|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01** Информатика и вычислительная техника

**Отчет**

|  |
| --- |
| **по Лабораторной работе № 3** |

**Название:** Создание БД для аналитики

**Дисциплина:** Базы данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-32Б |  |  | А. П. Плютто |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | М. А. Скворцова |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2023

Данная лабораторная состоит из 2 взаимосвязанных частей и является продолжением лабораторной работы №2. В каждой разделе есть теоретическая и практическая части лабораторной работы. В конце каждого раздела есть задания для выполнения в рамках лабораторной работы.

**Цель:**

• Сформировать у студента понимание особенностей создания аналитических баз данных и умение их настраивать и поддерживать.

**Задачи:**

• Получить теоретические знания денормализации.

• Узнать о основных методах денормализации.

• Ознакомится с массивами.

• Научится (изменять\добавлять\удалять) данные в массиве с помощью встроенных операций.

• Более подробно узнать о типе данных JSON.

• Узнать о ролях и пользователях.

• Научиться пользоваться командами для того, чтобы (определять\отзывать) доступ к данным, GRANT и REVOKE.

• Научится упрощать запросы к БД с помощью представлений.

**Практическое задание для первой части ЛР**

Задание №1 связано с проектированием схемы базы данных для

аналитики. Будем исходить из того, что приложение, для которого была

сделана база данных в лабораторной работе №2, стало очень популярным и по

нему каждый день можно собирать большой объем статистической

информации. Что это будет за статистика? Почему именно ее необходимо

собирать, обрабатывать и анализировать? Задачей студента является ответить

на эти вопросы, и, исходя из этого, изменить разработанную БД (допускается

разработка новой БД, но тематика остается та же) и дозаполнить ее

необходимыми атрибутами и данными. Результатом данного задания является

схема базы данных, скрипты доработки базы данных и ее заполнения,

обладающие следующими свойствами:

• Как минимум одна таблица должна содержать не меньше 100 млн.

записей, которые со временем теряют актуальность.

• Другая таблица, связанная с первой, должна содержать не меньше 1 млн.

записей.

• В одной из таблиц с количество записей больше 1 млн. должна быть

колонка с текстом, по которой будет необходимо настроить полнотекстовый

поиск.

• В одной из таблиц с количество записей больше 1 млн. должна быть

колонка с данными в json-формате.

• В одной из таблиц с количество записей больше 1 млн. должна быть

колонка с массивом.

При выполнении задания важно учитывать плюсы и минусы

денормализации схемы данных и использования массивов и json-формата. При

сдаче задания студент должен обосновать соответствие созданной схемы

поставленной задаче. Для проектирования схемы и построения диаграммы

можно использовать любые средства, в том числе те, которые были описаны в

лабораторной №1.

Код для генерации 100 млн. записей:

1. import psycopg2

2. import random

3. from datetime import datetime, timedelta

4. from urllib.parse import urlparse

5. from time import time\_ns

6.

7. def generate\_random\_name():

8. first\_names = ['Иван', 'Мария', 'Александр', 'Екатерина', 'Дмитрий', 'Ольга', 'Сергей', 'Анна', 'Андрей', 'Наталья']

9. last\_names = ['Иванов', 'Петров', 'Сидоров', 'Козлов', 'Смирнов', 'Кузнецов', 'Морозов', 'Николаев', 'Павлов', 'Васнецов']

10.

11. random\_name = random.choice(first\_names) + " " + random.choice(last\_names)

12. return random\_name

13.

14. def generate\_random\_city():

15. cities = ['Москва', 'Санкт-Петербург', 'Новосибирск', 'Екатеринбург', 'Нижний Новгород', 'Казань', 'Челябинск', 'Омск', 'Самара', 'Ростов-на-Дону', 'Уфа', 'Красноярск', 'Пермь', 'Воронеж', 'Волгоград', 'Краснодар', 'Саратов', 'Тюмень', 'Ижевск', 'Ульяновск']

16. unique\_cities = list(set(cities))

17. random\_city = random.choice(unique\_cities)

18.

19. return random\_city

20.

21. def generate\_random\_birthdate():

22. current\_year = 2023

23. min\_birth\_year = current\_year - 19

24. max\_birth\_year = current\_year - 18

25. birth\_year = random.randint(min\_birth\_year, max\_birth\_year)

26. birth\_month = random.randint(1, 12)

27. birth\_day = random.randint(1, 28)

28.

