УДК 004.415.2
DOI: 10.15827/0236-235X.140.644-653
Дата подачи статьи: 28.06.22, после доработки: 25.08.22
2022. Т. 35. № 4. С. 644–653

Динамическая схема GraphQL в реализации интегрированной информационной системы

Б.А. Черныш [1], *аспирант, borisblack@mail.ru*
А.В. Мурыгин [1], *д.т.н., профессор, зав. кафедрой информационно-управляющих систем, avm514@mail.ru*

[1] *Сибирский государственный университет науки и технологий им. академика М.Ф. Решетнева, г. Красноярск, 660037, Россия*

В статье рассматривается возможность использования инструментария GraphQL с динамически изменяемой схемой данных. Стандартным путем определения типов данных и операций в GraphQL является статическая схема. При ее использовании вся структура данных определяется заранее и не может быть изменена динамически во время работы приложения, обслуживающего запросы. Это обстоятельство не позволяет применять GraphQL в приложениях, где структура данных может динамически изменяться. Для решения задачи используется подход, состоящий в хранении схемы данных в памяти приложения и перегенерации этой схемы в случае изменения метаданных.

В данной работе приводится способ реализации этого подхода на примере разработанной авторами программной платформы SciCMS. Особенностью системы является ее ориентированность на требования по работе с данными о технически сложной продукции. Эти требования включают хранение данных в древовидном представлении с оптимизирующими механизмами выборки, контроль версий изделий, возможность использования разных языковых представлений изделий, управление жизненным циклом изделий, расширенные возможности интеграции с несколькими источниками данных.

В статье описаны приемы, методики и технологии, задействованные при построении системы, приводятся схемы и диаграммы UML основных структур и процессов ядра приложения. Описываются детали реализации отдельных подсистем платформы.

Проведены экспериментальные выборки данных с целью оценки эффективности выполнения запросов с использованием соединений (Join). На основании полученных данных выбраны наиболее эффективные инструменты оптимизации выборки иерархических данных. Представлены возможности платформы по ее интеграции с другими системами в рамках единого информационного пространства.

Ключевые слова: *GraphQL, SciCMS, Headless CMS, информационная система, программная платформа, ядро, интерфейс, API, система управления, версионирование, жизненный цикл, база данных, единое информационное пространство.*

Язык клиент-серверного взаимодействия GraphQL с каждым годом приобретает все более широкую популярность [1]. Его распространение связано как с выразительностью самого языка, так и с большим количеством инструментальных средств, позволяющих реализовать взаимодействие практически для любой современной программной среды. Многие технологические компании не только взяли на вооружение GraphQL, но и внесли (и продолжают вносить) существенный вклад в его развитие [2, 3]. Основные преимущества GraphQL перед традиционным архитектурным стилем REST заключаются в строгой типизации, иерархичности данных, в отсутствии проблем чрезмерной и недостаточной выборки данных, а также необходимости управления конечными точками REST [4]. Стандартным путем определения типов данных и операций в GraphQL является статическая схема. При ее использовании вся структура данных определяется заранее и не может быть изменена динамически во время работы приложения, обслуживающего запросы. Это обстоятельство не позволяет применять GraphQL в приложениях, где структура данных может динамически изменяться. Данная особенность послужила причиной отказа от использования технологии GraphQL при проектировании авторами ядра интегрированной информационной *системы управления документами* (СУД) [5]. Взамен для построения API был использован гибридный подход, основанный на REST и сочетающий преимущества иерархичных запросов с динамически изменяемой схемой данных. В процессе опытной эксплуатации системы были выявлены две важные особенности ее работы:

– динамическая природа структуры данных является источником многих ошибок в формировании, валидации и обработке запросов;

– частота изменения структуры данных настолько мала, что позволяет выполнять генерацию схемы динамически во время запуска приложения либо непосредственно перед запросом.

Эти факторы позволили пересмотреть вопрос о применении технологии GraphQL в варианте динамической генерации схемы в режиме runtime (во время выполнения приложения). Под динамической генерацией подразумевается возможность простого изменения определений структур метаданных с последующим пересозданием схемы GraphQL в памяти самим приложением.

Существующие подходы к построению динамического GraphQL предусматривают хранение метаданных схемы в кэширующем слое высокопроизводительного хранилища с возможностью их изменения на лету. Как быстрое хранилище может быть использована, например, резидентная система Redis [6]. В работе [7] описано применение в качестве основного хранилища DynamoDB с индексированием метаданных в Elasticsearch. Интересный способ описания метаданных и генерации схемы GraphQL приведен в [8]. Здесь используются не получивший признания среди разработчиков стандарт RDF и спецификации языков запросов к данным SPARQL и UGQL. В RDF в качестве единицы описания сущности применяется так называемая семантическая тройка, или триплет. Утверждение, высказываемое о сущности, имеет вид субъект–предикат–объект. Множество RDF-утверждений образуют граф, вершинами которого являются субъекты и объекты, а ребра отображают отношения. Ввиду несколько запутанной семантики и сложности реализации данная модель не получила широкого распространения.

