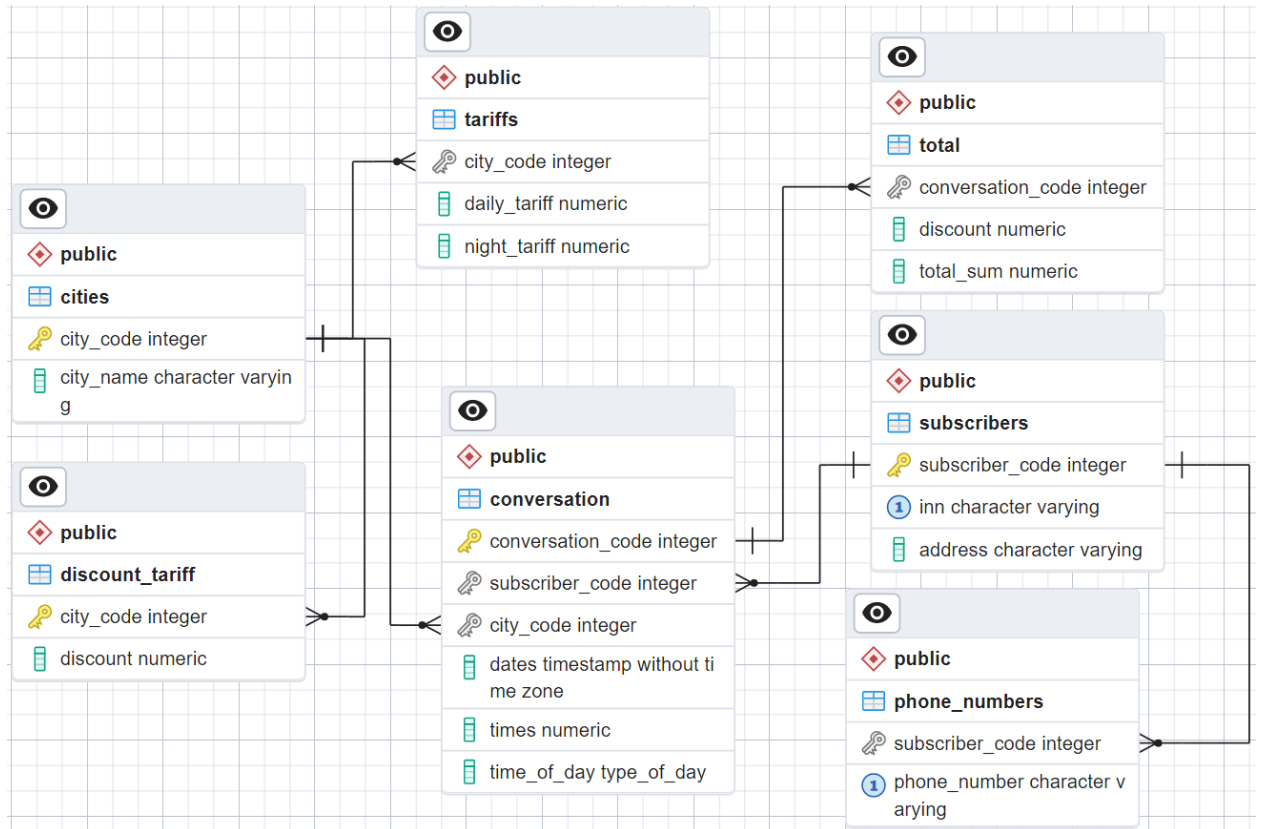


Рисунок 3 – физическая схемы базы данных

Далее модель была реализована при помощи PostgreSQL. Для создания таблиц и связей между ними были использованы SQL-запросы, которые указаны на рисунке 4.

```

CREATE TABLE IF NOT EXISTS subscribers(
    subscriber_code SERIAL PRIMARY KEY,
    inn VARCHAR UNIQUE,
    address VARCHAR
);

CREATE TABLE IF NOT EXISTS phone_numbers(
    subscriber_code INTEGER,
    phone_number VARCHAR NOT NULL UNIQUE,
    FOREIGN KEY (subscriber_code) REFERENCES subscribers(subscriber_code)
);

CREATE TABLE IF NOT EXISTS cities(
    city_code INTEGER PRIMARY KEY,
    city_name VARCHAR
);

CREATE TABLE IF NOT EXISTS tariffs(
    city_code INTEGER UNIQUE,
    daily_tariff NUMERIC NOT NULL,
    night_tariff NUMERIC NOT NULL,
    FOREIGN KEY (city_code) REFERENCES cities (city_code)
);

CREATE TYPE type_of_day AS ENUM('day', 'night');

CREATE TABLE IF NOT EXISTS conversation(
    conversation_code SERIAL PRIMARY KEY,
    subscriber_code INTEGER,
    city_code INTEGER,
    dates TIMESTAMP,
    times NUMERIC,
    time_of_day type_of_day,

    FOREIGN KEY (city_code) REFERENCES cities (city_code),
    FOREIGN KEY (subscriber_code) REFERENCES subscribers(subscriber_code)
);

CREATE TABLE discount_tariff(
    city_code INTEGER PRIMARY KEY,
    discount NUMERIC,

    FOREIGN KEY (city_code) REFERENCES cities (city_code)
);

CREATE TABLE IF NOT EXISTS total(
    conversation_code INTEGER UNIQUE,
    discount NUMERIC,
    total_sum NUMERIC,

    FOREIGN KEY (conversation_code) REFERENCES conversation(conversation_code)
);