29. birthdate = datetime(birth\_year, birth\_month, birth\_day)

30. return birthdate.strftime('%Y-%m-%d')

31.

32. def generate\_random\_year():

33. return 2022 if random.randint(0, 1) else 2023

34. url = urlparse("postgresql://pluttan:Pluttan@localhost:5432/lab2")

35. db\_params = {

36. 'host': url.hostname,

37. 'database': url.path[1:],

38. 'user': url.username,

39. 'password': url.password,

40. 'port': url.port,

41. }

42. connection = psycopg2.connect(\*\*db\_params)

43. cursor = connection.cursor()

44. a = time\_ns()

45. new\_b = a

46. for i in range(100\_000\_000):

47. cursor.execute("INSERT INTO student (sname, birsday, town, nyear, education\_id) VALUES (%s, %s, %s, %s, 1)",

48. (generate\_random\_name(), generate\_random\_birthdate(), generate\_random\_city(), generate\_random\_year())

49. )

50. if (i%10000 == 0):

51. b = time\_ns()

52. print(i, b-new\_b, b-a)

53. new\_b = b

54.

55. connection.commit()

56. cursor.close()

57. connection.close()

58.

Добавим новые поля к таблице в 1млн. записей и заполним ее.

1. ALTER TABLE education

2. ADD COLUMN data JSON,

3. ADD COLUMN datatext TEXT,

4. ADD COLUMN dataarray VARCHAR(60)[];

5. ALTER TABLE education

6. ADD COLUMN data\_tsvector TSVECTOR;

7. CREATE INDEX idx\_education\_data\_tsvector ON education USING GIN(data\_tsvector);

8. UPDATE education SET data\_tsvector = to\_tsvector('russian', datatext);

9.

1. from faker import Faker

2. from random import randint

3. import psycopg2

4. from urllib.parse import urlparse

5. from time import time\_ns

6. import json

7.

8. fake = Faker("ru\_RU")

9. sq = 1

10.

11. url = urlparse("postgresql://pluttan:Pluttan@localhost:5432/lab2")

12.

13. db\_params = {

14. "host": url.hostname,

15. "database": url.path[1:],

16. "user": url.username,

17. "password": url.password,

18. "port": url.port,

19. }

20.

21.

22. def generate\_tsvector(description):

23. connection = psycopg2.connect(\*\*db\_params)

24. cursor = connection.cursor()

25. cursor.execute("SELECT to\_tsvector('russian', %s)", (description,))

26. tsvector = cursor.fetchone()[0]

27. cursor.close()

28. connection.close()

29. return tsvector

30.

31.

32. def format\_description():

33. description = " ".join(fake.sentences(randint(1, 4)))

34. return (

35. description,

36. json.dumps(

37. {

38. f"Предложение {i + 1}": sentence

39. for i, sentence in enumerate(description.split(". "))

40. },

41. ensure\_ascii=False,

42. ),

43. description.split(" "),

44. generate\_tsvector(description),

45. )

46.

47.

48. def specialization():

49. return "ivt" if randint(0, 1) else "pi"

50.

51.

52. def squadAndYear():

53. global sq

54. sq += 1

55. year = 2023 if randint(0, 1) else 2022

56. return ("ИУ6-" + str(3 if year == 2022 else 1) + str(sq) + "Б", year)

57.

58.

59. def ship():

60. return randint(1000, 10000)

61.

62.

63. def fid():

64. return 1

65.

66.

67. connection = psycopg2.connect(\*\*db\_params)

68. cursor = connection.cursor()

69. a = time\_ns()

70. new\_b = a

71. for i in range(1\_000\_000):

72. cursor.execute(

73. "INSERT INTO education (specialization, squad, nyear, ship, faculty\_id, datatext, data, dataarray, data\_tsvector) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)",

74. (specialization(), \*squadAndYear(), ship(), fid(), \*format\_description()),

75. )

76. if i % 10 == 0:

77. b = time\_ns()

78. print(i, b - new\_b, b - a)

79. new\_b = b

80.

81. connection.commit()

82. cursor.close()

83. connection.close()

84.

**Практическое задание для второй части ЛР**

• Создать пользователя test и выдать ему доступ к базе данных.