СУД также реализует вышеописанные подходы хранения кэша схемы в памяти наряду с основным хранилищем в реляционной СУБД. Но при этом используется собственный формат описания и хранения метаданных посредством четко определенного языка DSL. Первичные метаданные описываются через конфигурационные файлы (формата YAML или JSON), на основании которых генерируются необходимые инструкции SQL для создания/модификации таблиц. При необходимости модификации схемы на лету выполняется управляющий API-запрос, содержащий данные о создаваемой/изменяемой сущности в аналогичном YAML/JSON-формате. Подобный декларативный подход (когда описывается, как должен выглядеть результат, а не какие изменения необходимо выполнить) в настоящее время широко применяется при проектировании информационных систем (яркий пример – экосистема Kubernetes [9]).

Платформа СУД относится к классу систем, называемых Headless CMS [10]. В отличие от традиционных CMS эти системы предполагают управление только контентом (телом), независимо от интерфейса (головы), в котором он будет использоваться. Существует большое количество готовых Headless CMS [10], однако ни одна из них полностью не соответствует требованиям к платформе СУД, а именно – отсутствует функционал версионирования и управления *жизненным циклом* (ЖЦ) объектов. Кроме того, большая часть решений является проприетарным ПО. В силу своей принадлежности к данному классу систем платформа СУД получила рабочее название Scientific CMS (SciCMS) как ориентированная на работу с высокотехнологичной продукцией.

Проектирование API

Принимая во внимание изначально заложенные требования к платформе и учитывая опыт эксплуатации предыдущей версии системы, в переработанном варианте использован GraphQL с его выразительными языковыми средствами и развитым инструментарием. Для обеспечения переносимости в качестве среды выполнения выбрана платформа Java, поддерживаемая большинством современных ОС. Ядро SciCMS построено на свободно распространяемом фреймворке Spring. Для реализации протокола GraphQL использована библиотека GraphQL Java. Поддерживаются обработка вложенных объектов и иерархические рекурсивные запросы. С целью оптимизации рекурсивные запросы реализуются непосредственно средствами СУБД [11]. Базовые операции с сущностями системы приведены в таблице 1. Поддерживаются версионность и мультиязычность. Имеется возможность настройки ЖЦ сущности (возможные состояния и переходы между ними) в специальном редакторе. Кроме того, платформа позволяет создавать пользовательские операции и привязывать их к сущностям определенного типа.

Сценарии взаимодействия с системой

Обобщенная диаграмма прецедентов приведена на рисунке 1.

Таблица 1

Базовые операции с сущностями

Table 1

Basic operations with entities

Операция	Описание
<имя_сущности>	Возвращает данные сущности
<имя_сущности_во_множественном_числе>	Возвращает данные сущностей с возможностями фильтрации, сортировки и постраничной выборки
create<имя_сущности >Version	Создает новую версию сущности
create<имя_сущности >Localization	Создает новую языковую локаль для сущности
update<имя_сущности>	Обновляет данные сущности
delete<имя_сущности>	Удаляет версию сущности
purge<имя_сущности>	Удаляет все версии сущности
lock<имя_сущности>	Блокирует сущность от изменения другими пользователями
unlock<имя_сущности>	Снимает с сущности блокировку от изменения другими пользователями
promote<имя_сущности>	Устанавливает новое состояние сущности. На переходы между состояниями могут создаваться специальные обработчики
<пользовательский_метод><имя_сущности>	На каждый тип сущности могут создаваться специальные обработчики с пользовательскими методами

Администратор, пользователь либо сторонняя система должны быть авторизованы в системе и иметь соответствующие разрешения на выполняемые операции. В случае внешней системы для авторизации используются *технологические учетные записи* (ТУЗ).

