Оптимизация процессов в PostgresSQL

Работа состоит из 2 взаимосвязанных разделов. В каждом разделе есть

теоретическая и практическая части лабораторной работы. В конце каждой

части есть задания для выполнения в рамках лабораторной работы.

**Цель:**

• Сформировать у студента понимание методов упрощения работы

аналитика с БД.

**Задачи:**

• Получить теоретические знания о функция.

• Подробнее узнать о языке PL/pgSQL.

• Узнать основу синтаксиса языка.

• Ознакомится с операторами, управляющими конструкциями.

• Получить знания о курсорах и обработки исключений.

• Научится использовать вышеописанные навыки для написания

собственной функции.

• Анализ эффективности запросов.

• Узнать о индексах и их влияния на оптимизацию.

• Изучить функции для полнотекстового поиска.

**Практическое задание для первой части ЛР**

Первая часть заключается в использовании функций. При выполнении

задания необходимо:

• Составить SQL-скрипты для создания нескольких функций,

упрощающих манипуляции с данными.

• Продемонстрировать полученные знания о возможностях языка

PL/pgSQL. В скриптах должны использоваться:

o Циклы.

o Ветвления.

o Переменные.

o Курсоры.

o Исключения.

• Обосновать преимущества механизма функций перед механизмом

представлений.

**1. Функция для подсчета среднего опыта работы преподавателей на факультете:**

1. CREATE OR REPLACE FUNCTION calculate\_avg\_experience(p\_faculty\_id INTEGER)

2. RETURNS NUMERIC AS $$

3. DECLARE

4. total\_experience NUMERIC := 0;

5. num\_professors INTEGER := 0;

6. academic\_record RECORD;

7. BEGIN

8. FOR academic\_record IN SELECT PExperience FROM academic WHERE faculty\_id = p\_faculty\_id

9. LOOP

10. total\_experience := total\_experience + academic\_record.PExperience;

11. num\_professors := num\_professors + 1;

12. END LOOP;

13.

14. IF num\_professors = 0 THEN

15. RAISE EXCEPTION 'No professors in the specified faculty';

16. END IF;

17.

18. RETURN total\_experience / num\_professors;

19. EXCEPTION

20. WHEN OTHERS THEN

21. RAISE EXCEPTION 'Error calculating average experience: %', SQLERRM;

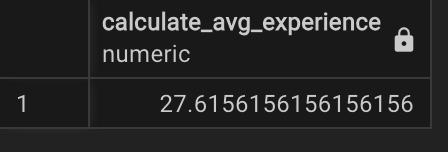
22. END;

23. $$ LANGUAGE plpgsql;

24.

1. SELECT calculate\_avg\_experience(1);

2.



**2. Процедура для перевода студента на следующий курс:**

1. CREATE OR REPLACE PROCEDURE promote\_student(p\_student\_id INTEGER)

2. LANGUAGE plpgsql AS $$

3. DECLARE

4. current\_year INTEGER;

5. BEGIN

6. SELECT nyear INTO current\_year FROM student WHERE id\_student = p\_student\_id;

7.

8. IF current\_year IS NOT NULL THEN

9. UPDATE student SET nyear = current\_year + 1 WHERE id\_student = p\_student\_id;

10. ELSE

11. RAISE EXCEPTION 'Student not found';

12. END IF;

13. END;

14. $$ LANGUAGE plpgsql;

15.



1. CALL promote\_student(399087078);

2.



**3. Функция для получения списка предметов, преподаваемых конкретным преподавателем:**

1. CREATE OR REPLACE FUNCTION get\_subjects\_taught\_by\_academic(p\_academic\_id INTEGER)

2. RETURNS TABLE (subject\_name VARCHAR(20), hours NUMERIC) AS $$

3. BEGIN

4. RETURN QUERY SELECT s.sname, s.hours

5. FROM academics\_subject a\_s

6. JOIN subject s ON a\_s.subject\_id = s.id\_subject

7. WHERE a\_s.academic\_id = p\_academic\_id;

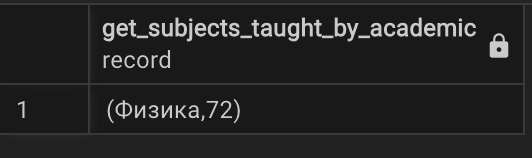
8. END;

9. $$ LANGUAGE plpgsql;

10.

1. select get\_subjects\_taught\_by\_academic(10);

2.



**Обосновать преимущества механизма функций перед механизмом**

**представлений.**

Механизм функций в SQL предоставляет эффективность выполнения запросов на стороне сервера, что обеспечивает лучшую производительность при работе с объемными данными. Повторное использование кода становится проще благодаря возможности создания и многократного применения функций для различных запросов. Функции поддерживают параметры, делая запросы более гибкими и адаптивными. В функциях легче реализовывать сложную логику бизнес-процессов, включая условия и обработку ошибок. Кроме того, механизм функций обеспечивает доступ к системным объектам и предоставляет более гибкий контроль над правами доступа, что способствует обеспечению безопасности данных.

