УДК (по ГОСТ 7.90-2007, например, 004.056)

**Система автоматического тестирования версионных миграций реляционной базы данных PostgreSQL**

**Милько П.А.**

*Ключевые слова: схема данных, SQL скрипты, аномалии модификации, автоматическое тестирование.*

## Введение

Для работы многих веб-приложений необходимо хранилище данных. Часто для хранения этих данных используются реляционные базы данных, например PostgreSQL. В реляционных базах данные хранятся структурировано, а структуры описываются схемами. Большие веб-приложения меняются со временем, и соответственно изменяются схемы в реляционных базах данных. Изменение схем часто реализуют при помощи версионных миграций, которые представляют собой версионированную последовательность из нескольких SQL скриптов. При применении этих скриптов могут возникать ошибки. Чтобы отследить некоторые из них достаточно применить миграции на пустой базе, но так же возможны ошибки связанные не с миграцией схемы, а с миграцией данных. В основном эти ошибки будут возникать из за нарушения CONSTRAINT-а.

CONSTRAINT это ограничения для столбцов и таблиц, которые дают возможность проверять корректность данных на этапе записи в базу. Такими ограничениями являются вторичные ключи таблиц, так как они не позволят создать запись которая ссылается на несуществующие данные в другой таблице. Типы колонок так же являются ограничениями, например нельзя записать строку в колонку, предназначенную для типа INTEGER. Но ограничения типов слишком грубые и дополнительные CONSTRAINT-ы позволяют дополнять их.

## Постановка задачи

Есть несколько подходов для отслеживания ошибок миграций, например статический и динамически анализ скриптов миграций. При статическом анализе нужен только SQL код скриптов для поиска ошибок. Для динамического же помимо самих скриптов понадобится рабочая база данных, на которой эти скрипты будут применяться. То есть при динамическом анализе не нужно знать что находится в скриптах миграций, достаточно только знать их версии. При статическом анализе доступны только скрипты, и всю логику по созданию схемы, которую делает база, придётся создавать отдельно от базы. Это невероятно трудоёмко, однако при статическом подходе есть и преимущество – каждая миграция будет разбираться пошагово, а значит все изменения будет легко отследить. В отличие от динамического анализа, при котором имеется лишь 2 состояния базы – до и после миграции. Вариант с динамическим анализом имеет свои недостатки, но статический анализ потребует намного больше усилий в реализации при том что он не сможет полностью повторить логику работы базы, и как следствие корректно тестировать миграции. Динамический анализ намного более привлекателен тем, что тестируется не логика миграции, а её результат. Достаточно лишь создать все возможные ситуации, при которых миграция может вызвать ошибку.

Как упоминалось ранее основная причина ошибок – нарушение CONSTRAINT-а. Наиболее простой пример – добавление NOT NULL ограничения для столбца. Тогда если в таблице существует запись с NULL значением столбца, то миграция завершится с ошибкой. Решением будет добавление значение DEFAULT к ограничению NOT NULL, тогда все NULL элементы получат указанное дефолтное значение. Это простейший пример, затрагивающий всего лишь один столбец в одной таблице, если же схема изменяется более существенно, добавляя и разделяя таблицы, то ошибки будут намного сложнее, так как для их определения необходима схема, заполненная определёнными данными. Более того, даже успешное применение миграций не гарантирует сохранность данных.

Все описанное выше больше относится к миграциям, которые повышают версию базы (миграции «вверх»), но также бывает необходимо откатить изменения с новой версии обратно на старую. Для этого существуют миграции «вниз», к которым применимы те же требования, что и к миграциям «вверх». То есть они должны без ошибок изменять схему и гарантировать миграцию данных. Это означает то что миграции «вверх» и «вниз» должны быть биективны друг относительно друга. Если это условие выполняется, то возможно проверять не только корректность миграции схемы, но и корректность миграции данных.

Объединяя все сказанное выше можно сформулировать требование к системе автоматического тестирования: система должна обнаруживать возможные нарушения CONSTRAINT-ов в версионных миграциях с исполь-зованием динамического анализа.