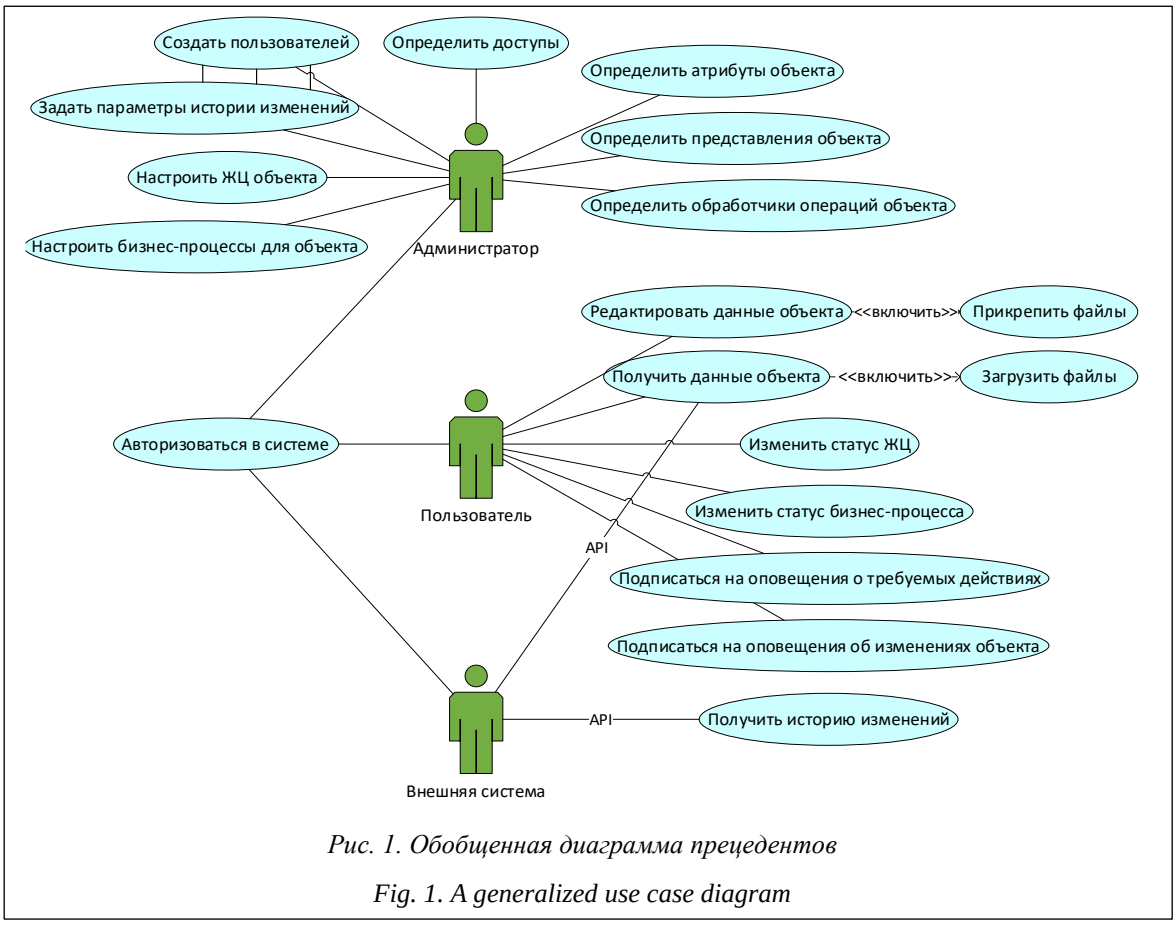

Рис. 1. Обобщенная диаграмма прецедентов

Fig. 1. A generalized use case diagram

В задачи администратора входят управление пользователями, ролями и доступами, а также определение структуры объектов системы, включая атрибуты, представления и обработчики. Кроме того, администратор настраивает параметры версионирования, локализации и управления ЖЦ.

Основные операции над объектами системы (ввод данных об объектах, добавление технической документации, изменение статуса ЖЦ) проводит инженерно-технический персонал. При необходимости любой пользователь может получать оповещения о необходимых операциях или изменениях, а также отслеживать историю изменений.

Внешняя система (приложение ERP, АСУ производства и т.д.) посредством прикладного программного интерфейса (API) также получает данные об учетных сущностях, включая историю изменений и состояние ЖЦ.

Базовый процесс редактирования данных об изделии представлен в виде диаграммы последовательности (см. http://www.swsys.ru/uploaded/image/2022-4/2022-4-dop/16.jpg).

Проектирование серверной части системы

Платформа SciCMS состоит из двух независимых модулей: клиентского (работающего в браузере) и серверного. Серверный модуль составляет ядро платформы. На рисунке 2 приведена диаграмма классов подсистемы загрузки метаданных этого модуля. Метаданные платформы определяются в виде YAML- или JSON-файлов, то есть для добавления новой сущности в систему достаточно описать ее характеристики и атрибуты в файле. При этом платформа сама создаст соответствующие структуры в БД и отследит их версии. Измене-

