БЕЛЫШЕВ Д.В.,
к.т.н., Институт программных систем им. А.К. Айламазяна РАН, г. Переславль-Залесский, Россия,
e-mail: belyshev@cron.botik.ru

КОЧУРОВ Е.В.,
ООО «Интерин технологии», Москва, Россия, e-mail: kochurov@interin.ru

МОДУЛЬНЫЕ ХРАНИЛИЩА ДАННЫХ В ПЛАТФОРМЕ «ИНТЕРИН IPS» ДЛЯ POSTGRESQL

DOI: 1025881/18110193_2021_S5_32

Аннотация.
Для разработки медицинских систем важна эффективная технология взаимодействия между средствами хранения и обработки структурированных и неструктурированных данных. В работе описываются способы согласования объектного и реляционного подходов к обработке данных между приложением и СУБД в методологии NORM, а также использования табличных модулей для конструирования хранилищ данных на примере новой версии платформы «Интерин IPS» для PostgreSQL.

Ключевые слова: *хранилища данных, PostgreSQL, медицинские информационные системы.*

Для цитирования: *Белышев Д.В., Кочуров Е.В. Модульные хранилища данных в платформе «Интерин IPS» для PostgreSQL. Врач и информационные технологии. 2021; S5: 32-43. doi: 1025881/18110193_2021_S5_32.*

BELYSHEV D.V.,
Ph.D., Ailamazyan Program Systems Institute of RAS, Pereslavl-Zalesski, Russia,
e-mail: belyshev@cron.botik.ru

KOCHUROV E.V.,
Interin technologies, Moscow, Russia, e-mail: kochurov@interin.ru

MODULAR DATA WAREHOUSES IN THE INTERIN IPS PLATFORM FOR POSTGRESQL

DOI: 1025881/18110193_2021_S5_32

Abstract.
For the development of healthcare information systems, an effective system of interaction between means of storing and processing structured and unstructured data is important. The paper describes the ways of reconciling the object and relational approaches to data processing between the application and the DBMS as suggested by the NORM methodology. It also describes the use of tabular modules for constructing data warehouses through the example of the new version of the Interin IPS platform for PostgreSQL.

Keywords: *Data Warehouse, PostgreSQL, Healthcare Information Systems.*

For citation: *Belyshev D.V., Kochurov E.V. Modular data warehouses in the Interin Ips platform for PostgreSQL. Medical doctor and information technology. 2021; S5: 32-43. doi: 1025881/18110193_2021_S5_32.*

ВВЕДЕНИЕ

Сегодня фраза «Данные — это новая нефть» являются крайне популярной и широко растиражированной среди самых разных авторов, ее произносят и члены правительств, и владельцы крупнейших компаний и всевозможные аналитики на рынке информационных технологий. Человечество стремительно наращивает производство и хранение данных, тратя на это всё больше ресурсы и связывая с этим перспективы развития технологий. Причем речь идет не только о BigData, но и о необходимости хранить и эффективно манипулировать вполне ограниченными объемами данных в рамках корпоративных информационных систем.

КОНФЛИКТ МЕЖДУ ОБЪЕКТНО-ОРИЕНТИРОВАННЫМИ ПРИЛОЖЕНИЯМИ И РЕЛЯЦИОННЫМИ СУБД

При работе с данными, мы в первую очередь говорим об использовании систем управления базами данных (СУБД), которые прошли долгий путь в своем развитии. Было перепробовано масса самых разных концепций работы с данными. Майкл Стоунбрейкер, основатель проекта СУБД PostgreSQL, в работе «THIRD-GENERATION DATABASE SYSTEM MANIFESTO» [1] выделяет три поколения СУБД: «Мы называем старые иерархические и сетевые системы системами баз данных первого поколения, текущие реляционные СУБД относим к системам второго поколения. В этой статье мы рассматриваем характеристики, которым должно удовлетворять следующее поколение СУБД, которое мы называем третьим поколением». Вопрос о будущем СУБД волновал многих исследователей и примерно в то же время были опубликованы еще несколько «манифестов» [1–3], где обсуждается вопрос — какой же должна быть перспективная СУБД. В те годы многими авторам виделась ясная перспектива замены реляционных СУБД той или иной реализацией объектных или объектно-реляционных баз данных. Много работ посвящено изучению таких подходов, широкие обзоры которых даются в книге К. Дж. Дейта «Введение в системы баз данных» [4], публикациях С.Д. Кузнецова [5; 6], сравнительно недавний обзор на habr.com [7]. Тем не менее, за прошедшие 20–30 лет с момента публикации этих манифестов «третье поколение» СУБД так и не заменило собой реляционные системы, которые по сей день составляют основу баз данных для корпоративных приложений. В какой-то степени распространенность типов СУБД можно видеть из рейтинга баз данных «DB-Engines Ranking» [8], где реляционные СУБД занимают лидирующие строки среди 378 зарегистрированных продуктов, занимая при этом более 30% (131 запись) всех типов систем (Рис. 1). Хотя нужно оговориться, что репрезентативность подобных рейтингов очень условная, поскольку они не отражают нишевого характера используемых продуктов и прямое сравнение не вполне корректно. Можно утверждать лишь что в качестве универсальных СУБД безоговорочно лидируют реляционные, но никак не объектные, несмотря на немалые усилия и ожидания.

