Р.А. СИМАКОВ, Д.С. ЛОГАШОВ

Муромский институт (филиал) ФГБОУ ВО «Владимирский государственный университет имени А.Г. и Н.Г. Столетовых», г. Муром

Red database: анализ интеграции с JSON

В данной статье рассмотрены проблемы, которые возникают при интеграции JSON с СУБД. Для анализа были выбраны несколько СУБД на примере которых раскрыты такие вопросы как: способ хранения JSON данных, необходимый функционал, преобразование в реляционный вид и индексация JSON.

Что такое JSON?

JSON – это текстовый формат обмена данными, основанный на JavaScript. За счет своей лаконичности и читаемости JSON используется для обмена данными в большинстве веб-приложений.

JSON состоит из двух базовых структур: коллекция пар ключ/значение и упорядоченный список значений.

Пример документа JSON:

```
{
   "id": 123,
   "shopperName": "Ivan Petrov",
   "shopperEmail": "ipetron@mail.com",
   "contents": [
     {
        "productId": 48,
        "productName": "bread",
        "quantity": 2
     },
     {
        "productId": 35,
        "productName": "milk",
        "quantity": 1
     }
   ]
}
```

Зачем это нужно?

JSON сейчас один из самых используемых форматов данных в разработке. Большинство современных сервисов возвращают информацию в виде JSON. JSON также предпочитаемый формат для хранения структурированной информации в файлах. Так как очень много данных используется в JSON-формате, то его поддержка становится актуальной в СУБД, а именно возможность импортировать и экспортировать данные в этом формате, загружать текст JSON в таблицы, извлекать значения из него, а также индексировать свойства в тексте JSON.

Можно выделить основные задачи, которые необходимо решить:

1. Способ хранения данных JSON;
2. Набор функций для работы с JSON;
3. Преобразование JSON в реляционный вид;
4. Индексирование данных JSON;

Для выбора верного решения данных задач проанализируем ряд СУБД, где работа с данными типа JSON уже реализована. Далее на основе проведенного анализа выберем наиболее подходящее решение задачи для СУБД Red database.

Для анализа были выбраны самые популярные реляционные СУБД такие как: MS SQL Server, PostgreSQL, Oracle Database, MySql.

Способ хранения данных JSON.

В выбранных СУБД существует 2 принципа хранения данных формата JSON. Первый это хранение данных в уже существующих типах, таких как varchar или BLOB, и добавление функций, которые позволять производить проверку корректности данных полей при добавлении и изменении. Такой подход реализован в СУБД MS SQL Server и Oracle Database. И второй, это создание в СУБД нового типа данных для хранения JSON с автоматической проверкой корректности, так было реализовано в PostgreSQL и MySql.

Так же существует проблема, в каком виде хранить данные, если хранить в обычном текстовом формате, то это не рационально со стороны использования свободного пространства и скорости обработки. Поэтому в каждой СУБД свои метод хранения JSON.

В MS SQL Server данные JSON могут храниться в текстовых полях nvarchar. Хотя JSON хранится в текстовых столбцах, это не просто обычный текст. В SQL Server есть механизм для оптимизации хранения текстовых столбцов с использованием различных механизмов сжатия, таких как сжатие UNICODE, которые позволяют экономить до 50% размера. Также вы можете хранить JSON в таблицах с columnstore индексами или сжимать их явно с использованием встроенный функции COMPRESS, которая использует алгоритм GZip [1]. Но сжатие не лучшее решение, потому что размер данных может быть уменьшен, но при этом сильно возрастут накладные расходы на обработку.

В PostgreSQL существуют два типа данных JSON: json и jsonb. Не смотря на то, что данные типы хранят JSON, данные между ними существует большая разница. Тип json сохраняет точную копию введённого текста, которую функции обработки должны разбирать заново при каждом выполнении, тогда как данные jsonb сохраняются в разобранном двоичном формате, что несколько замедляет ввод из-за преобразования, но значительно ускоряет обработку, не требуя многократного разбора текста [2]. Так же у типа jsonb есть возможность индексировать хранимые данные, в отличие от json. Преимущество json заключается в том, что он сохраняет форматирование документа и порядок ключей, так как хранит точную копию введенного текста.

