Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления (ИУ)»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии (ИУ7)»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

«Классификация методов оптимизации SQL-запросов»

Студент ИУ7-75Б _____ И. О. Артемьев
(Группа) (Подпись, дата) (И. О. Фамилия)

Руководитель _____ П. В. Клорикьян
(Подпись, дата) (И. О. Фамилия)

2023 г.

РЕФЕРАТ

В рамках данного исследования был проведен анализ области баз данных, а также подробно изучены различные методы оптимизации SQL-запросов, предоставив сравнительный анализ этих методов.

Ключевые слова: оптимизация запросов, SQL, PostgreSQL, индексы, материализованные представления, партиционирование.

Рассчетно-пояснительная записка к научно-исследовательской работе содержит 20 страницы, 1 иллюстрацию, 1 таблицу, 9 источников, 1 приложение.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В мире современных информационных технологий и баз данных, SQL-запросы служат основным средством взаимодействия с данными. Они выполняются миллионами раз в секунду, обрабатывая огромные объемы информации в реальном времени. Эффективное выполнение SQL-запросов имеет критическое значение для обеспечения производительности и отзывчивости приложений, работающих поверх баз данных. Оптимизация запросов становится неотъемлемой частью разработки и администрирования баз данных [1].

В рамках баз данных PostgreSQL, одной из самых популярных и мощных систем управления базами данных с открытым исходным кодом, оптимизация SQL-запросов имеет особое значение. Система управления базами данных (СУБД) предоставляет множество инструментов и механизмов для оптимизации запросов, и точное понимание их классификации и применения является ключевым аспектом успешной работы с этой системой.

Целью данной работы является классификация методов оптимизации SQL-запросов в PostgreSQL.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

— провести анализ предметной области;

— описать существующие методы оптимизации запросов в PostgreSQL;

— выделить критерии сравнения описанных методов;

— провести сравнение методов по выделенным критериям.

1 Анализ предметной области

1.1 Реляционная база данных

Реляционная база данных представляет собой структуру, где данные хранятся и организуются в таблицах, которые связаны на основе связанных данных. Основной целью такой структуры является возможность создания новых таблиц на основе данных из нескольких существующих таблиц с использованием одного запроса. Это также способствует пониманию взаимосвязей между данными, что может улучшить процесс принятия решений и выявление новых возможностей. Таблицы состоят из полей (столбцов) и наборов связанных данных (строк) [2].

Основное преимущество использования реляционных баз данных заключается в том, что они минимизируют избыточность данных, что в свою очередь снижает риск аномалий при выполнении операций вставки, обновления и удаления данных. Избыточность данных снижается на стадии проектирования базы данных с применением процесса нормализации. Разработчики баз данных часто используют схемы баз данных для начала создания структуры базы данных, и эти схемы определяются с использованием формального языка запросов SQL.

1.2 Системы управления базами данных (СУБД)

СУБД — это программа, которая служит для создания и обслуживания базы данных. Она также упрощает процесс определения, управления и обеспечения совместного доступа к базе данных нескольким пользователям и приложениям. Определение базы данных включает в себя установку ограничений на данные, определение типов данных, определение структур данных и другие аспекты. Эта информация обычно сохраняется в виде метаданных в каталоге пользователя СУБД, который используется пользователями СУБД и приложениями для получения информации о структуре базы данных [3].

Создание базы данных означает организацию данных таким образом, чтобы СУБД могла эффективно управлять ими, а совместное использование базы данных позволяет нескольким пользователям или приложениям одновременно получать доступ к данным и использовать их. Другие ключевые аспекты включают в себя изоляцию данных, возможность создания различных

представлений данных и уменьшение избыточности.

Изоляция данных гарантирует, что изменения в структуре данных, хранящейся в СУБД, не влияют на работу программы, что называется независимостью программы от данных. Возможность создания различных представлений данных означает, что данные из разных таблиц могут быть объединены и манипулированы для создания новых представлений. Уменьшение избыточности данных также важно и может быть достигнуто через контролируемую денормализацию, хотя иногда избыточность может быть использована для повышения производительности запросов [4].