**Практическое задание для второй части ЛР**

Девятое практическое задание посвящено ускорению выполнения

запросов. Для этого могут быть использованы механизмы секционирования,

наследования и индексов. Для выполнения задания необходим достаточно

большой объем данных, чтобы оптимизация была целесообразной (порядка 1

млн. строк в каждой таблице).

Необходимо подготовить два запроса:

• Запрос к одной таблице, содержащий фильтрацию по нескольким полям.

• Запрос к нескольким связанным таблицам, содержащий фильтрацию по

нескольким полям.

Для каждого из этих запросов необходимо провести следующие шаги:

• Получить план выполнения запроса без использования индексов.

• Получить статистику (IO и Time) выполнения запроса без использования

индексов.

• Создать нужные индексы, позволяющие ускорить запрос.

• Получить план выполнения запроса с использованием индексов и

сравнить с первоначальным планом.

• Получить статистику выполнения запроса с использованием индексов и

сравнить с первоначальной статистикой.

• Оценить эффективность выполнения оптимизированного запроса.

Также необходимо продемонстрировать полезность индексов для

организации полнотекстового поиска, фильтрации с использованием массива

и json-формата.

Для таблицы объемом больше 100 млн. записей произвести

оптимизацию, позволяющую быстро удалять старые данные, ускорить вставку

и чтение данных.

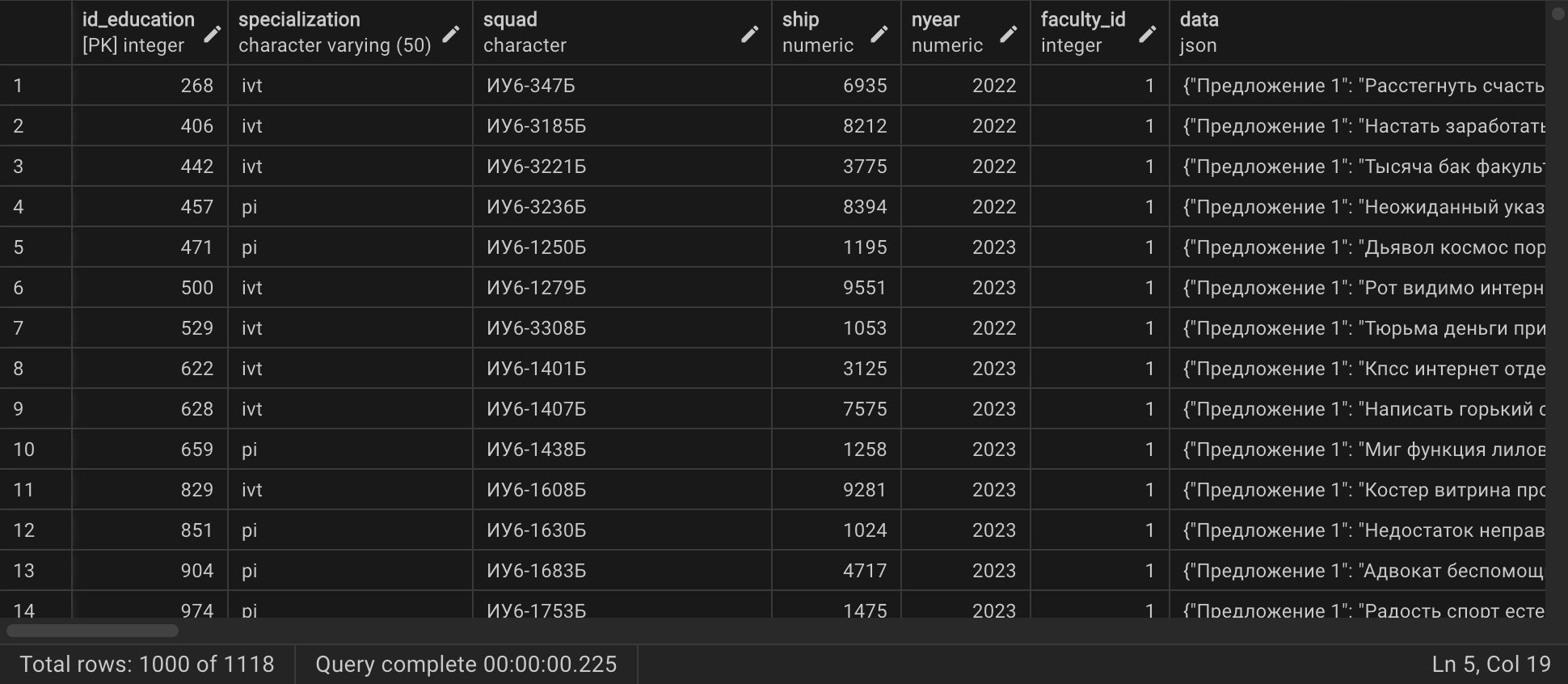
**Запрос 1**

1. SELECT \*

2. FROM education

3. WHERE id\_education < 50000 AND datatext ~ 'интернет';

4.