```


Рисунок 4 – Инициализация таблиц

## Описание и тестирование функций и триггеров

- **get\_info\_sub(int)** – функция, которая вычисляет общую стоимость разговора для заданного абонента. Входные данные: код абонента. Выходные данные: число, суммарная стоимость разговоров.

```
create or replace function get_info_sub(code int) returns numeric as $$  
    SELECT sum(total_sum) FROM conversation JOIN total on  
    total.conversation_code = conversation.conversation_code  
    where subscriber_code = code group by subscriber_code  
$$ language sql;
```



```
select * from get_info_sub(92);
```

get_info_sub	
numeric	
982.0865440000000000	

- **get\_info\_sub\_from\_cities(int)** – функция, которая вычисляет среднюю стоимость разговора для заданного клиента в каждом городе. Входные данные: код абонента. Выходные данные: таблица (название города, средняя стоимость для этого города).

```
create or replace function get_info_sub_from_cities(code int)  
returns table(cities_name varchar, avg_val numeric) as $$  
    select city_name, avg(total_sum) from conversation join total on  
    total.conversation_code = conversation.conversation_code join cities on  
    cities.city_code = conversation.city_code  
    where subscriber_code = code  
    group by subscriber_code, city_name  
$$ language sql;
```

```
select * from get_info_sub_from_cities(2);
```

cities_name	avg_val
character varying 	numeric 
Волгоград	61.2480200000000000
Москва	14.4661860000000000
Омск	75.2317000000000000
Орск	34.0753600000000000

- **get\_maxtime\_call()** – функция, которая ищет самый длинный разговор и его длительность. Входные данные: нет. Выходные данные: таблица (id разговора, id абонента, id города, дата разговора, время разговора, время суток).

```
create or replace function get_maxtime_call() returns
table(convers_code int, subs_code int, cities_code int,
      datess timestamp, timess numeric, times_of_day type_of_day) as $$
declare
    max_val numeric = 0.;
begin
    SELECT max(times) into max_val FROM conversation;
    return query SELECT * FROM conversation where times = max_val;
end;
$$ language plpgsql;

select * from get_maxtime_call();
```

convers_code	subs_code	cities_code	datess	timess	times_of_day
integer	integer	integer	timestamp without time zone	numeric	type_of_day
368	64	772	2020-05-16 09:18:45	40.92	day

- **auto\_total\_trigger** – триггер, отвечающий за автоматическое изменение таблицы «total» при изменении таблицы «conversation». Входные данные: нет. Выходные данные: нет. Срабатывание: после вставки в таблицу «conversation».

```
create or replace function auto_total() returns trigger as
$$
declare
    tariff numeric = 0.0;
    val numeric = 0.0;
    discounts numeric = 0.0;
begin
    if new.time_of_day = 'day'
    then select daily_tariff into tariff from tariffs;
    else select night_tariff into tariff from tariffs;
    end if;

    val = tariff * new.times;

    select discount_tariff.discount into discounts from discount_tariff
    where discount_tariff.city_code = new.city_code;

    insert into total values(new.conversation_code, val * discounts / 100, val * (100 - discounts) / 100);

    return new;
end;
$$ language plpgsql;
create trigger auto_total_trigger after insert on conversation
for each row execute procedure auto_total();

select * from total order by conversation_code;
```

497	6.2333920000000000	50.4338080000000000
498	9.9978000000000000	89.9802000000000000
499	2.5482800000000000	15.6537200000000000
500	9.1110500000000000	60.9739500000000000

```
insert into conversation values (501, 3, 777, '2023-05-05 05:05:05', 12.45);
```

```
select * from conversation order by conversation_code;
```

497	49	755	2019-03-21 02:08:32	26.48	night
498	55	772	2022-11-25 10:15:22	26.31	day
499	99	759	2018-02-10 15:35:03	4.79	day
500	58	777	2019-11-18 00:36:31	32.75	night
501	3	777	2023-05-05 05:05:05	12.45	night

```
select * from total order by conversation_code;
```

