УДК 004
https://doi.org/10.36906/KSP-2022/61

Широков И.А.
ORCID: 0000-0002-4106-6810
Университет ИТМО
г. Санкт-Петербург, Россия

ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ ХРАНЕНИЯ И ОБРАБОТКИ МНОГОУРОВНЕВЫХ ДАННЫХ В PostgreSQL

Аннотация. В статье анализируются проблемы, связанные с замещением в Российской Федерации компаний-поставщиков баз данных. В качестве альтернативы рассматривается open source база данных PostgreSQL, а также анализируется JSON формат как один из примеров хранения данных, позволяющий создавать уровни хранения. Далее, исследуется возможность хранения и обработки многоуровневых данных в виде JSON-объектов в PostgreSQL.

Ключевые слова: база данных; многоуровневые данные; JSON; PostgreSQL; pgsql.

Shirokov I.A.
ORCID: 0000-0002-4106-6810
ITMO University
Saint-Petersburg, Russia

RESEARCH OF THE POSSIBILITIES OF STORING AND PROCESSING MULTILEVEL DATA IN PostgreSQL

Abstract. The article analyzes the problems associated with the substitution of database companies in the Russian Federation. Open source database PostgreSQL is considered as an alternative, and JSON format is analyzed as one of the examples of data storage, which allows to create storage layers. Further, the possibility of storing and processing tiered data in the form of JSON objects in PostgreSQL is investigated.

Key words: database; storage layerss; JSON; PostgreSQL; pgsql.

На сегодняшний день существует большое количество реляционных и нереляционных систем управления базами данных (СУБД). Целевое применение данных СУБД часто бывает разным. Оно зависит от стека разработки, возможности обслуживания, доступности и других факторов.

В текущих реалиях на территории Российской Федерации осталось немного вариантов, способных полностью покрыть нужды разработчиков в проектировании баз данных для

проектов компаний. К компаниям, прекратившим поддержку и распространение своих продуктов, относятся такие компании как Oracle, Microsoft и MongoDB [2, с. 154].

Степень использования продуктов вышеуказанных компаний постепенно уменьшается за счет перехода на аналогичные продукты компаний, которые не покинули и не собираются покидать российский рынок, и на продукты отечественных компаний. В случае с реляционными СУБД выбор стоит между PostgreSQL, разработанный российской компанией Postgres Professional [1, с. 8] и Greenplum. Greenplum является доработкой PostgreSQL как более распределенная СУБД, что делает Greenplum легким для перехода разработчиков, которые ранее использовали PostgreSQL в своих проектах.

Несмотря на то, что Greenplum – проект компании Dell, его исходный код, как и у базового PostgreSQL остается открытым, что делает его предпочтительным для перехода без риска окончания поддержки или разных ограничений.

Так как Greenplum и PostgreSQL имеют идентичный синтаксис, pgsql – язык запросов и pl/pgsql – язык программирования, то в статье будут описаны примеры, выполненные на чистом PostgreSQL.

Как уже отмечалось выше, компании-разработчики реляционных и нереляционных СУБД уходят из России, но, если проблема ухода реляционных СУБД решается с помощью перехода на PostgreSQL-базированные базы данных, то есть ли возможность найти альтернативу для нереляционных баз данных?

По состоянию на октябрь 2022 года MongoDB не предоставляет возможности использовать продукт Atlas для пользователей из России, что делает невозможным полноценно использовать облачную версию MongoDB для стеков разработки MERN (MongoDB – Express – React – NodeJS) и MEAN (MongoDB – Express – Angular – NodeJS). MongoDB является документо-ориентированной СУБД, позволяющей хранить данные в виде иерархии, что является одним из удобных способов хранения в отношении «один-ко-многим» [3, с. 394].

Самым простым способом хранения в такой архитектуре является создание JSON файла, который будет постоянно пополняться данными. Такой вид хранения имеет много недостатков, в число которых входит:

– отсутствие возможности одновременной работы с базой данных;

– постоянное перезаписывание файла и хранение некоторое время его содержимого в оперативной памяти.

При рассмотрении пункта 1 можно прийти к выводу, что пользователь, который первый открыл файл базы данных будет вносить изменения, то второй и последующий пользователь будет иметь следующие проблемы:

– асинхронный доступ к данным;

– блокировка файла при его открытии другим пользователем.

Исходя из вышеописанных недостатков можно сделать вывод, что хранение документоориентированных баз данных в виде файлов в разрезе больших проектов невозможно.