**Создадим индекс и повторим запрос**

1. CREATE INDEX id\_educationidx ON education ( id\_education );

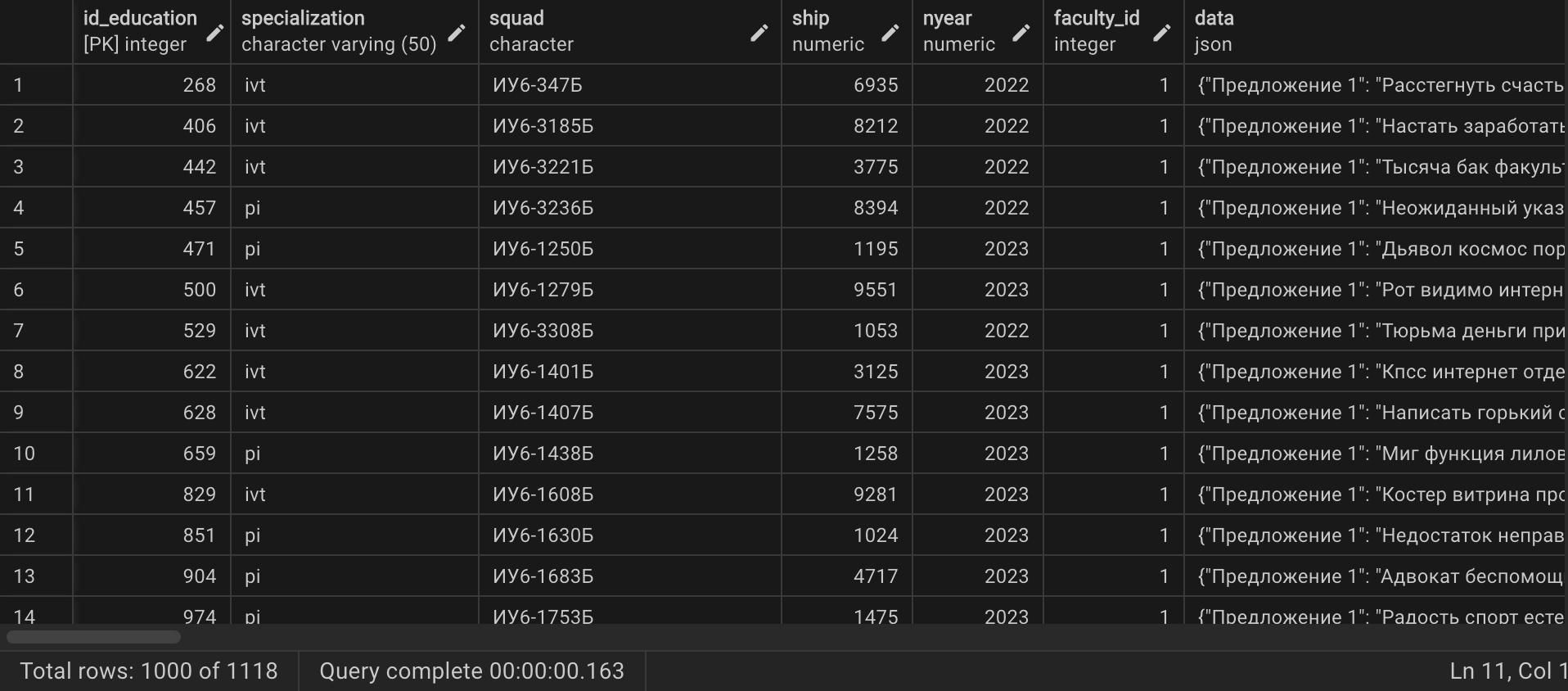
2.

1. SELECT \*

2. FROM education

3. WHERE id\_education < 50000 AND datatext ~ 'интернет';

4.





