Кафедра ЦТ
Институт информационных технологий
РТУ МИРЭА

Дисциплина «Разработка баз данных»

Практическая работа №8. Сложные типы данных и масштабирование в PostgreSQL

Постановка задачи: основываясь на индивидуальной схеме данных, составьте необходимые запросы.

Задание №1: хранение паролей (BYTEA)

Создайте справочник ролей и таблицу пользователей с полями для «сырого» пароля и хеша типа BYTEA, сохранив значения через функцию digest.

Напишите запрос для проверки соответствия введенного пароля сохраненному хешу с демонстрацией успешного и неуспешного сценариев.

Задание №2: секционирование таблицы логов

Реализуйте секционированную таблицу логов с использованием стратегии RANGE по полю даты.

Необходимо создать отдельные секции для 2-го полугодия 2024 и 1-го полугодия 2025 года, а также дефолтную секцию для данных, выходящих за эти рамки.

(продолжение на следующем слайде)

Практическая работа №8. Сложные типы данных и масштабирование в PostgreSQL

Постановка задачи: основываясь на индивидуальной схеме данных, составьте необходимые запросы.

Задание №3: генерация данных

Сгенерируйте скриптом PL/pgSQL 20 000 записей с различающейся JSON-структурой, даты которых охватывают все созданные диапазоны. Выведите результат выборкой, подтверждающей физическое распределение строк по разным таблицам-секциям.

Задание №4: анализ и поиск по JSONB

Напишите запросы для поиска записи по точному значению ключа и фильтрации данных с использованием операторов сравнения внутри JSONB.

Также выполните агрегацию числового поля из JSON-структуры, например, рассчитав сумму или среднее значение.

Задание №5: модификация данных JSONB

Выполните массовое добавление нового поля во все записи определенного типа.

Дополнительно продемонстрируйте точечное изменение значения конкретного ключа в одной выбранной записи.

(продолжение на предыдущем слайде)

BYTEA и ХЕШИРОВАНИЕ

BYTEA (Byte Array) и Хеширование

BYTEA или Byte Array – это тип данных для хранения бинарных строк (последовательности байт).

В отличие от **TEXT**, не зависит от кодировки и может хранить любые данные (хеши, картинки, файлы).

Хеширование vs Шифрование

- ✓ **Шифрование** – **обратимый** процесс. Есть **ключ** для *шифровки* и *расшифровки*.
- ✓ **Хеширование** – **односторонний** процесс. Из **хеша невозможно *напрямую*** восстановить исходный пароль.

Что такое «Соль», и зачем она нужна

Соль – это добавление **дополнительных символов** в **пароль** перед хешированием.

Соль может быть **уникальной** для **каждого пароля** и храниться в той же таблице *(делая даже **одинаковые пароли разными**)*, и/или **генерироваться** кодом по определённому **алгоритму**, **добавляя символы** в любые части пароля.

Соль необходима **для защиты от атак** через ***«радужные таблицы»*** *(заранее посчитанные хеши)*.

BYTEA (Byte Array) и Хеширование – синтаксис

Функция **digest()** – создает **бинарный хеш из строки** по **указанному алгоритму**.

```
digest('MyPassword', 'sha256')
```

Функция **encode()** – преобразует **бинарные данные** в **читаемый текстовый формат** (обычно hex).

```
encode(password_hash, 'hex')
```

Пример проверки пароля: здесь сравнение происходит **не по тексту пароля**, а **по его хешу**.

```
SELECT username, password_raw,
   encode(password_hash, 'hex') AS password_hash,
   (digest(password_raw, 'sha256') = password_hash) as is_valid_check
FROM system_users;
```

ВАЖНО: хранение в **реальных** проектах **незашифрованных паролей КАТЕГОРИЧЕСКИ ЗАПРЕЩЕНО**!!!

СЕКЦИОНИРОВАНИЕ (PARTITIONING)

Секционирование (Partitioning)

Секционирование – это разделение **одной большой логической** таблицы на **несколько физических** **таблиц** *(секций)*.

Как работает Partition Pruning *(отсечение секций)*:

- **планировщик запросов** анализирует, что указано в **WHERE** запроса;
- если **запрос ищет данные** за ***«Январь 2025»***, PostgreSQL читает **только соответствующую таблицу-секцию**, при этом **остальные файлы** на диске **игнорируются**.

Это дает колоссальный прирост производительности на больших объемах.

Отличие от VIEW

В отличие от **представлений**, **секционирование** – это **физическое разделение** **данных**, а не просто **логическая абстракция**.

Секционирование (Partitioning) – синтаксис

Синтаксис секционирования состоит из двух основных частей.

