ВЕСТНИК ТОМСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА
2018 Управление, вычислительная техника и информатика № 43

ИНФОРМАТИКА И ПРОГРАММИРОВАНИЕ

УДК 004.652.8
DOI: 10.17223/19988605/43/9

А.М. Бабанов, А.В. Петров

ПРОБЛЕМА НЕПРОТИВОРЕЧИВОСТИ СХЕМ ДАННЫХ И ЕЕ РЕШЕНИЕ В ERM-РЕПОЗИТОРИИ

Статья продолжает обсуждение основных архитектурных и проектных решений, положенных в основу разработанного для ERM-модели репозитория схем данных. В ней пойдет речь о проблеме обеспечения непротиворечивости этих схем. Широкие возможности описания семантики в ERM-модели делают эту проблему весьма актуальной и очень острой. Для ее решения в репозитории предусмотрен блок многочисленных проверок, гарантирующих высокое качество схем.
Ключевые слова: ERM-модель; непротиворечивость; схема данных; репозиторий; Oracle Designer.

В предыдущей статье авторов [1] описаны общие принципы и архитектура прототипа CASE-репозитория, созданного для апробации решений, которые принимаются в ходе исследований по развитию семантической модели данных «Сущность–Связь–Отображение» (ERM-модель) [2]. Важный аспект работы, а именно механизм проверки данных репозитория на непротиворечивость, был лишь кратко упомянут как важная составная часть системы. Однако особенности ERM-модели (богатые выразительные способности, синонимия элементов схемы, возможность наличия в схеме одновременно нескольких представлений одних и тех же явлений предметной области) требуют повышенного внимания к проблеме непротиворечивости схем данных. О решении этой проблемы в ERM-репозитории пойдет речь в настоящей статье.

1. Обострение проблемы непротиворечивости схем в ERM-модели

Задача реализации механизма проверки схем на непротиворечивость является одной из основных в рамках работ по созданию репозитория схем данных [3–7]. Важно, чтобы в любой момент времени данные, хранимые в репозитории, были целостными и непротиворечивыми. Чтобы предупредить создание ошибочных артефактов, в CASE-системе предусматривают соответствующие проверки формируемых в репозитории аналитических и проектных документов.

Р. Велке [3] указывает несколько классов ошибок проектировщиков:

– нарушение правил именования, дублирование имен;

– неправильный тип объекта;

– недопустимый тип объекта в связи;

– ошибочная комбинация объектов в связи;

– связь с опущенным обязательным объектом.

Это типичный набор ошибок для ER-модели и подобных ей моделей. Он является следствием их общего понятийного базиса и метасхемы (рис. 1) [8, 9].

В попытках повышения степени автоматизации проектирования и расширения спектра генерируемых исполняемых артефактов создатели ERM-модели существенно развили описательные возможности, позволяющие при ограниченном наборе понятий определять в схеме все структурные элементы и практически все бизнес-правила предметной области (ПрО). Об этом свидетельствует развитая метасхема модели. На рис. 2 представлена ее каркасная часть.

