МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
 «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Кибербезопасность информационных систем»

Лабораторная работа №8

Выполнил: обучающийся гр. ВКБ43

Ковалев Данил Петрович

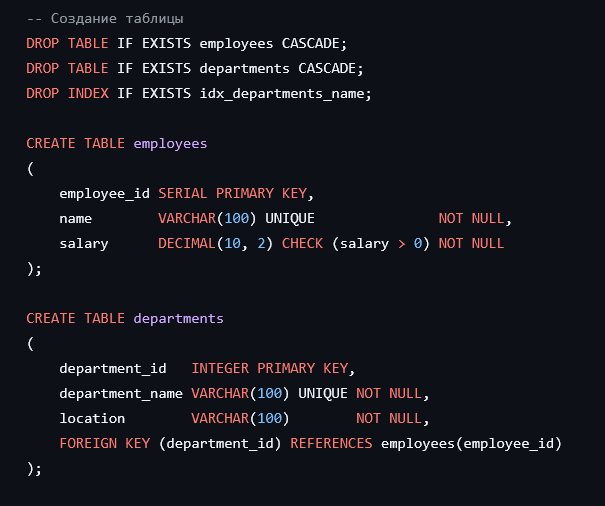
Проверил: Доцент Скляров А.В.

**Цель:** изучить принципы создания и использования индексов и ограничений в PostgreSQL для обеспечения целостности данных и повышения производительности запросов. А также добавить ограничения для обеспечения целостности данных и ограничить производительность запросов с использованием инструмента **EXPLAIN ANALYZE.**

**Вариант 6. Индивидуальное задание:** Создайте таблицу departments с полями department\_id, department\_name, location. Добавьте уникальный индекс на department\_name и внешний ключ department\_id, ссылающийся на таблицу employees. Проверьте влияние индексов на выборку по department\_name.

**Задание 1**. Создайте таблицы согласно вашему варианту, добавьте необходимые столбцы, типы данных и ограничения.

Создадим для начала таблицы, которые требуются по условию. В моем случае требуется создать 2 таблицы: departments, employees. Для первичных ключей использовался тип данных SERIAL, чтобы был автоматический автоинкеремент. Для таблицы departments был использован обычный INTEGER. Код представлен на рисунке 1.



**Рисунок 1 – таблицы для задания**

Рассмотрим ограничения, которые были выставлены. Для зарплаты – salary – было поставлено ограничение salary > 0, так как зарплата не может быть отрицательный, а также видно NOT NULL, говорящее нам о том, что поле обязательно должно заполняться. Имя сотрудника у уникальное и не может быть NULL. Для таблицы departments было выставлено ограничение на уникальность для departments\_name и NOT NULL, ну и location NOT NULL.

Departments связан с employes с помощью employee\_id, потому что каждый сотрудник может находиться только в 1 отделении, не более.

**Задание 2**. Заполните таблицы тестовыми данными, чтобы можно было оценить влияние индексов и ограничений на производительность и целостность данных.

Перейдем теперь к заполнению данных. Они были сгенерированы с помощью нейросети, приложу на рисунке 2 часть от данных, которыми заполняется база данных.

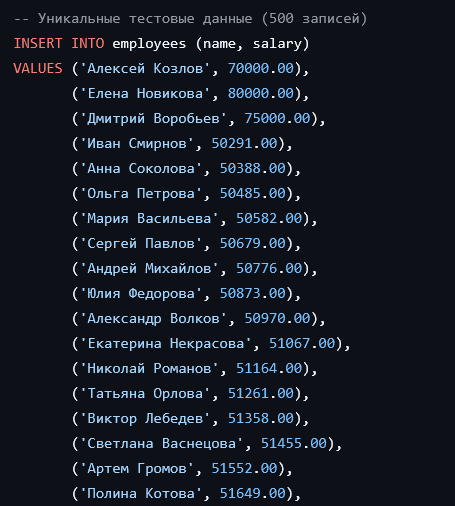


Рисунок 2 – часть тестовых данных для таблицы employess

Заполнение таблицы departments представлено на рисунке 3.