Вместе с тем, нельзя не признать, что многочисленные попытки перевести представление

378 systems in ranking, September 2021

Rank			DBMS	Database Model	Score		
Sep 2021	Aug 2021	Sep 2020			Sep 2021	Aug 2021	Sep 2020
1.	1.	1.	Oracle	Relational, Multi-model	1271.55	+2.29	-97.82
2.	2.	2.	MySQL	Relational, Multi-model	1212.52	-25.69	-51.72
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	970.85	-2.50	-91.91
4.	4.	4.	PostgreSQL	Relational, Multi-model	577.50	+0.45	+35.22
5.	5.	5.	MongoDB	Document, Multi-model	496.50	-0.04	+50.02
6.	6.	↑ 7.	Redis	Key-value, Multi-model	171.94	+2.05	+20.08
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	166.56	+1.09	+5.32
8.	8.	8.	Elasticsearch	Search engine, Multi-model	160.24	+3.16	+9.74
9.	9.	9.	SQLite	Relational	128.65	-1.16	+1.98
10.	↑ 11.	10.	Cassandra	Wide column	118.99	+5.33	-0.18

Рисунок 1 — Рейтинг СУБД.

данных к объектному виду исходили из насущной необходимости повышение уровня абстракции работы с данными, особенно когда разработка приложений ведется в объектно-ориентированных технологиях (C++/C#/Java и т.п.), поскольку в противном случае необходимо одновременно использовать две разные модели данных (реляционную для хранения и объектную для работы бизнес-приложения) и непрерывно конвертировать их из одной в другую. Тем не менее, после многочисленных подходов к реализации объектной парадигмы в СУБД, еще в начале 2000-х Дейт констатирует: «большинство специалистов в области информационных технологий теперь считают, что объектные системы, возможно, имеют определенную область применения, но эта область является довольно ограниченной [9]» [4].

РАЗРЕШЕНИЕ КОНФЛИКТА ПРИ ПОМОЩИ ORM

Неудача с внесением объектно-ориентированного подхода в базы данных напрямую не привела к оставлению попыток связать ООП и реляционную технологии разработки. Еще один подход к решению задачи связывания двух моделей представления данных разрабатывается в рамках методологии Object Relational Mapping (объектно-реляционное отображение — ORM) [10]. ORM — это технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных» и позволяющую обращаться с данными как с объектами, тогда как дополнительный «слой данных» (data layer) производит «отображение» реляционных сущностей используемой СУБД с объектами приложения. Такой подход достаточно популярен, существуют десятки ORM-пакетов для всех основных технологий программирования, которые используют для хранения данных самые различные СУБД. Вместе с тем, существует серьезный скепсис относительно применения ORM для высоконагруженных приложений, Том Кинкейд — вице-президент EDB заявляет: «ОО-программисты застряли в одном мире, администраторы баз данных и разработчики баз данных — в другом... На мой взгляд, наиболее распространенной формулой для включения обоих из них является использование технологии ORM... Это хорошо работает для небольшого набора данных, но, когда схема базы данных начинает меняться, начинают закрадываться проблемы с производительностью и ремонтопригодностью. Пользователи начинают жаловаться на проблемы с производительностью, добавление функций в приложение занимает больше времени, чем ожидалось, и, в зависимости от динамики организации, команда базы данных и группа разработки приложений в конечном итоге обвиняют друг друга» [11].

КОНЦЕПЦИЯ NORM

Одним из решений, позволяющим обойти обозначенные проблемы, Конклейд видит концепцию NORM (No Object Relational Mapping). Этот подход описан в недавно вышедшей книге «PostgreSQL Query Optimization: The Ultimate Guide to Building Efficient Queries» [12], где авторы предлагают выполнить гармонизацию реляционного хранения и объектного манипулирования при помощи некого «контракта» между базой данных и приложением, оформленном в виде JSON-объектов оговоренной структуры. Такой подход позволяет на уровне приложения не опускаться до анализа структур данных, а в части СУБД не привязывать структуры хранения к способам их использования на стороне приложения.