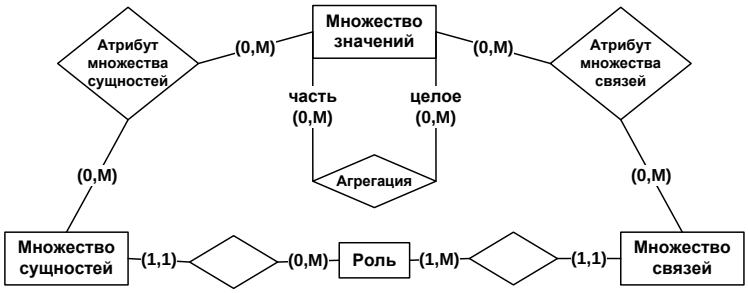

Рис. 1. Метасхема ER-модели

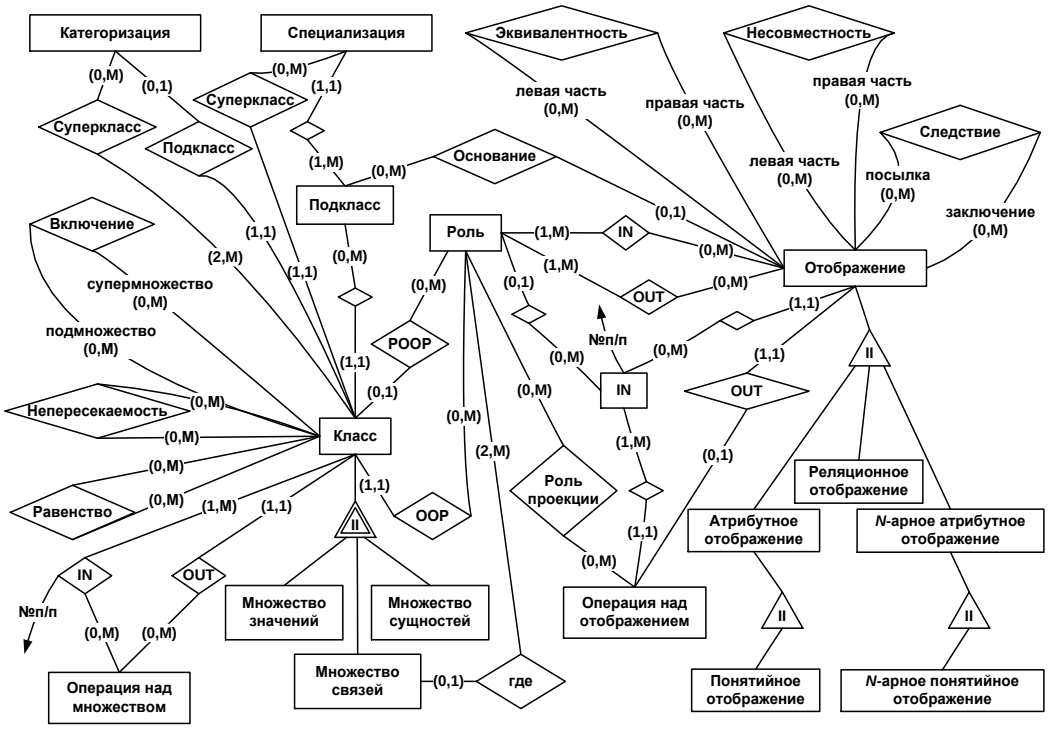

Рис. 2. Основная часть метасхемы ERM-модели

С целью упрощения работы проектировщиков наряду с базовым набором понятий (объект и отображение), обеспечивающим всю семантическую мощь модели, введены производные понятия, позволяющие оперировать ставшими традиционными терминами – «сущность», «связь», «атрибут». Когда для описания законов ПрО достаточно простых понятий, используются они, когда же их описательных способностей недостаточно, проектировщик для более точной передачи семантики прибегает к использованию базовых понятий. Более того, можно обеспечить автоматический переход между этими двумя понятийными базисами, что является важным элементом редактора ERM-схем [10]. В связи с этим открывается возможность наличия в одной и той же схеме одновременно разных форм представления явлений ПрО – на языке базовых и производных терминов.

Наконец, желание обеспечить хранение в репозитории всех схем ПрО (внешних, концептуальной, внутренней), а также взаимосвязей между элементами этих схем, представляющими одни и те же, а также родственные понятия [10], привело к дополнительному источнику противоречий. Но, несмотря на это обстоятельство, наличие в репозитории всех указанных артефактов существенно облегчает работу аналитиков и проектировщиков, способствует повышению автоматизации их труда и обеспечивает более высокое качество результата.

Таким образом, к ставшим уже традиционными проверкам пришлось добавить гораздо большее количество новых проверок схем данных на непротиворечивость. Для корректной реализации таких проверок требовалось предварительно их проанализировать и классифицировать.

