УДК 004.652.4

ХРАНЕНИЕ ЭЛЕКТРОННЫХ ПАСПОРТОВ ТОВАРОВ В РЕЛЯЦИОННОЙ БАЗЕ ДАННЫХ С ВЕРСИОНИРОВАНИЕМ ИЗМЕНЕНИЙ

Е.П. ЖИДЕЦКИЙ
(Представлено: Е.Р. СУХАРЕВ)

Реляционные базы данных по определению плохо подходят для хранения данных с изменяющейся структурой, но иногда такая необходимость возникает. Существуют различные методы решения этой задачи в зависимости от дополнительных требований. Рассмотрен один из вариантов ее решения.

Для хранения данных с изменяющейся структурой в большинстве случаев лучшим решением будет использование документно-ориентированной базы данных [1]. Такой подход упростит логику DAO-слоя приложения, даст высокую скорость записи и чтения данных. Но иногда характер сохраняемой информации, а также бизнес логика работы с ней требуют реляционного подхода.

Примером такой модели данных может быть информация о товарах для системы управления мастер-данными. В таких системах особенно важным является удобство заполнения каталогов и редактирования отдельных товаров, а также скорость доступа к отдельным атрибутам конкретного товара. Кроме того для различных пользователей структура товара может различаться, это обусловлено различными внешними и внутренними факторами: требования рынка, государственный стандарт электронного паспорта товара, характер внутренних бизнес процессов компании и др. Для описания информации о товаре существует единый мировой стандарт GS1, включающий в себя более 500 атрибутов – что является избыточным для большинства пользователей [2]. Дополнительным требованием было хранение истории изменений товаров для возможности определения момента и автора изменения любого атрибута товара, а также отката изменений товара до определенного момента. Эти требования делали модель реляционной и ограничивали множество кандидатов на роль хранилища данных.

Для анализа модели данных использовался стандарт GS1, который давал информацию о максимально детальном описании товара, за исключением атрибутов, используемых во внутренних бизнес процессах компаний пользователей. В ходе анализа было определено множество типов данных, покрывающее все атрибуты:

– число. Обычная числовая характеристика: процентное содержание алкоголя, штрих код, остаток на складе;
– строка. Обычная строковая характеристика: код GTIN, GLN владельца;
– дата. Обычная календарная характеристика: дата создания, дата отгрузки, дата окончания срока годности;
– логическое да/нет. Различные флажки: подлежит ли возврату, указана ли цена на упаковке, подлежит ли переработке;
– значение из словаря. Выбор из списка вариантов: страна производитель, тип штрих кода, канал торговли;
– многоязычная строка. Строковая характеристика, представленная на разных языках: наименование на упаковке, описание, рекомендации;
– число с единицей измерения. Физические величины: метрики, цена с указанием валюты;
– файл. Различные присоединенные файлы: изображение товара, сертификат.

Каждый атрибут, кроме логических, может быть представлен как единичным значением, так и массивом значений соответствующего типа данных.

Для получения возможности хранить товары с гибкой структурой атрибутов при минимальной избыточности было решено построить реляционное хранилище по следующим принципам:

– отдельные таблицы для атрибутов каждого типа данных: ссылка на товар, имя атрибута, индекс, значение (в зависимости от типа данных);
– таблица для ключевых данных товара: версия, автор, хеш (высчитывается из ключевых атрибутов) для быстрого поиска;
– отдельные таблицы для сохранения истории атрибутов каждого типа данных;
– таблица для сохранения истории товаров, для группировки истории атрибутов.

Такая структура базы данных позволила убрать влияние размера истории изменений товара на скорость доступа к его актуальному состоянию, поместив данные об истории изменений в отдельный набор таблиц, а при использовании стратегии вертикального шардинга – даже на отдельный сервер [3]. Также в структуре было зафиксировано максимальное количество статичной информации о модели дан-

ных без потери гибкости в добавлении в модель новых атрибутов. Схема базы данных представлена на рисунке.

Для работы с данными был разработан ряд алгоритмов, позволяющий выполнять базовые операции максимально эффективно. Операции чтения, создания и удаления имели только одну особенность: необходимость работы с несколькими таблицами для доступа к данных по всем требуемым типам атрибутов – что не было проблемой благодаря использованию механизма транзакций и пакетного исполнения запросов JDBC [4].

