$$\left.\begin{aligned}
\mathbf{M_{д1}} &= \mu_1 \cdot (\omega_2 - \omega_3 \cdot \mathbf{u_1});\\
\mathbf{M_{д2}} &= \mu_2 \cdot (\omega_4 / \mathbf{u_2} - \omega_5);\\
\mathbf{M_{д3}} &= \mu_3 \cdot (\omega_6 / \mathbf{u_3} - \omega_7);\\
\mathbf{M_{д4}} &= \mu_4 \cdot (\omega_7 / \mathbf{u_4} - \omega_8 \cdot \mathbf{u_5});\\
\mathbf{M_{д5}} &= \mu_5 \cdot (\omega_8 - \omega_9).
\end{aligned}\right\} \qquad (3)$$

В качестве граничных условий для системы (1) выступают: минимальная ($\omega_{xx\ min}$) и максимальная ($\omega_{xx\ max}$) угловые скорости холостого хода двигателя (угловая скорость холостого хода двигателя зависит от положения педали акселератора γ и может принимать значение из интервала $\omega_{xx\ min} \le \omega_{xx} \le \omega_{xx\ max}$); момент двигателя, описываемый функцией $\mathbf{M_1 = f(\omega_{двиг.}, \gamma)}$; момент сцепления ведущих колес с дорогой, представленный в виде функции $\mathbf{M_{ф4} = f(\varphi)}$, где φ – коэффициент сцепления ведущих колес с дорогой.

Преимущества предлагаемого алгоритма формирования структуры динамических моделей

Отличительными особенностями алгоритма являются простота и удобство.

Простота алгоритма заключается в том, что все связи ЭДМ должен указывать пользователь. Удобство состоит в том, что при указании пользователем полюсов ЭДМ, соединяемых между собой, осуществляется проверка правил формирования динамической схемы. Таким образом, в ПО *SMM_Model* исключена вероятность неправильного построения схем. Кроме того, предлагаемый алгоритм не требует от пользователя соблюдения правил очередности размещения ЭДМ. Они могут быть размещены на области построения динамических схем в любой последовательности независимо от типа.

Важным преимуществом алгоритма является возможность представления трансформаторных элементов сложной конфигурации (приемно-контрольных приборов и различного рода механических редукторов) в виде, близком к принципиальным или кинематическим схемам. Это достигается за счет специфической конфигурации соединительных линий и позволяет реализовывать графические образы сложной конфигурации.

Эффективность предлагаемого алгоритма заключается в его универсальности, он может использоваться для построения любой математической модели, состоящей из совокупности элементов, взаимодействующих между собой посредством соединительных линий.

В заключение можно сделать следующие выводы.

Разработанный алгоритм автоматизированного перехода от двумерного графического представления структуры динамических моделей к матричной форме позволяет избежать соблюдения правил очередности размещения ЭДМ на поле, представить графический образ динамических схем в наиболее удобном для восприятия пользователем виде, осуществить контроль правильности построения динамических схем.

Предложенный алгоритм формирования матричного представления динамических моделей может быть применен и для автоматизации формирования структуры математических моделей в матричном виде с целью любого схемного представления технического объекта.

Разработанные алгоритм и методика автоматизированного формирования динамических моделей позволяют легко и просто осуществить построение динамической модели на дисплее монитора, приложение источников внешних воздействий (сил и вращающих моментов к сосредоточенным массам), редактирование динамических схем, описание нелинейных характеристик ЭДМ, представление структуры модели в матричной форме.

Литература

1. Тарасик В.П. Математическое моделирование технических систем: учебник для вузов. Минск: ДизайнПРО, 2004. 640 с.

2. Тарасик В.П., Евсеенко И.А. Прикладное программное обеспечение для моделирования объектов макроуровня // Автоматизация и современные технологии. 2007. № 4. С. 11–18.

3. Альгин В.Б. Динамика, надежность и ресурсное проектирование трансмиссий мобильных машин. Минск: Наука и техника, 1995. 256 с.