497	6.2333920000000000	50.4338080000000000
498	9.9978000000000000	89.9802000000000000
499	2.5482800000000000	15.6537200000000000
500	9.1110500000000000	60.9739500000000000
501	3.4635900000000000	23.1794100000000000

- **auto\_time\_of\_day\_trigger** – триггер, отвечающий за автоматическое определение времени суток по времени начала разговора. Входные данные: нет. Выходные данные: нет. Срабатывание: перед вставкой или обновлении таблицы «conversation».

```
create or replace function auto_time_of_day() returns trigger as
$$
```

```
begin
```

```
    if extract(hour FROM new.dates) between 0 and 5
```

```
        then new.time_of_day = 'night';
```

```
    else new.time_of_day = 'day';
```

```
    end if;
```

```
    return new;
```

```
end;
```

```
$$ language plpgsql;
```

```
create trigger auto_time_of_day_trigger before insert or update on conversation
for each row execute procedure auto_time_of_day();
```

```
insert into conversation values (501, 3, 777, '2023-05-05 05:05:05', 12.45);
```

```
select * from conversation order by conversation_code;
```

497	49	755	2019-03-21 02:08:32	26.48	night
498	55	772	2022-11-25 10:15:22	26.31	day
499	99	759	2018-02-10 15:35:03	4.79	day
500	58	777	2019-11-18 00:36:31	32.75	night
501	3	777	2023-05-05 05:05:05	12.45	night

- **correct\_conversation\_input\_values\_check\_trigger** – триггер, отвечающий за введение валидных значений в таблицу «conversation». Проверяет, чтобы код абонента был в списке абонентов, код города был в списке городов и так далее. Входные данные: нет. Выходные данные: нет. Срабатывание: перед вставкой или обновлении таблицы «conversation».

```
create or replace function correct_conversation_input_values_check() returns trigger as
$$
begin
    if new.subscriber_code not in (select subscriber_code from subscribers)
        then raise notice 'Данный абонент не является нашим клиентом';
        return null;
    end if;
    if new.city_code not in (select city_code from cities)
        then raise notice 'Наша фирма не поддерживает звонки из этого города';
        return null;
    end if;
    if new.times < 0
        then raise notice 'Время разговора не может быть отрицательным';
        return null;
    end if;
    return new;
end;
$$ language plpgsql;

create trigger correct_conversation_input_values_check_trigger before insert or update on
conversation for each row execute procedure correct_conversation_input_values_check();

insert into conversation values (501, 3, 99999, '2023-05-05 05:05:05', 12.45);
```

ЗАМЕЧАНИЕ: Наша фирма не поддерживает звонки из этого города  
INSERT 0 0

Запрос завершён успешно, время выполнения: 42 мсек.

- **parse\_mobile\_trigger** – триггер, отвечающий за преобразование номера формата «79996667788» в номер формата «+7 (999) 666 77 88». Входные данные: нет. Выходные данные: нет. Срабатывание: перед вставкой в таблицу «phone\_numbers».



```



create or replace function parse_mobile() returns trigger as
$$
begin
    new.phone_number = concat ('+7', '(', substr(new.phone_number, 2, 3), ') ',
    substr(new.phone_number, 5, 3), ' ', substr(new.phone_number, 8, 2), ' ',
    substr(new.phone_number, 10, 2), ' ', substr(new.phone_number, 12, 2));
    return new;
end;
$$ language plpgsql;

create trigger parse_mobile_trigger before insert on phone_numbers
for each row execute procedure parse_mobile();

insert into phone_numbers values (3, 79879988776);

select * from phone_numbers where subscriber_code = 3;

```

subscriber_code 	phone_number 
integer	character varying
3	+7(987) 998 87 76

- **delete\_sub\_trigger** – триггер, отвечающий за полное удаление данных об абоненте при удалении его из таблицы «subscribers». Триггер удаляет его номер телефона, а также все записи о разговорах пользователя. Входные данные: нет. Выходные данные: нет. Срабатывание: перед удалением данных из таблицы «subscribers».

```

create or replace function delete_sub()
returns trigger as $$
declare
    conversation_codes integer[];
begin
    select array_agg(conversation_code)
    into conversation_codes from conversation
    where subscriber_code = old.subscriber_code;

    delete from phone_numbers
    where subscriber_code = old.subscriber_code;

    delete from total
    where conversation_code = any(conversation_codes);

    delete from conversation
    where subscriber_code = old.subscriber_code;

    return old;
end;
$$ language plpgsql;

create trigger delete_sub_trigger before delete on subscribers
for each row execute function delete_sub();

