ПРИНЦИПЫ ИДЕМПОТЕНТНОСТИ И КОНВЕРГЕНТНОСТИ В ОБЛАСТИ МИГРАЦИИ ДАННЫХ

Данильчик В. В., Прытков В. А.
Кафедра электронных вычислительных машин, Белорусский государственный университет информатики и радиоэлектороники
Минск, Республика Беларусь
E-mail: danilchican@mail.ru, prytkov@bsuir.by

Миграция схем данных является неотъемлемой задачей при переходе от одной структуры схем данных к другой. Сложность миграции данных заключается в наличии большой разницы между текущей структурой схемы данных и новой, а также достаточно большую сложность составляет наличие существующих зависимостей между объектами. В данной статье рассмотрены принципы, которые помогают определить алгоритм действий для миграции схемы данных и ее сущностей.

Актуальность анализа способов миграции схем данных обусловлена необходимостью постоянной поддержки структуры базы данных и ее последующей оптимизации при выходе новой версии программного продукта. В настоящее время процессу миграции данных часто сопутствуют проблемы их дублирования и возникновения ошибок, связанных с изменением структуры данных.

У большинства подходов есть общий принцип: им необходимо основание — некоторое эталонное состояние базы данных (БД), от которого можно отталкиваться. Что из себя представляет основание — это дамп структуры базы данных для версии, которая принята за базовую. Имея основание, впоследствии всегда можно будет создать БД с нуля. Чтобы получить БД со структурой самой последней версии, нужно применить к этой базе данных все миграции, созданные в процессе разработки.

При разработке программного продукта и, соответственно, написании кода разработчики пользуются системами контроля версий, такими как Git или Subversion. Данные системы позволяют вести совместный процесс разработки несколькими сотрудниками, хранить только разницу между версиями кода, отслеживать и, при необходимости, возвращаться к конкретной версии исходного кода. В то же время применение систем контроля версий при работе со структурами схем данных довольно ограничено.

Также, одной из причин миграции схем данных является наличие каких-либо аномалий либо увеличение нагрузки при анализе работы программного продукта с базой данных, т.к. быстрота работы БД напрямую зависит от ее структуры и выполненной оптимизации. В этом случае при миграции схемы данных, могут быть добавлены новые индексы на определенные столбцы таблиц.

Для того, чтобы повысить производительность системы, нужно проанализировать производительность базы данных. Для этого существует Схема Производительности (см. рис.1). Она фокусируется прежде всего на данных о производительности [1], следит за развитием событий сервера. «Событие» - что-либо, что сервер делает, который занимает время. Собранные события сохранены в таблицах в базе данных. Эти таблицы могут быть запрошены, используя «SELECT» операторы как другие таблицы.

Database - Refresh	Collation	Tables	Size - Compute
information_schema	utf8_general_ci	?	?
[redacted]	utf8mb4_unicode_ci	?	?
[redacted]	utf8mb4_unicode_ci	?	?
[redacted]	utf8mb4_unicode_ci	?	?
[redacted]	utf8mb4_unicode_ci	?	?
[redacted]	utf8mb4_unicode_ci	?	?
[redacted]	utf8mb4_unicode_ci	?	?
[redacted]	utf8mb4_unicode_ci	?	?
[redacted]	utf8mb4_unicode_ci	?	?
mysql	utf8mb4_unicode_ci	?	?
performance_schema	utf8_general_ci	?	?
[redacted]	utf8mb4_unicode_ci	?	?

Рис. 1 – Схема Производительности

Схема Производительности предназначается, чтобы обеспечить доступ к полезной информации о выполнении сервера, оказывая минимальное влияние на производительность сервера. Реализация следует за этими целями проекта. Контроль сервера происходит непрерывно и незаметно с очень небольшими издержками. Активирование Схемы Производительности не делает сервер неприменимым

В результате в процессе развития программного продукта отследить изменения структуры данных весьма проблематично ввиду того, что DDL, язык определения схем данных, не предусматривает документирование изменения структуры на каждом шаге [2].

Решением данной задачи является создание в каталоге проекта пронумерованных скриптов модификации структуры базы данных при этом, не прибегая к использованию дополнительных инструментов. В таком случае все изменения будут накапливаться в проекте при условии соблюдения заранее определенных правил участниками проекта.