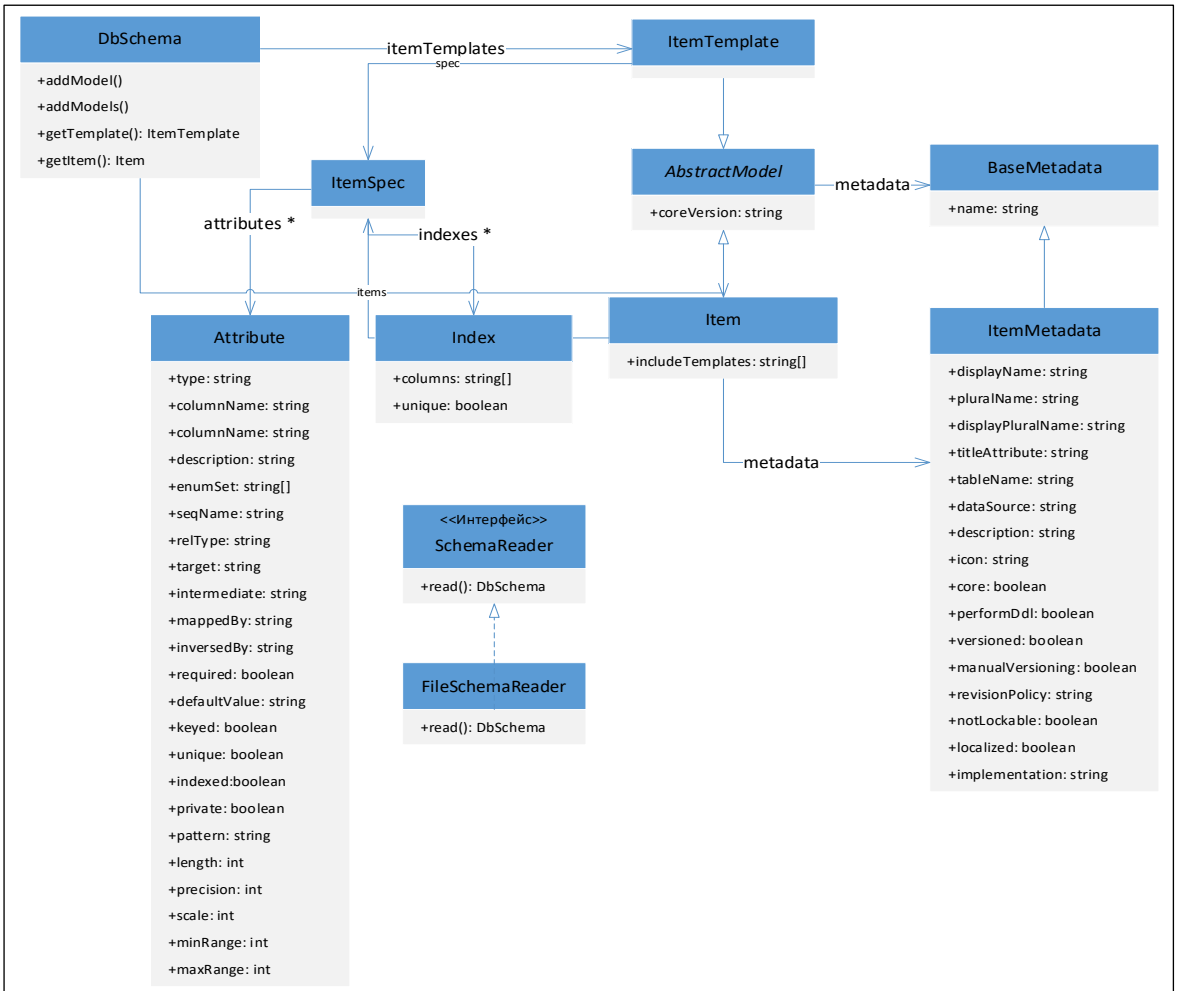

Рис. 2. Подсистема загрузки метаданных

Fig. 2. A metadata loading subsystem

ние файла схемы вызывает изменение в метаданных с последующим выполнением операций изменения (пересоздания) таблиц и перегенерации схемы GraphQL. Обработка файлов реализуется в классе FileSchemaReader через вызов метода read(). Данный метод возвращает объект типа DbSchema, который, в свою очередь, содержит коллекцию объектов Item с метаданными схемы.

Структура метаданных платформы

Структура метаданных [5] была переработана в пользу более эффективной организации связанных данных. Так, например, таблица core_items (см. http://www.swsys.ru/uploaded/image/2022-4/2022-4-dop/17.jpg) с метаданными сущностей предметной области в своем поле SPEC содержит описание атрибутов соответствующей сущности в JSON-формате. В качестве хранилища данных платформы используется реляционная СУБД. Поддерживаются БД Oracle и PostgreSQL. Схема данных разбита на наборы таблиц, которые отвечают за различные подсистемы (ядро, контроль доступа, управление ЖЦ, собственно данные) и именуются соответствующим префиксом. Все таблицы, помимо основных атрибутов, содержат служебные атрибуты аудита, управления версиями и ЖЦ.

Атрибут versioned определяет, является ли сущность версионируемой. Для целей версионирования каждая таблица хранит набор служебных атрибутов: generation, major_rev, minor_rev, is_current. Версии могут присваиваться автоматически (на основании заданной политики версий – атрибут revision_policy_id сущности) либо вручную. Данное поведение определяется атрибутом manual_versioning.

Аналогично versioned атрибут localized определяет, поддерживает ли сущность мультиязычность. Для этого в каждую таблицу включается атрибут locale.

Помимо версионирования и управления ЖЦ, еще одной важной особенностью SciCMS, отличающей ее от других подобных систем, является возможность использования связанных данных из разных источников в рамках единого информационного пространства. С этой целью каждая сущность содержит поле data_source. Все настройки источников данных задаются в конфигурационных файлах. Данная особенность системы позволяет с высокой эффективностью использовать SciCMS как интеграционную основу для многих систем, использующих РСУБД в качестве хранилища данных. Использование нескольких источников данных в рамках одной GraphQL-схемы дает возможность объединения различных сервисов с их собственными подграфами в единый интегрированный шлюз и является эффективной альтернативной таким мощным интеграционным решениям, как, например, GraphQL Federation [12].