1.3 Язык структурированных запросов (SQL)

Язык структурированных запросов играет ключевую роль в управлении реляционными базами данных (СУБД). SQL предоставляет операторы как для создания и описания структуры данных, так и для выполнения запросов и обновлений данных, что делает его одновременно Языком Определения Данных (DDL) и Языком Управления Данными (DML). DDL позволяет работать с описанием базы данных, включая создание схем, а DML занимается манипуляцией данными в базе. Язык SQL используется для создания схемы базы данных, запросов к данным и управления базой данных [4].

1.4 PostgreSQL

PostgreSQL — это система управления реляционными базами данных с открытым исходным кодом, которая использует SQL и предоставляет дополнительные функции, такие как внешние ключи, обновляемые представления и дополнительные возможности для расширения функциональности. PostgreSQL работает в архитектуре клиент-сервер, где сервер управляет данными и обрабатывает запросы клиентов, а клиентские приложения взаимодействуют с сервером для выполнения операций с данными. PostgreSQL поддерживает работу с реляционными данными, где данные хранятся в таблицах, и можно создавать схемы баз данных. Он также поддерживает внешние ключи для поддержания ссылочной целостности данных, и их поведение может быть настроено разработчиком [4].

1.5 Оптимизатор запросов в PostgreSQL

В PostgreSQL оптимизатор запросов разрабатывает план выполнения для каждого поступающего запроса. С помощью команды EXPLAIN можно получить доступ к планам, которые планировщик создает для запросов. План запроса представляет собой дерево с узлами, где листьями являются узлы сканирования, которые извлекают строки из таблицы [5].

Существует разнообразие типов узлов сканирования, в зависимости от характера выполнения запроса. Если запрос включает в себя другие операции, такие как объединение или сортировка, над узлами сканирования могут появиться другие узлы. Вывод команды EXPLAIN предоставляет информацию о каждом узле в дереве плана, его типе и предполагаемых затратах на выполнение. Затраты измеряются в произвольных единицах, которые зависят от параметров затрат, установленных планировщиком. Стоимость верхнего узла включает в себя стоимость всех его дочерних узлов.

Важно отметить, что планировщик учитывает только те аспекты, которые влияют на его решения, и не включает в расчет передачу результатов запроса клиенту. Это важно, потому что существуют другие факторы, влияющие на производительность, которые планировщик не учитывает, и это может означать, что оптимизация запроса не решает все проблемы с эффективностью.

Для проверки точности оценок, предоставляемых планировщиком, можно использовать команду EXPLAIN ANALYZE. Эта команда выполняет запрос и предоставляет информацию о количестве обработанных строк и времени выполнения для каждого узла плана, а также сравнивает эти результаты с предсказанными оценками. Для выполненных планов измеряется время в миллисекундах, в отличие от произвольных единиц, используемых в выводе команды EXPLAIN.

1.6 Представления и материализованные представления

Представление (view) представляет собой именованный запрос, который удобно использовать для часто выполняемых запросов в базе данных. Это является важным аспектом хорошего дизайна базы данных с использованием

SQL. Представления могут быть использованы практически так же, как реальные таблицы, и даже создаваться на основе других представлений. Однако следует помнить, что представления не сохраняют данные как таблицы, а являются всего лишь ссылками на запросы. Это означает, что каждый раз, когда вы обращаетесь к представлению, выполняется запрос, на котором оно основано.

Материализованное представление (materialized view) использует ту же концепцию, но сохраняет результат запроса как реальную таблицу. Основное различие между материализованным представлением и обычной таблицей заключается в том, что материализованное представление нельзя обновлять напрямую. Вместо этого сохраняется сам запрос, который используется для создания материализованного представления, и его можно периодически обновлять при необходимости обновления данных. Материализованные представления могут обеспечивать более быстрый доступ к данным по сравнению с обычными таблицами, что может быть полезно во многих сценариях, даже если данные в них не всегда актуальны [6].

1.7 Индексы

Индексы в базах данных — это мощное средство оптимизации выполнения SQL-запросов и ускорения доступа к данным. Они представляют собой структуры данных, создаваемые в базе данных, с целью улучшить производительность операций поиска, сортировки и фильтрации данных в таблицах [7].