```

```
select * from conversation where subscriber_code = 2;
```

conversation_code [PK] integer	subscriber_code integer	city_code integer	dates timestamp without time zone	times numeric	time_of_day type_of_day
86	2	756	2019-05-12 10:38:36	10.19	day
148	2	734	2023-05-11 21:54:14	18.11	day
223	2	755	2023-01-20 00:37:24	39.50	night
257	2	777	2019-03-05 03:33:00	7.77	night

```
delete from subscribers where subscriber_code = 2;
```

```
select * from conversation where subscriber_code = 2;
```

conversation_code [PK] integer	subscriber_code integer	city_code integer	dates timestamp without time zone	times numeric	time_of_day type_of_day
-----------------------------------	----------------------------	----------------------	--------------------------------------	------------------	----------------------------

В итоге физического проектирования было создано 7 таблиц, 5 триггеров и 3 функции.

### 3 ИСПОЛЬЗОВАНИЕ БАЗЫ ДАННЫХ

#### Создание DataSet

Таблица «cities»

Код города	Название города
777	Москва
798	Санкт-Петербург
734	Волгоград
759	Пермь
...	...

Таблица «tariffs»

Код города	Дневной тариф	Ночной тариф
777	3.80	2.14
798	1.34	1.24
734	1.97	2.77
759	1.58	5.56
...	...	....

Таблица «subscribers»

Код абонента	ИНН	Адрес
1	3876043806766	Цветной бульвар, 17
2	7773719027786	Большая Садовая улица, 98
3	4827508798337	Цветной бульвар, 48
4	7590531120978	Большой Кисловский переулок, 97
...	...	...

Таблица «phone\_numbers»

Код абонента	Номер телефона
81	+7(976) 716 74 70'
22	+7(268) 166 50 61
74	+7(203) 728 81 38
33	+7(713) 287 94 94
...	...

Таблица «discount\_tariff»

Код города	Размер скидки
777	13
798	14
734	11
759	14
...	...

Таблица «conversation»

Код разговора	Код абонента	Код города	Дата	Время разговора
1	63	798	26.10.2022 14:47	24.79
2	94	777	13.11.2023 5:12	22.82
3	76	734	24.06.2020 12:35	37.70
4	29	777	24.08.2023 18:27	17.38
...	...	...	...	...

Таблица «total»

Заполняется автоматически триггером **auto\_total\_trigger**.

### Добавление данных в базу

Таблица «cities»

```
INSERT INTO cities VALUES(777, 'Москва');  
INSERT INTO cities VALUES(798, 'Санкт-Петербург');  
INSERT INTO cities VALUES(734, 'Волгоград');  
INSERT INTO cities VALUES(759, 'Пермь');
```

Таблица «subscribers»

```
INSERT INTO subscribers(inn, address) VALUES('3876043806766', 'Цветной бульвар, 17');  
INSERT INTO subscribers(inn, address) VALUES('7773719027786', 'Большая Садовая улица, 98');  
INSERT INTO subscribers(inn, address) VALUES('4827508798337', 'Цветной бульвар, 48');  
INSERT INTO subscribers(inn, address) VALUES('7590531120978', 'Большой Кисловский переулок, 97');
```

Таблица «phone\_numbers»

```
INSERT INTO phone_numbers VALUES(81, '+7(976) 716 74 70');  
INSERT INTO phone_numbers VALUES(22, '+7(268) 166 50 61');  
INSERT INTO phone_numbers VALUES(74, '+7(203) 728 81 38');  
INSERT INTO phone_numbers VALUES(33, '+7(713) 287 94 94');
```

Таблица «discount\_tariff»

```
INSERT INTO discount_tariff VALUES(777, 13);  
INSERT INTO discount_tariff VALUES(798, 14);  
INSERT INTO discount_tariff VALUES(734, 11);  
INSERT INTO discount_tariff VALUES(759, 14);
```

Таблица «tariffs»

```
INSERT INTO tariffs VALUES(777, 3.80, 2.14);  
INSERT INTO tariffs VALUES(798, 1.34, 1.24);  
INSERT INTO tariffs VALUES(734, 1.97, 2.77);  
INSERT INTO tariffs VALUES(759, 1.58, 5.56);
```

Таблица «conversation»

```
INSERT INTO conversation VALUES(1, 63, 798, '2022.10.26 14:47:57', 24.79);  
INSERT INTO conversation VALUES(2, 94, 777, '2023.11.13 5:12:59', 22.82);  
INSERT INTO conversation VALUES(3, 76, 734, '2020.6.24 12:35:20', 37.70);  
INSERT INTO conversation VALUES(4, 29, 777, '2023.8.24 18:27:30', 17.38);
```

Таблица «total»

Заполняется автоматически триггером **auto\_total\_trigger**.

