УДК 004

МЕТОДЫ ВЕРСИОНИРОВАНИЯ БАЗ ДАННЫХ ПРИ РАЗРАБОТКЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Асмоловский В.В., Мартынов Я.А.
Научный руководитель: Мартынов Я.А.

Национальный Исследовательский Томский политехнический университет,
634050, Россия, г. Томск, пр. Ленина, 30
E-mail: vva1@tpu.ru

This article describes different methods of versioning databases, like storing file in version control system or use migration mechanism.

Key words: *database versioning, database migration, database in version control system.*

Ключевые слова: *версионность базы данных, миграции в базах данных, хранение баз данных в системе управления версий.*

Введение

Современная разработка программных продуктов, в том числе информационных систем не производится одним программистом за одним компьютером. В настоящее время повсеместно используется распределенная разработка с использованием систем управления версиями [1]. Этот инструмент позволяет разработчикам использовать одну кодовую базу, находясь географически в разных местах, защищает код от случайного удаления, и позволяет отследить изменения, производимые с исходным кодом.

В большинстве информационных систем используется база данных, но за её целостностью и за производимыми в ней изменениями следят не так часто. При этом теряется часть информации, которую можно узнать из кодовой базы. Так, добавление полей, хранимых в базе данных, приводит к изменению исходного кода, и автора этих изменений легко узнать, что не всегда возможно, если рассматривать взаимодействие с базой данных напрямую. Наличие возможности следить за изменениями в схемах баз данных позволяет точно сопоставить версию базы данных и разрабатываемой информационной системы. Это особенно важно в случае автоматизированного обновления приложения и базы данных.

Варианты использования базы данных

При разработке информационных систем возможно два варианта использования базы данных, и их комбинации:

- Использование централизованной базы данных (подходит для небольших команд).
- Развёртка экземпляра базы данных на компьютере каждого разработчика.

В первом случае может возникнуть ситуация, когда состояние и актуальность базы данных не контролируется никак. К базе данных можно подключиться на сервер, и система контроля версий ни в каком виде не знает о ней. У этого подхода есть очевидные минусы – сложность построения истории изменений, сложность возврата к предыдущему состоянию, возможность потери данных, и т. д.

Второй случай подразумевает возможность передачи схемы базы данных между разработчиками и необходимость хранения её в системе контроля версий. Таких способов можно выделить множество, но в рамках статьи будут рассмотрены следующие:

- Создание миграций [2] (данный термин рассмотрен далее) и их хранение в системе контроля версий.
- Хранение физического файла базы данных в системе контроля версий.

Создание миграций

В контексте данной статьи можно рассмотреть миграции как обновление структуры базы данных от одной версии к другой. Миграции можно разбить на две меньшие категории:

- Создание миграций с помощью *SQL* файлов и с помощью библиотек, предлагаемых на разрабатываемом языке. Такое разделение необходимо, так как второй способ возможен не для всех языков разработки.
- Создание миграций с помощью библиотек. Этот подход заключается в том, что все изменения, производимые с хранимыми в базе данных сущностями, особым образом переводятся в вызовы функций языка разработки. Далее, при развёртывании миграции, библиотека, отвечающая за миграции, переводит сгенерированные команды в *SQL* синтаксис.

При этом чаще всего, в самой базе данных хранится информация о том, какие миграции были применены, а какие нет. У такого подхода большое количество плюсов, таких как:

- Возможность отследить изменения, производимые в структуре базы данных и сопутствующие изменения в системе контроля версий.
- Возможность написания или корректирования, при необходимости, миграций на том же языке разработки, на котором написан исходный код.
- Возможность изменения версии базы данных так же, как и версии информационной системы.

При таком подходе можно абстрагироваться от физической структуры базы данных, и работать с ней на уровне сущностей. (Не все библиотеки позволяют произвести такой переход, и не все базы данных обладают одинаковыми возможностями)

Из примеров языков разработки, где возможно использование такого подхода, можно назвать *Ruby, Python, C#* и т. д.

Написание миграций на *SQL* по сути тоже самое, что написание их с использованием библиотеки, но, накладывает необходимость разработчику отслеживать самому версионность файлов, так же возможна ситуация, когда множество *SQL* файлов создают сложность в понимании конечной схемы базы данных. Такой подход не привязан к языку разработки, но привязан к *SQL* диалекту базы данных.

Хранение физического файла базы данных

Такой подход практически не отличается от использования централизованной базы данных. Чаще всего, несмотря на то, что база данных находится под системой контроля версий, изменения схемы невозможно отследить, при этом она занимает большое количества места, т. к. при каждом изменении создаётся новая копия, а не изменения. В таком случае практически невозможно корректно разрешить конфликты при одновременном изменении базы данных несколькими разработчиками. Положительной стороной использования описанного подхода является возможность отслеживания и изменения версии базы данных.

Заключение

При разработке современных информационных систем использование методов версионирования базы данных так же важно, как и версионирование исходного кода, а механизм миграций обеспечивает оптимальный функционал для этой задачи.

Список литературы

1. Система управления версиями [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Система_управления_версиями.
2. Active Record Migrations [Электронный ресурс]. – Режим доступа: http://edgeguides. rubyonrails.org/active_record_migrations.html.

УДК 004

АВТОМАТИЗАЦИЯ СОЗДАНИЯ ПЕРЕХВАТЧИКОВ СОБЫТИЙ В СИСТЕМЕ УПРАВЛЕНИЯ ПРОЕКТАМИ TEAM FOUNDATION SERVER С ПОМОЩЬЮ REST API ДЛЯ VISUAL STUDIO TEAM SERVICES И TEAM FOUNDATION SERVER

Чебоксаров В.А.
Научный руководитель: Шестаков Н.А., к.т.н., доцент

Национальный Исследовательский Томский политехнический университет,
634050, Россия, г. Томск, пр. Ленина, 30
E-mail: eboxarov22@gmail.com

В данной статье рассмотрен метод автоматизации процесса создания перехватчиков событий для проектов в системе управления проектами Team Foundation Server с помощью REST API для Visual Studio Team Services и Team Foundation Server. В качестве примера рассмотрено создание перехватчика событий для проектов, работающий с системами контроля версий TFVC.

This article describes the method of automation of the service hook setting process for projects in the Project Management System Team Foundation Server using the REST API for Visual Studio Team Services and Team Foundation Server. By way of example, the creation of the service hook for projects operating with TFVC version control systems is considered.

***Ключевые слова**: перехватчики событий, автоматизация, система управления проектами, система контроля версий.*

***Key words**: Team Foundation Server, TFS, service hooks, project management, source code management, Team Foundation Version Control, TFVC, REST API for VS Team Services and TFS.*

Team Foundation Server (сокр. TFS) – продукт корпорации Microsoft, представляющий собой комплексное решение, объединяющее в себе систему управления версиями, сбор данных, построение отчётов, отслеживание статусов и изменений по проекту и предназначенное для совместной работы над проектами по разработке программного обеспечения [1].

Перехватчики событий являются одним из нововведений в Team Foundation Server 2015. Основной целью перехватчиков является отправка оповещений сторонним сервисам о