Индексы играют важную роль в улучшении производительности базы данных, позволяя быстро находить нужные записи в таблице. Вместо того чтобы сканировать всю таблицу для выполнения запроса, база данных может использовать индексы для поиска и выборки данных. Это особенно важно при работе с большими объемами данных, где операции сканирования могут быть крайне медленными.

Индексы могут быть одностолбцовыми или многостолбцовыми, в зависимости от того, для каких столбцов они создаются. Они также могут быть уникальными, гарантируя уникальность значений в индексированных столбцах. Кроме того, существуют полнотекстовые индексы, которые облегчают поиск и фильтрацию текстовых данных.

Однако важно помнить, что индексы не являются панацеей. Они занимают дополнительное дисковое пространство и могут замедлять операции вставки, обновления и удаления данных в таблице. Поэтому выбор и проектирование индексов должно быть обдуманным и зависеть от конкретных потребностей вашей базы данных [8].

Использование индексов — это важная стратегия оптимизации работы с базами данных, и правильное их применение может значительно улучшить производительность выполнения SQL-запросов.

1.8 Партиционирование

Партиционирование представляет собой метод организации данных, при котором большая таблица разделяется на меньшие части, называемые партициями. Каждая партиция содержит данные, которые соответствуют определенным условиям, заданным на основе ключевого столбца. Этот подход предоставляет несколько выгод, включая улучшенную производительность запросов и упрощенное управление данными [8].

2 Методы оптимизации SQL-запросов

2.1 Обзор методов оптимизации SQL-запросов

В данном разделе определим базу данных и запрос к ней, затем рассмотрим несколько методов оптимизации, которые могут значительно упростить выборку данных из базы. Эти методы включают использование индексов, представлений и партиций. Понимание и эффективное применение этих методов позволяют повысить производительность выполнения SQL-запросов и снизить нагрузку на базу данных.

Создание базы данных

Создадим базу данных, изображенную на диаграмме 2.1, которая содержит три сущности. Это поможет нам сравнить различные методы оптимизации в работе с базой данных.