## Запросы к базе данных

1. Посчитайте суммарную длительность всех разговоров в базе данных.

```
select sum(times) from conversation
```

	sum numeric
1	10852.06

2. Посчитайте среднюю длительность разговоров в каждом городе.

```
select city_name, avg(times) from conversation join cities on  
cities.city_code = conversation.city_code group by city_name;
```

city_name character varying	avg numeric
Уфа	23.1609756097560976
Ульяновск	22.1984444444444444
Казань	18.4681578947368421
Пермь	21.4942105263157895
Тюмень	24.7604651162790698

3. Посчитайте общую стоимость всех разговоров для каждого клиента.

```
select conversation.conversation_code, sum(total_sum)  
from conversation join subscribers on  
subscribers.subscriber_code = conversation.subscriber_code join total on  
total.conversation_code = conversation.conversation_code  
group by conversation.conversation_code
```

conversation_code [PK] integer	sum numeric
1	81.0137200000000000
2	42.4862760000000000
3	127.5014000000000000
4	57.4582800000000000

4. Посчитайте общую стоимость разговоров в каждое время суток (день и ночь).

```
select time_of_day, sum(times) from conversation  
group by time_of_day
```

time_of_day type_of_day	sum numeric
day	8383.67
night	2468.39

5. Найдите клиента с наибольшей суммарной стоимостью разговоров в каждом городе.

```
select city_name, subscriber_code, total_times
from (
    select city_name, subscriber_code, sum(times) as total_times,
           row_number() over (partition by city_name order by sum(times) desc)
    as row_num from conversation join cities on
    conversation.city_code = cities.city_code
    group by city_name, subscriber_code
) as subquery
where row_num = 1;
```

city_name character varying	subscriber_code integer	total_times numeric
Волгоград	7	83.79
Воронеж	76	89.28
Казань	41	58.79
Москва	57	55.04
Омск	35	68.26
Орск	51	108.91
Пермь	31	63.01
Санкт-Петербург	63	112.71
Тюмень	41	71.38
Ульяновск	58	69.10
Уфа	49	51.73

6. Найдите клиента, который совершил самый длительный разговор в каждом городе.

```
select city_name, subscriber_code, total_times
from (
    select city_name, subscriber_code, max(times) as total_times,
           row_number() over (partition by city_name order by max(times) desc)
    as row_num from conversation join cities on
    conversation.city_code = cities.city_code
    group by city_name, subscriber_code
) as subquery
where row_num = 1;
```

city_name character varying	subscriber_code integer	total_times numeric
Волгоград	96	40.23
Воронеж	21	40.55
Казань	62	40.35
Москва	54	40.88
Омск	5	39.63
Орск	51	40.17
Пермь	37	40.56
Санкт-Петербург	56	36.55
Тюмень	64	40.92
Ульяновск	50	40.86
Уфа	23	40.69

7. Посчитайте общую стоимость разговоров для каждого клиента в каждое время суток.

```
select subscriber_code, sum(total_sum), time_of_day from conversation join total on
total.conversation_code = conversation.conversation_code
group by subscriber_code, time_of_day order by subscriber_code, time_of_day
```

	subscriber_code integer	sum numeric	time_of_day type_of_day
1	2	95.3233800000000000	day
2	2	89.6978860000000000	night
3	3	133.7094600000000000	day
4	3	103.6449080000000000	night
5	4	284.7488200000000000	day
6	4	68.8930200000000000	night
7	5	292.6361000000000000	day
8	5	68.7560600000000000	night
9	6	308.1644200000000000	day
10	6	35.2624920000000000	night
11	7	569.2118800000000000	day
12	7	159.7794620000000000	night
13	8	269.7969600000000000	day
14	8	74.2318920000000000	night
15	9	361.8580400000000000	day
16	10	447.7836400000000000	day
17	10	94.6241660000000000	night
18	11	246.0496200000000000	day
19	12	385.2280400000000000	day
20	13	150.7205400000000000	day

8. Найдите клиента с наибольшей суммарной стоимостью разговоров в каждое время суток.

```
select time_of_day, subscriber_code, total_times
from (
    select time_of_day, subscriber_code, max(times) as total_times,
           row_number() over (partition by time_of_day order by max(times) desc)
           as row_num from conversation
    group by time_of_day, subscriber_code
) as subquery
where row_num = 1;
```

time_of_day type_of_day	subscriber_code integer	total_times numeric
day	64	40.92
night	54	40.88