• Составить и выполнить скрипты присвоения новому пользователю прав

доступа к таблицам, созданным в практическом задании 1. При этом права

доступа к различным таблицам должны быть различными, а именно:

o По крайней мере, для одной таблицы новому пользователю

присваиваются права SELECT, INSERT, UPDATE в полном объеме.

o По крайней мере, для одной таблицы новому пользователю

присваиваются права SELECT и UPDATE только избранных столбцов.

o По крайней мере, для одной таблицы новому пользователю

присваивается только право SELECT.

• Создать стандартную роль уровня базы данных, присвоить ей право

доступа (UPDATE на некоторые столбцы) к представлению, созданному в

практическом задании №3.3, назначить новому пользователю созданную роль.

• Выполнить от имени нового пользователя некоторые выборки из таблиц

и представления. Убедиться в правильности контроля прав доступа.

• Выполнить от имени нового пользователя операторы изменения таблиц

с ограниченными правами доступа. Убедиться в правильности контроля прав

доступа.

• Составить SQL-скрипты для создания нескольких представлений,

которые позволяли бы упростить манипуляции с данными или позволяли бы

ограничить доступ к данным, предоставляя только необходимую

информацию.

**Создать пользователя test и выдать ему доступ к базе данных.**

1. CREATE USER test WITH PASSWORD 'password';

2. GRANT CONNECT ON DATABASE lab2 TO test;

3.

**Составить и выполнить скрипты присвоения новому пользователю прав**

**доступа к таблицам, созданным в практическом задании 1. При этом права**

**доступа к различным таблицам должны быть различными, а именно:**

**- По крайней мере, для одной таблицы новому пользователю**

**присваиваются права SELECT, INSERT, UPDATE в полном объеме.**

**- По крайней мере, для одной таблицы новому пользователю**

**присваиваются права SELECT и UPDATE только избранных столбцов.**

**- По крайней мере, для одной таблицы новому пользователю**

**присваивается только право SELECT.**

1. GRANT SELECT, INSERT, UPDATE ON education TO test;

2. GRANT USAGE, SELECT ON SEQUENCE education\_faculty\_id\_seq TO test;

3. GRANT SELECT (id\_student, education\_id), UPDATE (education\_id) ON student TO test;

4. GRANT SELECT ON faculty TO test;

5.

**Создать стандартную роль уровня базы данных, присвоить ей право**

**доступа (UPDATE на некоторые столбцы) к представлению, созданному в**

**практическом задании №3.3, назначить новому пользователю созданную роль.**

1. CREATE VIEW students\_cafs AS

2. SELECT sname, squad

3. FROM student, education

4. WHERE education\_id = id\_education;

5.

1. CREATE ROLE test\_role;

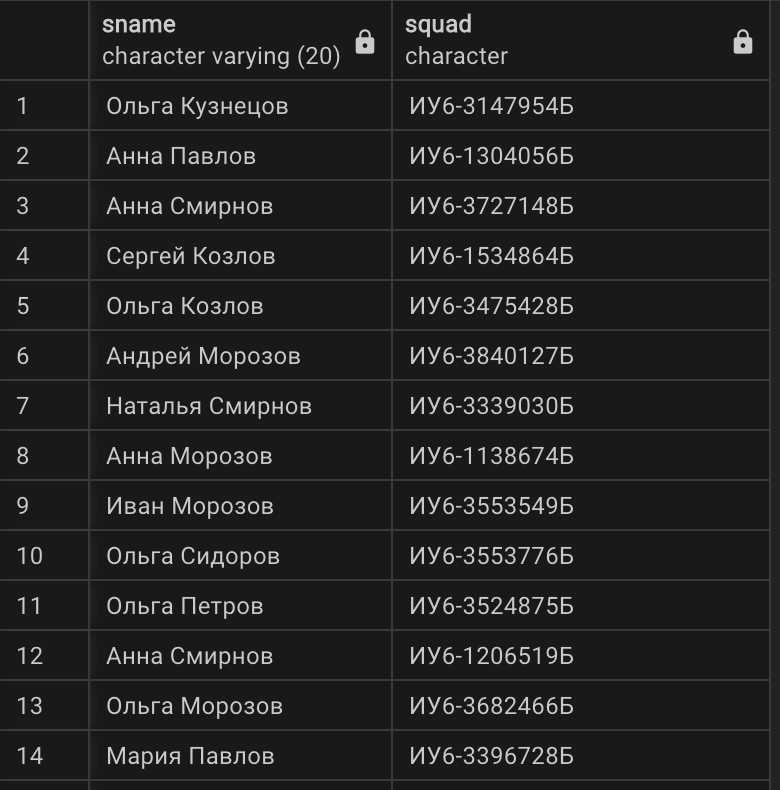
2. GRANT SELECT, UPDATE ON students\_cafs TO test\_role;

3. GRANT test\_role TO test;

4.