2. Система классификации проверок ERM-схемы

Р. Велке [3] выделяет следующие стратегии проектировщиков схем данных по борьбе с нецелостностью спецификаций (мы предпочитаем использовать для схем термин «противоречивость», оставляя по традиции термин «нецелостность» для данных):

– делать проверки и исправления в момент ввода или изменения описаний;

– делать их по окончании всего процесса задания спецификаций;

– оставлять проверки для другой группы людей – верификаторов.

В рамках CASE-репозитория может быть реализована любая из этих стратегий. Однако первая из них является предпочтительной, так как позволяет определить и исправить ошибку сразу после ее возникновения и вообще исключает нецелостность репозитория, даже на короткий период.

Также Велке предлагает три основных уровня проверок целостности [3]:

– проверки на уровне метасхемы;

– проверки на уровне представлений (диаграмм);

– проверки на уровне методики.

«Проверки уровня метасхемы (или репозитория) касаются правил законного существования экземпляров типов репозитория, зафиксированных в метасхеме. Для объектов они включают проверку уникальности имен и правильности значений свойств; для связей – проверку легальности комбинации ассоциированных объектов, требуемой кардинальности и правильности значений свойств. Проверки уровня репозитория должны осуществляться незамедлительно, чтобы гарантировать, что правила, указанные в метасхеме, не нарушены.

Проверки уровня представлений могут выполняться в любой момент проектирования в конкретном окне (или диаграмме), например в DFD (Data Flow Diagram – диаграмма потоков данных). Можно ввести информацию, которая корректна на уровне метасхемы, но нарушает директиву конструирования представлений. Например, связь «вызов» между двумя процессами может быть абсолютно корректна на уровне типов репозитория, но неправильна в контексте DFD. Тонкость заключается в том, что один и тот же процесс может быть только на одной DFD. Более очевидный пример – процесс с входом, но без выхода.

Реализация целостности уровня представлений может иметь две формы. Сначала фиксируем (commit) информацию спецификации, затем в постпроцессе проверяем, что эта информация удовлетворяет контексту, в котором она введена, и устраняем недостатки последующей модификацией репозитория. Второй вариант – не фиксируем в репозитории информацию представления до тех пор, пока она не пройдет все поверки уровня представления. Несмотря на то, что последний подход соответствует принципу «чистого» репозитория, он приводит к проблемам для аналитика-конструктора.

Проверки уровня методики принципиально осуществляются над всеми спецификациями репозитория. Они обнаруживают и устраняют аномалии между представлениями (такие как балансировка DFD с ее родителем и ее братьями, балансировка диаграммы потоков управления с ассоциированной с ней диаграммой переходов состояний), проверяют устойчивость проекта по всем стадиям разработки, трассируют требования по отношению к элементам репозитория и т.д. Для проведения таких проверок в репозитории должна иметься полная аналитическая и проектная информация по системе. Следовательно, это наиболее важная задача постпроцессинга» [3. С. 22].

Сложная, обладающая богатыми выразительными средствами модель данных требует серьезного анализа источников противоречивости схем и систематизации проверок, необходимых для исключения некорректности. При построении системы классификации проверок схемы на непротиворечивость необходимо использовать строгие логические принципы, главным из которых является принцип единства основания деления. Он гласит, что, выполняя деление родительского понятия (класса проверок), следует выделять дочерние понятия (подклассы проверок) на основании одного

и того же признака. Деление методом дихотомии порождает два понятия: одно объединяет объекты, у которых признак есть, другое – объекты, у которых его нет. Такой принцип деления осуществляется, как правило, на основании свойств объектов. Второй вариант деления понятия – деление по видоизменению признака – некоторой характеристики, принимающей разные значения для объектов различных понятий.

В соответствии с этим принципом можно выделить следующие главные основания (признаки) для классификации проверок:

– свойство проверки, говорящее о том, что она формально представлена в метасхеме модели (Metascheme);

– характеристика времени проверки (проверки непосредственно при изменении схемы (Immediate), выполняемые как часть операции изменения схемы, и отложенные проверки (Deferred), как правило, комплексные, транзакционные, запускающиеся специальным скриптом);

– характеристика места проверки (проверки на уровне элементов управления диалога и в коде клиентского приложения (Client) не являются задачей репозитория; проверки на уровне структур и декларативных ограничений целостности репозитория (Repository или Scheme) обеспечивает базовая СУБД; проверки на уровне триггеров и процедур API (Server или Code) требуют программирования на языке сервера СУБД);

– характеристика охвата элементов схемы (локальные проверки только изменяемого элемента (Local); глобальные проверки, затрагивающие неизменяемые элементы (Global)).