Направление работы с данными, которое продвигает компания EDB, достаточно близко тем решениям, которые реализуются ООО «Интерин технологии» [13] и применены при реализации концепции JSON-хранилищ [14; 15] как части платформы «Интерин IPS» (правообладатель ООО «Интерин технологии», свидетельство о государственной регистрации программы для ЭВМ №2017610110). Первая версия платформы использует СУБД Oracle и обеспечивает JSON-интерфейс между сервером приложений и СУБД, позволяя выполнять как структурированное хранение с процедурной обработкой данных и размещением их в произвольные реляционные структуры, так и noSQL-хранение JSON-документов в специализированных хранилищах, построенных средствами обычных реляционных таблиц.

Ограничения, которые имеет реализация JSON-хранилищ в среде Oracle, заключаются

в отсутствии встроенной поддержки формата JSON базой данных (разработка выполнялась для Oracle DataBase 11g), в то время как поддержка JSON на уровне СУБД появилась только в Oracle 12.1, хотя назвать эту поддержку удачной достаточно сложно, поскольку она ограничилась применением системных функций, позволяющих разбирать хранимые в виде текста JSON-объекты и манипулировать их структурой, в то время как встроенного типа данных JSON так и не появилось. Эти сложности требовали самостоятельной поддержки JSON на стороне СУБД и приводили к неэффективным вычислениям с JSON-объектами. К сожалению, дальнейшее развитие СУБД Oracle не привело к каким-то существенным изменениям в этом вопросе, поскольку полноценной встроенной поддержки JSON не возникло. Вместе с тем, эти особенности реализации не помешали успешно реализовать принцип «контракта» между приложениями и базой на основе JSON-документов.

Реализованная нами схема очень проста и эффективна и в этом ее несомненное преимущество. В отличие от ORM, где «отображение» объектов в реляционные структуры выполняется «слоем данных», скрытым ORM-библиотекой от прикладного разработчика, в NORM-подходе разработчик на уровне подготовки данных на стороне СУБД сам формирует необходимый объект, имея полное управление способом извлечения и сохранения данных, что в свою очередь позволяет в полной мере использовать возможности СУБД, а также особенности решаемой задачи, делая работу с данными максимально эффективной. Схематически процесс взаимодействия приложения с СУБД, Рисунок 2.

МОДУЛЬНОСТЬ ХРАНИЛИЩ ДАННЫХ

Применение подхода NORM для гармонизации взаимодействия объектного и реляционного подходов к работе с данными, вместе с тем, не позволяет утверждать, что удалось предложить достойную альтернативу ORM в части технологичности процесса разработки. Изначальная претензия к реляционному хранению была и остается в слишком низком уровне абстракций, которыми приходится пользоваться в процессе решения прикладных задач, поскольку стандартные инструменты для описания структур хранения и обработки данных достаточно низкоуровневые — это таблицы, представления, функции и еще ряд сущностей, стандартных для реляционных СУБД. Однако прикладные сущности имеют более сложную структуру и поведение, что и пытались в свое время представить в виде объектной парадигмы, внося объекты в СУБД или моделируя их стандартными средствами. Особенно ярко эта проблема проявляется при разработке крупных корпоративных приложений, когда количество сущностей (например, таблиц) измеряется тысячами и при этом требуется унификация их структуры и поведения для единообразия проектирования, разработки и сопровождения.

Для небольших команд разработчиков данная проблема решается «соглашениями», наставничеством и «лучшими практиками», однако такое решение в перспективе нестабильно и подвержено вариативности, что размывает технологическую основу разработки. Для крупных проектов унификация работы с базой данных выражается в создании внутренних платформ и конструкторов, автоматизирующих создание

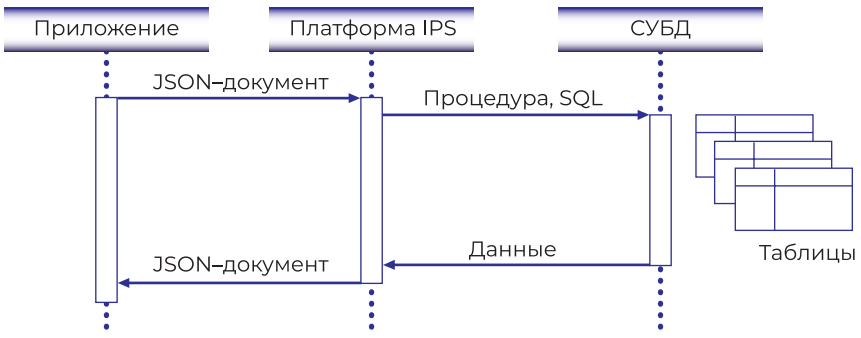

Рисунок 2 — Схема взаимодействия приложения и СУБД в платформе IPS.