1. SELECT \* FROM students\_cafs limit 1000;

2.

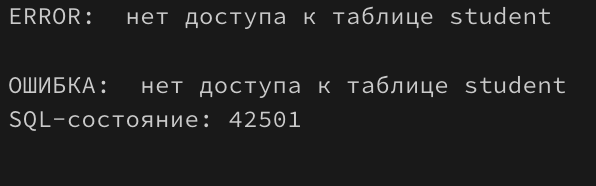


**Выполнить от имени нового пользователя некоторые выборки из таблиц**

**и представления. Убедиться в правильности контроля прав доступа.**

1. SELECT \* FROM student;

2.



**Составить SQL-скрипты для создания нескольких представлений,**

**которые позволяли бы упростить манипуляции с данными или позволяли бы**

**ограничить доступ к данным, предоставляя только необходимую**

**информацию.**

1. CREATE VIEW student\_id AS

2. SELECT id\_student, education\_id FROM student;

3.

1. CREATE VIEW students\_cafs AS

2. SELECT sname, squad

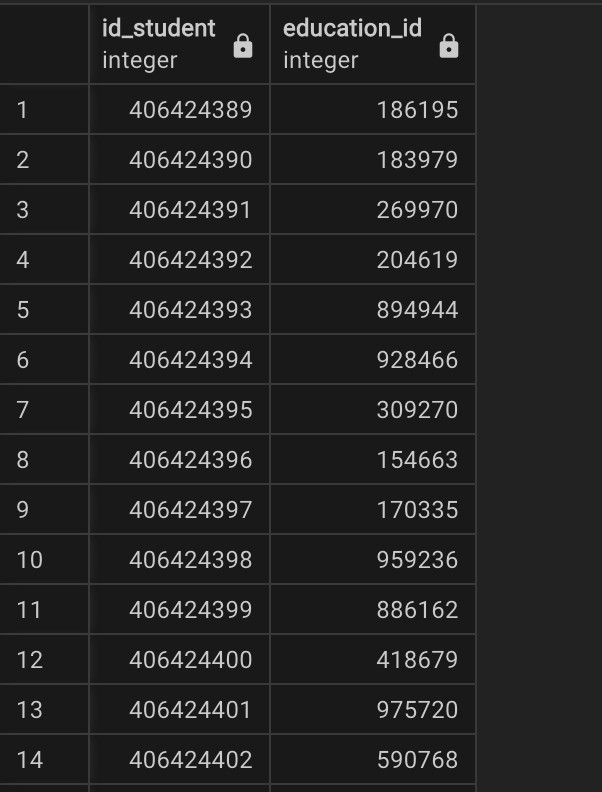
3. FROM student, education

4. WHERE education\_id = id\_education;

5.

1. SELECT \* FROM student\_id limit 1000;

2.



**Выполнить от имени нового пользователя операторы изменения таблиц**

**с ограниченными правами доступа. Убедиться в правильности контроля прав**

**доступа.**

1. INSERT INTO education (squad) values (1000);

2.