Каждая проверка в зависимости от своих свойств и характеристик по каждому из критериев попадает в тот или иной класс, и в силу этого для ее осуществления применяются соответствующие средства.

3. Классификации проверок ERM-схемы

В результате анализа возможных противоречий ERM-схемы был получен документ, содержащий 179 различных проверок. К полученному списку проверок применена классификация по следующим признакам (наряду с главными признаками, упомянутыми в предыдущем разделе, использовались дополнительные признаки, которые принимались во внимание в ходе реализации проверок):

– тип объекта согласно метамодели репозитория (элементный или ассоциативный);

– структурный элемент ERM-модели (класс, подкласс специализации, эквивалентность отображений и т.д.);

– действие в репозитории (добавление, изменение, удаление структурного элемента модели);

– время инициирования проверки (немедленно, по окончании транзакции);

– охват элементов схемы (локальные проверки только изменяемого элемента, глобальные проверки, затрагивающие неизменяемые элементы);

– наличие ограничения в текущей ERM-метасхеме;

– форма реализации в рамках репозитория (обеспечивается структурой репозитория или специально написанным программным кодом);

– возможность задания декларативными инструментами реляционной модели.

Наличие последнего признака позволяет осуществлять проверки средствами реляционной СУБД, так как репозиторий – не что иное, как реляционная БД. А значит, данные БД (элементы ERM-схем) должны удовлетворять всем ограничениям, декларированным в схеме этой БД (ERM-метасхеме).

По результатам классификации проверок был получен список проверок, фрагмент которого приведен в таблице 1.

Таблица 1

Фрагмент списка проверок ERM-схем на непротиворечивость

Элемент ERM-модели	Проверка	Immediate / Deferred	Local / Global	Meta scheme	Scheme / Code	Можно задать декларативными инструментами реляционной модели
Множество сущностей	Проверка соединения с множествами связей через роль	Immediate	Local	Да	Scheme	Да
Множество сущностей	Проверка соединения с множествами значений через атрибут	Immediate	Local	Да	Scheme	Да
Множество связей	Проверка соединения с множествами сущностей через роль	Immediate	Local	Да	Scheme	Да
Множество связей	Проверка соединения с множествами значений через атрибут	Immediate	Local	Да	Scheme	Да
Операция над классами	Проверка правильности атрибута «Тип»	Immediate	Local	Нет	Code	Да
Операция над классами	Проверка правильности атрибута «Кандидат»	Immediate	Local	Нет	Code	Да
Операция над классами	Проверка наличия одного класса-результата	Deferred	Local	Да	Code	Да
Операция над классами	Проверка наличия минимум одного аргумента	Deferred	Local	Да	Code	Да
Операция над классами	Проверка отсутствия некандидатов в аргументах операции	Deferred	Local	Нет	Code	Нет
Операция над классами	Проверка отсутствия некандидатов в результатах операции	Deferred	Local	Нет	Code	Нет
Операция над классами	Проверка наличия нумерации аргументов только у разности	Deferred	Local	Нет	Code	Нет
Операция над классами	Проверка правильности нумерации аргументов у разности	Deferred	Local	Нет	Code	Нет
Операция над классами	Проверка одинаковости всех типов классов	Deferred	Local	Нет	Code	Нет

В соответствии с этим списком и были реализованы проверки схемы на непротиворечивость в ходе разработки репозитория. Часть из них удалось выразить декларативными средствами реляционной модели (модели репозитория Oracle Designer), другая часть потребовала использования программных средств этой модели (триггеров), третий класс проверок осуществляется в процедурах API репозитория.