9. Найдите клиента с наибольшей суммой скидки

```
select subscribers.subscriber_code, sum(discount) from conversation join total on
total.conversation_code = conversation.conversation_code join subscribers on
subscribers.subscriber_code = conversation.subscriber_code
group by subscribers.subscriber_code
order by sum desc, subscribers.subscriber_code limit 1
```

subscriber_code [PK] integer	sum numeric
92	121.0754560000000000

10. Найдите города, в которых были совершены разговоры во время НОЧНЫХ ЧАСОВ.

```
select city_name from conversation join cities on
cities.city_code = conversation.city_code where time_of_day = 'night'
group by city_name having count(*) > 0
```

city_name character varying
Уфа
Ульяновск
Казань

11. Найдите клиентов, которые совершили разговоры только в одном городе.



```
select distinct subscriber_code, (
    select count(distinct city_code) from conversation as d
    where d.subscriber_code = sc.subscriber_code
) from conversation as sc
where (
    select count(distinct city_code) from conversation as d
    where d.subscriber_code = sc.subscriber_code
) = 1;
```

subscriber_code integer	count bigint
48	1
82	1
98	1

12. Найдите клиентов, которые не совершили ни одного разговора.

```
select subscriber_code from subscribers where subscriber_code
not in (select subscriber_code from conversation)
```

subscriber_code [PK] integer
18
22

13. Посчитайте общую сумму денег, потраченных на разговоры в каждый день недели.

```
select case
    when extract(dow from dates) = 0 then 'Понедельник'
    when extract(dow from dates) = 1 then 'Вторник'
    when extract(dow from dates) = 2 then 'Среда'
    when extract(dow from dates) = 3 then 'Четверг'
    when extract(dow from dates) = 4 then 'Пятница'
    when extract(dow from dates) = 5 then 'Суббота'
    when extract(dow from dates) = 6 then 'Воскресенье'
end as days_of_the_week, sum(total_sum)
from conversation join total on
total.conversation_code = conversation.conversation_code
group by days_of_the_week order by sum
```

days_of_the_week text	sum numeric
Вторник	4242.6945460000000000
Суббота	4448.5287480000000000
Воскресенье	4513.2000580000000000
Среда	4769.1818320000000000
Четверг	4944.7388080000000000
Понедельник	5073.1487140000000000
Пятница	5228.9625000000000000

14. Посчитайте процентное соотношение длительности разговоров в дневное время и ночное время для каждого клиента.

```
select distinct subscriber_code,
  case
    when
      (select count(*) from conversation as pz where
       pz.time_of_day = 'night' and pz.subscriber_code = con.subscriber_code) = 0
    then null
    else
      round(((select count(*) from conversation as pz where
              pz.time_of_day = 'day' and pz.subscriber_code = con.subscriber_code)::numeric
              /
              (select count(*) from conversation as pz where
               pz.time_of_day = 'night' and pz.subscriber_code = con.subscriber_code)::numeric, 5)
            end as day_per_night
from conversation as con order by subscriber_code;
```

subscriber_code integer	day_per_night numeric
2	1.00000
3	0.66667
4	4.00000
5	4.00000
6	3.00000
7	1.50000

15. Посчитайте общую стоимость разговоров для каждого клиента за каждый месяц.

```
select subscriber_code, extract(month from dates) as months, count(*)
from conversation group by subscriber_code, extract(month from dates)
order by subscriber_code, months, count(*)
```

subscriber_code integer	months numeric	count bigint
2	1	1
2	3	1
2	5	2
3	2	1
3	4	2
3	9	1
3	12	1
4	2	1
4	6	2
4	10	1
4	11	1

16. Напишите запрос, который будет выводить информацию о клиенте:

«Непостоянный клиент» - если суммарное время разговоров меньше 90 минут

«Регулярный клиент» - если суммарное время разговоров больше 90 минут, но меньше 170 минут

«Постоянный клиент» - если суммарное время разговоров больше 170 минут

```
select subscriber_code, sum(times),
       case
         when sum(times) < 90 then 'Непостоянный клиент'
         when sum(times) between 90 and 170 then 'Регулярный клиент'
         when sum(times) > 170 then 'Постоянный клиент'
       end as charasteristic
from conversation group by subscriber_code
```

subscriber_code integer	sum numeric	charasteristic text
55	170.66	Постоянный клиент
27	80.65	Непостоянный клиент
23	114.84	Регулярный клиент
56	123.40	Регулярный клиент
91	140.84	Регулярный клиент
58	255.93	Постоянный клиент
8	116.81	Регулярный клиент
87	55.54	Непостоянный клиент
74	93.89	Регулярный клиент
54	97.95	Регулярный клиент