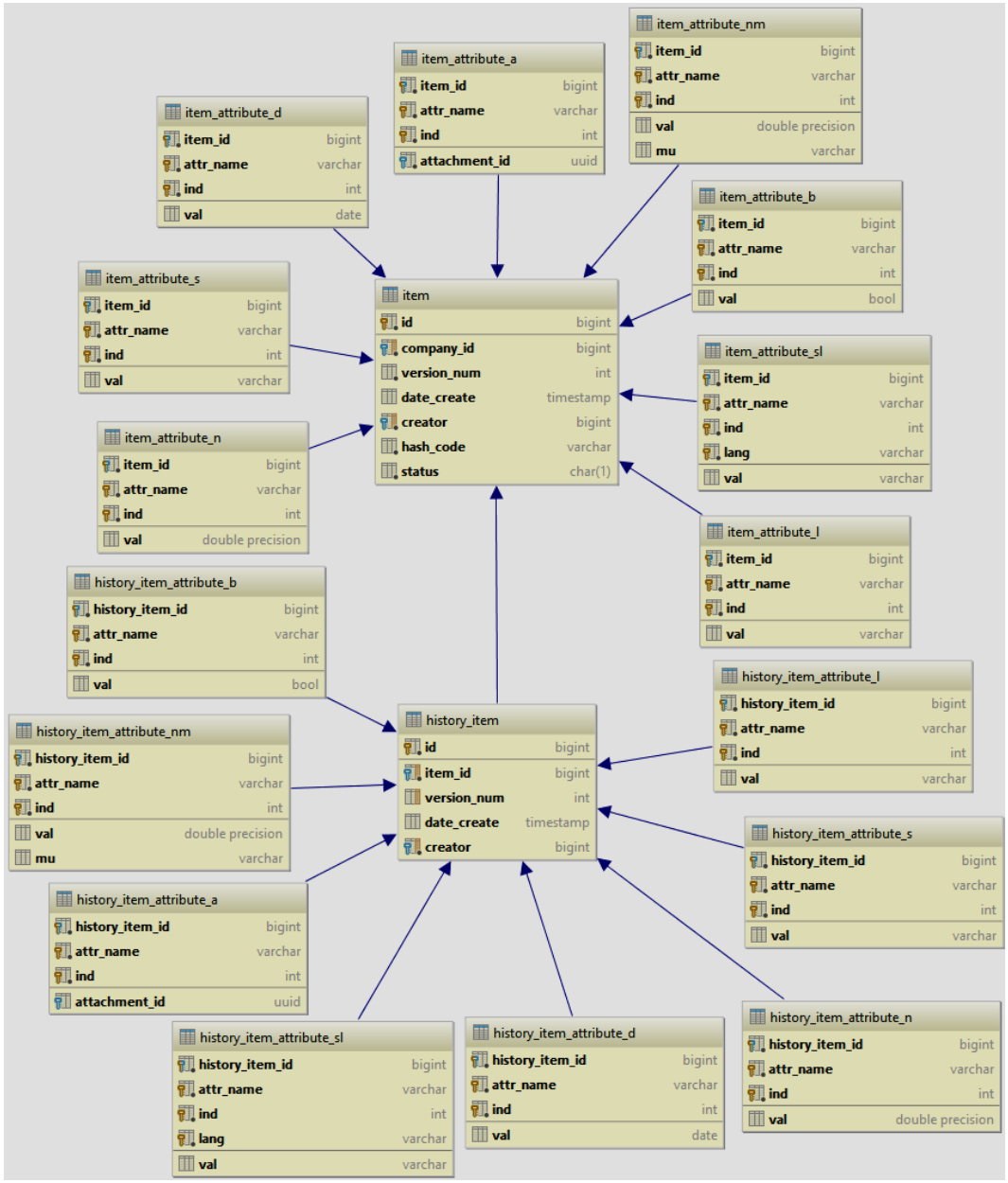

Рис. Схема базы данных с информацией о товарах и версиях изменений

Наиболее сложной операцией было обновление данных, поскольку только в этой операции требовалось вести версионность изменений. Классические алгоритмы ведения разностных изменений не подходили, поскольку они рассчитаны на накопление изменений от некоторого фиксированного базового состояния. При таком подходе существует прямая зависимость между вычислительной сложностью получения актуального состояния товара и длиной истории его изменений, а значит, чем больше было изменений в товаре, тем дольше будет выполняться доступ к его актуальному состоянию. Кроме этого, при классическом подходе урезание истории редактирования до определенной глубины является довольно сложной операцией, в которой изменения в не актуальной части истории перед удалением последова-

тельно накатываются на базовое состояние и меняют его. Эти факты привели к необходимости разработки собственного «обратного» алгоритма сохранения изменений. Его ключевые особенности:

– в базе всегда хранится актуальная версия товара;
– в актуальной версии товара не сохраняются атрибуты с пустым значением;
– восстановление ранней версии товара представляет собой итерационное применение изменений от актуальной версии до требуемой;
– вычислительная сложность получения ранней версии прямо пропорциональна количеству версий между актуальной и требуемой;
– урезание истории выполняется путем простого удаления данных об устаревших изменениях.