Следует отметить, что в соответствии с принципами разработки, управляемой моделями (Model Driven Development – MDD) [7, 11], следовало представить все условия непротиворечивости схем в ERM-метасхеме и осуществлять проведение всех проверок на основании только этой декларативной информации. В случае такой реализации проверок добавление новой проверки осуществлялось бы как расширение метасхемы без специального программирования.

Но задача синтеза полной (и в плане структурных компонентов, и в плане условий непротиворечивости) метасхемы ERM-модели находится в стадии решения, поэтому пока для проведения проверок используются самые различные инструменты.

4. Реализация проверок схемы в ERM-репозитории на базе Oracle Designer

Для того, чтобы при выполнении комплексных модификаций иметь возможность их отменить, в системах хранения, как правило, применяются транзакции. Генерируемое Oracle Designer (OD) API предоставляет возможность управления начальной и конечной точками транзакции, что и было использовано при создании механизма проверки.

Накапливаемые в ходе транзакции изменения схемы проверяются по мере возможности при наличии в репозитории необходимых для этого элементов. В случае отрицательного результата проверки все изменения данных, осуществленные в ходе транзакции, должны быть отменены, и репозиторий должен принять то состояние, которое он имел на момент открытия транзакции.

Для того, чтобы репозиторий своевременно выдавал ошибки при вводе неправильных элементов схемы, необходимо сделать размер транзакции как можно меньше. С другой стороны, для поддержания непротиворечивости хранимых в репозитории данных требуется, чтобы работа с одним экземпляром структурного понятия ERM-модели не разбивалась на несколько транзакций. В частности, это важно для сложных типов, создание экземпляра которых порождает создание более одного элементного или ассоциативного объектов. В соответствии с этим была принята модель транзакции, согласно которой каждый экземпляр структурного понятия ERM-модели добавляется, изменяется или удаляется в рамках одной транзакции.

Одним из важных вопросов является расположение механизма проверки в системе. Проверка должна производиться при работе во всех инструментах, осуществляющих изменения хранимых в репозитории данных. Как видно на рис. 3, работа внешних приложений (показано одно из них – графический редактор ERM-схем) осуществляется путем обращения к функциям пакета доступа ERM_API, работа же инструмента Repository Object Navigator (RON), разработанного до появления ERM_API и недоступного для изменения, осуществляется путем обращения непосредственно к OD API. Следовательно, расположение механизма проверки схем на непротиворечивость должно быть не выше уровня OD API.

