Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Семенов Александр Сергеевич

Транзакционное хранилище для версионирования бинарных данных в PostgreSQL

Учебная практика

Научный руководитель:
доц. каф. СП, к.ф.-м.н. Луцив Д.В.

Санкт-Петербург
2023

Оглавление

Введение

Сегодня сложно представить себе организацию, деятельность которой обходится без информационных систем. Одной из основных составляющих любой информационной системы являются данные. Для их хранения и управления используются различные технологии, например, базы данных. Для настраивания и администрирования баз данных применяется набор программных средств, называемый системой управления базами данных (СУБД).

Одной из самых популярных объектно-реляционных СУБД является PostgreSQL, которая широко используется в различных областях, включая финансы, науку и образование, а также имеет открытый исходный код, поддерживает репликации, хранимые процедуры и прочие полезные функции, а также совместима с набором требований ACID. Одним из требований ACID является атомарность, которая гарантирует, что транзакция либо будет выполнена полностью, либо не выполняется совсем. Это позволяет сохранять согласованность данных в базе, что может быть критично для многих информационных систем, например, для банковских или систем, взаимодействующих с рынком ценных бумаг.

Для хранения данных внутри базы в PostgreSQL предусмотрено множество типов, таких как integer, text, boolean и прочие. Эти типы данных позволяют эффективно хранить и обрабатывать структурированные данные, однако они не подходят для неструктурированных данных, например, бинарных. Основное отличие этих типов данных в том, что неструктурированные данные не соответствуют заранее определенной структуре данных, из-за этого подходы к хранению и обработке такого рода информации должны отличаться от соответствующих подходов работы со структурированными данными. Примерами бинарных данных, которые может быть полезно хранить информационной системе, могут быть pdf-документы, картинки, архивы, электронные письма и тп.

Если бинарные данные имеют небольшой размер, то часто для их

версионирования заводят отдельную таблицу, в которой хранят метаданные, такие как дата последнего изменения файла и ссылка на конкретную версию файла. Для такого подхода необходимо хранить все версии файлов отдельными записями. Проблемы возникают, когда появляется необходимость хранить большие файлы, так как из-за множества их версий разрастается сама база данных, а также это будет сказываться на её производительности.

В данной работе будут рассмотрены подходы к версионированию бинарных файлов внутри СУБД PostgreSQL, предложен подход, позволяющий делать это эффективно с точки зрения занимаемого дискового пространства и времени, а также поддерживающий транзакционность операций над бинарными данными.

1. Постановка задачи

Данная работа выполняется в рамках проекта, над которым работают два человека. Целью именно этой работы является написание алгоритма для версионирования бинарных данных, а также предоставление API для написания расширения для PostgreSQL, позволяющее управлять данными при помощи синтаксиса SQL.

Для достижения цели были поставлены следующие задачи:

- Изучить существующие подходы к решению задачи версионирования бинарных данных в PostgreSQL.
- Определить подход к версионированию бинарных данных.
- Реализовать выбранный подход к версионированию бинарных данных.
- Реализовать API для взаимодействия с расширением PostgreSQL.
- Провести тестирование полученного решения и сравнение с аналогами.

2. Обзор

2.1. Стандартные средства PostgreSQL

В СУБД PostgreSQL для хранения бинарных данных предусмотрены типы bytea и BLOB[1] [1]. Тип bytea является двоичной строкой переменной длины. Этот тип использует механизм TOAST[2], который позволяет разбивать большие файлы на множество мелких. Для этого заводится специальная таблица, которая хранит ссылки на части файла и с помощью которой впоследствии можно восстановить файл из частей [3]. Однако этот механизм не был разработан для поддержки обновлений данных. Поэтому в случае обновления данных sql-операцией UPDATE, механизм TOAST будет дублировать все части сохраненных данных, даже если изменения затрагивают какую-то одну часть из них [2]. Помимо этого, такой подход накладывает ограничения на производительность СУБД при работе с большими данными.

Помимо bytea, в PostgreSQL есть средство хранения больших объектов BLOB, который обеспечивает потоковый доступ к данным, однако для работы с ним нужно использовать отличающийся от стандартного интерфейс, и этот механизм не предусмотрен для версионирования данных.

Несмотря на то, что эти способы хранения данных поддерживают транзакционность, они не позволяют версионировать бинарные данные так, чтобы размер базы не увеличивался из-за количества их версий.

2.2. Система контроля версий Git

Когда речь заходит о версионировании, сразу вспоминается система контроля версий Git. Она позволяет хранить историю изменений, поддерживает ветвления и работу с конфликтами. Наибольшую популярность эта система обрела среди разработчиков, потому что позволяет эффективно работать с изменением текстовых файлов с исходными

[1]Binary Large Object

[2]The Oversized Attribute Storage Technique

кодами программ. Помимо текстовых файлов, Git позволяет версионировать и бинарные файлы. Можно было бы представить себе решение поставленной задачи следующим образом: завести внешнее хранилище данных, настроить там Git и подключить его к СУБД. Однако здесь возникает существенный недостаток: Git умеет определить, что изменился бинарный файл, но не умеет определять в каком именно месте произошло изменение. Из-за этого любое изменение файла, даже самое небольшое, будет приводить к созданию копии файла, а если мы имеем дело с большими файлами, то размер нашего хранилища будет существенно расти.