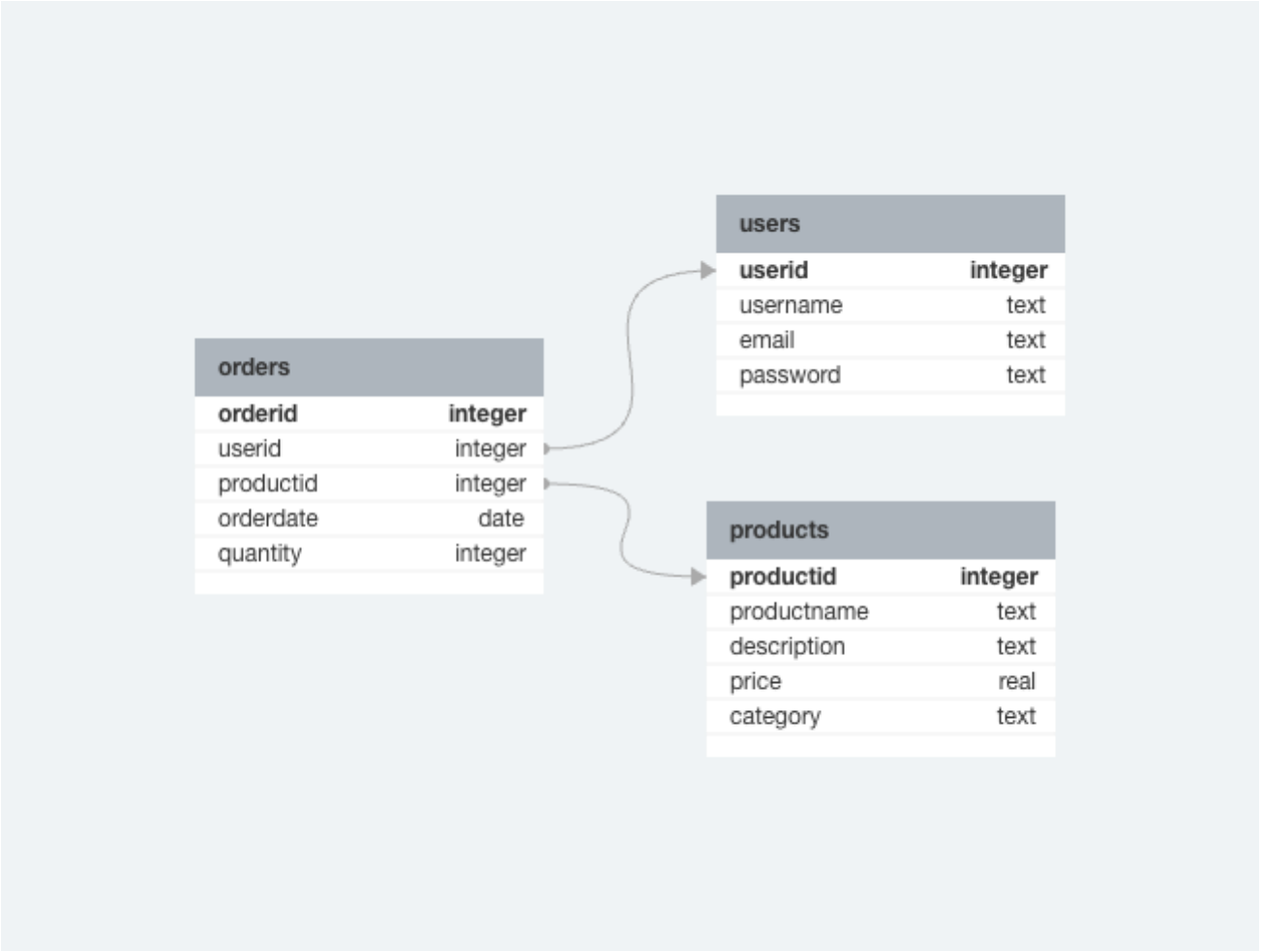

Рисунок 2.1 – Схема базы данных

Листинг 2.1 – Создание сущностей

```
CREATE TABLE IF NOT EXISTS Users
(
    UserID SERIAL PRIMARY KEY,
    Username TEXT NOT NULL,
    Email TEXT NOT NULL,
    Password TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS Products
(
    ProductID SERIAL PRIMARY KEY,
    ProductName TEXT NOT NULL,
    Description TEXT,
    Price REAL NOT NULL,
    Category TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS Orders
(
    OrderID SERIAL PRIMARY KEY,
    UserID INTEGER NOT NULL,
    ProductID INTEGER NOT NULL,
    OrderDate DATE NOT NULL,
    Quantity INTEGER NOT NULL,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
);
```

Таблица **Users** содержит информацию о пользователях:

1. UserID — Уникальный идентификатор пользователя (SERIAL PRIMARY KEY);
2. Username — Имя пользователя (TEXT NOT NULL);
3. Email — Адрес электронной почты пользователя (TEXT NOT NULL);
4. Password — Пароль пользователя (TEXT NOT NULL).

Таблица **Products** содержит информацию о продуктах:

1. ProductID — Уникальный идентификатор товара (SERIAL PRIMARY KEY);
2. ProductName — Название товара (TEXT NOT NULL);

3. Description — Описание товара (TEXT);
4. Price — Цена товара (REAL NOT NULL);
5. Category — Категория, к которой относится товар (TEXT NOT NULL).

Таблица **Orders** содержит информацию о заказах:

1. OrderID — Уникальный идентификатор заказа (SERIAL PRIMARY KEY);
2. UserID — Идентификатор пользователя, совершившего заказ (INTEGER NOT NULL);
3. ProductID — Идентификатор товара, включенного в заказ (INTEGER NOT NULL);
4. OrderDate — Дата размещения заказа (DATE NOT NULL);
5. Quantity — Количество товаров в заказе (INTEGER NOT NULL).

Определение запроса

Определим запрос, на котором будем проводить сравнение методов. Данный запрос (листинг 2.2) выводит информацию о заказах, где пользователь с именем ilya заказал продукт chipsiki в количестве 1 единицы.