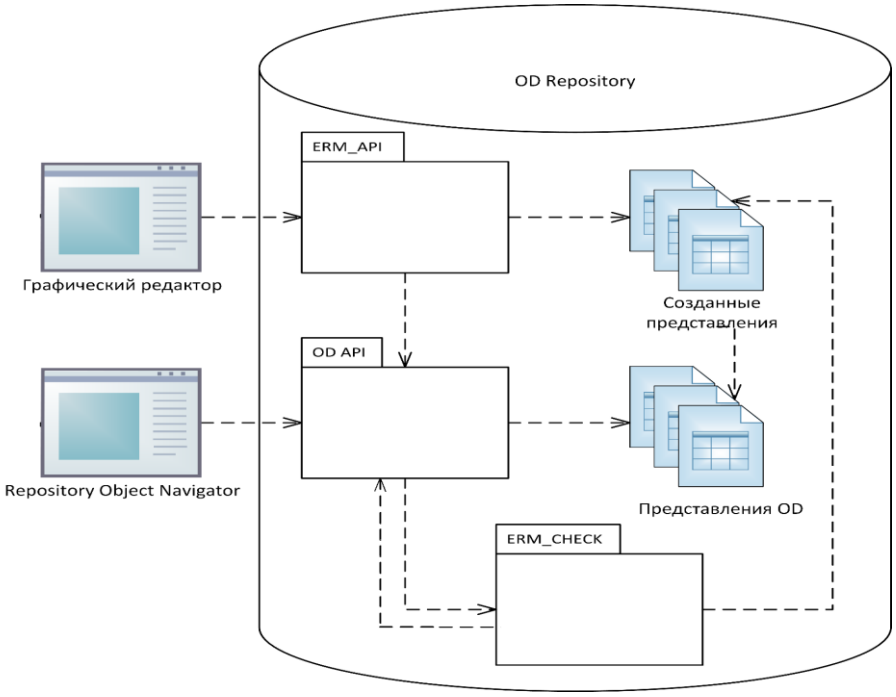

Рис. 3. Архитектура ERM-репозитория

Вызов процедур пакета проверки схем на непротиворечивость (ERM_CHECK) осуществляется в функциях API инструмента Oracle Designer. Для каждого создаваемого пользовательского типа автоматически генерируется пакет для работы с ним. Каждая из функций этих пакетов осуществляет вызов аналогичной функции (добавления, изменения, удаления или выборки) базового пакета API. Для элементных типов это пакет CIO_EXTENDED_ELEMENT, а для ассоциативных – CIO_EXTENDED_STRUCTURE_ELEMENT. Поскольку вызов функций этих пакетов осуществляется при любой операции с репозиторием посредством API, было решено расположить вызов функций проверки ERM-схем в функциях этих пакетов.

Для каждого элементного и ассоциативного типа ERM-объектов имеется собственная процедура, реализующая его проверку. В каждой функции можно выделить 3 блока.

Первым из них является блок обязательных проверок. Как уже говорилось ранее, работа по созданию, изменению или удалению сложного объекта ERM-схемы может включать в себя совокупность нескольких действий с точки зрения репозитория. Так, например, при создании отображения следует также создать связи с ролями, составляющими его область определения (ООО) и область значений (ОЗО). Для обеспечения целостности изменяемой схемы такая совокупность действий должна быть реализована в рамках одной транзакции. Однако модель инструмента RON использует только атомарные транзакции, т.е. в данном случае каждое действие с элементом репозитория будет проведено в рамках отдельной транзакции.

Решено было смоделировать свои, более крупные транзакции. Для этого каждому элементному типу было добавлено еще одно свойство – поле Candidate, обозначающее факт окончательного завершения работы над объектом. В рамках первого блока производится проверка локальных ограничений, которые относятся к одному элементу репозитория и должны соблюдаться вне зависимости от значения поля Candidate. Например, тип отображения должен иметь корректное значение вне зависимости от того, завершена ли работа с данным объектом или нет.

Далее следует блок проверок для объектов-кандидатов. В рамках данного блока реализованы проверки, характерные для сложных объектов ERM-схемы, работа над которыми уже завершена (Candidate = Yes). Так, например, в рамках данного блока производится проверка наличия в ООО и ОЗО отображения хотя бы по одной роли.

Завершающим является блок проверок внешних объектов. Зачастую при совершении какой-либо операции с объектом приходится осуществлять проверку других связанных с ним объектов (глобальную проверку). Так, например, при удалении подкласса специализации следует проверить саму специализацию на предмет наличия у нее хотя бы одного подкласса. Данный блок реализует проверку тех объектов, которые потенциально могут стать ошибочными после проведения операции.

Благодаря этим многочисленным разноплановым проверкам ERM-репозиторий гарантирует непротиворечивость схем данных. Причем изменения репозитория могут осуществляться как во вновь разрабатываемых диалоговых инструментах, использующих специализированный ERM_API, так и в ранее разработанных средствах, пользующихся OD API.

Заключение

В статье описаны основные принципы, положенные в основу реализации механизма проверки ERM-схем на непротиворечивость. Прототип ERM-репозитория создан на базе CASE-системы Oracle Designer в виде ее расширения. Открытость расширения обеспечила возможность внедрения инструментов верификации в ядро OD. Это позволило осуществлять проверки элементов репозитория в предопределенных диалоговых средствах системы, позволяющих создавать ERM-схемы без программирования специализированных клиентских мест.

Перспективы развития механизма проверки ERM-схем на непротиворечивость заключаются в последовательном использовании принципов разработки, управляемой моделями [7, 11]. В соответствии с ними необходимо декларативно представить в ERM-метасхеме все условия непротиворечивости схем и реализовать программно универсальный верификатор, осуществляющий необходимые проверки на основании только этой информации. Семантическая полнота ERM-модели в состоянии обеспечить такие описания.

