УДК 004.652.5

ОНТОЛОГИЧЕСКАЯ МОДЕЛЬ БЛОКА ТРАНЗАКЦИОННЫХ ДАННЫХ РЕЛЯЦИОННОЙ СУБД

И.С. Решетников [1], ***П.Н. Коснырев*** [2]

[1] ООО «Газпром центрремонт», Москва
[2] Шуйский госуниверситет

i.reshetnikov@gcr.gazprom.ru

Поступила в редакцию 14.07.2010

Рассматривается способ организации физической структуры данных в реляционной СУБД, основанный на подмножестве метамодели языка UML для диаграмм классов.

Ключевые слова: метамодель, пакет, класс, атрибут, объект, отношение, обобщение, ассоциация, агрегирование, композиция.

Введение

При построении корпоративных информационно-вычислительных сетей, имеющих дело с описанием сложных по структуре производственных комплексов, одной из основных задач является построение системы надёжного и гибкого хранения данных об оборудовании. В большинстве случаев такие системы строятся на базе современных реляционных СУБД, и блок транзакционных данных представляет собой сильно нормализованную структуру, исключающую избыточность и обеспечивающую целостность данных. Однако нормализация структуры не всегда обеспечивает нужный уровень связности данных, т.к. не всегда с её помощью возможно описать сложные системы с внутренней иерархией с обеспечением необходимого уровня согласованности. Вторым недостатком нормализации является сложность внесения корректировок в структуру данных, необходимость которых возникает при изменении модели описания единиц оборудования.

В настоящей статье рассматривается альтернативный подход к организации транзакционных данных в реляционных СУБД на примере СУБД Oracle, имеющий ряд преимуществ при реализации распределённых систем и обеспечивающий возможность расширять логическую структуру данных в процессе своего функционирования на уровне конечного пользователя информационной системы.

Поставленная задача, на первый взгляд, близка к задаче систем класса PLM/PDM [1], но имеет и существенное отличие, т.к. необходимо не описание статической электронной модели объекта, а динамическое описание комплекса сложных объектов с динамической внутренней структурой при наличии редких структурных перестроек с условиями хранения истории изменений и обеспечения целостности. Такие дополнительные условия делают неприменимым использование стандартных технологий вроде CALS или построения исполнительной документации «Как построено». Применимость данного вида технологий ограничивается разовым описанием системы при дальнейшем внесении несущественных изменений, не затрагивающих структуру объекта, при этом за целостность отвечает, как правило, оператор, осуществляющий ввод данных.

Концепция описания логической структуры данных

Задача репликации подразумевает под собой единую для всех серверов физическую структуру базы данных. Отсюда следует, что для соответствия требованиям, предъявляемым к расширению структуры предметной области, значения характеристик для всех описываемых объектов должны содержаться в некой единой таблице, а принадлежность значения к той или иной характеристике того или иного объекта должна описываться с помощью набора метаданных [2]. Таким образом, для формирования логической структуры предметной области в базу данных необходимо внести метаданные, описывающие объекты и их характеристики.

Подход к разработке структуры метаданных, описываемый в работе, базируется на метамодели языка UML [3], а, точнее, той её части, которая относится к диаграмме классов. По аналогии с терминологией языка UML для описания предметной области были введены понятия *объект* (конкретная сущность реального

мира), *класс* (совокупность объектов с общими атрибутами, отношениями и семантикой, экземпляр класса – объект), *атрибут* (именованное свойство класса).

Для описания связей вводятся понятия *отношение* (связь между сущностями), *обобщение* (отношение между суперклассом, или родителем, и его конкретным воплощением – субклассом, потомком), *ассоциация* (связь между объектами разных классов), у каждой ассоциации два *полюса*. Для описания отношения типа «часть-целое», в котором один из классов имеет более высокий ранг (целое) и состоит из нескольких меньших по рангу (частей), используется понятие *агрегирование* как частный случай ассоциации. Отношение агрегирования является транзитивным и асимметричным. Вариация агрегирования – *композиция*, форма агрегирования с четко выраженным отношением владения, причем время жизни частей и целого совпадает.

В том случае, когда для некоторых связей помимо ссылок на объекты необходимо определить свои собственные значения атрибутов, используют *класс-ассоциацию*.