Листинг 2.2 – Запрос к базе данных

```
SELECT
    u.Username AS User,
    p.ProductName AS Product,
    o.OrderDate,
    o.Quantity
FROM
    Users u
JOIN
    Orders o ON u.UserID = o.UserID
JOIN
    Products p ON o.ProductID = p.ProductID
WHERE
    u.Username = 'ilya' AND
    p.ProductName = 'chipsiki' AND
    o.Quantity = 1;
```

Метод оптимизации с использованием индексов

Для оптимизации запроса, можно создать индексы на столбцах, которые используются в условиях фильтрации и объединения таблиц (листинг 2.3).

Листинг 2.3 – Создание индексов

```
CREATE INDEX idx_users_username ON Users (Username);
CREATE INDEX idx_products_productname ON Products (ProductName);
CREATE INDEX idx_orders_quantity ON Orders (Quantity);
CREATE INDEX idx_orders_userid ON Orders (UserID);
CREATE INDEX idx_orders_productid ON Orders (ProductID);
```

Создав эти индексы, запрос будет выполняться более эффективно, так как база данных сможет быстро найти и отфильтровать соответствующие записи и объединить таблицы, уменьшая нагрузку на запрос и улучшая производительность.

Метод оптимизации с использованием материализованного представления

Для оптимизации запроса и улучшения производительности, можно создать материализованное представление (листинг 2.4). Материализованное представление представляет собой предварительно вычисленные данные, которые хранятся в виде таблицы, что позволяет уменьшить нагрузку на базу данных при выполнении запросов [6].

Листинг 2.4 – Создание материализованного представления

```
CREATE MATERIALIZED VIEW OrderView AS
SELECT
    u.Username AS User,
    p.ProductName AS Product,
    o.OrderDate,
    o.Quantity
FROM
    Users u
JOIN
    Orders o ON u.UserID = o.UserID
JOIN
    Products p ON o.ProductID = p.ProductID
WHERE
    u.Username = 'ilya' AND
    p.ProductName = 'chipsiki' AND
    o.Quantity = 1;
```

Метод оптимизации с использованием партиционирования

Для оптимизации запроса можно использовать партиции (листинг 2.5). Этот метод помогает улучшить производительность и управляемость базы данных, особенно в случаях, когда таблицы содержат большие объемы данных и когда часто выполняются запросы на фильтрацию данных по определенным критериям [9].

Листинг 2.5 – Переопределение базы и создание партиций

```
CREATE TABLE UsersMain
(
    UserID SERIAL,
    Username TEXT NOT NULL,
    Email TEXT NOT NULL,
    Password TEXT NOT NULL,
    PRIMARY KEY (UserID, Username)
)
PARTITION BY LIST (Username);

CREATE TABLE UsersPart1 PARTITION OF UsersMain FOR VALUES IN ('ilya', 'gadzhi', 'kirill');
CREATE TABLE UsersPart2 PARTITION OF UsersMain FOR VALUES IN ('khamit', 'sergey', 'anton');


CREATE TABLE ProductsMain
(
    ProductID SERIAL,
    ProductName TEXT NOT NULL,
    Description TEXT,
    Price REAL NOT NULL,
    Category TEXT NOT NULL,
    PRIMARY KEY (ProductID, ProductName)
)
PARTITION BY LIST (ProductName);

CREATE TABLE ProductsPart1 PARTITION OF ProductsMain FOR VALUES IN ('chipsiki', 'suhariki', 'ribka');
CREATE TABLE ProductsPart2 PARTITION OF ProductsMain FOR VALUES IN ('kvas', 'voda', 'sok');


CREATE TABLE OrdersMain
(
    OrderID SERIAL,
```

```
33      UserID INTEGER NOT NULL,
34      Username TEXT NOT NULL,
35      ProductID INTEGER NOT NULL,
36      ProductName TEXT NOT NULL,
37      OrderDate DATE NOT NULL,
38      Quantity INTEGER NOT NULL,
39      FOREIGN KEY (UserID, Username) REFERENCES UsersMain(UserID, Username),
40      FOREIGN KEY (ProductID, ProductName) REFERENCES ProductsMain(ProductID
            , ProductName),
41      PRIMARY KEY (OrderID, Quantity)
42  )
43  PARTITION BY LIST (Quantity);
44
45  CREATE TABLE OrdersPart1 PARTITION OF OrdersMain FOR VALUES IN (1, 2, 3);
46  CREATE TABLE OrdersPart2 PARTITION OF OrdersMain FOR VALUES IN (4, 5, 6);
```

2.2 Сравнение и оценка методов

Сравнение эффективности методов оптимизации SQL-запросов проводилось в условиях, где в каждой из трех таблиц было около 10 000 записей, а условию фильтрации удовлетворяли всего 100 записей.