УДК

ПРОГРАММА ДЛЯ АВТОМАТИЗИРОВАННОЙ ВЕРИФИКАЦИИ ОГРАНИЧЕНИЙ ЦЕЛОСТНОСТИ БАЗ ДАННЫХ

М.Л. Глухарев; А.П. Косаренко; А.Д. Хомоненко, д.т.н.
(Петербургский государственный университет путей сообщения, mlgluharev@yandex.ru, ale.k@mail.ru, khomon@mail.ru)

В работе делается обзор методов и средств тестирования реляционных БД. Описывается программа Constraints Validator, реализующая новый подход к обеспечению автоматизированной верификации ограничений целостности БД

на основе проверки соответствия формальной спецификации БД и существующих в исходных кодах ограничений целостности и триггеров. Приводится пример обработки триггеров программой Constraints Validator для проверки ограничений целостности в БД.

Ключевые слова: *БД, верификация, формальные методы, ограничения целостности, тестирование, контроль ограничений целостности.*

PROGRAM FOR AUTOMATED VERIFICATION OF INTEGRITY CONSTRAINTS IN DATABASES

Glukharev M.L.; Kosarenko A.P.; Khomonenko A.D.*, Ph.D. (Saint-Petersburg State Railway Transport University, mlgluharev@yandex.ru, ale.k@mail.ru, khomon@mail.ru)*

Abstract. In this paper, review of database testing tools is proposed. Program Constraints Validator that realizing the new approach to support of automated verification of constraints of integrity of databases on the basis of check of correspondence of the formal specification of a database and limitations existing in initial codes of integrity and triggers is described. The example of processing of triggers by program Constraints Validator for check of cconstraints of integrity in a database is resulted.

Keywords: database, verification, formal methods, integrity constraints, testing, integrity constraints checking.

Поддержка целостности информации является одной из основных функций современных реляционных СУБД. В каждой БД, помимо таблиц, присутствуют ограничения целостности и триггеры – программные объекты, предназначенные для поддержания целостности хранимой и обрабатываемой информации. Правильность и полнота реализации этих программных объектов обеспечивают высокую защищенность данных и, как следствие, способствуют повышению качества БД и *информационной системы* (ИС) в целом.

Тестирование и проверка ограничений целостности при проектировании и сопровождении БД являются залогом сохранности данных. Механизмам работы ограничений целостности и способам их тестирования посвящено большое число работ. К примеру, в [1] делается обзор современных методов и программных средств верификации реляционных БД. В работе [2] представлена утилита для автоматической генерации тестовых данных. Она случайным образом генерирует тестовые записи согласно критериям, заданным инженером по тестированию, и особенностям тестируемой БД. Критерии составляют схема БД, логические взаимосвязи между столбцами в таблицах, ссылочная целостность БД, количество генерируемых записей и т.д. Главной задачей утилиты является наполнение БД записями для ее (БД) последующего тестирования. Это может быть полезно, например, при проведении нагрузочного тестирования БД.

При тестировании ограничений целостности распределенных БД возникает своя специфика. Описанный в [3] инструмент извлекает некоторые данные из схемы БД и создает набор *update*-операций (так называемых шаблонов), которые могут нарушить ограничения целостности. Далее составляются интеграционные тесты. Таким образом, для одного ограничения целостности могут быть составлены один или несколько шаблонов, которые содержат один или несколько тестов. В процессе выполнения выбираются ограничение целостности, соответствующие ему шаблоны тестирования и интеграционные тесты. Интеграционные тесты ранжируются, для чего предложен оригинальный способ ранжирования, учитывающий расположение серверов распределенной БД. Таким образом, выбирается тест, максимально использующий локальную информацию БД и минимизирующий данные, которые необходимо передать по сети для выполнения теста. Это заметно снижает нагрузку на БД при проведении тестирования, так как, согласно [4], в распределенных БД стоимость доступа к удаленным данным в наибольшей степени влияет на производительность этой БД.