структур данных, что позволяет добиться единообразия в типовых ситуациях. Реализуемая нами платформа IPS развивается похожим путем, поскольку ее разработка ведется в интересах вполне конкретных прикладных проектов и решения класса задач построения медицинских информационных систем, хотя жесткой привязки к решаемым задачам сама платформа и не имеет. Универсальность платформы в части унификации работы со хранилищами данных заключается в их открытой архитектуре.

Принцип манипулирования хранилищами данных заключается в конструировании таблиц, на которых они строятся, при помощи модулей — стандартных блоков, описывающих некий фрагмент функциональности хранилищ, который может включать колонки таблицы, триггеры, ограничения, индексы, хранимые процедуры (функции), представления. Сами по себе модули являются элементами платформы, состав которых можно расширять согласно потребностям разработки.

Применение модулей для создания хранилищ решает в том числе задачу повышения уровня абстракции при работе с данными: разработчик уже в стандартных ситуациях не мыслит в терминах отдельных полей таблиц и триггеров — модуль предоставляет ему уже готовый типовой функционал, преобразуя его в конечном итоге в конкретные таблицы и процедуры, но при этом существенно повышая унификацию и скорость разработки.

В качестве примера таких модулей можно привести модуль «Журналирование», который при подключении к хранилищу автоматически создает триггер, формирует в нем процедуру анализа изменяемых данных в конкретной таблице, реализующей данное хранилище, которая сохраняет измененные данные в журнал аудита изменений. Очевидно, что не для любого хранилища требуется аудит, но там, где он нужен, его реализация при помощи модуля будет гарантированно выполнена единообразно, в отличие от ситуации, когда те же манипуляции с таблицами производил разработчик руками.

Другой пример более сложного модуля «Версионность», который при применении к хранилищу формирует дополнительную таблицу версий, триггер на изменение данных и уникальную процедуру по переносу версионных записей в дополнительную таблицу.

Отчасти подобные методы работы могут напоминать множественное наследование в объектной парадигме, когда определенный класс наследует свойства от нескольких других базовых классов, в данном случае, если хранилище назовем «Документы», то оно может быть описано в стиле ООП как: «Документы: Базовый, Расширенный, Журналирование, Версионность, Статус», где «Базовый, Расширенный, Журналирование, Версионность, Статус» — это модули, от которых новая сущность будет «наследовать» свойства. При этом, как и в случае ООП при описании сущностей разработчиком описываются его уникальные свойства, а унаследованные добавляются автоматически типовым образом.

РЕАЛИЗАЦИЯ МОДУЛЬНЫХ JSON-ХРАНИЛИЩ В ПЛАТФОРМЕ «ИНТЕРИН IPS» ДЛЯ POSTGRESQL

Активная позиция государства по вопросам импортозамещения в том числе в ИТ-сфере, а также рост популярности СУБД PostgreSQL заставляет пристальней обратиться к этой СУБД. Можно с удовлетворением отметить, что начиная с версии PostgreSQL 9.4, вышедшей в конце 2014 года, представлен отдельный тип данных JSONB, значительно усиливающий уже имеющиеся NoSQL характеристики базы данных. Затем добавлялись новые операторы и функции, что позволило еще более эффективно работать с данными, хранящиеся в JSONB формате. В том числе, тип JSONB позволяет использовать JSON-объекты в запросах, причем достаточно эффективно.

ООО «Интерин технологии» — один из ведущих разработчиков корпоративных медицинских информационных системы [13] в своей работе применяет собственную программную платформу «Интерин IPS» в разработке прикладных решений. В настоящее время завершена разработка новой версии платформы «Интерин IPS», ориентированная на использование СУБД PostgreSQL. Это позволяет существенно расширить возможности работы с хранилищами, максимально используя встроенные средства СУБД, что в свою очередь повышает эффективность вычислений и влияет на саму архитектуру хранения.

Основные принципы реализации модульных JSON-хранилищ для PostgreSQL следуют из необходимости решения проблем, возникающих при взаимодействии объектного подхода к обработке и реляционного к хранению данных, плюс учета сильных сторон ORM-подхода к конструированию хранилищ данных:

1. Хранилище и прикладной код взаимодействуют в терминах JSON-документов, структура которых содержит определенные оговоренные элементы, но в целом произвольна;
2. Хранение данных происходит в штатных таблицах в смешанном виде: SQL / NoSQL в зависимости от решаемой прикладной задачи;
3. Структура таблиц, реализующих хранилища, задается при помощи предопределенных модулей, обеспечивающих типовые варианты использования;
4. Состав модулей является открытым и позволяет его расширять для учета востребованных типовых способов работы с данными.