Вообще, может существовать любое количество экземпляров классы (*кратность*). В случае наличия ограничений на кратность выделяется абстрактный класс (0), синглетный (1), *n*-кратный (не более *n*). Кратность встречается также у ассоциаций.

При описании систем с потенциально большим количеством классов возникает необходимость организовывать эти сущности в более крупные блоки. Для организации моделирующих сущностей в группы мы, по аналогии с семантикой языка UML, будем использовать *пакеты*.

Пакет может владеть другими элементами: классами и пакетами. Владение — это композитное отношение, означающее, что элемент объявлен внутри пакета. Если пакет удаляется, то уничтожается и принадлежащий ему элемент. Элемент может принадлежать только одному пакету. Взятое само по себе *имя* пакета называется простым. К составному (или полному) имени спереди добавлено полное имя объемлющего его пакета через разделитель «:». Класс записывается как элемент пакета в форме «полное_имя_пакета::имя_класса».

Пакет задаёт границы *видимости* для содержащихся в нём классов, т.е. определяет, какие классы могут иметь отношения между собой. По умолчанию для определённого класса видимыми являются все классы, находящиеся в пакетах, объемлющих заданный класс. Для того чтобы класс, находящийся в одном пакете, имел доступ к классу, находящемуся в другом пакете, при условии, что оба пакета являются равноправными (отсутствует вложенность), устанавливается связь типа *импорт*. С целью жесткого ограничения видимости класса рамками одного пакета класс может быть объявлен как закрытый.

Реализация предложенной концепции формирования метаданных позволяет создавать сложные логические структуры для описания модели предметной области, состоящей из большого количества различных объектов, с формализацией ограничений и обеспечением целостности данных.

Физическая структура данных

Для практической реализации предложенной объектной модели была разработана физическая модель данных, представленная на рис. 1. Для упрощения некоторые вспомогательные таблицы на схеме опущены. В качестве платформы использована СУБД Oracle 10g.

Все таблицы в базе данных можно разделить на две категории: таблицы для хранения метаданных (таблицы для описания пакетов, классов, атрибутов и отношений) и таблицы для хранения собственно данных (объекты и их характеристики). В качестве значений для первичных ключей применяется UUID, его с приемлемым уровнем уверенности можно считать уникальным.

В таблице PACKAGE описываются пакеты. Изначально в указанной таблице содержится единственная запись для корневого пакета, которая создаётся при первичной инициализации структуры данных. Все создаваемые пакеты, напрямую или опосредованно, должны являться вложенными в корневой пакет. Вложенность пакетов задается рекурсивной связью через внешний ключ ENCLOSING_PACKAGE_ID. Импортируемые пакеты описываются в таблице IMPORTED_PACKAGE, которая имеет два поля, являющихся внешними ключами к таблице PACKAGE – для пакета, в который производится импорт и для импортируемого пакета. Комбинация значений данных полей образует составной первичный ключ.

Классы описываются в таблице CLAZZ. В этой таблице хранятся имена классов (поле NAME), количество допустимых экземпляров (или кратность) классов (поле MULTIPLICITY), информация о том, является ли класс видимым вне своего пакета (поле IS_PRIVATE) и другие данные. Для логического (булева) типа данных используется тип CHAR(l), принимающий значения 'Y' для «истина» и 'N' для «ложь».