17. Найдите абонентов, у которых сумма дневных расходов по разговорам за последний месяц превышает среднюю дневную сумму расходов всех абонентов.

```
with avg_expenses as (
    select avg(con.times * t.daily_tariff) as average_expenses
    from conversation con
    join tariffs t on con.city_code = t.city_code
    where time_of_day = 'day' and con.dates >= current_date - interval '1 month'
)
select s.subscriber_code, s.inn, s.address,
       sum(con.times * t.daily_tariff) as total_expenses
from subscribers s
join conversation con on s.subscriber_code = con.subscriber_code
join tariffs t on con.city_code = t.city_code
where time_of_day = 'day' and con.dates >= current_date - interval '1 month'
group by s.subscriber_code, s.inn, s.address
having sum(con.times * t.daily_tariff) > (select average_expenses from avg_expenses);
```

subscriber_code [PK] integer	inn character varying	address character varying	total_expenses numeric
42	1159114668006	Большая Никитская улица, 74	99.2348
29	9512980723372	Каширское шоссе, 163	66.0440
88	5541870042446	Арбат, 78	72.8178
41	1701228722423	Тверская улица, 44	274.8504
51	7700369754105	Пресненская набережная, 102	394.9857
35	3139179336722	Бауманская улица, 87	105.6330
72	1787585661207	Мясницкая улица, 11	82.4373
24	9869077496612	Большой Кисловский переулок, 69	126.3500
92	7883448738248	Ходынский бульвар, 86	115.3580
31	7861329788834	Проспект Мира, 58	94.4478
30	1779177490997	Пресненская набережная, 78	73.4052
17	8833536805229	Мясницкая улица, 32	136.6161
27	6314157586369	Большая Никитская улица, 166	79.6422

18. Найдите все разговоры, продолжительность которых превышает среднюю продолжительность разговоров в их городе.

```
with avg_duration as (
    select c.city_code, avg(con.times) as average_duration
    from cities c
    join conversation con on c.city_code = con.city_code
    group by c.city_code
)
select con.conversation_code, con.subscriber_code, con.city_code,
con.dates, con.times, con.time_of_day
from conversation con join avg_duration avg on
con.city_code = avg.city_code
where con.times > avg.average_duration;
```

conversation_code [PK] integer	subscriber_code integer	city_code integer	dates timestamp without time zone	times numeric	time_of_day type_of_day
1	63	798	2022-10-26 14:47:57	24.79	day
2	94	777	2023-11-13 05:12:59	22.82	night
3	76	734	2020-06-24 12:35:20	37.70	day
6	50	734	2021-04-27 16:12:25	38.16	day
8	34	773	2018-08-12 07:41:08	28.22	day
9	67	772	2018-05-14 14:05:59	33.53	day
10	76	736	2022-02-09 17:04:15	30.40	day
13	24	759	2022-05-26 06:08:53	30.52	day
15	16	777	2022-09-15 07:07:53	29.83	day
16	26	798	2020-11-15 02:59:32	36.40	night
18	92	759	2021-01-19 09:39:50	34.78	day
19	78	702	2020-05-04 19:10:16	29.89	day
20	10	798	2018-01-08 00:38:15	22.49	night

19. Найдите абонентов, которые совершили разговоры из всех доступных городов.

```
select s.subscriber_code, s.inn, s.address
from subscribers s
join (
    select subscriber_code, count(distinct city_code) as total_cities
    from conversation
    group by subscriber_code
    having count(distinct city_code) = (select count(*) from cities)
) sub_count on s.subscriber_code = sub_count.subscriber_code;
```

subscriber_code [PK] integer	inn character varying	address character varying
92	7883448738248	Ходынский бульвар, 86

20. Найдите топ-5 городов с наибольшим общим количеством разговоров и суммой расходов по разговорам для каждого города.

```
select c.city_code, c.city_name, count(*) as total_conversations,  
       sum(con.times * t.daily_tariff) as total_expenses  
from cities c  
join conversation con on c.city_code = con.city_code  
join tariffs t on c.city_code = t.city_code  
group by c.city_code, c.city_name  
order by total_conversations desc, total_expenses desc  
limit 5;
```

city_code [PK] integer	city_name character varying	total_conversations bigint	total_expenses numeric
759	Пермь	57	1935.7686
734	Волгоград	51	1960.8198
777	Москва	49	4165.2560
756	Орск	48	4975.3407
755	Омск	47	4096.1283

## **4 ЗАКЛЮЧЕНИЕ**

В заключение данной курсовой работы следует отметить, что все поставленные цели были достигнуты, а задачи успешно выполнены. В процессе выполнения работы была проведена обширная литературная исследовательская работа, направленная на изучение основных принципов проектирования и разработки баз данных с использованием СУБД PostgreSQL.