Рисунок 3 – заполнение таблицы departments

Запустим наш скрипт, проверим, что все данные были успешно созданы. В дальнейшем при анализе индексов тестовые данные, таблицы и ограничения меняться не будут. Будет изменение только в определении индекса, которые будут расписаны в дальнейшем.



Рисунок 4 – успешный прогон скрипта

Проверим, что ограничения работают правильно. Результат корректной работы представлен на рисунке 5.

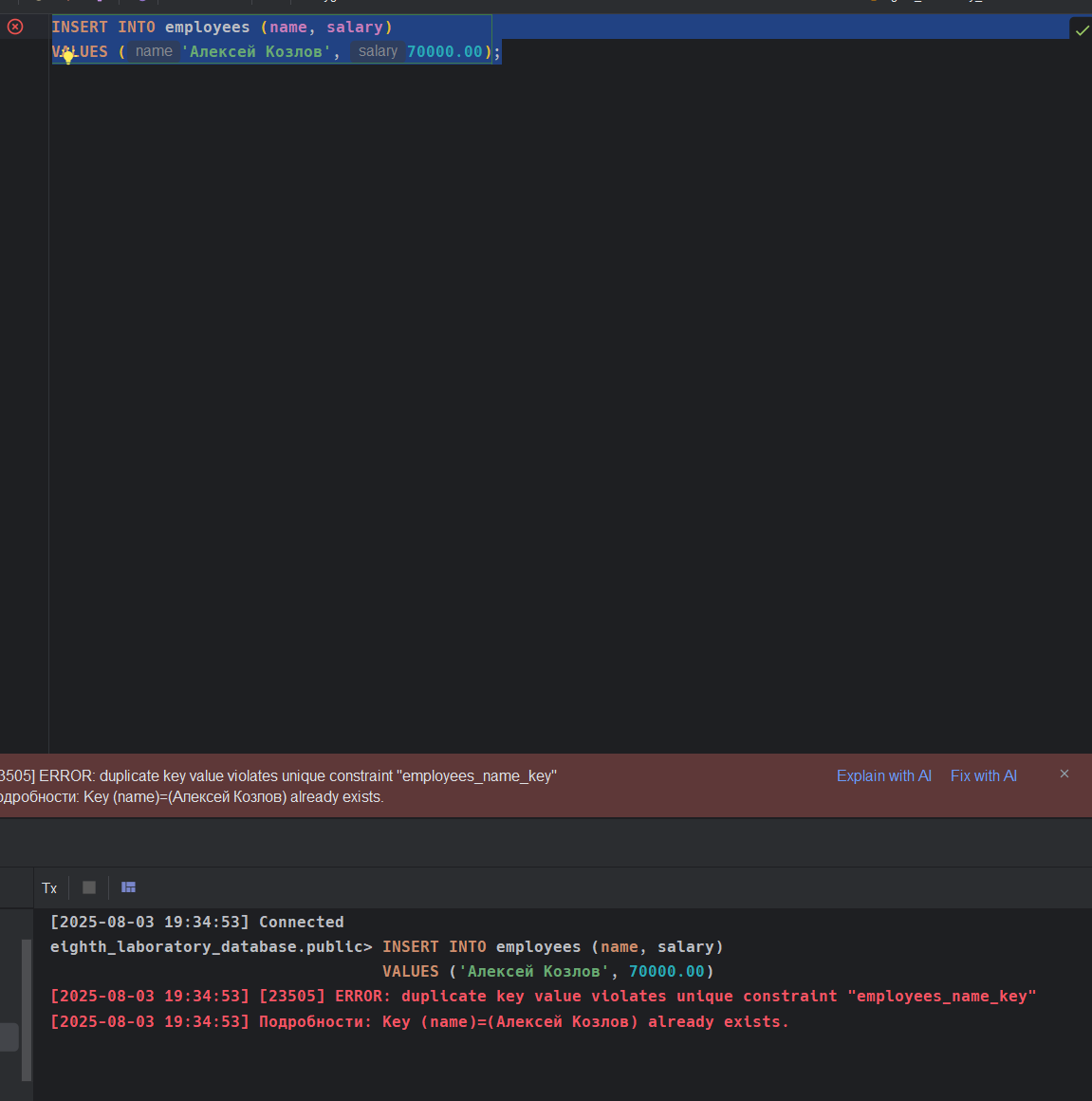


Рисунок 5 – ограничения на UNIQUE работает корректно

**Задание 3**. Создайте индексы для столбцов, указанных в вашем варианте. Используйте команды CREATE INDEX для настройки индексов.

1. Создание индекса типа **B-Tree** для столбца department\_name:

Проведем в начале тест для индекса B-Tree. Для создания индекса использовалась команда, которая представлена на рисунке 4.

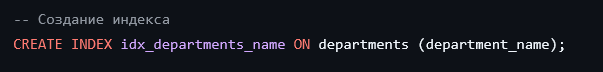


Рисунок 4 – создание индекса B-Tree

Теперь посмотрим процесс выполнения команды, задержки и какие решения принял оптимизатор PostgreSQL в процессе запроса. Команда для выполнения запроса представлена на рисунке 5.