Сравнение проводилось по следующим критериям:

— К1 — насколько быстрее (во сколько раз) выполняется запрос после применения оптимизации по сравнению с исходным запросом без оптимизации;

— К2 — влияние метода на размер и структуру базы данных;

— К3 — дополнительные расходы на другие операции с таблицей (удаление, вставка);

— К4 — дополнительные расходы на ручной контроль.

Результаты сравнений приведены в таблице 2.1.

Таблица 2.1 – Сравнение методов

Методы	**К1**	**К2**	**К3**	**К4**
Использование индексов	3	+	+	-
Использование материализованного представления	33	+	-	+
Использование партиционирования	5	+	+	-

2.3 Вывод

Сравнив результаты анализа, можно утверждать, что метод оптимизации с использованием материализованного представления оказался наиболее эффективным, ускорив выполнение запроса в 33 раза по сравнению с исходной версией. Однако этот метод также требует дополнительных затрат на контроль и оказывает влияние на размер и структуру базы данных.

Метод с использованием индексов также дал заметное улучшение, ускорив запрос в 3 раза, но он требует дополнительных ресурсов при операциях вставки и удаления данных.

Метод с использованием партиционирования показал ускорение запроса в 5 раз, но не стоит забывать, что ускорение зависит от партиций и тех данных, которые в них лежат.

ЗАКЛЮЧЕНИЕ

В ходе выполнения научно-исследовательской работы были классифицированы методы оптимизации SQL-запросов.

На основании результатов сравнения методов можно сделать вывод о том, что в задаче оптимизации SQL-запросов выбор метода оптимизации зависит от конкретных потребностей и компромиссов между производительностью и дополнительными расходами.

При написании данной работы:

— проведен анализ предметной области;

— описаны существующие методы оптимизации запросов в PostgreSQL;

— выделены критерии сравнения описанных методов;

— проведено сравнение методов по выделенным критериям.

Таким образом, поставленные задачи были выполнены, цель научно-исследовательской работы была достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Ferrari L.*, *Pirozzi E.* Learn PostgreSQL: Build and manage high-performance database solutions using PostgreSQL 12 and 13. — Packt Publishing, 2020.
2. *Juba S.*, *Volkov A.* Learning PostgreSQL 11: A Beginner's Guide to Building High-Performance PostgreSQL Database Solutions, 3rd Edition. — 3rd. — Packt Publishing, 2018.
3. *Johnson C.*, *Brown D.* Optimizing SQL Queries for Performance: A Comprehensive Guide // International Journal of Advanced Computer Science and Applications. — 2016. — Т. 7, № 1. — С. 50—57.
4. *Smith B.* A Survey of SQL Query Optimization Methods // International Journal of Computer Applications. — 2017. — Т. 162, № 5. — С. 28—33.
5. *Kumar S.*, *Gupta P.* Efficient Techniques for SQL Query Optimization: A Review // International Journal of Computer Science and Information Technologies. — 2021. — Т. 2, № 1. — С. 73—77.
6. *Смирнов Д. Н.* Классификация и анализ методов оптимизации SQL-запросов // Россия, Москва, Информационные технологии и вычислительные системы. — 2020.
7. *Wang J.*, *Li Q.* A Comparative Study of SQL Query Optimization Techniques // Journal of Computer and Communications. — 2019. — Т. 7, № 2. — С. 11—21.
8. *Johnson M.*, *Davis L.* An Overview of SQL Query Optimization Techniques // International Journal of Computer Applications. — 2017. — Т. 156, № 3. — С. 12—19.
9. *Patel A.*, *Shah R.* A Survey on SQL Query Optimization Techniques // International Journal of Computer Applications. — 2015. — Т. 113, № 15. — С. 20—26.

ПРИЛОЖЕНИЕ А