В Oracle данные JSON могу храниться, в SQL типах VARCHAR2, BLOB и CLOB. Так как JSON храниться в Unicode, то лучше использовать BLOB для хранения JSON, потому что он хранится в бинарном виде. Типы VARCHAR2, CLOB хранятся в текстовом виде, поэтому если кодировка БД не будет соответствовать кодировке Unicode или UTF-8, то будут производиться автоматические символьные преобразования, которые могут влиять на производительность и привести к потере данных [3].

В MySQL существует отдельный тип данных JSON с автоматической проверкой допустимости. Сохраненный в столбцах JSON, преобразуется во внутренний формат, который обеспечивает быстрое чтение элементов документа. Двоичный формат структурирован, чтобы позволить серверу искать подобъекты или вложенные значения непосредственно по ключу или индексу

массива, не читая все значения прежде или после них в документе [4].

Из выше сказанного следует, что создание отдельного типа данных для хранения JSON в разобранном бинарном виде позволит наиболее быстро обрабатывать данные, так как не будет нужды каждый раз делать разбор данных при чтении, но это замедлит их ввод. Но как правило чтении совершается наиболее часто чем запись, поэтому данный вариант реализации выглядит очень перспективно, но имеет более сложную реализацию чем хранить JSON в виде текста и не позволит сохранить форматирование текста. Если же хранить JSON в виде текста, то можно использовать уже существующие типы данных, но при каждом чтение данных необходимо будет делать синтаксический анализ текста.

Набор функций для работы с JSON.

MS SQL Server и Oracle имеют ограниченный набор функций для работы с JSON. Основными из них присутствующих в обоих СУБД являются:

1. ISJSON – проверяет валидность JSON на соответствие спецификации. С помощью этой функции вы можете накладывать ограничения на столбцы, содержащие JSON

2. JSON_VALUE(jsonText, path) разбирает jsonText и выделяет отдельные значения по определенному «пути»

3. JSON_QUERY(jsonText, path) разбирает jsonText и выделяет объекты или массивы по определенному «пути»

В MS SQL Server существует функция редактирования JSON, которая называется JSON_MODIFY(jsonText, path, newValue) – изменяет значение какого-либо свойства по определенному «пути». Данная функция позволяет добавлять и изменять объекты и поля в документе JSON, но возможность удаления отсутствует. В Oracle Database явные функции для модификации JSON полей отсутствуют, но существует такая функция как JSON_EXISTS, которая по определенному пути проверяет наличие указанного поля, может служить фильтром для выборки данных и использоваться в CASE выражениях и в условие WHERE в операторе SELECT.

СУБД PostgreSQL и MySQL имеет более богатый набор функций для работы с JSON.

В MySQL существует набор функций для создания типа JSON (JSON_ARRAY([val[, val]...]), JSON_OBJECT([key, val[, key, val]...]), JSON_QUOTE(json_val)), которые позволяют создавать данные типа JSON на основе массивов, объектов ключ/значение и преобразование текста в формате JSON в формат СУБД. В PostgreSQL так же имеется аналоги данных функции, а так же несколько дополнительных функций.

Так же есть набор функций для доступа к данным JSON. Функции JSON_EXTRACT (или оператор «->») и JSON_UNQUOTE (или оператор «->>»), данные функции соответствуют функциям JSON_QUERY и JSON_VALUE в MS SQL Server и Oracle Database. В PostgreSQL используются только операторы «->» и «->>» для выполнения данных функций. Использование операторов выглядит более лаконично и читабельно, чем использование функций для данных целей. В PosgresSQL существуют еще несколько операторов для доступа к данным «#>» и «#>>» данные операторы выдают объект JSON по заданному пути, первый возвращает результат в формате JSON, а второй в виде текста. В MySQL есть еще две интересны функции, которых нет в других СУБД JSON_KEYS, которая возвращает список ключей, на определенном уровне, и JSON_SEARCH, которая возвращает путь до искомой строки в JSON.