При эксплуатации программной системы достаточно часто необходима ее модификация. Соответственно возникает необходимость убедиться в том, что работающая БД соответствует документации. Для этого можно использовать приложение, описанное в [5]. Оно анализирует *ER*-диаграмму и введенные в него ограничения целостности. На основе этой информации генерируются запросы (*create*, *update*, *delete*), совокупность которых полностью покрывает все имеющиеся в БД ограничения целостности и позволяет проверить БД на соответствие спецификации. Далее приложение анализирует результаты выполнения этих операций и составляет набор запросов (*alter*), позволяющих привести существующую БД к требуемому виду.

В процессе верификации ПО возникает необходимость проверки соответствия БД формальной спецификации. В этом случае целесообразно применение формальных методов верификации по отношению к реляционным БД. Требования целостности реляционных БД, как правило, хорошо формализуются, причем для формализации подходит язык реляционной алгебры, знакомый любому специалисту по реляционным БД. Разработка методики и программной системы для формальной верификации реляционных БД на соответствие требованиям целостности является актуальной задачей в области обеспечения и оценивания качества автоматизированных ИС.

Программная система для формальной верификации реляционных БД

Оригинальный метод формальной верификации реляционных БД в части ограничений целостности и триггеров реализует *Constraints Validator*. Используемый метод предполагает логико-алгебраическое моделирование требований целостности. Каждое требование целостности в спецификации должно быть описано с помощью предиката, называемого ***формальный описатель*** (логическое выражение, используемое для формулирования требования на выбранном языке спецификаций; представляет собой логическое утверждение того, что некоторое множество состояний данных в указанных условиях является допустимым). Спецификация на БД в целом представляет собой набор формальных описателей всех требований целостности. Безошибочность описателей проверяется на ранних стадиях жизненного цикла БД, после чего список описателей фиксируется и далее не изменяется. Таким образом, описатели выражают формальные требования заказчика к БД.

Формальное описание ***требований целостности*** (правила, при помощи которого задается множество допустимых значений некоторого атрибута сущности/связи или принцип логического согласования записей БД между собой) согласуется с логической схемой данных. Если создается реляционная БД, то схема данных описывается в терминах теории отношений, а требования целесообразно описывать на языке реляционной алгебры. В общем случае формальный описатель требования целостности имеет следующий вид: $\mathbf{Expr(tables)} \mid \mathbf{ins(T_1)} \vee \ldots \vee \mathbf{ins(T_n)} \vee \mathbf{upd(T_1)} \vee \ldots \vee \mathbf{upd(T_n)} \vee \mathbf{del(T_1)} \vee \ldots \vee \mathbf{del(T_n)}$, где $\mathbf{tables}=\{T_1, T_2, \ldots, T_n\}$ – множество таблиц, связываемых данным ограничением; **ins()**, **upd()**, **del()** – предикаты выполнения *DML*-операторов вставки, обновления и удаления строк таблиц соответственно. Перечень предикатов в описателе после знака «|» показывает, при выполнении каких операций должно быть истинным логическое условие **Expr(tables)**.

Спецификация оформляется в виде текстового файла и используется в качестве входных данных программы. Каждой реляционной операции в описателях соответствует некоторый символьный псевдокод.

Для общего требования целостности можно не указывать перечень операций в явном виде. Например, требование «масса груза всегда положительна» является общим, на что указывает слово «всегда». Предположим, для хранения сведений о грузах используется таблица **loads**, массы грузов записываются в столбец **mass**. Тогда формальный описатель требования «масса груза неотрицательна» будет выглядеть следующим образом: $\sigma_{\mathbf{mass}\le 0}\mathbf{(loads)}=\varnothing$, а в программной системе его можно представить с использованием псевдокода реляционных операций: **[mass≤0](loads)=0**.

Кроме файла спецификаций, программная система *Constraints Validator* принимает в качестве входной информации *SQL*-сценарии, содержащие программную реализацию требований целостности. Такие требования могут быть реализованы в БД двумя способами: при помощи ограничений целостности (декларативная реализация) и при помощи триггеров (процедурная реализация). Во многих случаях для реализации требования целостности необходима триггерная связка, то есть совокупность триггеров, совместно реализующих одно требование.