Система позволяет хранить все возможные виды связей между сущностями (один-к-одному, многие-к-одному, один-ко-многим, многие-ко-многим). Для моделирования связей «многие-ко-многим» используются дополнительные промежуточные таблицы.

Для иерархической выборки связанных сущностей используется механизм Data Fetchers, эффективно решающий так называемую проблему N+1, заключающуюся в необходимости выполнять дополнительный запрос к связанной сущности для каждого элемента текущей выборки [13].

Изначально данная проблема в СУД решалась путем выполнения дополнительных соединений (Join) в исходном запросе [5]. В процессе опытной эксплуатации подход не оправдал себя при соединении трех и более таблиц, приводя к существенному замедлению выполнения запросов. В связи с этим было проведено небольшое экспериментальное исследование по измерению времени выполнения запросов в СУБД Oracle с различным числом соединений (от 0 до 6). Размер экспериментальной выборки составляет 500 записей. Все столбцы, участвующие в соединении, проиндексированы. Результаты измерений, среднее время и стандартное отклонение приведены в таблице 2. Стандартное отклонение рассчитано по формуле $S_0 = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(t_i - \bar{t})^2}$, где t_i – значение времени i-й выборки (мс); $\bar{t}$ – среднее значение; n – число запросов.

Зависимость времени выполнения и стандартного отклонения от числа соединений показана на рисунке 3.

График зависимости показывает, что изменение эффективности выполнения запросов происходит нелинейно, крутизна ее нарастает с каждым дополнительным соединением. Если при одном соединении среднее время составляет 148 мс, то для шести соединений оно превышает 1 секунду. По такому же закону увеличивается и стандартное отклонение, что гово-

Таблица 2

Время выполнения запросов с соединениями (мс)

Table 2

Query execution time with connections (ms)

Выборка	Число соединений						
	0	1	2	3	4	5	6
1	57	204	355	406	397	463	1771
2	40	149	277	408	631	763	1043
3	49	140	267	392	614	755	1018
4	42	141	301	400	622	753	1002
5	41	144	273	407	609	753	1011
6	49	134	306	408	620	805	1035
7	36	149	279	418	629	754	994
8	34	134	283	397	614	762	1038
9	30	145	292	418	628	755	1019
10	50	141	280	398	615	765	1035
t_i	42,8	148,1	291,3	405,2	597,9	732,8	1096,6
S_0	15,1	48,4	90,5	121,5	191,3	237,6	398,6

рит еще и о деградации стабильности времени выполнения запроса. Это, в свою очередь, требует тщательной настройки индексов таблиц, а также учета особенностей работы оптимизаторов конкретных СУБД. Описать математически закономерность изменения времени выполнения не представляется возможным, так как оно в большой степени зависит от таких факторов, как структура таблиц, размер данных, алгоритмы компиляции и оптимизации СУБД, характеристики вычислительной среды, топология кластеров и сети и т.д.

Суть работы механизма Data Fetchers заключается в выборке связанных сущностей одним запросом к БД по условию принадлежности этих сущностей к одной из уже выбранных записей верхнего уровня и последующем связывании их непосредственно в приложении. Такой способ гарантирует линейное нарастание времени выполнения от числа соединений и избавляет от проблемы повторной выборки уже запрошенных однажды данных. Запросы при этом выполняются независимо друг от друга, что при необходимости позволяет использовать различные источники данных для каждого из них.

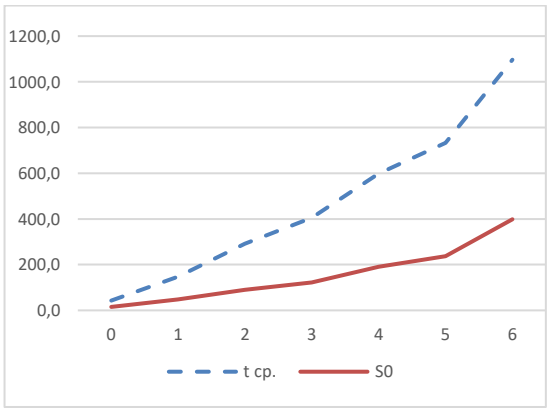

Рис. 3. Среднее время выполнения запроса и стандартное отклонение в зависимости от числа соединений

Fig. 3. Average query execution time and standard deviation depending on the number of connections

Управление ЖЦ

Для управления ЖЦ служит таблица метаданных core_lifecycles (рис. 4). Каждая таблица в схеме имеет собственный атрибут lifecycle_id, являющийся внешним ключом к первичному ключу ID этой таблицы. В столбце SPEC хранится информация об этапах ЖЦ и переходах между ними. Атрибут implementation хранит название обработчика ЖЦ, позволяя назначать пользовательские обработчики событий переходов ЖЦ. Таблица core_allowed_lifecycles определяет, какие ЖЦ могут быть выбраны для определенной сущности.