Так же нельзя игнорировать то что не всегда реальные ограничения явно описаны в самой схеме, они могут обеспечиваться работой приложения, использующего базу. Тогда изменение типа столбца TEXT → VARCHAR(20) не будет вызывать ошибку на реальной базе. Так что желательно иметь возможность указать внешние (неявные) ограничения.

## Метод решения

Для обнаружения ошибок, связанных с миграцией данных необходимо сначала заполнить базу. Учитывая связи между таблицами нельзя добавлять данные в случайном порядке, так как CONSTRAINT-ы связей не позволят добавлять некорректные данные, которые ссылаются на ещё не добавленные записи в других таблицах. То есть нужен определённый порядок вставки записей в таблицы. Чтобы определить порядок необходимо иметь представление о схеме данных. Все данные о таблицах и их связях PostgreSQL позволяет получить при помощи служебных таблиц. Если отобразить все связи между таблицами, то они будут представлять собой направленный мультиграф. Это значит что возможны множественные связи между двумя таблицами и циклически связанные таблицы. Циклические зависимости особенно усложняют вставку данных, так записи должны добавляться в таблицы в определённом порядке. Стоит отметить что в изначально пустые циклически связанные таблицы невозможно добавить данные, если связи между ними представляют собой 1:N связь. Но если в цикле есть хотя бы одна связь 0-1:N, то можно добавить данные в одну таблицу, а значит и во все остальные дальше по циклу. Работа с циклически связанными таблицами затруднительна и возникновение цикла чаще всего связано с ошибкой проектирования, но существование циклической связи вполне возможно.

Описание всех связей между таблицами действительно создаёт мультиграф, но в предыдущем абзаце сказано что не все связи являются жёсткими, некоторые связи опциональны. Тогда если предположить что в схеме нет жёсткого цикла из таблиц, то структура связей будет больше похоже на дерево. А в дереве всегда можно найти корневую вершину, с которой можно начать заполнение базы данными.

## Эффективность метода

Если гарантировать что для каждой связи между таблицами будет создана хотя бы одна запись, то можно обнаружить ошибки CONSTRAINT-ов при изменении связей таблиц. В худшем случае каждая таблица связана со всеми остальными и тогда для каждой связи в ней будет ровно записей. Тогда для всех таблиц будет записей. То есть получается ассимптотика, где – количество таблиц или , где – количество связей между таблицами.

Текст тезисов статьи (строго 5000 - 9000 тыс. символов) должен включать следующие заголовки: **введение**, **промежуточные заголовки**, **выводы**, **литература**.

Не рекомендуются к публикации тезисы статьи, в которых отсутствует формализованное решение задачи.

Все формулы должны быть пронумерованы справа в круглых скобках, например:

, (1)

Рисунки и таблицы должны быть пронумерованы последовательно в порядке упоминания в тексте, ссылки на них приводятся в круглых скобках, например: (рис.1), (табл.1). Рисунки дополнительно предоставляются в графических форматах tiff или jpg.

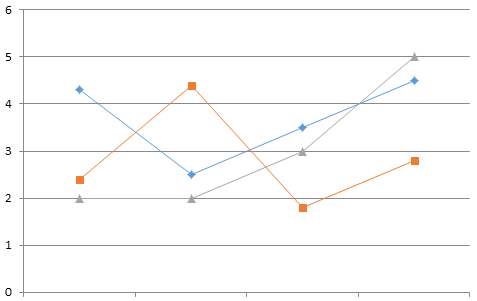


Рис.1. Подпись к рисунку

Таблица 1.

Заголовок таблицы

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Поле 1** | **Поле 2** |
| 1 | Значение 11 | Значение 12 |
| 2 | Значение 21 | Значение 22 |

В случае указания в тексте тезисов фрагмента программного кода следует его приводить в формате моноширинного шрифта, например:

if bit==0 then

{ print(«Веселый привет!»);

return();

}

В **выводе** в обязательном порядке следует отметить:

- что сделано,

- какой достигнут эффект в какой научной области.

Отдельным абзацем следует строго отметить практическое подтверждение/достоверность предлагаемых научных решений.