Перед обновлением данных загружается текущее состояние товара из базы. Это необходимо для формирования разностных данные между существующим состоянием и новым, а также предотвращает множество проблем при одновременном редактировании товара разными пользователями: позволяет определить конфликт версий и провести слияние. В историю сохраняются полученные разностные данные в инвертированном виде, что позволяет, применяя сохраненные в базе изменения, переходить от текущего состояния на предыдущее. Для каждого изменившегося значения атрибута, который однозначно идентифицируется в товаре при помощи составного ключа из имени атрибута и индекса, применяется алгоритм в зависимости от характера изменения атрибута. Алгоритм включает в себя изменение атрибута в актуальном состоянии, и в истории предыдущей версии товара. Возможные варианты представлены в таблице.

Таблица

Работа с хранилищем при обновлении товара в зависимости от характера изменения атрибута

Характер изменения	Признак	Действие	
		Актуальная версия	Предыдущая версия
Добавление	В текущей версии товара у атрибута отсутствовало значение, а в новой – присутствует	Вставить новое значение	Вставить NULL
Удаление	В текущей версии товара у атрибута присутствовало значение, а в новой – отсутствует	Удалить значение	Вставить старое значение
Изменение	Значение атрибута присутствует в текущей и новой версии товара, но оно различно	Изменить на новое значение	Вставить старое значение

С целью оптимизации работы с базой, описанный выше алгоритм был несколько усложнен: изменения атрибутов конкретного товара группировались по изменяемым таблицам. Это позволило применить пакетную обработку запросов JDBC и увеличить производительность операций [4].

При нагрузочном тестировании, в базу было внесено более 9 000 000 товаров по 30 заполненных атрибутов в каждом. Шардинг базы данных не применялся. В качестве СУБД выступал PostgreSQL 9.3, развернутый на машине средней мощности: Intel Core i5 2,2GHz 4 CPU, 8 GB RAM – и настроенный в соответствии с лучшими практиками [5]. Размер хранилища данных составил 40 ГБ. Средняя скорость выборки по ключевым атрибутам – 78 запросов/сек, по не ключевым – 11 запросов/секунду. Т.о. система, с хранилищем, построенным на описанных выше принципах, показала высокую производительность при нагрузках, а также надежность при одновременной работе множества пользователей с одним и тем же набором товаров.

Описанный в статье подход к хранению и работе с данных динамичной структуры в реляционном хранилище показал свою эффективность на тестовом стенде и в настоящее время работает в production-окружении. Он может применяться для хранения объектов из широкого спектра предметных областей, где характер данных не позволяет работать в документно-ориентированной среде.

ЛИТЕРАТУРА

1. Seguin, K. Когда использовать MongoDB / K. Seguin // Маленькая книга о MongoDB [Электронный ресурс]. – 2015. – Режим доступа: http://jsman.ru/mongo-book/Glava-5-Kogda-ispolzovat-MongoDB.html. – Дата доступа: 27.09.2015.
2. Standards [Electronic resource] / GS1 The Global Language of Business. – 2015. – Mode of access: http://www.gs1.org/standards. – Date of access: 27.09.2015.
3. Шардинг и репликация [Электронный ресурс] / HighLoad.com. – 2009. – Режим доступа: http://ruhighload.com/index.php/2009/05/06/шардинг-партиционирование-репликац. – Дата доступа: 28.09.2015.
4. JDBC – Batch Processing [Electronic resource] / TutorialsPoint. – 2015. – Mode of access: http://www.tutorialspoint.com/jdbc/jdbc-batch-processing.htm. – Date of access: 28.09.2015.
5. Installation and Administration Best practices [Electronic resource] / PostgreSQL Wiki. – 2015. – Mode of access: http://wiki.postgresql.org/wiki/Installation_and_Administration_Best_practices. – Date of access: 28.09.2015.