Средства безопасности и контроля доступа

Система контроля доступа была во многом переработана с целью более гибкого распределения разрешений. Метаданные ACL также хранятся в реляционной СУБД. На ER-диаграмме представлены основные таблицы (рис. 5).

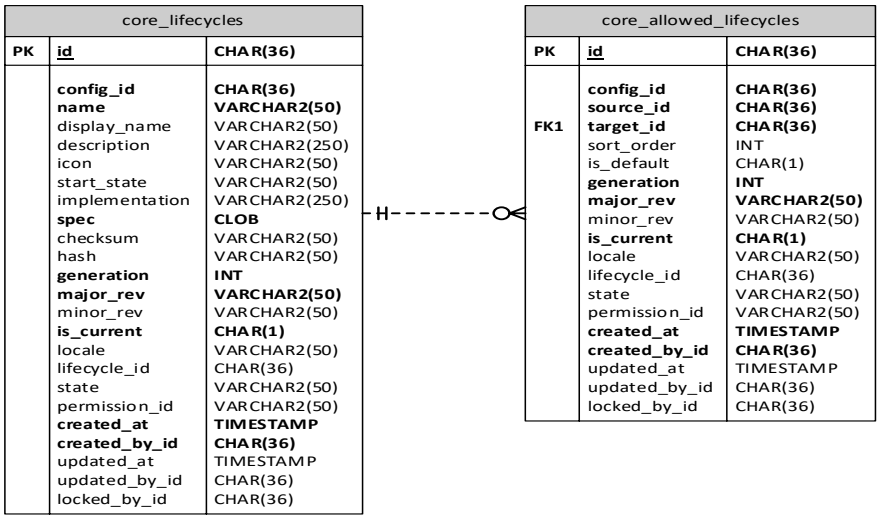

Рис. 4. Таблицы для управления ЖЦ

Fig. 4. Life cycle management tables

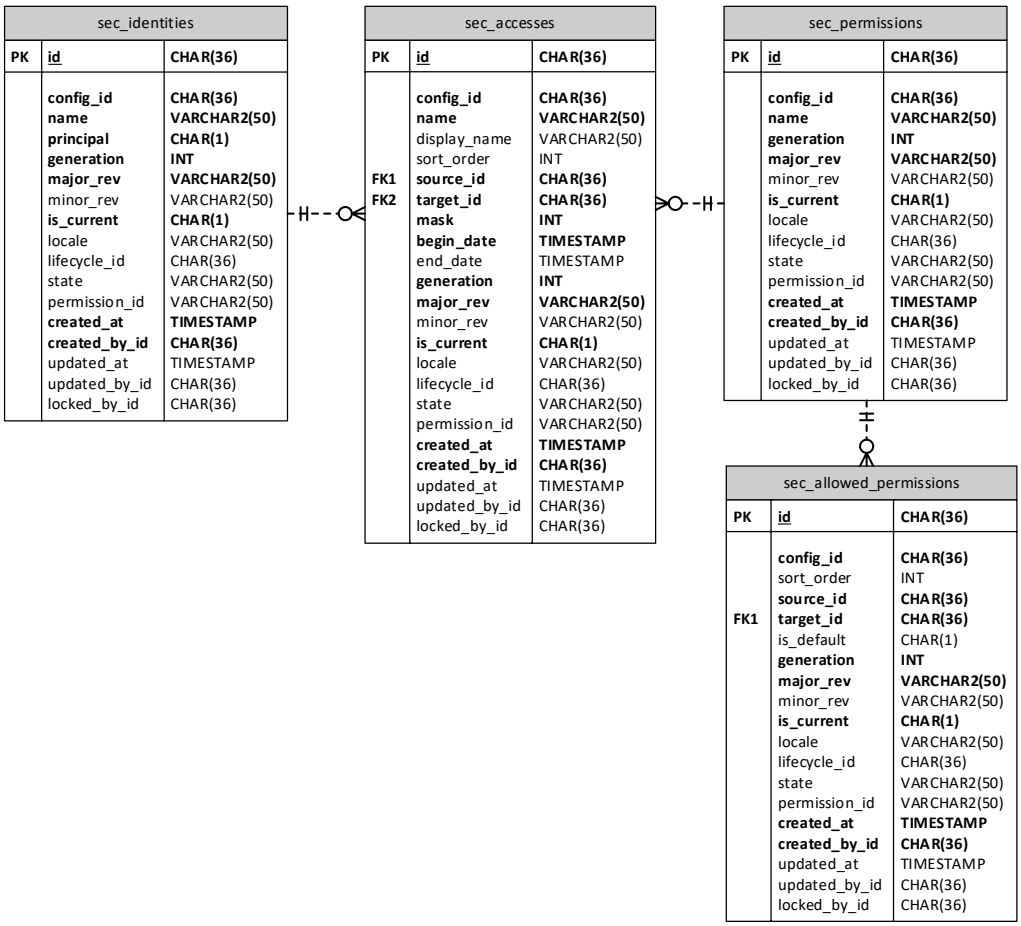

Рис. 5. Таблицы ACL

Fig. 5. ACL tables

Основу ACL составляет механизм разрешений (permission). Каждое разрешение включает в себя список контроля доступа для определенных пользователей и ролей с указанием двоич-

ной маски доступа. Маска доступа включает в себя флаги чтения (R – read), редактирования (W – write), создания (C – create), удаления (D – delete) и администрирования (A – administration) и позволяет в одном целом числе от 0 до 31 гибко сконфигурировать набор прав. При программном построении структуры будущих SQL-запросов вызываются специальные обработчики, ответственные за фильтрацию данных и операций на основании ACL.