Решением данной проблемы можно назвать использование многомерности в таблицах реляционных СУБД. Сегодня СУБД умеют поддерживать OLAP кубы, что делает их более гибкими в финансовых организациях. Также современные СУБД умеют хранить в поле JSON-объекты, что позволяет делать запись настолько глубокой, насколько разработчику нужно.

Для описания эксперимента работы с многомерными таблицами будет использован стек PostgreSQL – NodeJS. Целью эксперимента является доказательство релевантности работы с многомерными таблицами в реляционной СУБД на примере стека PostgreSQL – NodeJS.

Был выбран именно такой стек технологий по причине того, что проект по сопряжению PostgreSQL и NodeJS (pg) актуален и развивается наравне с проектом по спряжению MongoDB и NodeJS (mongoose).

Для эксперимента была создана тестовая таблица в СУБД Postgres. В отличие от СУБД Oracle, где JSON-объекты представляются CLOB-объектами [4, с. 53], в PostgreSQL существует отдельный одноименный тип данных, позволяющий хранить JSON-объекты в чистом виде. DDL-код представлен на рисунке 1.

```
CREATE TABLE JSONTEST (
    ID NUMERIC,
    JS JSON
)
```

Рис. 1. DDL создания таблицы для хранения JSON-объекта

Благодаря отделению в отдельный тип данных, JSON-объект с помощью DML добавляется в виде строки и при запросе представляется как обычный JSON (рис. 2).

	id numeric	js json
1	1	{"lastname":"Shirokov","firstname":"Ivan"}

Рис. 2. Результат выполнения запроса к JSON-объекту

Хранение более сложных структур требует использование типа данных JSONB, таким образом возможно использование дополнительного перечисления (рис. 3) [5, с. 95].

```
INSERT INTO
    JSONTEST
    VALUES (1, '{
                "lastname":"Shirokov",
                "firstname":"Ivan",
                "university":{
                    "bachelor":"NARXOZ",
                    "master":"ITMO",
                    "phd":"ITMO"}}'::jsonb);
```

Рис. 3. Пример вставки вложенного JSON-объекта

Встроенными инструментами PostgreSQL данную структуру можно развернуть в обычный двумерный вид (рис. 4, 5).

```
SELECT
    JS ->> 'lastname' AS LASTNAME,
    JS ->> 'firstname' AS FIRSTNAME,
    JS -> 'university'  ->> 'bachelor' AS BACHELOR,
    JS -> 'university'  ->> 'master' AS MASTER,
    JS -> 'university'  ->> 'phd' AS PHD
FROM JSONTEST
```

Рис. 4. Запрос вложенного JSON-объекта

	lastname text	firstname text	bachelor text	master text	phd text
1	Shirokov	Ivan	NARXOZ	ITMO	ITMO

Рис. 5. Результат запроса вложенного JSON-объекта

Был проведен эксперимент по хранению массива данных внутри JSON-объекта, и в результате этого эксперимента была выявлена проблема вывода данных из массива, если они не числового формата (рис. 6).

```
INSERT INTO
    JSONTEST
    VALUES (1, '{
                "lastname":"Shirokov",
                "firstname":"Ivan",
                "university":{
                    "bachelor":"NARXOZ",
                    "master":"ITMO",
                    "phd":"ITMO"},
                "hobbies":["IT", "Aviation"]}'::jsonb);
```

Рис. 6. Пример вставки JSON-объекта с массивом

Согласно документации PostgreSQL (https://clck.ru/32MYfz), для вызова массива с помощью ключа используется оператор “#>>”, который выводит массив в виде строки (рис. 7). Таким образом деление массива на элементы затрагивает сам синтаксис массива как текст, что «загрязняет» данные как показано на рисунке 8.

```
SELECT
    JS ->> 'lastname' AS LASTNAME,
    JS ->> 'firstname' AS FIRSTNAME,
    unnest(string_to_array(JS #>> '{hobbies}',',')) as hobbies
FROM JSONTEST
```

Рис. 7. Запрос вывода массива данных

	lastname text	firstname text	hobbies text
1	Shirokov	Ivan	["IT"
2	Shirokov	Ivan	"Aviation"]

Рис. 8. Результат вывода массива данных

Для решения данной проблемы предложено использование программной платформы NodeJS с официальным пакетом для работы с СУБД Postgres “pg” (https://clck.ru/32MXF7).