В PostgreSQL и MySQL имеет набор функций для модификации JSON данных изменение и добавление новых полей, но в MySQL в отличие от своих аналогов, есть функция JSON_REMOV, которая позволяет удалять объекты из JSON.

В результате для взаимодействия с JSON данными необходимо реализовать следующий минимальный набор функций: функция проверки на валидность JSON, функция доступа к значениям свойств и к объектам JSON по заданному пути, возможность модификации данных JSON.

Преобразование JSON в реляционный вид.

В СУБД MS SQL Server и Oracle существуют функции OPENJSON и JSON_TABLE соответственно, которые производят

синтаксический анализ текста JSON и позволяют представлять данные в реляционном виде. Данные функции возвращают таблицу, с которой можно работать как с обычной реляционной таблицей, используя инструкции SQL. Так же в этих функциях указывается схема, которая отображает зависимость полей данных JSON с полями результирующей таблицы.

Пример синтаксиса функции OPENJSON в MS SQL Server.

```
SELECT *
FROM OPENJSON(@json, N'$.Orders')
WITH (
Number VARCHAR(200) N'$.Order.Number',
Date DATETIME N'$.Order.Date',
Customer VARCHAR(200) N'$.Account',
Quantity INT N'$.Item.Quantity' )
```

Данный синтаксис лаконичен и удобен, он наглядно отображает, какую реляционную таблицу мы хотим получить, и какие свойства JSON будут соответствовать полям таблицы.

Так же все необходимые данные можно получить с помощь функций JSON_VALUE или JSON_QUERY, но в документации Oracle говориться, что если используются функции JSON_VALUE или JSON_QUERY несколько раз в запросе или в сочетании друг с другом, то лучше использовать JSON_TABLE для доступа к данным. Это связано с тем, что вызов функций JSON_VALUE или JSON_QUERY может привести к новому разбору данных JSON, а с помощь функции представления JSON данных в реляционном виде, это можно сделать за один раз [3]. В документации Microsoft об этом ничего не написано, но скорее всего эта проблема присутствует и в MS SQL Server, так как данные хранятся в текстовом виде.

В PostgreSQL и MySQL функций для преобразования в JSON данных в реляционный вид нет, но извлечь данные JSON можно за счет богатого набора функций для доступа к данным в обоих СУБД. Так же плюсом данных систем является отсутствие проблемы, описанной выше, которая присутствует в Oracle Database, так как данные JSON хранятся в специальном бинарном виде, то нет необходимости, при вызове функций поиска данных, проводить синтаксические анализ и разбор данных JSON.

Следовательно, специальные функции для преобразования данных в реляционный вид обязательно нужны, если JSON храниться в текстовым виде, т.к. они позволят за один синтаксический разбор извлекать набор нужных данных. В случае хранения в разобранном бинарном виде можно обойтись без этих функций, это не скажется на производительности, но лишней она тоже не будет, потому что позволяет удобно поучить необходимый набор данных из JSON документа.

Индексирование данных JSON.

В зависимости от особенностей и функциональных возможностей СУБД поддержка индексации JSON данных реализована в них разными методами.

В MS SQL Server значения JSON можно индексировать их как обычные значения в столбцах и использовать некластеризованные или полнотекстовые индексы. Если необходимо создать индекс для какого-либо свойства JSON, которое часто используется в запросах, можно создать вычисляемый столбец, который ссылается на нужное свойство, затем создать обычный индекс по этому полю. Т.к. JSON это просто текст, можно использовать полнотекстовый индекс. Полнотекстовые индексы могут быть созданы на массиве значений. Вы создаете полнотекстовый индекс на столбце, который содержит массив JSON, или можете создать вычисляемый столбец, который ссылается на массив и добавить полнотекстовый индекс на этот столбец. Это может быть полезно, если вам нужно оптимизировать запросы, которые ищут какое-либо значение в массиве JSON [5].