ЛИТЕРАТУРА

1. Бабанов А.М., Петров А.В. Реализация репозитория ERM-модели в CASE-системе Oracle Designer // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. 2017. № 4 (41). С. 47–54.

2. Бабанов А.М. Семантическая модель «Сущность–Связь–Отображение» // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. 2007. № 1. С. 77–91.
3. Welke R. The case repository: more than another database application // Proceedings of 1988 INTEC Symposium Systems Analysis and Design: a Research Strategy, Atlanta, Georgia, 1988 / W.W. Cotterman, J.A. Senn (eds.); Georgia State University. Meta Systems, Ltd., 1988.
4. Welke R. The case repository: more than another database application // Challenges and strategies for research in systems development. New York : John Wiley & Sons, Inc., 1992. P. 181–218.
5. Dahanayake A. Computer-Aided Method Engineering: Designing CASE Repositories for the 21st Century. Idea Group Publishing, 2001. 550 p.
6. Agarwal B.B., Tayal S.P., Gupta M. Software Engineering and Testing. Jones & Bartlett Publishers, 2010. 516 p.
7. Hamid B. A model-driven approach for developing a model repository: Methodology and tool support // Future Generation Computer Systems. 2017. V. 68, March. P. 473–490.
8. Chen P. The entity-relationship model – towards a unified view of data // ACM Transactions on Database Systems. 1976. No. 1 (1). P. 9–36.
9. Barker R. CASE Method: Entity Relationship Modelling. Addison-Wesley. 1990. 240 p.
10. Бабанов А.М. Перспективы проектирования БД, открывающиеся с применением современных семантических моделей данных // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. 2015. № 2 (31). С. 73–80.
11. Brambilla M., Cabot J., Wimmer M. Model-Driven Software Engineering in Practice, 2nd ed. Morgan & Claypool Publishers, 2017. 280 p.

Поступила в редакцию 19 декабря 2017 г.

Babanov A.M., Petrov A.V. (2018) PROBLEM OF DATA SCHEME CONSISTENCY AND ITS SOLUTION IN THE ERM REPOSITORY**.** *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie vychislitelnaja tehnika i informatika* [Tomsk State University Journal of Control and Computer Science]. 43. pp. 72–80

DOI: 10.17223/19988605/43/9

The task of implementing a mechanism for verifying consistency of schemes is one of the main tasks in the development of a data scheme repository. It is important that at any time the data stored in the repository are complete and consistent. To prevent erroneous artifacts created in CASE system, it provides for appropriate checks of the analytical and design documents formed in the repository.

In attempts to increase the degree of automation of design and the widening of the spectrum of generated, executable artifacts, the creators of the ERM model significantly expanded the descriptive possibilities allowing, with a limited set of concepts, to determine in the scheme all the structural elements and practically all the business rules of the application domain (AD).

In order to simplify the work of designers, along with a basic set of concepts (object and mapping) that provide the entire semantic power of the model, derivative concepts are introduced that allow us to operate with the traditional terms - entity, relationship, attribute. When simple concepts are enough to describe the semantics of AD, they are used, when their descriptive abilities are not enough, the designer uses the basic concepts for a more accurate transfer of semantics. In this case, it becomes possible to have simultaneously in the same scheme different forms of representing the phenomena of AD, in the language of basic and derived terms. The desire to provide storage in the repository of all schemes of AD (external, conceptual, internal), as well as interrelations between the elements of these schemes representing the same and related concepts, has led to an additional source of contradictions.

In accordance with the principle of unity of the basis of notion division, the following main characteristics can be distinguished for classification of inspections:

– property of verification to be formally represented in the model's metascheme;
– characteristic of the time of verification;
– characteristic of the place of verification;
– characteristic of the coverage of scheme elements.

Each check, depending on its properties and characteristics, falls into one or another class for each of the criteria, and accordingly, appropriate means are applied for its implementation. As a result of the analysis of possible contradictions in the ERM scheme, a document containing 179 various checks was received.