Рисунок 5 – запуск EXPLAIN ANALYZE для запроса

В результате запуска запроса вышло то, что представлено на рисунке 6. Можно заметить, что больше всего задержка происходит на уровне оптимизатора, который предпринимает меры как лучше сделать запрос.

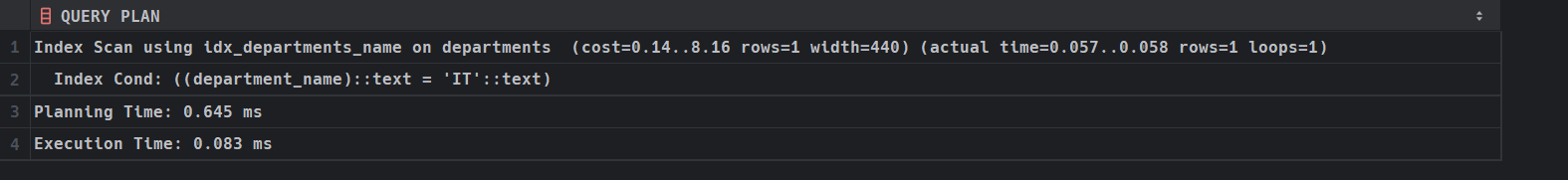


Рисунок 6 – процесс выполнения запроса

1. **Создание индекса типа Hash для столбца department\_name**

**Для создания индекса использовалась команда, которая представлена на рисунке 7.**

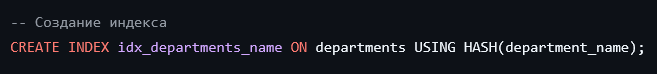


Рисунок 7 – создание индекса с типом HASH

Теперь посмотрим процесс выполнения команды, задержки и какие решения принял оптимизатор PostgreSQL в процессе запроса. Команда для выполнения запроса представлена на рисунке 8.



Рисунок 8 – запрос для EXPLAIN ANALYZE

В результате запуска запроса вышло то, что представлено на рисунке 9. Можно заметить, что больше всего задержка происходит на уровне оптимизатора, который предпринимает меры как лучше сделать запрос.

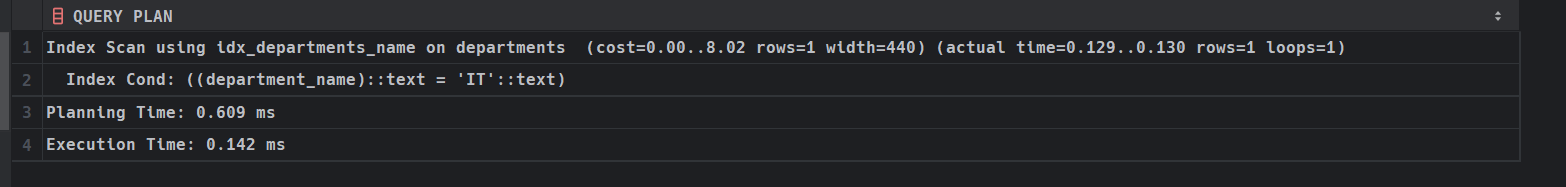


Рисунок 9 – процесс выполнения запроса

3. Создание индекса типа **Gin** для столбца department\_name.