Для подключения данных обработчиков используется широко применяемый механизм (паттерн) Chain of Responsibility (цепочка обязанностей) [14] с целью дополнить структуру будущего запроса ограничивающими выражениями на основании ACL. Таким образом, контроль доступа осуществляется на уровне SQL-запросов, что исключает избыточность данных, избавляет от необходимости их последующей обработки и существенно уменьшает потоки ввода-вывода.

Выводы

Технология GraphQL является одним из лидеров среди инструментов предоставления публичного API в распределенных средах. В силу преимуществ перед традиционным архитектурным стилем REST (строгая типизация, иерархичность данных, отсутствие проблем overfetching и underfetching, отсутствие множества конечных точек) многие технологические компании успешно применяют GraphQL в промышленной разработке. Несмотря на статическую природу, схема GraphQL может быть заново сгенерирована в процессе работы приложения. Именно такая динамическая перегенерация положена в основу работы системы SciCMS. Создание/изменение сущностей и связей осуществляется декларативным путем с использованием YAML- или JSON-формата (такой подход нашел широкое применение, например, в экосистеме Kubernetes). Функционал SciCMS предусматривает возможность хранения объектов предметной области в древовидном представлении с оптимизирующими механизмами выборки, контроль версий, мультиязычность, управление ЖЦ, а также использование нескольких источников данных. Последнее обстоятельство позволяет применять SciCMS как интеграционную основу для многих систем, использующих РСУБД в качестве хранилища данных в рамках ЕИП. Использование нескольких источников данных в рамках одной GraphQL-схемы дает возможность объединения различных сервисов с их собственными подграфами в единый интегрированный шлюз и является эффективной альтернативной таким мощным интеграционным решениям, как, например, GraphQL Federation.

Литература

1. Тонкушин М.В., Гудков К.В. Сравнительный анализ технологий GraphQL и REST // Современные информационные технологии. 2019. № 29. С. 127–131.

2. Apollo GraphQL. URL: https://www.apollographql.com/ (дата обращения: 28.05.2022).

3. Netflix DGS Framework. URL: https://netflix.github.io/dgs/ (дата обращения: 28.05.2022).

4. Конин М.В., Соколова О.Д. Программные интерфейсы для управления данными, описывающими объекты с иерархической структурой // Проблемы оптимизации сложных систем: сб. тр. 2019. С. 616–620.

5. Черныш Б.А., Картамышев А.С. Разработка ядра интегрированной информационной системы // Программные продукты и системы. 2021. Т. 34. № 2. С. 561–566. DOI: 10.15827/0236-235X.134.237-244.

6. Кетов Д.К. Разработка API-доступа к данным с динамически изменяемой структурой на основе GraphQL // АСУИТ: мат. конф. 2019. С. 108–114.

7. Silveria A. GraphQL live querying with DynamoDB. ArXiv, 2020. URL: https://arxiv.org/abs/2008.00129 (дата обращения: 28.05.2022).

8. Gleim L., Holzheim T., Koren I., Decker S. Automatic bootstrapping of GraphQL endpoints for RDF triple stores. ASLD, 2020, pp. 119–134.

9. Манойло В.Е. Использование технологий Docker и Kubernetes для построения инфраструктуры сложных систем // Современное образование: Традиции и инновации. 2022. № 1. С. 135–139. DOI: 10.51623/23132027_122_135.

10. Артемьев Е.В., Гафаров Ф.М. Headless CMS. Концепция и преимущества // Интернаука. 2020. № 22-1. С. 10–12.

11. Chernysh B.A., Kartamyshev A.S., Murygin A.V. Hierarchical data model choosing in the information systems design in relational DBMS. Proc. FarEastCon, 2021, pp. 1–5. DOI: 10.1109/FarEastCon50210.2020.9271190.

12. Stunkel P., Bargen O., Rutle A., Lamo Y. GraphQL federation: A model-based approach. J. of Object Technology, 2020, vol. 19, no. 2, pp. 18:1–21. DOI: 10.5381/jot.2020.19.2.a18.

13. Иванов И.А. Решение проблемы «SELECT N+1» при обращении к базе данных при имплементации GraphQL на серверной и клиентской логике // Педагогика. Образование. Практика: сб. тр. конф. 2019. С. 93–95.

14. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Publ., 1995, 395 p.