3. Предлагаемое решение

3.1. Описание решения

В данной работе предлагается решить проблему хранения и версионирования бинарных данных в СУБД PostgreSQL следующим образом: заводится сетевое файловое хранилище с файловой системой, которая позволяет на её основе реализовать необходимый для отслеживания версий файлов функционал. Предполагается, что у сервера базы данных есть доступ к этому сетевому хранилищу.

На основе файловой системы на сетевом хранилище будет реализован API, позволяющий взаимодействовать с файлами через стандартный синтаксис SQL, то есть поддерживающий стандартные операции SELECT, INSERT, UPDATE и DELETE. Далее будет написано расширение для PostgreSQL, использующее этот API и позволяющее сохранять и версионировать бинарные файлы из СУБД. Благодаря этому будет возможность взаимодействовать с файлами стандартными средствами СУБД, не заботясь о том, как эти файлы на самом деле хранятся.

3.2. Выбор файловой системы

Были выделены основные критерии отбора файловой системы для написания алгоритма версионирования бинарных данных и API для поддержки написания расширения для PostgreSQL:

- Наличие открытого API для взаимодействия с файловой системой;
- Поддержка создания снапшотов для реализации алгоритма версионирования;
- Поддержка транзакций через механизм copy-on-write.

Снапшоты – это специальные снимки, которые хранят полное состояние диска или файла в определенный момент времени. Благодаря

этому механизму, появляется возможность хранить несколько снимком определенного файла или директории и возвращаться к любому снимку в случае необходимости.

Механизм copy-on-write позволяет при изменении данных не менять их напрямую, а создает копию данных в другом месте и изменяет её. Это позволяет избежать ошибок в данных при неудачных изменениях. Также эта технология позволяет увеличить скорость записи, так как она стирает старые данные только когда появляется в этом необходимость.

Выбор происходил между файловыми системами EXT4, BTRFS и ZFS, так как эти файловые системы чаще всего используются для сетевых хранилищ, а также у всех них есть открытый и хорошо задокументированный API для взаимодействия с ними.

Несмотря на то, что EXT4 является самой стабильной из рассматриваемых файловых систем, а также по умолчанию используется на большинстве операционных систем семейства Unix, она не поддерживает механизм copy-on-write, поэтому дальше эта файловая система рассматриваться не будет.

Файловые системы BTRFS и ZFS обе поддерживают механизм copy-on-write и создание снапшотов. Однако, при работе с большими объёмами данных, BTRFS показывает более низкую эффективность в сравнении с ZFS [4].

Поэтому в дальнейшем для решения поставленной задачи будет использоваться файловая система ZFS. Это хороший выбор для хранения и версионирования бинарных данных в PostgreSQL, так как помимо того, что она поддерживает атомарные операции над файлами и создание снапшотов за константное время, в этой файловой системе реализовано сжатие данных, что позволит эффективнее использовать дисковое пространство [5].

Заключение

В результате работы над учебной практикой были решены следующие задачи:

- Изучены существующие подходы к решению задачи версионирования бинарных данных в PostgreSQL.
- Определён подход к версионированию бинарных данных.

Планы для дальнейшей работы:

- Реализовать выбранный подход к версионированию бинарных данных.
- Реализовать API для взаимодействия с расширением PostgreSQL.
- Провести тестирование полученного решения и сравнение с аналогами.

Список литературы

[1] BinaryFilesInDB // PostgreSQL Wiki. — 2021. — Access mode: https://wiki.postgresql.org/wiki/BinaryFilesInDB (online; accessed: 25.11.2022).

[2] Fittl Lukas. 5mins of Postgres E3: Postgres performance cliffs with large JSONB values and TOAST // pgAnalyze Blog. — 2022. — Access mode: https://pganalyze.com/blog/5mins-postgres-jsonb-toast (online; accessed: 02.01.2023).

[3] TOAST // PostgreSQL Documentation. — 2015. — Access mode: https://www.postgresql.org/docs/current/storage-toast.html (online; accessed: 25.11.2022).

[4] Vondra Tomas. Postgres vs. File Systems: A Performance Comparison // EDB Blog. — 2022. — Access mode: https://www.enterprisedb.com/blog/postgres-vs-file-systems-performance-comparison (online; accessed: 02.01.2023).

[5] Меликов Георгий. ZFS: архитектура, особенности и отличия от других файловых систем // «Завтра облачно», журнал о цифровой трансформации от VK Cloud Solutions. — 2020. — Access mode: https://mcs.mail.ru/blog/zfs-arhitektura-osobennosti-i-otlichija (online; accessed: 02.01.2023).