Полученные теоретические знания были успешно применены на практике при разработке базы данных для учета телефонных разговоров. PostgreSQL, как мощная система управления базами данных с открытым исходным кодом, предоставила широкий набор инструментов и возможностей для эффективной организации и анализа данных в сфере телекоммуникаций.

Разработка базы данных в PostgreSQL для учета телефонных разговоров является актуальной и востребованной задачей, особенно в сфере телекоммуникаций. Применение PostgreSQL позволяет создавать гибкие и масштабируемые базы данных, а также использовать продвинутые функции и возможности, такие как поддержка транзакций, создание пользовательских функций и хранимых процедур.

В результате выполнения данной курсовой работы были закреплены и систематизированы полученные теоретические и практические умения по разработке баз данных с использованием СУБД PostgreSQL. Это позволит в дальнейшем успешно применять эти навыки при проектировании и разработке баз данных в различных проектах и отраслях, а также продолжать изучение и совершенствование знаний в области баз данных и СУБД.

## 5 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19.201-78. Режим доступа: <https://www.swrit.ru/doc/espd/19.201-78.pdf> (дата обращения: 18.04.2023).
2. Официальная документация разработчика PostgreSQL. Режим доступа: <https://www.postgresql.org/> (дата обращения: 22.06.2023).
3. Статья «Изучаем PostgreSQL. Часть 1. Знакомимся с архитектурой». Режим доступа: <https://habr.com/ru/companies/otus/articles/706346/> (дата обращения: 22.06.2023).
4. «PostgreSQL изнутри» — Е. Рогов. Режим доступа: <https://postgrespro.ru/education/books/internals> (дата обращения 22.06.2023).
5. «Postgres: первое знакомство» — П. Лузанов, Е. Рогов, И. Лёвшин. Режим доступа: <https://postgrespro.ru/education/books/introbook> (дата обращения 22.06.2023).
6. ГОСТ 7.32-2017. Режим доступа: <https://files.stroyinf.ru/Data2/1/4293742/4293742537.pdf> (дата обращения: 22.06.2023).
7. Курс в ЛМС Московского политеха «Разработка технических текстов и документации». Режим доступа: <https://online.mospolytech.ru/course/view.php?id=1689> (дата обращения: 18.04.2023).
8. «Оптимизация запросов PostgreSQL» — Г. Домбровская, Б. Новиков, А. Бейликова. Режим доступа: <https://dmkpress.com/files/PDF/978-5-97060-963-7.pdf> (дата обращения 22.06.2023).
9. «Основы технологий баз данных: учебное пособие» — Б. Новиков, Е. Горшкова, Н. Графеева. Режим доступа: <https://postgrespro.ru/education/books/dbtech> (дата обращения: 22.06.2023).
10. «PostgreSQL. Основы языка SQL» — Е. Моргунов. Режим доступа: <https://postgrespro.ru/education/books/sqlprimer> (дата обращения 22.06.2023).



11. Руководство по подготовке курсовых работ (проектов) и выпускных квалификационных работ. Режим доступа: [http://www.skf-mtusi.ru/files/vkr/MU\\_DP\\_KP\\_MTUSI.pdf](http://www.skf-mtusi.ru/files/vkr/MU_DP_KP_MTUSI.pdf) (дата обращения 22.06.2023)
12. Правила оформления курсовых и дипломных работ. Режим доступа: <https://studfile.net/preview/3009968/page:5/> (дата обращения 22.06.2023).
13. Правила оформления отчета к лабораторным и курсовым работам. Режим доступа: <https://publications.hse.ru/mirror/pubs/share/direct/227003831> (дата обращения 22.06.2023).
14. Триггерные процедуры в PostgreSQL. Режим доступа: <https://postgrespro.ru/docs/postgresql/9.6/plpgsql-trigger> (дата обращения 22.06.2023)
15. Форум по PostgreSQL Wiki.postgresql. Режим доступа: [https://wiki.postgresql.org/wiki/Main\\_Page/ru](https://wiki.postgresql.org/wiki/Main_Page/ru) (дата обращения 22.06.2023).
16. Статья «Моделирование данных: обзор». Режим доступа <https://habr.com/ru/articles/556790/> (дата обращения 22.06.2023).
17. Нотация Питера Чена. Режим доступа: [https://studme.org/77222/informatika/notatsiya\\_pitera\\_chena](https://studme.org/77222/informatika/notatsiya_pitera_chena) (дата обращения 22.06.2023).
18. Презентация «Модель «сущность-связь». Режим доступа: <https://foreva.susu.ru/courses/db/lecture3.pdf> (дата обращения 22.06.2023)