**Вопросы для самостоятельного изучения**

1. **Денормализация** — это процесс организации данных в базе данных для улучшения производительности при их извлечении. Она противоположна нормализации, которая направлена на уменьшение избыточности данных и предотвращение аномалий в базе данных. Денормализация используется в тех случаях, когда производительность чтения данных становится критическим фактором, и разрешается некоторая избыточность в данных ради повышения быстродействия запросов.
2. **Виды денормализации:**
   * **Предварительная денормализация (Precomputed Denormalization):** Создание и хранение дополнительных данных в момент вставки или обновления данных для ускорения последующих запросов.
   * **Постобработка (Query-Time Denormalization):** Денормализация данных в момент выполнения запроса, а не заранее. Это может включать объединение данных из нескольких таблиц в один результат.
   * **Кеширование (Caching):** Хранение результатов запросов для повторного использования вместо повторного выполнения запросов.
3. **Методы реализации денормализации:**
   * **Добавление дублированных данных:** Копирование данных из одной таблицы в другую для уменьшения необходимости объединения таблиц при запросах.
   * **Использование вычисляемых столбцов:** Создание дополнительных столбцов в таблице для хранения результатов вычислений, которые могут быть использованы в запросах.
   * **Хранение предварительно вычисленных агрегатов:** Сохранение результатов агрегатных функций (например, суммы, средние значения) для ускорения запросов агрегации.
   * **Материализованные представления (Materialized Views):** Создание и хранение предварительно вычисленных представлений данных для оптимизации выполнения запросов.
4. **Преимущество использования массивов:**
   * **Эффективность:** Массивы обеспечивают эффективное хранение и доступ к данным. Использование массивов позволяет сократить объем избыточной информации и уменьшить время доступа к данным.
   * **Простота кодирования и обработки данных:** Массивы упрощают работу с наборами данных, особенно когда данные имеют структуру и могут быть представлены в виде многомерных массивов.
   * **Поддержка множества операций:** Массивы обеспечивают поддержку различных операций, таких как индексация, сортировка и фильтрация, что делает их удобным инструментом для работы с данными.
   * **Экономия места:** По сравнению с дублированием данных в нескольких таблицах, массивы позволяют эффективнее использовать пространство хранения.
5. **Разница между JSON и JSONB в PostgreSQL:**
   * **JSON (JavaScript Object Notation):** Это стандартный текстовый формат данных, представляющий собой набор пар "ключ-значение". В PostgreSQL, столбец типа JSON сохраняет данные в текстовом формате без дополнительной обработки. Поиск в данных JSON может быть медленнее из-за неоптимизированной структуры хранения.
   * **JSONB (Binary JSON):** Это формат хранения данных JSON в бинарном формате. JSONB обеспечивает более эффективное хранение и индексацию данных, что улучшает производительность запросов по сравнению с обычным JSON. Он поддерживает более сложные типы данных и операции сравнения, что делает его более мощным и эффективным для запросов.
6. **Для чего нужны роли в СУБД (Система Управления Базами Данных):**
   * **Аутентификация и авторизация:** Роли используются для аутентификации пользователей и предоставления им определенных прав доступа к базам данных.
   * **Управление доступом:** Роли позволяют определять права доступа к таблицам, представлениям, функциям и другим объектам базы данных.
   * **Ограничение доступа:** Роли позволяют ограничивать доступ к чувствительным данным, предоставляя различные уровни привилегий разным пользователям.
7. **Что такое схема в контексте базы данных:**

**Схема (Schema):** Это логическая группировка объектов базы данных, таких как таблицы, представления, индексы и т. д. Схемы используются для организации базы данных и изоляции объектов от других схем. Они помогают управлять именами объектов, предоставляют пространство имен и улучшают структурирование базы данных.

1. **Директивы GRANT и REVOKE в SQL:**
   * **GRANT:** Эта директива используется для предоставления прав доступа к объектам базы данных (например, таблицам, представлениям, функциям) пользователям или ролям. GRANT может предоставлять различные привилегии, такие как SELECT, INSERT, UPDATE, DELETE и другие.

1. GRANT SELECT, INSERT ON table\_name TO user\_name;

* + **REVOKE:** Эта директива используется для отзыва предоставленных прав доступа. REVOKE отменяет ранее предоставленные привилегии.

1. REVOKE SELECT, INSERT ON table\_name FROM user\_name;

GRANT и REVOKE вместе обеспечивают механизм управления безопасностью и доступом к данным в базе данных.

1. **Роль PUBLIC:**

Роль PUBLIC в базе данных PostgreSQL является предопределенной ролью, которая используется для управления правами доступа, применяемыми ко всем пользователям базы данных. Объекты, к которым предоставлен доступ через PUBLIC, становятся доступными всем пользователям в базе данных без явного предоставления прав.