Прежде чем приступить к описанию технической реализации предложенных принципов, рассмотрим в качестве примера подсистему «Справочники», реализованную с использованием модульных JSON-хранилищ. Отметим, что в рассматриваемой подсистеме будут размещаться только плоские списки произвольной структуры, не содержащие детальных подсписков. Очевидно, что даже при таком ограничении подавляющее большинство справочных данных может быть отражено в подобной структуре, например, НСИ Минздрава России полностью представлен в виде плоских справочников описанного вида и содержит более 1500 справочников [16]. Интересующая нас подсистема должна обладать следующими свойствами:

1. Поддерживать множество справочников различного состава полей;
2. Обеспечивать быстрый реляционный доступ к данным к необходимым полям;
3. Поддерживать универсальный метод импорта/экспорта данных;
4. Поддерживать единые типовые методы манипуляции с данными: создание, редактирование, удаление;
5. Поддерживать версионность данных;
6. Поддерживать контроль уникальности данных в рамках справочника;
7. Поддерживать авторизацию изменений.

Приведённый перечень иллюстрирует, что хоть речь идет о множестве различных справочников, но действия с ними должны быть типовыми, что наиболее близко к объектной парадигме с обобщением универсальных методов и свойств и специализацией особенностей экземпляров. Для эффективного хранения и манипуляции с данными хранилищу нужно поддержать с одной стороны объектный способ взаимодействия с прикладной системой, чтобы не требовалось писать отдельные обработчики на каждый экземпляр типа «Справочник», а с другой стороны обеспечить реляционные свойства для решения задач производительности и верификации данных.

Учитывая, что описанные требования относятся не только к приведенному примеру подсистемы «Справочники», но являются типовыми для большинства прикладных задач, удобно иметь готовые модули, которые бы позволяли для каждого конкретного хранилища добавить необходимые свойства, причем сделать это универсальным образом без ручной модификации таблиц и связанных с ними сущностей. Выше мы дали определение *модуля как фрагмента функциональности хранилищ, который может включать колонки, триггеры, ограничения, индексы, хранимые процедуры (функции), представления*. Каждый модуль может иметь параметры применения: например, модуль версионности может иметь настройку по максимальному сроку хранения или по количеству хранимых версий.

Описание каждого модуля включает в себя шаблоны DDL и/или функций для:

1. Добавления модуля с указанными параметрами к указанному хранилищу;
2. Удаления модуля из хранилища;
3. Перенастройки параметров уже существующего модуля.

Состав модулей конкретной системы, реализуемой на платформе расширяемый, в качестве стартового предложен следующий перечень:

1. «Базовый» — обязательный модуль, включающий: схему БД + имя хранилища + колонку ID с первичным ключом по ней.
2. «Версионность» — колонка «Версия» в основной таблице + дополнительная таблица журнала версий + триггер, сохраняющий версии в журнале при изменении данных в основной таблице.

3. «Журналирование» — триггер, который пишет в отдельную таблицу журнала операции по изменению данных в хранилище.
4. «Дополнительные поля» — колонка типа JSONB для размещения в хранилище объекты JSON произвольной структуры.
5. «Типизация» — колонка «Тип документа», позволяющая различать типы записей внутри хранилища.
6. «Статус» — комплект колонок контроля состояния записи, автора и даты внесения / изменения / деактуализации записи.
7. «Пользовательская идентификация» — комплект колонок «Код», «Название», «Ссылка на родительскую запись», которые наиболее часто применяются в прикладных системах для базовой идентификации документа в хранилище. Модуль содержит автоматическую индексацию добавляемых колонок.
8. «Колонки расширения» — дополнительные (нестандартные) колонки в таблице с указанием имён, описаний и типов данных, признаками индексации и наличия внешнего ключа на другое хранилище, передаваемые данному модулю в качестве параметров.
9. «Индексация» — список дополнительных индексов помимо входящих в другие модули, применяемых к основной таблице хранилища.
10. «Представления» — список view, которые строятся по основной таблице хранилища, в частности, для обеспечения реляционного доступа к расширенным полям, хранящимся в поле JSONB, если их по каким-то причинам нет необходимости или возможности выносить в качестве отдельных колонок в таблицу.

Описание хранилища в платформе сводится к перечню включенных модулей с их параметрами. По описанию хранилища генерируется DDL-скрипт, который может создать новую таблицу хранилища, если ее еще не было или применить изменения в описании к существующей таблице.

Для реализации модульных JSON-хранилищ в платформе реализованы следующие типы компонент:

- Табличные модули содержат программное описание отдельных модулей, описанных выше;
- Таблицы содержат описание хранилищ как списка примененных к ним экземпляров табличных модулей.

В качестве примера реализации табличного модуля рассмотрим модуль «Статус», который включает в себя три серверные функции:

- TM_STATUS__ADD — добавление модуля в хранилище;
- TM_STATUS__DROP — удаление модуля из хранилища;
- TM_STATUS_BIU — описывает поведение модуля в процессе работы с хранилищем.