Для создания индекса использовалась команда, которая представлена на рисунке 10.

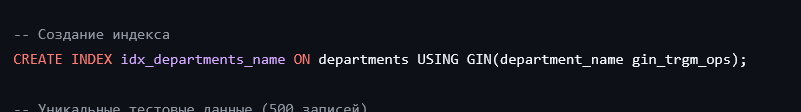


Рисунок 10 – создание индекса с типом Gin

Теперь посмотрим процесс выполнения команды, задержки и какие решения принял оптимизатор PostgreSQL в процессе запроса. Команда для выполнения запроса представлена на рисунке 11.



Рисунок 11 - запрос для EXPLAIN ANALYZE

В результате запуска запроса вышло то, что представлено на рисунке 12. Можно заметить, что больше всего задержка происходит на уровне оптимизатора, который предпринимает меры как лучше сделать запрос.

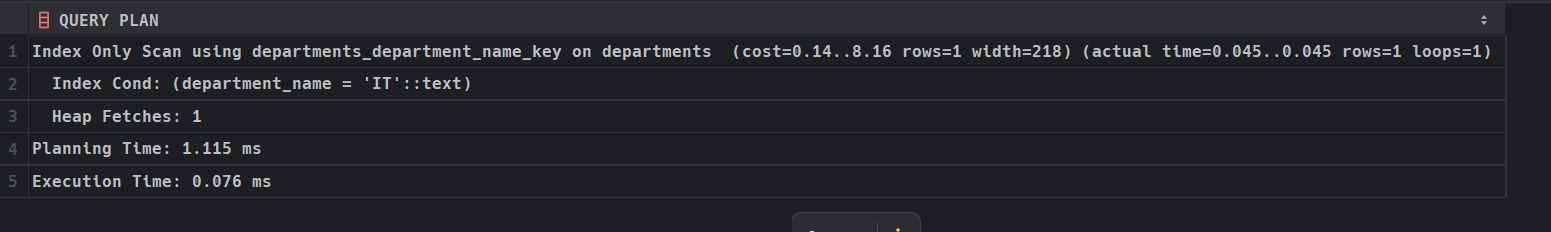


Рисунок 12 – процесс выполнения запроса

В этот раз при первом запросе используется Index Scan, но в дальнейшем используется обычный Seq Scan, указывая на то, что оптимизатор PostgreSQL не думает рациональным использование индекса.

4. Создание индекса **Gist** для столбца department\_name

Для создания индекса использовалась команда, которая представлена на рисунке 13.

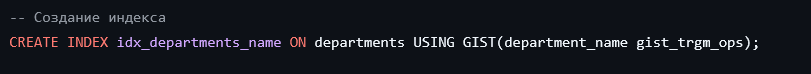


Рисунок 13 – создание Gist индекса

Теперь посмотрим процесс выполнения команды, задержки и какие решения принял оптимизатор PostgreSQL в процессе запроса. Команда для выполнения запроса представлена на рисунке 14.



Рисунок 14 – запрос для EXPLAIN ANALYZE

В результате запуска запроса вышло то, что представлено на рисунке 15. Можно заметить, что больше всего задержка происходит на уровне оптимизатора, который предпринимает меры как лучше сделать запрос.