Чтобы провести верификацию БД, программа строит модель реализации ограничений целостности и триггеров и сравнивает ее со спецификацией. Очевидно, что модель реализации должна содержать множество описателей, но они отражают не исходные требования заказчика, а *действительно реализованные* в БД ограничения. Построение модели реализации на основе анализа *SQL*-кодов объектов-ограничений является одной из важнейших функций рассматриваемой программной системы.

Получение описателей по исходным кодам объектов-ограничений называется ***восстановлением описателей***. БД можно считать корректно реализованной в части объектов-ограничений, если каждому описателю, восстановленному по объектам-ограничениям, соответствует исходный описатель заранее декларированного требования целостности. Если же имеются несоответствия, то это может говорить о том, что:

- часть требований целостности не была реализована (есть описатели заявленных требований, которым не соответствует ни один восстановленный описатель);
- в БД имеются лишние объекты-ограничения или нужные объекты-ограничения содержат недекларированные возможности (некоторые из восстановленных описателей не соответствуют в точности ни одному из описателей исходных требований).

Обработка типовых ограничений целостности

В реляционных БД существует ряд типовых ограничений целостности: первичный ключ, уникальность значений, определенность значений, ограничение домена (ограничение по логическому условию) и внешний ключ. Типовые требования целостности, как и любые другие, могут быть смоделированы с помощью формальных описателей, которые также можно назвать типовыми. Такие конструкции сравнительно легко распознаются программной системой.

Восстановление описателей по ограничениям

целостности не представляет трудностей. Каждому типовому ограничению соответствует определенный типовой описатель, и его восстановление сравнительно легко реализуется программно.

Например, к первичному ключу предъявляются два требования – уникальность и определенность каждого значения. Следовательно, ни одно значение не будет повторяться дважды и ни одно значение не будет **NULL**. Пусть имеется отношение **R** и в нем атрибут **x** (возможно, составной). Тогда ограничение целостности «Первичный ключ» будет выглядеть следующим образом:

$$\sigma_{COUNT(x)>1 \vee (x \text{ is null})}(\gamma_x(\mathbf{R}))=\varnothing,$$

а в программной системе будет представлено как **[COUNT(x)>1 & (x is null)](x[R])=0.**

Обработка триггеров

Триггеры являются более гибким средством поддержания корректности данных в БД. Каждый триггер содержит процедурную реализацию некоторого алгоритма поддержки целостности, поэтому для триггеров не могут существовать типовые описатели.

Восстановление описателей по триггерам – более сложная процедура. Чтобы восстановить описатель по триггеру или триггерной связке, программной системе необходимо проанализировать *SQL*-коды, после чего, пользуясь *формальной* методикой, построить описатель, являющийся инвариантом для триггера или связки.

Восстановление описателя по коду триггера проходит в несколько этапов. Вначале программная система восстанавливает промежуточные описатели по каждому простому оператору в теле триггера с применением словаря промежуточных описателей. Если программе не удается автоматически подобрать по словарю для некоторого *SQL*-оператора промежуточный описатель, этим занимается эксперт. Затем промежуточные описатели соединяются в один описатель путем применения ряда правил восстановления описателей по условному и циклическому операторам и различным вариантам следования друг за другом простых, условных и циклических конструкций. Для вывода правил восстановления промежуточных описателей авторы использовали аппарат логики Ч. Хоара.

Чтобы восстановить описатель по триггерной связке, необходимо восстановить описатели по каждому триггеру БД, а затем найти описатели с одинаковыми **Expr(tables)**. Для этого, возможно, потребуется выполнить ряд эквивалентных преобразований, чтобы привести несколько описателей к одному виду. Решение о выполнении эквивалентных преобразований принимается экспертом, а сами преобразования выполняются программной системой.

Результатом работы программы являются количество правильно реализованных ограничений (из числа заявленных), списки лишних ограничений (с указанием файла и строки, где они находятся) и недостающих ограничений, а также список конфликтных ситуаций, когда программа не может принять решение самостоятельно и требуется вмешательство эксперта.

Обработка триггеров программой Constraints Validator

Рассмотрим для примера упрощенную БД системы пассажирских перевозок, в которй имеются таблицы **coaches** (вагоны поезда) и **seats** (места). Требование целостности состоит в следующем:

Если вагон является купейным, то номер любого места в нем не больше 36.