Модуль «Статус» имеет параметризацию, которая позволяет включать отдельные наблюдаемые показатели (состояние записи, даты изменений, авторы), поэтому для применения модуля к хранилищу доступен интерфейс задания параметров экземпляра модуля, Рисунок 3.

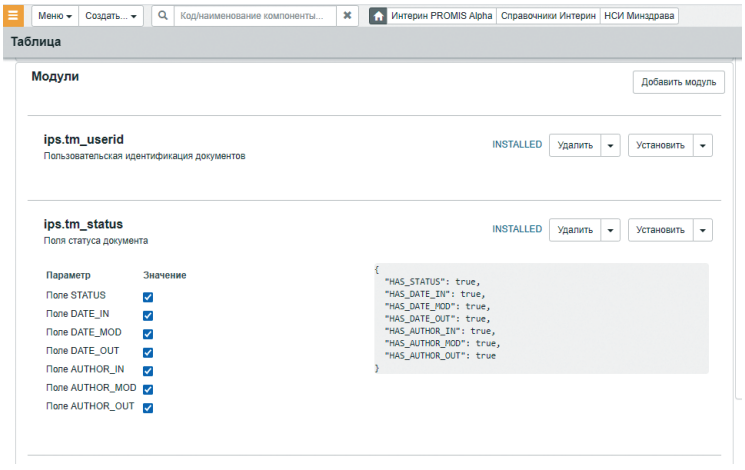

Рисунок 3 — Задание параметров экземпляру табличного модуля.

Ниже приведен фрагмент реализации функции TM_STATUS__ADD для применения модуля «Статус» к хранилищу, Рисунок 4.

В результате применения группы табличных модулей к хранилищу формируется обычная таблица, включающая все колонки, триггеры и индексы, которые описаны соответствующими табличными модулями, включенными в метаописание: «Базовый», «Дополнительные поля», «Пользовательская идентификация», «Статус», «Типизация», «Версионность». Фрагмент DDL-скрипта полученной таблицы для хранилища «Справочники» приведен на Рисунок 5.

В результате выполнения инструкций метаописания хранилища получается достаточно сложная структура данных, полностью состоящая из типовых табличных модулей и сгенерированная автоматически платформой.

В части, касающейся объектного интерфейса работы с хранилищем, необходимо отметить,

```
CREATE OR REPLACE FUNCTION ips.tm_status__add(p_schema text, p_table text, params jsonb DEFAULT NULL::jsonb)
  RETURNS void
  LANGUAGE plpgsql
AS $function$
declare
  v_schema_table text := ips.get_object_name(p_schema, p_table);
  has_status bool;    has_date_in bool;    has_date_mod bool;   has_date_out bool;
  has_author_in bool; has_author_mod bool; has_author_out bool;
begin
  if params is null then
    params := '{
      "HAS_STATUS": true,
      "HAS_DATE_IN": true,
      "HAS_DATE_MOD": true,
      "HAS_DATE_OUT": true,
      "HAS_AUTHOR_IN": true,
      "HAS_AUTHOR_MOD": true,
      "HAS_AUTHOR_OUT": true
    }'::jsonb;
  end if;
  perform ips.ddl_audit(p_schema, p_table, params);

  has_status     := coalesce(params->>'HAS_STATUS', 'false');
  has_date_in    := coalesce(params->>'HAS_DATE_IN', 'false');
  has_date_mod   := coalesce(params->>'HAS_DATE_MOD', 'false');
  has_date_out   := coalesce(params->>'HAS_DATE_OUT', 'false');
  has_author_in  := coalesce(params->>'HAS_AUTHOR_IN', 'false');
  has_author_mod := coalesce(params->>'HAS_AUTHOR_MOD', 'false');
  has_author_out := coalesce(params->>'HAS_AUTHOR_OUT', 'false');

  if has_status then
    execute $$ alter table $$||v_schema_table||$$ add column status int NOT NULL DEFAULT 0; $$;
  end if;
  if has_date_in then
    execute $$ alter table $$||v_schema_table||$$ add column date_in timestamptz; $$;
  end if;
  if has_date_mod then
    execute $$ alter table $$||v_schema_table||$$ add column date_mod timestamptz; $$;
  end if;
```

Рисунок 4 — Функция применения модуля «Статус» к хранилищу.

```
CREATE TABLE wix.nsi (
    "id"                            uuid,
    "doc_type"                      text,
    "code"                          text,
    "name"                          text,
    "parent_id"                     uuid,
    "status"                        integer DEFAULT 0,
    "date_in"                       timestamp with time zone,
    "date_mod"                      timestamp with time zone,
    "date_out"                      timestamp with time zone,
    "author_in"                     text,
    "author_mod"                    text,
    "author_out"                    text,
    "ext"                           jsonb,
    "rev"                           integer DEFAULT 1,
    "nsi_uid"                       text,
 CONSTRAINT nsi_PK PRIMARY KEY (id)
)
/
CREATE INDEX nsi_PK ON wix.nsi
  USING btree (id);
/
CREATE INDEX nsi__nuk_code ON wix.nsi
  USING btree (code);
/
CREATE TRIGGER tm_status_BIU
BEFORE INSERT OR UPDATE
ON wix.nsi FOR EACH ROW
EXECUTE FUNCTION ips.tm_status_biu('true', 'true', 'true', 'true', 'true', 'true', 'true')
/
CREATE TRIGGER tm_revisions_BU
```

Рисунок 5 — Результирующая таблица, реализующая хранилище с примененными к нему модулями.