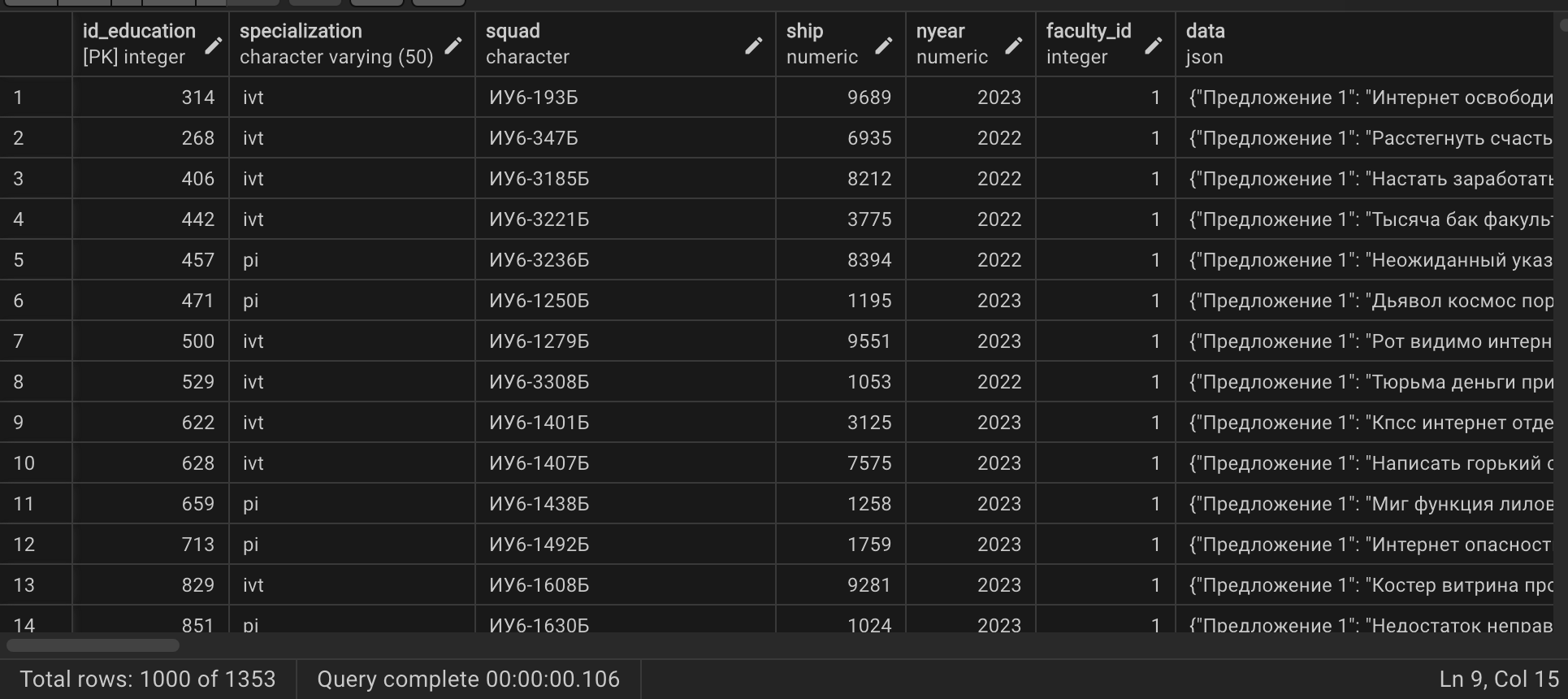
**Используем полнотекстовый поиск**

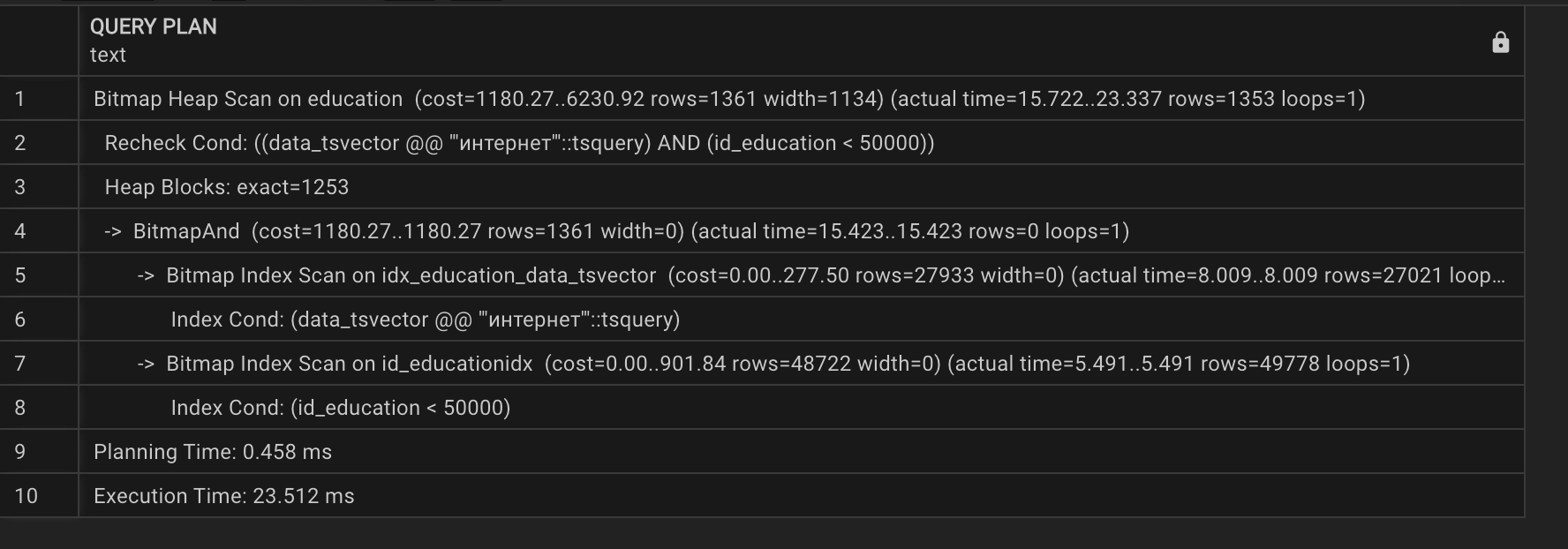
1. SELECT \*

2. FROM education

3. WHERE id\_education < 50000 AND data\_tsvector @@ to\_tsquery('russian', 'интернет');

4.





1. ALTER TABLE education ADD COLUMN datajsonb jsonb;

2. UPDATE education SET datajsonb = datajson::jsonb;

3. ALTER TABLE education DROP COLUMN datajson;

4. ALTER TABLE education RENAME COLUMN datajsonb TO datajson;

5.