Одним из наиболее современных и практичных способов контроля структуры схем данных является использование дополнительного инструмента Liquibase [3]. Основной задачей системы является контроль номера последнего выполненного скрипта изменения структуры дан-

ных и обеспечение единовременного линейного выполнения скриптов по изменению структуры данных. Однако, использование инструментов, в основе которых лежит принцип накопления change-скриптов (см. рис.2), не позволяет решить задачу так же легко, как написать программный код приложения. Главным минусом данного подхода является необходимость хранения огромного количества неактуальных change-скриптов, что затрудняет получение представления о фактически используемой структуре схемы данных. У liquibase есть еще один существенный недостаток — автоматическое возвращение изменений работает лишь только в том случае, если миграция описана в виде файла XML. Однако, Liquibase позволяет писать миграции и на SQL, но с одной ремаркой — автоматическое возвращение теперь не работает и секцию, которая выполняется при отмене миграции нужно писать вручную.

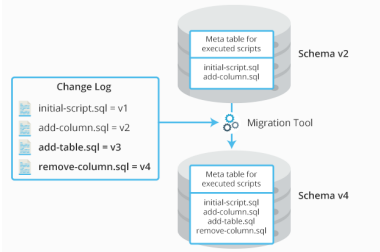

Рис. 2 – Change-скрипты

Время от времени, при обновлении версии базы данных, нужно обновлять не только структуру БД, но и хранящиеся в ней данные. В качестве примера можно привести перенос данных из таблицы со старой структурой в новые таблицы — в целях нормализации. Поскольку данные уже существуют и используются, недостаточно просто создать новые таблицы и удалить старые, нужно еще и перенести имеющиеся данные.

Есть несколько решений этой задачи:

- хранить изменения данных путем накапливания миграционных скриптов;
- нигде не хранить запросы-изменения данных, а в процессе изменения структуры схемы данных подготовить данные для импорта, которые будут соответствовать новой схеме.

Для решения поставленных задач используются такие принципы, как конвергентность и идемпотентность. Соблюдение приведенных принципов в разработанном модуле миграции позволяет наиболее эффективно контролировать состояние схемы данных и не допускать повторного выполнения скрипта миграции данных при его успешном запуске ранее.

Наиболее оптимальным решением задачи миграции схем данных и является следование данным принципам: идемпотентности и конвергентности [4]. Отличительной особенностью такого подхода является то, что скрипт, который удовлетворяет условиям принципов, описывает состояние, к которому объект должен быть приведен, а не действия, которые требуется произвести над ним.

Идемпотентность обеспечивает отсутствие изменений объекта при выполнении скрипта повторно в случае, если скрипт был выполнен успешно при первом его запуске. То есть основная идея этого подхода — написание миграционных файлов таким образом, чтобы их можно было выполнить больше одного раза. При первой попытке выполнить любую из SQL-команд, изменения будут применены; при всех последующих попытках ничего не произойдет.

Соблюдение данного принципа позволяет избежать дублирования и искажения данных. Однако, в случае невыполнения скрипта или неудачного завершения его выполнения, система, при повторном выполнении данного скрипта, будет пытаться прийти к тому состоянию, которое было задано в качестве желаемого. Такое поведение системы обусловлено принципом конвергентности, который направлен на приведение системы в нужное состояние при каждом следующем запуске одного и того же скрипта без выполнения операций, успешно выполненных ранее. Т.е. изменения, не выполненные при одной попытке, система будет пытаться выполнить еще раз при повторном запуске.

1. Performance Schema Quick Start [Электронный ресурс] - Режим доступа: https://dev.mysql.com/doc/refman/5.6/en/performance-schema-quick-start.html
2. Клеппман. М. Высоконагруженные приложения. Программирование, масштабирование, поддержка/М. Клеппман // СПб: Питер, 2018.
3. Версионирование структуры БД с помощью Liquibase [Электронный ресурс] - Режим доступа: http://easy-code.ru/lesson/database-versioning-liquibase
4. Database Migration: What It Is and How to Do It [Электронный ресурс] - Режим доступа: https://rollout.io/blog/database-migration