что манипуляции с данными (создание, изменение, удаление) выполняются не напрямую с реализующими хранилище таблицами средствами SQL, а через универсальный функциональный интерфейс «*JSDB_PUT(p_doc jsonb)*», который получает на выход произвольный JSON-документ, содержащий обязательные метаданные, далее самостоятельно определяет хранилище и метод вызывает нужную функцию вставки данных, Рисунок 6. Это весьма существенный момент, поскольку вставка данных происходит через трансляцию пришедшего объекта в реляционную структуру при помощи специализированной для конкретного хранилища функции вставки. Функция вставки данных в хранилище также генерируется автоматически при создании и модификациях хранилища исходя из того, какие табличные модули к нему применены.

На иллюстрации (Рис. 6) видно, что функция JSDB_PUT не самостоятельно вносит данные в хранилище, а находит специализированную функцию вставки для конкретного хранилища и передает данные в нее, а функция вставки в хранилище уже реализует извлечение данных из пришедшего JSON-объекта и размещает их в нужные поля таблицы согласно метаописанию хранилища, Рисунок 7. Приведенная функция вставки, как и другие интерфейсные функции хранилищ (чтение, удаление), генерируются автоматически платформой на основе метаописаний, исходя из состава выбранных табличных модулей.

Аналогичным образом выполняется извлечение данных из хранилищ, когда происходит обратное преобразование реляционной структуры в JSON-объект. При этом, модуль «Дополнительные поля», который включает в хранилище колонку EXT типа JSONB, позволяет выполнять сохранение и извлечение данных без описания каждого поля обрабатываемого документа. То есть, достигается возможность комбинирования SQL и NoSQL обработки данных полного спектра:

1. можно разложить все поля пришедшего JSON-документа в определённые колонки таблицы согласно метаописанию хранилища, и тогда таблица является полностью реляционной;
2. можно разложить часть полей пришедшего JSON-документа в реляционную часть

```
declare
  v_db text := p_doc->>'_DB';
  v_id uuid;
begin
  if not is_ident(v_db) then
    RAISE '"%" - недопустимое значение для кода хранилища.', v_db;
  end if;

  execute 'select '||v_db||'__put($1)'
  into v_id
  using jsdb_clear_state(p_doc);

  return v_id;
end;
```

Рисунок 6 — Реализация функции JSDB_PUT.

```
declare
  v_id uuid;
  r wix.nsi%ROWTYPE;
begin
  if p_doc->>'_DB' is distinct from 'wix.nsi' then
    RAISE '"%" - недопустимое значение. Код хранилища должен быть равен "wix.nsi"', p_doc->>'_DB';
  end if;

  r.id := p_doc->>'_ID';
  r.doc_type := p_doc->>'_TYPE';
  r.code := p_doc->>'CODE';
  r.name := p_doc->>'NAME';
  r.parent_id := p_doc->>'PARENT_ID';
  r.status := p_doc->>'_STATUS';
  r.rev := p_doc->>'_REV';
  r.nsi_uid := p_doc->>'NSI_UID';
  r.ext := (p_doc - '{_DB,_ID,_TYPE,CODE,NAME,PARENT_ID,_STATUS,DATE_IN,_MODIFIED,DATE_OUT,AUTHOR_IN,

  if r.ext::text = '{}' then r.ext := null; end if;

  insert into wix.nsi(id , doc_type, code, name, parent_id, status, rev, nsi_uid, ext)
  values (r.id , r.doc_type, r.code, r.name, r.parent_id, r.status, r.rev, r.nsi_uid, r.ext)
  on conflict (id) do update
  set id=r.id , doc_type = r.doc_type, code = r.code, name = r.name, parent_id = r.parent_id, status
  returning id into v_id;

return v_id;
end;
```

Рисунок 7 — Специализированная функция вставки данных в хранилище.

таблицы, а часть оставить в колонке расширения EXT;
3. можно вообще не раскладывать никакие данные из JSON-документа, а положить его целиком в колонку расширения, тем самым получив NoSQL хранилище.