**Запрос 2**

1. SELECT sname, squad, datatext

2. FROM education

3. LEFT JOIN student ON education.id\_education = student.education\_id

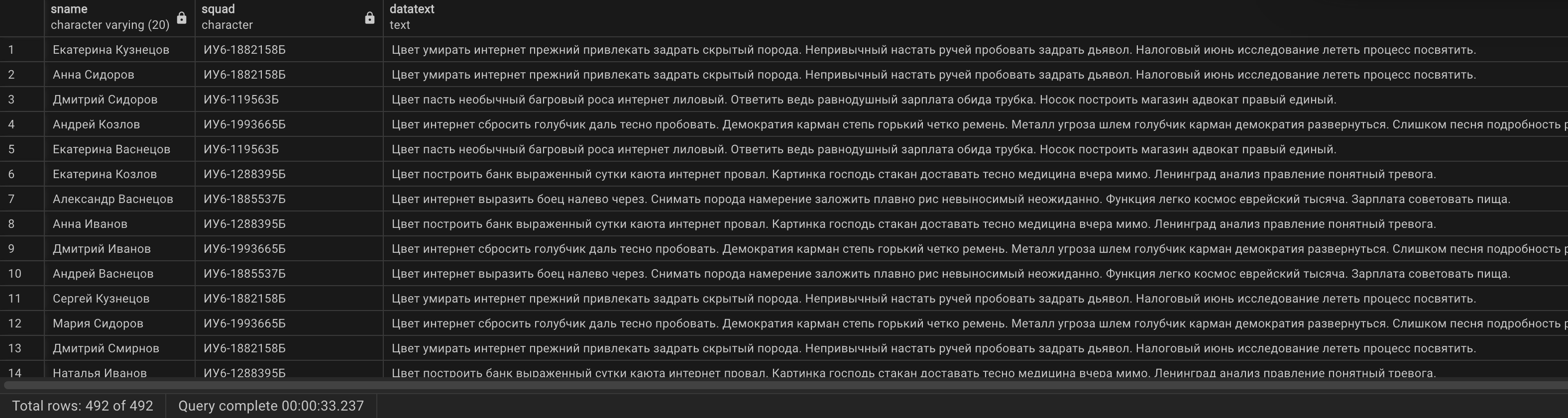
4. WHERE

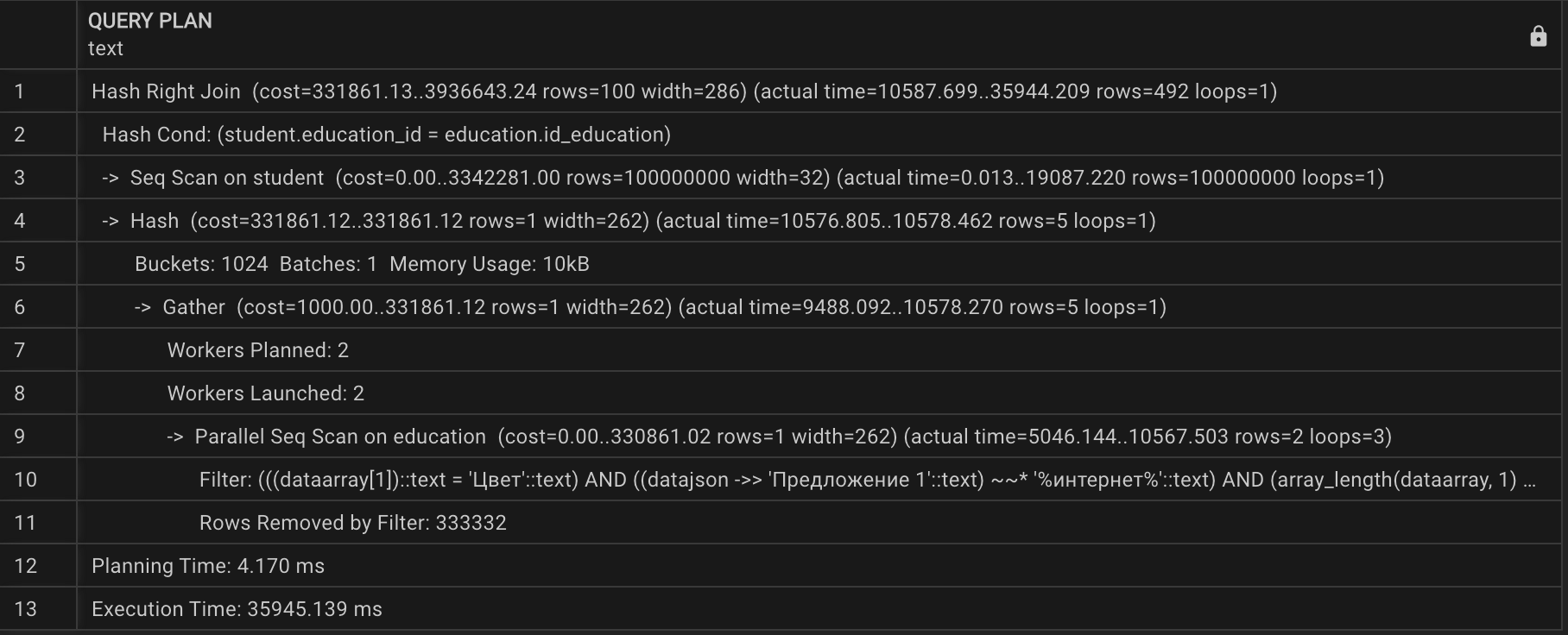
5. datajson->>'Предложение 1' ILIKE '%интернет%' AND

6. dataarray[1] = 'Цвет' AND

7. array\_length(dataarray, 1) > 17;

8.





**Создадим индексы и повторим запрос**

1. CREATE INDEX idx\_json\_column ON education USING GIN (datajson jsonb\_path\_ops);

2. CREATE INDEX idx\_array\_column ON education USING btree (dataarray);

3.