Номер места хранится в столбце **seat_number** таблицы **seats**, тип выгона – в столбце **coach_type** таблицы **coaches**.

Требование целостности можно переформулировать следующим образом: ни в какой момент в соединении таблиц **seats** и **coaches** не должно быть строк, в которых одновременно указаны купейный тип вагона и номер места больше 36. Формальный описатель данного требования в аналитической форме выглядит так:

$$\sigma_{coach\_type='Купе' \wedge seat\_number>36}(\mathbf{seats} \triangleright\triangleleft \mathbf{coaches}) = \varnothing,$$

а в псевдокоде, используемом программой *Constraints Validator*, следующим образом: **[coach_type='Купе'&seats_number>36](seats*coaches)=0.**

Данное требование целостности реализовано в БД с помощью связки из двух триггеров: **tr_seat_num** и **tr_coach_seat_num**. Ниже приведены коды триггеров на языке *Transact-SQL*.

```
create trigger tr_seat_num
   on seats
   for insert, update
   as
   if exists (select * from coaches, seats
          where coach_type = 'Купе'
          and seat_number > 36
          and coaches.id_coach=seats.id_coach)
   rollback tran
create trigger tr_coach_seat_num
   on coaches
   for update
   as
   if exists (select * from coaches, seats
          where coach_type = 'Купе'
          and seat_number > 36
          and coaches.id_coach = seats.id_coach)
   rollback tran
```

Требуется провести верификацию этих триггеров, то есть проверить, корректно ли они реализуют заданное требование.

На вход программы *Constraints Validator* подаются файл спецификации с исходным описателем и *SQL*-скрипт с кодами триггеров. Затем эксперт запускает процесс автоматизированного восстановления описателей по триггерам, которое

происходит в следующем порядке. Тело первого триггера начинается с оператора **if** и содержит единственную конструкцию вида **if C rollback tran**. Здесь **C** – это условие **exists(select**…). Вначале восстанавливается промежуточный описатель **Desc(C)**. Здесь речь идет о том, что существует множество строк, получаемых при применении некоторой последовательности реляционных операций к таблицам **coaches**, **seats**, то есть получаем:

$$\mathbf{Desc(C)} \equiv \sigma_{\mathbf{coach\_type='Купе' \wedge seat\_number>36}}(\mathbf{seats} \triangleright\triangleleft \mathbf{coaches}) \neq \emptyset .$$

Запись при помощи используемого в программе псевдокода будет выглядеть следующим образом: **[coach_type='Купе' & seats_number>36] (seats * coaches) !=0.**

Из тела триггера видно: если БД приходит в состояние **Desc(C)**, триггер откатывает транзакцию. В результате **Desc(C)** перестает быть истиной. Очевидно, что постусловием данной конструкции будет:

$$\overline{\mathbf{Desc(C)}} \equiv \overline{\sigma_{\mathbf{coach\_type='Купе' \wedge seat\_number>36}}(\mathbf{seats} \triangleright\triangleleft \mathbf{coaches}) \neq \emptyset} \equiv$$

$$\equiv \sigma_{\mathbf{coach\_type='Купе' \wedge seat\_number>36}}(\mathbf{seats} \triangleright\triangleleft \mathbf{coaches}) = \emptyset,$$

или **[coach_type='Купе' & seats_number>36] (seats * coaches)=0.**

Больше в теле триггера операторов нет. Учитывая, что триггер срабатывает на вставку и обновление в таблице **seats**, имеем описатель по триггеру **tr_seat_num**:

$$\mathbf{Desc(tr\_seat\_num)} \equiv$$

$$\equiv \sigma_{\mathbf{coach\_type='Купе' \wedge seat\_number>36}}(\mathbf{seats} \triangleright\triangleleft \mathbf{coaches}) =$$

$$= \mathbf{\emptyset \mid ins(seats) \vee upd(seats)},$$

или **[coach_type='Купе' & seats_number>36] (seats * coaches) = 0 | ins(seats), upd(seats)**.