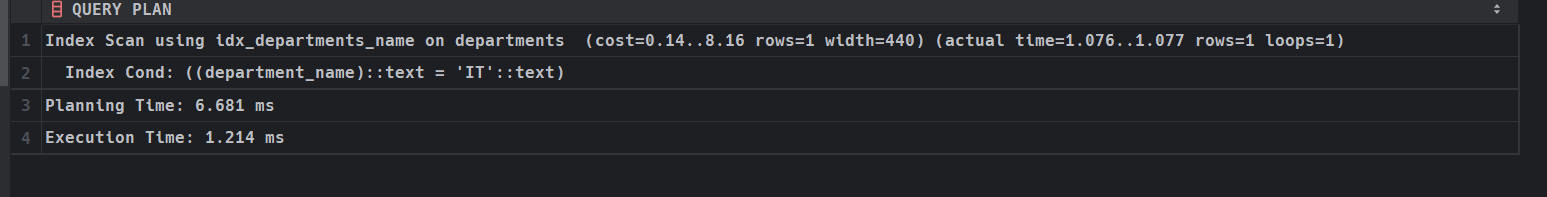


Рисунок 15 – процесс выполнения запроса

5. Создание индекса **Sp-Gist** для столбца department\_name

Для создания Sp-Gist индекса используется код, который представлен ниже на рисунке 16.



Рисунок 16 – создание SpGist индекса

Теперь посмотрим процесс выполнения команды, задержки и какие решения принял оптимизатор PostgreSQL в процессе запроса. Команда для выполнения запроса представлена на рисунке 17.



Рисунок 17 – запрос для EXPLAIN ANALYZE

В результате запуска запроса вышло то, что представлено на рисунке 18. Можно заметить, что больше всего задержка происходит на уровне оптимизатора, который предпринимает меры как лучше сделать запрос.

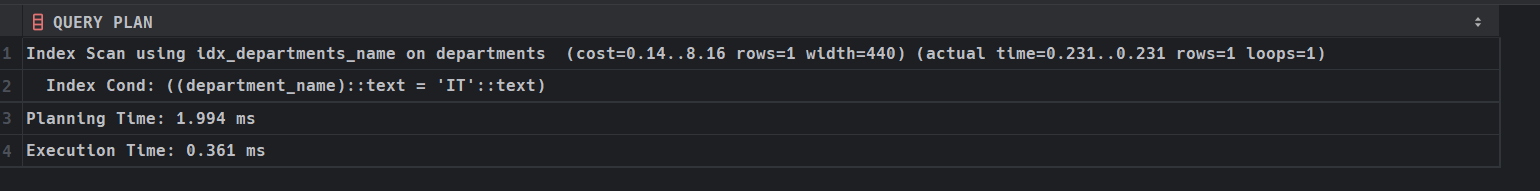


Рисунок 18 – процесс выполнения запроса

6. Создание **Brin** индекса для столбца department\_name

Для создания Brin индекса используется код, который представлен ниже на рисунке 19.



Рисунок 19 – создание Brin индекса

Теперь посмотрим процесс выполнения команды, задержки и какие решения принял оптимизатор PostgreSQL в процессе запроса. Команда для выполнения запроса представлена на рисунке 20.



Рисунок 20 – запрос для EXPLAIN ANALYZE

В результате запуска запроса вышло то, что представлено на рисунке 21. Можно заметить, что больше всего задержка происходит на уровне оптимизатора, который предпринимает меры как лучше сделать запрос.