In order to be able to cancel complex modifications, in the storage systems, as a rule, transactions are applied. Generated by Oracle Designer (OD) API provides the ability to manage the start and end points of a transaction, which was used when creating the verification mechanism.

The scheme changes that are accumulated during the transaction are checked as soon as possible in the presence of elements necessary for this. In the case of a negative result of the check, all data changes made during the transaction must be rollbacked, and the repository must accept the state it had at the time the transaction was opened. So, transaction model was adopted according to which each instance of the structural concept of the ERM model is added, modified or deleted within a single transaction.

One of the important issues is the location of the verification mechanism in the system. The check should be performed when working in all tools that change the data stored in the repository.

For each element and association type of ERM-objects there is its own procedure that realizes its verification. There are 3 blocks in each function: a block of mandatory checks, a block of checks for candidate objects and a block of checks of external objects.

Thanks to these numerous versatile checks, the ERM repository ensures data scheme consistency.

Prospects for the development of a mechanism of verifying ERM schemes for consistency are in the consistent use of principles of Model Driven Development. In accordance with them, it is necessary to declaratively present in the ERM metascheme all the conditions of consistency of schemes and implement universal program verifier that performs the necessary checks based only on this information. The semantic completeness of the ERM model is able to provide such descriptions.

Keywords: ERM-model; consistency; data scheme; repository; Oracle Designer.

BABANOV Alexey Mihailovich (Candidate of Technical Sciences, Associate Professor, National Research Tomsk State University, Russian Federation**).**
E-mail: babanov2000@mail.ru
PETROV Alexander Viktorovich (National Research Tomsk State University, Russian Federation).
E-mail: flamingcode@mail.ru

REFERENCES

1. Babanov, A.M. & Petrov, A.V. (2017) Implementation of the ERM-model repository in CASE-system Oracle Designer. *Vestnik Tomskogo gosudarstvennogo universiteta. Upravleniye, vychislitel'naya tekhnika i informatika – Tomsk State University Journal of Control and Computer Science*. 4(41). pp. 47–54. (In Russian). DOI: 10.17223/19988605/41/6
2. Babanov, A.M. (2007) Semantic model "Entity – Relationship – Mapping". *Vestnik Tomskogo gosudarstvennogo universiteta. Upravleniye, vychislitel'naya tekhnika i informatika – Tomsk State University Journal of Control and Computer Science*. 1. pp. 77–91. (In Russian).
3. Welke, R. (1988) *The case repository: more than another database application*. Meta Systems, Ltd.
4. Welke, R. (1992) *Challenges and strategies for research in systems development*. New York: John Wiley & Sons, Inc. pp. 181–218.
5. Dahanayake, A. (2001) *Computer-Aided Method Engineering: Designing CASE Repositories for the 21st Century*. Idea Group Publishing.
6. Agarwal, B.B., Tayal, S.P. & Gupta, M. (2010) *Software Engineering and Testing*. Jones & Bartlett Publishers.
7. Hamid, B. (2017) A model-driven approach for developing a model repository: Methodology and tool support. *Future Generation Computer Systems*. 68. pp.473–490. DOI: 10.1016/j.future.2016.04.018〉
8. Chen, P. (1976) The entity-relationship model – towards a unified view of data. *ACM Transactions on Database Systems*. 1(1). pp. 9–36. DOI: 10.1145/362384.362685
9. Barker, R. (1990). *CASE Method: Entity Relationship Modelling*. Addison-Wesley.
10. Babanov, A.M. (2015) Database design prospects opening with application of modern semantic data models. *Vestnik Tomskogo gosudarstvennogo universiteta. Upravleniye, vychislitel'naya tekhnika i informatika – Tomsk State University Journal of Control and Computer Science.* 2(31). pp. 73–80. (In Russian).
11. Brambilla, M., Cabot, J. & Wimmer, M. (2017) *Model-Driven Software Engineering in Practice*. 2nd ed. Morgan & Claypool Publishers.