При выполнении запроса таким методом, возвращается массив строк, сохраняя при этом первоначальный тип данных (рис. 9). При обращении непосредственно к полю, содержащему JSON-объект, обращение происходит как к уже форматированному объекту для JavaScript, как показано на рисунке 10.

```
import pg from "pg";

const pool = new pg.Pool({
  user: "postgres",
  password: null,
  host: "localhost",
  port: 5432,
  database: "postgres",
});

const data = await pool.query('SELECT * FROM "PG_TEST".jsontest');
console.log(data.rows[0].js);
```

Рис. 9. Инициализация доступа к Postgres посредством NodeJS

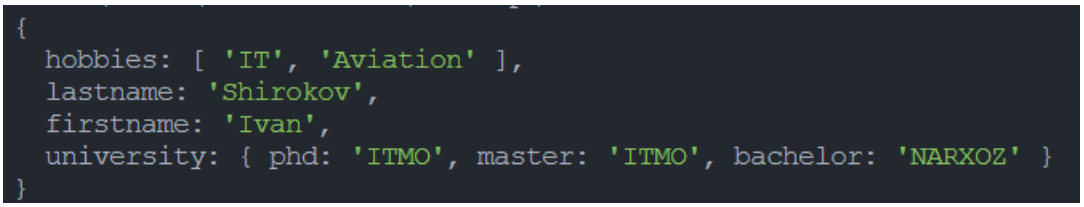

Рис. 10. Результат запроса с помощью пакета pg

При работе с NodeJS есть возможность обновления записи, используя нативные методы JavaScript с последующим вызовом update-запроса для подтверждения изменений в базе данных. Алгоритм представлен на рисунке 11.

Рис. 11. Пример обновления данных

В вышеуказанном примере было добавлено значение под ключом “hobbies”, которое имело тип данных «массив» при вставке данных. Обновленное значение с помощью пакета “pg” изменяется в базе данных. Результат выполнения показан на рисунке 12.

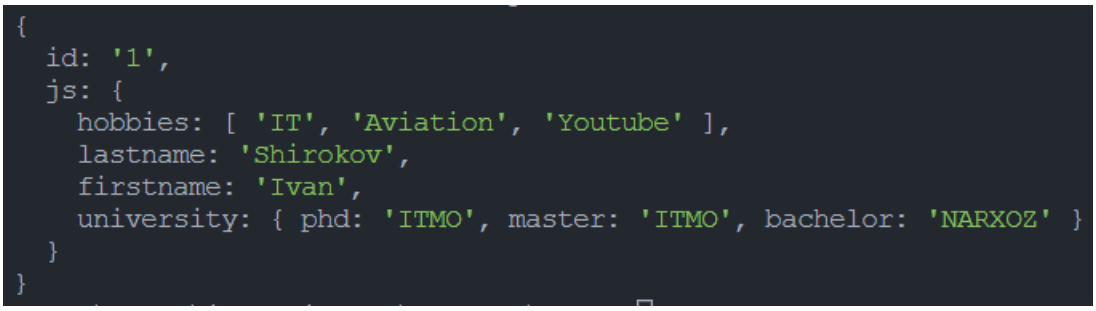

Рис. 12. Результат обновления данных

Исходя из вышеперечисленного можно сделать выводы, PostgreSQL-подобные СУБД:

– подходят для хранения JSON-объектов и, соответственно, могут заменить документно-ориентированные СУБД (например, MongoDB) с возможностью более гибкого использования,

– могут развернуть вложенные JSON-объекты в обычную таблицу, что делает возможным создание развернутых представлений в базе данных,

– некорректно работают с текстовыми массивами внутри JSON-объектов, что создает необходимость использования сторонних инструментов,

– имеют постоянно поддерживаемые инструменты для подключения к веб-приложениям, построенным на NodeJS.

Результат исследования выявил возможность использования PostgreSQL-подобных СУБД для более гибкого проектирования информационных систем, которые используют параллельно с реляционной СУБД и нереляционную.

Литература

1. Бажанова С.В., Сырямина Н.А. Независимость информационных ресурсов как элемент информационной безопасности государства // Вестник ВУиТ. 2019. № 3. С. 5-12.

2. Волушкова В.Л., Волушкова А.Ю. Структура данных для хранения информации в социальных сетях // Образовательные ресурсы и технологии. 2014. № 2 (5). С. 153-157.

3. Дьяконов А.В., Козлова Ю.Б. О современных тенденциях хранения данных в документо-ориентированных СУБД // Актуальные проблемы авиации и космонавтики. 2015. № 11. С. 394-396.

4. Лямин А.В. XML-реляционное преобразование с использованием системы продукционных правил // Компьютерные инструменты в образовании. 2018. № 1. С. 51-63.

5. Свиридов А.А. Хранение топологии сети в реляционных базах данных // Научно-исследовательские публикации. 2013. № 1. С. 94-105.