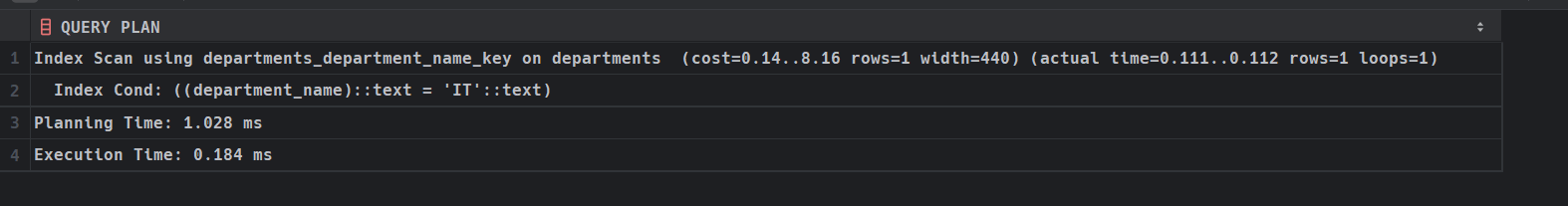


Рисунок 21 – процесс выполнения запроса

**Задание 4**. Проанализируйте, какие типы индексов подойдут для каждого столбца, и выберите наиболее подходящий тип индекса (например, B-Tree для сравнения значений, GIN для полнотекстового поиска).

Так как значение в нашем случае строковое, то кажется самым рациональным это использовать – это полнотекстовый поиск, так как Gin заточен лучше всего под данные действия. Опять-таки можно и обойтись обычным B-Tree, если нет потребности для полнотекстового поиска. Но если же мы будем проверять только по четкому равенству и значений будет очень много, то самым рациональным – это будет использование Hash. Если будет использоваться какой-то поиск по ключевым сообщениям департамента, то лучше всего использовать Gin индекс.

**Контрольные вопросы**

### 1. ****Что такое индекс в PostgreSQL, и как он влияет на производительность запросов?****

Индекс в PostgreSQL — это структура данных, которая помогает ускорить поиск и извлечение данных из таблиц. Он используется для оптимизации операций поиска, сортировки и фильтрации данных. Индексы позволяют базе данных находить строки с определенными значениями гораздо быстрее, чем при полном сканировании таблицы.

**Как индекс влияет на производительность**:

* **Преимущества**:
  + Ускоряет операции поиска, фильтрации, сортировки и соединения таблиц.
  + Снижает время выполнения запросов, особенно при работе с большими объемами данных.
* **Недостатки**:
  + Индексы требуют дополнительных ресурсов для их создания и хранения.
  + Они могут замедлить операции вставки, обновления и удаления данных, поскольку необходимо обновлять индекс при изменении данных.

### 2. ****Какие типы индексов существуют в PostgreSQL, и в каких случаях каждый из них наиболее эффективен?****

В PostgreSQL существует несколько типов индексов, каждый из которых подходит для различных типов запросов:

**B-tree**:

* + **Применение**: По умолчанию используется для индексации столбцов, которые часто используются в операциях сравнения, таких как =, >, <, >=, <=, BETWEEN.
  + **Эффективность**: Наиболее эффективен для поиска по диапазону значений и для простых сравнений.
  + **Пример**: Индексирование столбца с датами или числовыми значениями.

**Hash**:

* + **Применение**: Используется для индексации столбцов, которые часто используются в операциях равенства (=).
  + **Эффективность**: Менее эффективен для диапазонных запросов, но подходит для точных поисков.
  + **Пример**: Поиск записей по уникальному значению.

**GIN (Generalized Inverted Index)**:

* + **Применение**: Используется для индексации документов с полями, содержащими массивы, JSONB или текстовые данные для полнотекстового поиска.
  + **Эффективность**: Хорош для полнотекстового поиска, поиска по массивам или данным с несколькими значениями.
  + **Пример**: Индексация текстовых документов или JSONB данных.

**GiST (Generalized Search Tree)**:

* + **Применение**: Используется для индексации данных, которые не могут быть эффективно обработаны с помощью обычного B-tree индекса. Это могут быть географические данные, данные о похожих объектах и другие нетрадиционные типы данных.
  + **Эффективность**: Хорош для работы с геометрическими данными, деревьями и многими другими специфическими случаями.

**SP-GiST (Space-partitioned Generalized Search Tree)**:

* + **Применение**: Для индексации данных, которые могут быть разделены на несколько областей, таких как географические данные (например, точки в пространстве).
  + **Эффективность**: Подходит для высокоразмерных данных, например, для геопространственных запросов.