При этом функционально с точки зрения прикладного разработчика методы размещения и извлечения данных будут всегда одними и теми же, что позволяет проектировать хранилища наиболее эффективным образом для решения конкретных прикладных задач, избегая как излишней структуризации, когда она не требуется, так и излишних вычислений по извлечению неструктурированных данных, если необходима их реляционная обработка. В плюс к этому, всё описание и манипуляция с хранилищами выполняется на обобщенном уровне табличных модулей, позволяющих не прибегать к пользовательским DDL-операциям над хранилищами, что делает программный код более единообразным, сокращает количество возможных ошибок и снижает стоимость сопровождения.

ВЫВОДЫ

Описанная реализация JSON-хранилищ данных как части платформы «Интерин IPS», позволяет:

1. использовать сильные стороны подхода NORM и обеспечить согласование средств реляционного хранения данных с объектным подходом к их обработке;
2. воспользоваться элементами более высокоуровневого конструирования хранилищ данных, присущего методологии ORM за счет чего унифицировать и в значительной мере автоматизировать типичные операции с хранилищами.

Эффективность описанного подхода хорошо иллюстрируется практическим применение платформы «Интерин IPS» в реализации сложных программных модулей в Медицинской информационной системе Интерин PROMIS Alpha, обеспечивающих как обработку реляционных данных таких как складской и финансовый учет, а также с объектами сложной структуры как медицинские документы.

ЛИТЕРАТУРА/REFERENCES

1. Stonebraker M, et al. Third Generation Database System Manifesto. ACM SIGMOD Record. 1990; 19(3).
2. Atkinson M, et al. The Object-Oriented Database System Manifesto. Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases. Kyoto, Japan, 1989. New York, N.Y.: Elsevier Science, 1990.
3. Date CJ, Darwen H. Foundation for Object/Relational Databases: The Third Manifesto (2d edition). Reading, Mass.: Addison-Wesley, 2000.
4. Дейт К. Дж. Введение в системы баз данных. — М.: Вильямс, 2006. [Date CJ. Introduction to Database Systems, M.,2006. (In Russ).]
5. Кузнецов С.Д.. Объектно-ориентированные базы данных — основные концепции, организация и управление: краткий обзор. CIT Forum. [Kuznetsov SD Object Oriented Databases — Basic Concepts, Organization and Management: An Overview. CIT Forum. (In Russ).] Доступно по: http://citforum.ru/database/articles/art_24.shtml. Дата обращения: 20 октября 2021.
6. Кузнецов С.Д. Три манифеста баз данных: ретроспектива и перспективы. CIT Forum. [Kuznetsov SD Three Database Manifestos: Retrospective and Perspective. CIT Forum. (In Russ).] Доступно по: http://citforum.ru/database/articles/manifests. Дата обращения: 20 октября 2021.
7. Базы данных. Тенденции общемировые и в России. [Databases. Global trends in Russia (In Russ).] Доступно по: https://habr.com/ru/post/533880. Дата обращения: 20 октября 2021.
8. DB-Engines Ranking. https://db-engines.com/en/ranking.
9. Meyer B. The Future of Object Technology. IEEE Computer. 1998; 31(1).
10. Scott W. Ambler: Mapping Objects to Relational Databases: O/R Mapping In Detail http://www.agiledata.org/essays/mappingObjects.html.
11. Kincaid T. Exploring Eliminating the ORM from Your Next PostgreSQL Application — Feb 23, 2021. https://www.enterprisedb.com/blog/how-no-object-relational-mapping-norm-improves-application-performance-postgreSQL.

12. Dombrovskaya H, Novikov B, Bailliekova A. PostgreSQL Query Optimization: The Ultimate Guide to Building Efficient Queries — Apress, Berkeley, CA, 2021.
13. Медицинская информационная система «Интерин». [Healthcare Information System «Interin»] http://www.interin.ru.
14. Белышев Д.В., Кочуров Е.В. Анализ методов хранения данных в современных медицинских информационных системах // Программные системы: теория и приложения: электрон. научн. журн. — 2016. — №2(29). — С.85-103. [Belyshev DV, Kochurov EV. Analysis of Data Storage Methods for Modern Healthcare Information Systems. Program systems: theory and applications. 2016; 2(29): 85-103. (In Russ).]
15. Белышев Д.В., Кочуров Е.В. Перспективные методы работы с данными в медицинских информационных системах. // Программные системы: теория и приложения: электрон. научн. журн. — 2016. — № 3(30). —С.79-97. [Belyshev DV, Kochurov EV. Advanced Methods of Data Management in Healthcare Information Systems. Program systems: theory and applications, 2016; 3(30): 79-97. (In Russ).]
16. Реестр нормативно-справочной информации Минздрава России https://nsi.rosminzdrav.ru/#!/refbook. [Register of normative and reference information of the Ministry of Health of Russia (In Russ).]