Аналогичным образом *Constraints Validator* восстановит описатель по триггеру **tr_coach_seat_name**, который работает так же, как и **tr_seat_num**, но реагирует на обновление строк таблицы **coaches**. Поэтому восстановленный по его коду описатель выглядит так:

$$\mathbf{Desc(tr\_coach\_seat\_num)} \equiv$$

$$\equiv \sigma_{\mathbf{coach\_type='Купе' \wedge seat\_number>36}}(\mathbf{seats} \triangleright\triangleleft \mathbf{coaches}) =$$

$$= \mathbf{\emptyset \mid upd(coaches)},$$

или **[coach_type='Купе' & seats_number>36] (seats * coaches) = 0 | upd(coaches)**.

Следующая стадия – поиск восстановленных описателей с эквивалентной левой частью (частью до вертикальной черты). Следует обратить внимание, что именно **два восстановленных описателя одинаковы в левой части**. Беря их за основу, получим описатель реализованного в БД ограничения:

$$\mathbf{Desc(tr\_seat\_num; tr\_coach\_seat\_num)} \equiv$$

$$\equiv \sigma_{\mathbf{coach\_type='Купе' \wedge seat\_number>36}}(\mathbf{seats} \triangleright\triangleleft \mathbf{coaches}) =$$

$$= \mathbf{\emptyset \mid ins(seats) \vee upd(seats) \vee upd(coaches).}$$

Эквивалентная запись в псевдокоде *Constraints Validator* будет выглядеть следующим образом: **[coach_type='Купе'&seats_number>36](seats* coaches) = 0 | ins(seats), upd(seats), (coaches)**.

Сравним этот описатель с исходным описателем требования в спецификации:

$$\sigma_{\mathbf{coach\_type='Купе' \wedge seat\_number>36}}(\mathbf{seats} \triangleright\triangleleft \mathbf{coaches}) = \emptyset$$

[coach_type='Купе' & seats_number>36](seats * coaches)=0.

Строго говоря, описатели не совсем совпадают: исходный описатель распространяется на все триггерные события в таблицах **seats** и **coaches**, а восстановленный – только на 3 из них. Следовательно, программа не сможет самостоятельно установить полное соответствие между спецификацией и реализацией и сообщит эксперту лишь о найденном частичном соответствии. Установить, не нарушается ли в реализованной БД условие

$$\sigma_{\mathbf{coach\_type='Купе' \wedge seat\_number>36}}(\mathbf{seats} \triangleright\triangleleft \mathbf{coaches}) = \emptyset$$

при выполнении операций **del(seats)**, **ins(coaches)** и **del(coaches)** – это задача эксперта.

Литература

1. Глухарев М.Л. Современные методы и программные средства тестирования и верификации реляционных баз данных. Инновации на железнодорожном транспорте-2009: докл. юбилейной науч.-технич. конф. (Санкт-Петербург, 28–29). СПб: ПГУПС, 2009. С. 68–73.

2. Piriyakitpaiboon K. and Suwannasart T. RealGen: A Test Data Generation Tool to Support Software Testing / In the proc. of the second international conference on information and communication technologies (ICT2004). Thailand, Nov. 28–19, 2004.

3. Alwan A.A., Ibrahim H., Udzir N.I. A Framework for Checking Integrity Constraints in a Distributed Database. 2008. ICCIT '08. Third International Conference on Convergence and Hybrid Information Technology. Vol. 1. Nov. 11–13, 2008.

4. Ibrahim H., Gray W.A., and Fiddian N.J. Optimizing Fragment Constraints – a Performance Evaluation / International Journal of Intelligent Systems – Verification and Validation Issues in Databases, Knowledge-Based Systems, and Ontologies, John Wiley & Sons Inc, 2001. № 16(3), pp. 285–306.

5. Tongrak P., Suwannasart T. A Tool for Generating Test Case from Relational Database Constraints Testing / Computer Science and Information Technology (ICCSIT 2009). 2nd IEEE International Conference on Date: Aug. 8–11, 2009, pp. 435–439.

УДК

НЕЙРОННЫЕ СЕТИ И МОДЕЛИ ARIMA ДЛЯ ПРОГНОЗИРОВАНИЯ КОТИРОВОК