**BRIN (Block Range INdexes)**:

* + **Применение**: Используется для очень больших таблиц, где данные имеют порядок или могут быть эффективно объединены в блоки (например, временные метки или отсортированные данные).
  + **Эффективность**: Очень эффективен при работе с таблицами, содержащими огромные объемы данных, где данные физически упорядочены по определенному признаку.

### 3. ****Что такое ограничения целостности данных, и как они обеспечивают безопасность данных в базе данных?****

Ограничения целостности данных — это правила, которые обеспечивают точность, согласованность и достоверность данных в базе данных. Эти ограничения защищают базу данных от некорректных или неполных данных и помогают поддерживать правильные связи между таблицами.

**Типы ограничений**:

* **NOT NULL**: Обеспечивает, что столбец не может содержать пустые значения (NULL).
* **UNIQUE**: Гарантирует, что все значения в столбце или комбинации столбцов будут уникальными.
* **PRIMARY KEY**: Составляет уникальный идентификатор для каждой строки в таблице, сочетая ограничения NOT NULL и UNIQUE.
* **FOREIGN KEY**: Обеспечивает связь между таблицами, гарантируя, что значение в столбце соответствует значению в другом столбце (например, ссылка на другую таблицу).
* **CHECK**: Устанавливает проверку значений в столбце по заданному условию.
* **EXCLUSION**: Ограничение, предотвращающее перекрытие значений в столбцах, например, для работы с географическими данными.

**Как они обеспечивают безопасность данных**:

* Обеспечивают, что данные в базе данных остаются согласованными и целостными.
* Предотвращают неправильные или поврежденные данные.
* Помогают в соблюдении бизнес-логики, например, предотвратить вставку записей с неверными датами или несоответствующими внешними ключами.

### 4. ****Как можно использовать EXPLAIN ANALYZE для анализа производительности запросов?****

EXPLAIN ANALYZE позволяет получить подробную информацию о том, как PostgreSQL планирует и выполняет запрос. Это помогает анализировать, насколько эффективно работает запрос и какие части могут быть оптимизированы.

**Как использовать**:

Выполните запрос с командой EXPLAIN ANALYZE перед SQL-запросом. Например:

EXPLAIN ANALYZE

SELECT \* FROM shipments WHERE shipment\_date = '2024-11-01';

**Анализ вывода**:

* + **Index Scan**: Указывает на использование индекса для запроса.
  + **Seq Scan**: Указывает на полное сканирование таблицы (что может быть медленно, если таблица большая).
  + **Cost**: Оценка стоимости выполнения запроса.
  + **Rows**: Ожидаемое количество строк, которые будут возвращены.
  + **Actual Time**: Фактическое время выполнения запроса.

EXPLAIN ANALYZE помогает выявить узкие места в запросах, такие как неэффективные сканирования таблиц, которые могут быть улучшены с помощью индексов или изменений в запросе.

### 5. ****Какие преимущества и недостатки у индексов и ограничений с точки зрения производительности базы данных?****

**Преимущества индексов**:

* **Ускорение поиска**: Индексы позволяют значительно ускорить выполнение запросов, особенно при поиске по большим таблицам.
* **Ускорение сортировки**: Индексы могут улучшить производительность сортировки данных.
* **Повышение производительности соединений**: Индексы могут ускорить выполнение операций соединения (JOIN).

**Недостатки индексов**:

* **Дополнительное использование памяти**: Индексы требуют места на диске для хранения.
* **Снижение производительности при вставке, обновлении и удалении**: Каждое изменение данных в таблице требует обновления индекса, что может замедлить эти операции.

**Преимущества ограничений**:

* **Целостность данных**: Ограничения обеспечивают целостность данных и предотвращают внесение ошибочных данных.
* **Упрощение разработки**: Логика проверок данных реализуется на уровне базы данных, что снижает количество ошибок в приложении.

**Недостатки ограничений**:

* **Снижение производительности при вставке/обновлении**: Некоторые ограничения, такие как FOREIGN KEY, требуют дополнительной проверки данных, что может замедлить операции вставки и обновления.