1. **Родительская таблица** – определяет **схему данных** и **метод разбиения** (***RANGE***, ***LIST***, ***HASH***).

ВАЖНО: в родительской таблице **данные НЕ ХРАНЯТСЯ**.

```
CREATE TABLE user_logs (
   ...
   created_at TIMESTAMPTZ NOT NULL,
   ...
) PARTITION BY RANGE (created_at);
```

2. **Дочерние таблицы *(секции)*** – **хранят** реальные данные. Их задача – чётко **описывать диапазоны**.

```
CREATE TABLE user_logs_2024_h2 PARTITION OF user_logs
FOR VALUES FROM ('2024-07-01') TO ('2025-01-01');
```

ВАЖНО: верхняя граница (**TO** …) **не включается** в диапазон.

Секционирование (Partitioning) – важные правила

- **Секция DEFAULT**

  Обязательна для стратегии **RANGE**, если есть вероятность появления данных **за пределами описанных диапазонов** *(например, будущие даты)*.

  Без неё попытка **вставки данных «мимо» секций** приведет к **ошибке транзакции**.

```
CREATE TABLE user_logs_default PARTITION OF user_logs DEFAULT;
```

- **Индексы создаются** на **родительской таблице** и **автоматически наследуются** всеми **существующими** и **будущими секциями**.

- **Ограничения (Constraints)**

  PRIMARY KEY или **UNIQUE** ограничения в секционированной таблице **обязаны включать ключ секционирования** *(в данном случае – created_at)*.

ХРАНЕНИЕ JSON
(JSONB)

JSON vs JSONB

В PostgreSQL есть **два типа данных** для хранения **JSON**.

JSON – тип данных, позволяющий хранить стандартный JSON.

- Хранит данные **как текст**.
- **Сохраняет форматирование** и **пробелы**.
- **Медленный поиск** *(нужен парсинг каждый раз)*.

JSONB (binary) – специальный тип данных PostgreSQL, предпочтительный для 99% задач с JSON.

- Хранит данные в **разложенном бинарном формате** – т.е. в специальной **оптимизированной структуре**.
- **Поддерживает индексы *(GIN)***.
- **Не сохраняет порядок ключей** и **лишние пробелы**.
- Позволяет **хранить атрибуты**, которые **меняются** от записи к записи
  (например, разные поля для события 'login' и 'sale').

Работа с JSONB – операторы навигации

Самая **частая ошибка** – путаница между **получением объекта** и **получением текста**.

- **Оператор `->`** *(вернуть **ОБЪЕКТ**)*

  Используется для **навигации «вглубь»** структуры (**объект внутри объекта**). **Результат** – **JSONB**.

  ```
  event_data -> 'details' вернет {"qty": 5}
  ```

- **Оператор `->>`** *(вернуть **ТЕКСТ**)*

  Используется на **последнем шаге** для получения значения. **Результат** – **TEXT**.

  ```
  event_data ->> 'event' вернет 'sale'
  ```

ВАЖНО: для **сравнения с числами** результат нужно **приводить к типу**:

```
(data ->> 'sum')::numeric > 100
```

Работа с JSONB – поиск и фильтрация

- **Оператор вхождения @>** (***containment***) – проверяет, содержит ли **JSON слева структуру**, указанную **справа**.

  Это **«подмножество»**, т.е. **проверяются** только **указанные ключи** и **значения**, **остальные игнорируются**.

Пример: найти все записи, где внутри **details** количество **quantity = 5**.

```
SELECT * FROM user_logs
WHERE event_data @> '{"details": {"quantity": 5}}';
```

ВАЖНО: такие запросы могут использовать GIN-индекс, что делает их крайне быстрыми, по сравнению с обычным текстовым поиском.

- **Оператор проверки наличия ключа ?** – проверяет, есть ли в **JSONB**-объекте *(или массиве)* **указанный ключ/элемент**.

  `WHERE event_data ? 'ip'` – **TRUE**, если в поле **event_data** **есть ключ ip**.

Работа с JSONB – модификация данных

В PostgreSQL есть **два способа модификации** данных типа **JSONB**

- **Оператор конкатенации** `||` – **объединяет** два **JSON-объекта**.

  Если **ключ существует** – он **перезаписывается**, **если нет** – **добавляется**.

```
SET event_data = event_data || '{"source": "web"}'::jsonb;
```

- **Точечное изменение** `jsonb_set`

  Изменяет значение глубоко **внутри структуры**, **не затрагивая остальные поля**.

```
jsonb_set(
    target_json,          -- Изменяемое поле (название столбца)
    '{path, to, key}',    -- Путь как массив строк
    'new_value'::jsonb    -- Новое значение (обязательно jsonb)
)
```

Спасибо за внимание