Software & Systems
DOI: 10.15827/0236-235X.140.644-653

Received 28.06.22, Revised 25.08.22
2022, vol. 35, no. 4, pp. 644–653

A GraphQL dynamic schema in integrated information system implementation

B.A. Chernysh [1], *Postgraduate Student, borisblack@mail.ru*
A.V. Murygin [1], *Dr.Sc. (Engineering), Professor, Head of Department of Information and Control Systems, avm514@mail.ru*

[1] *Reshetnev Siberian State University of Science and Technology, Krasnoyarsk, 660037, Russian Federation*

Abstract. The paper discusses the possibility of using the GraphQL toolkit with a dynamically changing data schema. The standard way to define data types and operations in GraphQL is a static schema. When using it, the entire data structure is determined in advance and cannot be changed dynamically while the application serving requests is running. This circumstance does not allow using GraphQL in applications where the data structure can change dynamically. To solve this problem, there is an approach that consists in storing the data schema in the application memory, and regenerating this schema in case of metadata changes.

This paper presents a method for implementing this approach using the SciCMS software platform developed by the authors as an example. A feature of the system is its focus on the requirements for working with data on technically complex products. These requirements include data storage in a tree view with optimizing retrieval mechanisms, product version control, the ability to use different language representations of products, product lifecycle management, and advanced integration with multiple data sources.

The paper outlines the techniques and technologies involved in building the system, provides UML diagrams and diagrams of the main structures and processes of the application core. It also describes the implementation details of individual platform subsystems.

Experimental data sampling was carried out in order to evaluate the efficiency of executing queries using joins. The most effective tools for optimizing the selection of hierarchical data were selected based on the data obtained. The paper presents the possibilities of the platform for its integration with other systems within a single information space.

Keywords: GraphQL, SciCMS, Headless CMS, information system, program platform, core, interface, API, control system, versioning, lifecycle, database, single information space.

References

1. Tonkushin M.V., Gudkov K.V. Comparative analysis of GraphQL and REST technologies. *Modern Information Technologies*, 2019, no. 29, pp. 127–131 (in Russ.).

2. *Apollo GraphQL*. Available at: https://www.apollographql.com/ (accessed May 28, 2022).

3. *Netflix DGS Framework*. Available at: https://netflix.github.io/dgs/ (accessed May 28, 2022).

4. Konin M.V., Sokolova O.D. Programming interfaces for managing data describing objects with a hierarchical structure. *Proc. OPCS*, 2019, pp. 616–620 (in Russ.).

5. Chernysh B.A., Kartamyshev A.S. The core design of an integrated information system. *Software & Systems*, 2021, vol. 34, no. 2, pp. 561–566. DOI: 10.15827/0236-235X.134.237-244 (in Russ.).

6. Ketov D.K. Developing a GraphQL-based API to dynamic structured data. *Proc. ASUIT*, 2019, pp. 108–114 (in Russ.).

7. Silveria A. GraphQL live querying with DynamoDB. *ArXiv*, 2020. Available at: https://arxiv.org/abs/2008.00129 (accessed May 28, 2022).

8. Gleim L., Holzheim T., Koren I., Decker S. Automatic bootstrapping of GraphQL endpoints for RDF triple stores. *ASLD*, 2020, pp. 119–134.

9. Manoylo V.E. Using Docker and Kubernetes technologies to build the infrastructure of complex systems. *Modern Education: Traditions and Innovations*, 2022, no. 1, pp. 135–139. DOI: 10.51623/23132027_122_135 (in Russ.).

10. Artemyev E.V., Gafarov F.M. Headless CMS. Concept and benefits. *Internauka*, 2020, no. 22-1, pp. 10–12 (in Russ.).

11. Chernysh B.A., Kartamyshev A.S., Murygin A.V. Hierarchical data model choosing in the information systems design in relational DBMS. *Proc. FarEastCon*, 2021, pp. 1–5. DOI: 10.1109/FarEastCon50210.2020.9271190.

12. Stunkel P., Bargen O., Rutle A., Lamo Y. GraphQL federation: A model-based approach. *J. of Object Technology*, 2020, vol. 19, no. 2, pp. 18:1–21. DOI: 10.5381/jot.2020.19.2.a18.

13. Ivanov I.A. Solving the ″SELECT N+1″ problem when accessing the database when implementing GraphQL on the server and client logic. *Proc. Pedagogy. Education. Practice*, 2019, pp. 93–95 (in Russ.).

14. Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley Publ., 1995, 395 p.

Для цитирования

Черныш Б.А., Мурыгин А.В. Динамическая схема GraphQL в реализации интегрированной информационной системы // Программные продукты и системы. 2022. Т. 35. № 4. С. 644–653. DOI: 10.15827/0236-235X.140.644-653.

For citation

Chernysh B.A., Murygin A.V. A GraphQL dynamic schema in integrated information system implementation. *Software & Systems*, 2022, vol. 35, no. 4, pp. 644–653 (in Russ.). DOI: 10.15827/0236-235X.140.644-653.