1. SELECT sname, squad, datatext

2. FROM education

3. LEFT JOIN student ON education.id\_education = student.education\_id

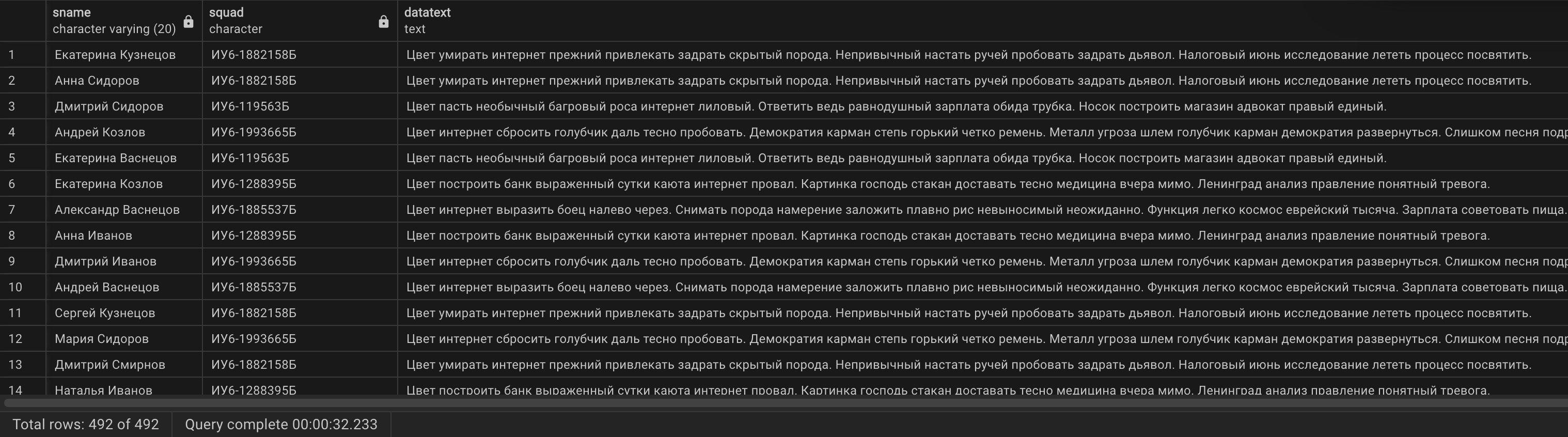
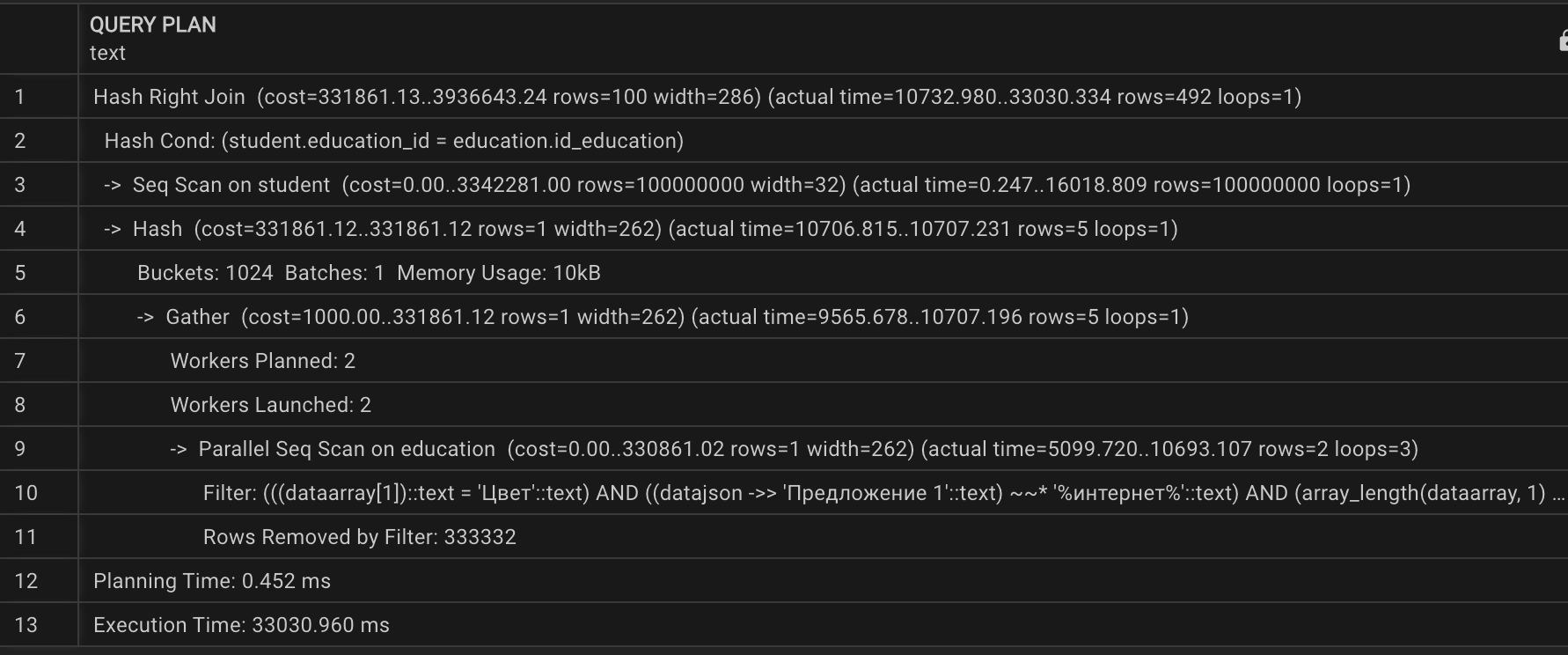
4. WHERE

5. datajson->>'Предложение 1' ILIKE '%интернет%' AND

6. dataarray[1] = 'Цвет' AND

7. array\_length(dataarray, 1) > 17;

8.

**Вопросы для самостоятельного изучения**

**1. Целесообразность создания функций в SQL:**

* **Сложные бизнес-правила:** Используйте функции при наличии сложных бизнес-правил, которые требуют повторного использования.
* **Параметризация запросов:** Функции подходят, когда необходима параметризация запросов для адаптации под различные сценарии.
* **Эффективность выполнения:** Создавайте функции для выполнения операций на стороне сервера и обеспечения лучшей производительности.
* **Логика бизнес-процессов:** Используйте функции для реализации сложной логики бизнес-процессов, таких как обработка ошибок и циклов.