*Если работа проводилась при финансовой поддержке (к примеру, в рамках гранта РФФИ), в конце выводов (или в дополнительной сноске) это целесообразно указать.*

Список **литературы** нумеруется в порядке, упоминаемом в тексте (примеры ссылок по тексту: [1], [2, 5-7], [3, 123 c.]).

Количество литературных источников: 10-20.

Настоятельно рекомендуется ссылаться на журнальные статьи, представленные в elibrary.ru за последние 5 лет (2014-2018 гг.).

**Запрещается** ссылаться на неиндексируемые литературные источники, как-то: стандарты, законы и анонимные (без авторов) публикации, а также на ненаучные источники (например, википедию, газеты, новости ТиВи, ОБС).

Литературный источник приводится строго по ГОСТ Р 7.0.5-2008 либо как он отображается в elibrary.ru (или scopus).

Пример перечня литературы представлен ниже.

### Литература

1. Lee S.W., Kim H.J. A model of schema versions for object-oriented databases based on the concept of rich base schema. Журнал Information and Software Technology. 1998. Т. 40. № 3. С. 157-173.
2. Tom Laszewski, Prakash Nauduri, Migrating to the Cloud, 2012.
3. Удинцев, П. В. Анализ подходов миграций баз данных в корпоративных информационных системах / П. В. Удинцев. — Текст : непосредственный // Молодой ученый. — 2018. — № 22 (208). — С. 62-64. — URL: <https://moluch.ru/archive/208/50955/> .
4. Patrik Neu, Write Better Migrations with SQL Tests, – URL: <https://www.ory.sh/testing-sql-migrations/> April 27, 2020

1. Марков А.С., Цирлов В.Л., Барабанов А.В. Методы оценки несоответствия средств защиты информации. М.: Радио и связь, 2012. 192 с.

2. Горшков Ю.Г. Тестирование средств засекречивания речи // Вопросы кибербезопасности. 2015. № 2 (10). С. 26-30.

3. Авезова Я.Э., Фадин А.А. Вопросы обеспечения доверенной загрузки в физических и виртуальных средах // Вопросы кибербезопасности. 2016. № 1 (14). C. 24-30.

4. Марков А.С., Матвеев В.А., Фадин А.А., Цирлов В.Л. Эвристический анализ безопасности программного кода // Вестник Московского государственного технического университета им. Н.Э.Баумана. Серия: Приборостроение. 2016. № 1 (106). С. 98-111. DOI: 10.18698/0236-3933-2016-1-98-111.

5. Абрамова О.С., Постернак Е.В., Шахалов И.Ю. Некоторые аспекты управления рисками в программном продукте класса GRC для организаций банковской системы РФ. Сборник трудов Восьмой всероссийской научно-технической конференции «Безопасные информационные технологии» (БИТ-2017) / Под. ред. М.А.Басараба. – М.: МГТУ им. Н.Э.Баумана, НУК «Информатика и системы управления», 2017. С. 3-6.

6. Barabanov A.V., Markov A.S., Tsirlov V.L. Statistics of Software Vulnerability Detection in Certification Testing // Journal of Physics: Conference Series. 2018. V. 1015. P. 042033.

7. Varfolomeev A.A. About Some Perspective Training Cryptography Disciplines. In Selected Papers of the VIII All-Russian Scientific and Technical Conference on Secure Information Technologies (BIT 2017). CEUR Workshop Proceedings, 2017, Vol-2081, pp. 135-138.

…

20 Alexey Zhukov. Lightweight Cryptography: Modern Development Paradigms. In Proceedings of the 8th International Conference on Security of Information and Networks (Sochi, Russian Federation, September 08-10, 2015). SIN '15. ACM New York, NY, USA, 2015, pp. 7-7. DOI: 10.1145/2799979.2799981.

**Научный руководитель или консультант** (если все авторы не имеют ученых степени или звания): Фамилия Имя Отчествополностью, ученое звание, ученая степень, должность и место работы, email. Не допускается совпадение одного из авторов с указанным здесь научным руководителем/консультантом. 