В Oracle Database более гибкие инструменты для создания индексов, в которых нет необходимости в создание вычисляемых полей. Oracle позволяет при создание индексов использовать функции для работы с JSON, такие как JSON_VALUE, JSON_EXISTS, так же можно использовать «dot notation». Существует возможность создания составных индексов, но для этого нужно создать виртуальные столбцы по необходимым свойствам JSON, и на основе эти виртуальных полей можно создать композитный B-tree индекс [3]. Так же существует возможность полнотекстовых запросов с использованием JSON индексов.

В PostgreSQL для эффективного поиска ключей или пар ключ/значение можно применять GIN индексы, они могут работать только с данными jsonb. Класс операторов GIN по умолчанию для jsonb поддерживает запросы с операторами существования ключа на верхнем уровне (?, ?& и ?|) и оператором существования пути/значения (@>). Тип jsonb также поддерживает индексы btree и hash. Они полезны, только если требуется проверять равенство JSON-документов в целом [2].

В MySQL столбцы типа данных JSON не могут быть проиндексированы непосредственно. Реализация индексация JSON данных аналогичен методу, который использован в MS SQL. Чтобы создать индекс, который ссылается на данные JSON косвенно, вы можете определить генерируемый столбец. В него извлекается информация, которая должна быть проиндексирована, а затем создается индекс на генерируемый столбец.

Заключение

Для начала лучшим решением будет реализовать хранение JSON в текстовом виде и хранить в уже существующих типах данных СУБД Red database, таких как VARCHAR или BLOB и реализовать минимальный набор функций для взаимодействия. К таким функциям можно отнести:

1. Проверка на валидность (IS_JSON);
2. Доступа к данным (JSON_VALUE, JSON_QUERY);
3. Представление JSON в реляционном виде (OPEN_JSON);
4. Модификации данных (JSON_MODIFY);

Так как данные буду храниться в текстовом виде, то реализация функции OPEN_JSON будет необходима во избежание многократного синтаксического разбора.

В Red database не поддерживается индексирование BLOB полей, но возможно создание индексов, на основе вычисляемого выражения, с помощью конструкции COMPUTED BY, данным образом, возможно, реализовать индексацию данных внутри JSON по аналогии с MS SQL Server и Oracle Database.

Реализация данного набора задач позволит быстрым путем интегрировать поддержку JSON в Red database и увидеть его возможности.

Следующим шагом создание специализированного типа данных для хранения JSON в бинарном виде. Это реализация интеграции JSON трудней и требует больше времени, чем первая. Трудность заключается в выборе двоичной спецификации для реализации хранения, необходимо проанализировать уже существующие, такие как BSON, BJSON, Smile и в результате модифицировать данные спецификации под Red database или создать свою. Хранение в бинарном виде позволит быстрее обрабатывать данные и получать к ним доступ, но замедлит их ввод. Так же это даст возможность реализовать дополнительные функции, которые будут возвращать глубину или длину JSON документа, используя мало ресурсов. Еще одним плюсом бинарного формата, является больше возможностей в организации индексирования JSON данных.

В результате получиться неплохая поддержка JSON, которая позволит в зависимости от потребностей клиентов использовать текстовый или бинарный формат хранения данных. Текстовый формат логично использовать, когда важно форматирование документа и данные чаще добавляются, чем обрабатываются. В остальных случаях бинарный формат будет более целесообразен.

Литература

1. MSDN. Данные JSON (SQL Server): //URL: https://msdn.microsoft.com/ru-ru/library/dn921897.aspx (дата обращения: 10.01.2017)

2. PosgreSQL Documentation. JSON types: //URL: https://www.postgresql.org/docs/9.6/static/datatype-json.html (дата обращения: 12.01.2017)

3. Oracle Help Center. JSON on Oracle Database: //URL: https://docs.oracle.com/database/121/ADXDB/json.htm#ADXDB6293 (дата обращения: 12.01.2017)

4. MySQL Documentation. The JSON data types: //URL: https://dev.mysql.com/doc/refman/5.7/en/json.html (дата обращения: 14.01.2017)

5. Работаем с JSON в SQL Server 2016: //URL: https://habrahabr.ru/post/317166/ (дата обращения: 05.01.2017)