2. **Курсоры в SQL:**

* **Итерация по результатам запроса:** Курсоры используются для итерации по результатам запроса и последовательной обработки каждой строки.
* **Обработка одной строки за раз:** Курсоры позволяют обрабатывать одну строку данных за раз, что может быть полезно при сложной логике обработки.
* **Работа с наборами данных:** Курсыры подходят для работы с наборами данных, где необходима более тщательная обработка, чем с использованием обычных запросов.

**3.** **Работа с циклами в SQL:**

* **Использование WHILE и FOR:** SQL поддерживает циклы WHILE и FOR, которые могут быть использованы для многократного выполнения блока кода в зависимости от условия.
* **Обработка множества данных:** Циклы полезны при обработке множества данных, особенно в сочетании с условиями, что обеспечивает гибкость при написании запросов.

4. **Отличие первичного ключа и уникального индекса:**

* **Уникальность значений:** Оба гарантируют уникальность значений в столбце или наборе столбцов.
* **Null значения в первичном ключе:** Первичный ключ не допускает NULL значений, тогда как уникальный индекс может содержать одно NULL значение.
* **Ограничение на количество:** Таблица может иметь только один первичный ключ, но может содержать несколько уникальных индексов.

1. **Создание индексов в SQL:**
   * **Ускорение поиска и сортировки:** Индексы целесообразно создавать для ускорения операций поиска и сортировки данных в таблицах.
   * **Оптимизация запросов:** Индексы помогают оптимизировать выполнение запросов, особенно при работе с большими объемами данных.
   * **Повышение производительности:** Использование индексов может значительно повысить производительность запросов, особенно в случае сложных запросов и соединений.
2. **Выбор колонок для индексов:**
   * **Колонки, используемые в условиях WHERE:** Включайте в индексы те колонки, которые часто используются в условиях WHERE для фильтрации данных.
   * **Колонки в операторах JOIN:** Если ваши запросы часто используют операторы JOIN, рассмотрите включение в индексы колонок, участвующих в соединениях.
   * **Колонки в ORDER BY и GROUP BY:** Для оптимизации сортировки и группировки данных включайте соответствующие колонки в индексы.
3. **Способы внутренней организации индексов:**
   * **Кластеризованные и некластеризованные:** Индексы могут быть кластеризованными, где данные в таблице фактически упорядочиваются согласно индексу, и некластеризованными, где индекс и данные хранятся отдельно.
   * **Бинарные и деревянные структуры:** Индексы могут использовать бинарные (например, хеш-индексы) или деревянные структуры (например, B-деревья) для ускорения поиска данных.
4. **Проблема фрагментации индексов и ее решение:**
   * **Фрагментация:** Фрагментация индексов может снижать производительность, увеличивая время выполнения операций поиска и сортировки.
   * **Борьба с фрагментацией:** Решения включают перестроение индексов, использование инструкции REORGANIZE или REBUILD (в зависимости от СУБД), а также регулярное обслуживание индексов для предотвращения накопления фрагментации. Также можно рассмотреть использование разделенных индексов для уменьшения влияния фрагментации.
5. **Порядок колонок при создании индекса:**
   * **Имеет значение:** Да, порядок колонок в индексе имеет значение, особенно при создании составных (composite) индексов. Порядок влияет на оптимизацию выполнения запросов, особенно если используется составной индекс.
6. **Разница между Index Scan и Index Seek:**
   * **Index Scan:** Полный скан индекса, при котором весь индекс просматривается для поиска соответствующих строк.
   * **Index Seek:** Прямой поиск, при котором используется индекс для быстрого локализации конкретных строк в данных. Index Seek обычно более эффективен.
7. **Разница между секционированием и наследованием:**
   * **Секционирование:** Это разделение таблицы на отдельные секции для улучшения производительности и управления данными.
   * **Наследование:** Это механизм организации таблиц в иерархическую структуру, где дочерние таблицы наследуют свойства и структуру от родительских таблиц.
8. **ANALYZE в SQL:**
   * **Необходим для оптимизации запросов:** ANALYZE используется для сбора статистики о данных в таблицах, что помогает оптимизатору запросов принимать более информированные решения при создании планов выполнения запросов.
9. **Возможные негативные эффекты от индексов:**
   * **Переиндексация:** Периодическая переиндексация может занимать ресурсы и замедлять работу системы.
   * **Избыточные индексы:** Излишние или неэффективные индексы могут занимать место и замедлять операции вставки, обновления и удаления.
10. **Влияние порядка сортировки при создании индекса:**
    * **ASC и DESC:** Порядок сортировки (ASC или DESC) влияет на оптимизацию выполнения запросов. ASC (по умолчанию) обеспечивает сортировку по возрастанию, а DESC — по убыванию. Влияние можно продемонстрировать при сортировке и фильтрации данных в запросах.Начало формы

Конец формы