1. **Добавление нового пользователя в текущую базу данных:**

Для добавления нового пользователя в текущую базу данных в PostgreSQL, вы можете использовать команду **CREATE USER**. Например:

1. CREATE USER имя\_пользователя WITH PASSWORD 'пароль';

1. **Позволение пользователю заходить на сервер:**

Пользователь должен иметь не только учетную запись в базе данных (CREATE USER), но и права на подключение к серверу. Это обычно выполняется с использованием команды **GRANT**:

1. GRANT CONNECT ON DATABASE имя\_базы\_данных TO имя\_пользователя;

1. **Виды прав доступа (привилегии) в PostgreSQL:**
   * **SELECT:** Разрешает выполнение операции SELECT (чтение данных) на таблицах или представлениях.
   * **INSERT:** Разрешает выполнение операции INSERT (вставка данных) в таблицы.
   * **UPDATE:** Разрешает выполнение операции UPDATE (обновление данных) в таблицах.
   * **DELETE:** Разрешает выполнение операции DELETE (удаление данных) из таблиц.
   * **USAGE:** Разрешает использование объекта, такого как схема, таблица, представление, последовательность и т. д.
   * **CREATE:** Разрешает создание новых объектов, таких как таблиц, представлений, индексов и т. д.
   * **ALTER:** Разрешает изменение структуры существующих объектов, таких как таблицы или представления.
   * **DROP:** Разрешает удаление объектов, таких как таблицы, представления, индексы и т. д.
   * **GRANT OPTION:** Разрешает предоставление (или отзыв) указанной привилегии другим пользователям.
   * **CONNECT:** Разрешает подключение к базе данных.
   * **TEMPORARY:** Разрешает создание временных объектов в базе данных.

Эти привилегии могут быть предоставлены пользователям или ролям с использованием команд GRANT и REVOKE.

1. **Исправить ошибки в обязательной части.**
2. **Сменить владельца базы данных.**

Для смены владельца базы данных используется команда ALTER DATABASE.

1. ALTER DATABASE имя\_базы\_данных OWNER TO новый\_владелец;

1. **Смена пароля для пользователя:**

* Для смены пароля пользователя используется команда **ALTER USER**. Например:

1. ALTER USER test WITH PASSWORD 'newPassword';

2.

16. **Определить роль с заданными правами.**

1. CREATE ROLE trole WITH LOGIN PASSWORD 'пароль' CREATEDB CREATEROLE;

2.

Здесь **LOGIN** разрешает пользователю вход на сервер, **CREATEDB** разрешает создание баз данных, а **CREATEROLE** разрешает создание других ролей. Укажите необходимые права в соответствии с вашими требованиями.

1. **CHECK OPTION в представлениях:**

* **CHECK OPTION** - это опция, которая может быть применена при создании представления в PostgreSQL. Она определяет условие, которое записи должны удовлетворять, чтобы быть включенными в представление при выполнении операции вставки или обновления через представление.
* Например, если у вас есть представление, которое выводит только строки, соответствующие определенному условию, вы можете использовать **CHECK OPTION** для того, чтобы гарантировать, что новые данные, вставляемые через представление, также будут соответствовать этому условию.
* Пример создания представления с **CHECK OPTION**:

1. CREATE VIEW my\_view AS SELECT \* FROM my\_table WHERE column\_name > 10 WITH CHECK OPTION;

2.

В этом примере **WITH CHECK OPTION** гарантирует, что при вставке или обновлении данных через это представление будут удовлетворять условию **column\_name > 10**.

1. **Модификация данных через представления:**

* В PostgreSQL представления могут использоваться для модификации данных в базе данных через операции вставки (INSERT), обновления (UPDATE) и удаления (DELETE). Однако есть некоторые ограничения, и не все представления могут быть модифицированы.
* Представление должно удовлетворять определенным требованиям, чтобы быть обновляемым. Например, представление должно содержать все ключевые столбцы таблицы, и оно не должно включать агрегатные функции или группировки.
* Пример обновления данных через представление:

1. UPDATE my\_view SET column\_name = 'новое\_значение' WHERE some\_condition;

2.

1. **Вставка данных через представления:**

* Вставка данных через представление возможна, если представление соответствует определенным критериям обновления. Вставка данных происходит как если бы данные вставлялись непосредственно в базовую таблицу, и представление служит просто абстракцией для удобства запросов.
* Пример вставки данных через представление:

1. INSERT INTO my\_view (column1, column2) VALUES ('значение1', 'значение2');

2.

Здесь **my\_view** - это представление, и данные вставляются в соответствующие столбцы. Однако, как уже упоминалось, есть ограничения на обновляемость представлений, и они должны быть учтены при модификации данных через них.