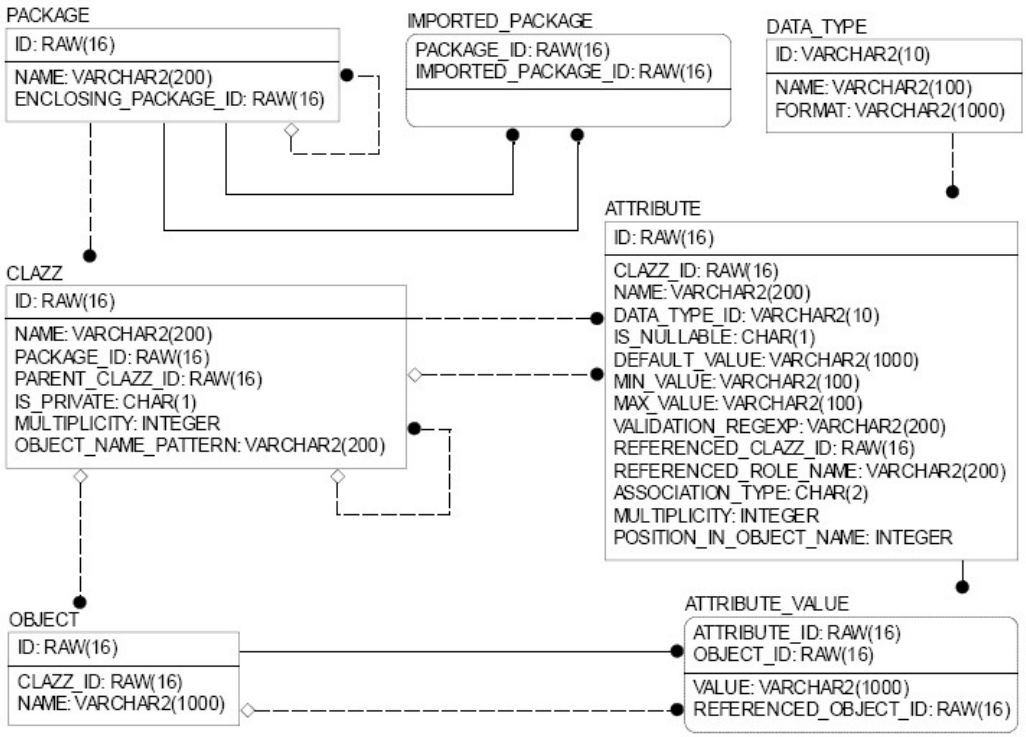

Рис. 1. Диаграмма физической структуры данных в нотации IDEF1x

Отношения обобщения описываются рекурсивной связью через внешний ключ PARENT_CLASS_ID, в котором указывается идентификатор родительского класса. В случае, если класс не имеет родителя, значение указанного поля должно быть пустым.

Исходя из опыта разработки прикладного программного обеспечения для работы с данными, можно говорить о том, что подавляющее большинство описываемых объектов предметной области должно иметь некое наименование. Для ряда объектов название задается явно при их создании или редактировании. Однако для некоторых объектов наименование удобнее формировать из значений их атрибутов по определенному шаблону, например: для некоторого узла в составе сборочной единицы название можно формировать как название сборочной единицы плюс номер узла по спецификации к технологической схеме. Шаблон для автоматического формирования наименования объекта задается в поле OBJECT_NAME_PATTERN. Так, для примера, описанного выше, шаблон может выглядеть как '%1, узел №%2', где %01 и %02 – порядковые номера атрибутов в названии объекта (в данном случае для атрибутов «сборочная единица» и «номер по спецификации», см. далее).

Всего в системе определено восемь типов данных: «число», «строка», «текстовое поле», «булев тип», «год», «дата», «время» и «дата-время». Для каждого типа в таблице DATA_TYPE имеется запись, которая содержит название типа (поле NAME) и формат типа (поле FORMAT). Формат представляет собой PL/SQL код, преобразующий входные данные к виду, в котором они будут храниться в базе данных. Например, для типа «год» формат будет иметь следующий вид: to_char(to_date(decode(length(trim('%s')), 4, '01.01.' || '%s', '%s'), 'dd.mm.yyyy'), 'yyyy'). При редактировании данных в системе все вхождения %s в строке формата заменяются на новое значение атрибута, после чего полученный код исполняется и результат сохраняется в базе данных.

Кроме указания типа, на множество значений атрибута можно наложить дополнительные ограничения. Так, например, атрибуты, имеющие тип «число» или «дата», можно ограничить минимальным и максимальным значением (поля MIN_VALUE и MAX_VALUE), а атрибуты с типом «строка» или «текстовое поле» можно проверить на соответствие шаблону регулярного выражения (поле VALIDATION_REGEXP). Также для каждого атрибута может быть задано значение по умолчанию (поле DEFAULT_VALUE) и указано, может ли данный атрибут принимать пустое значение (поле IS_NULLABLE).

Помимо значений простого типа («число», «строка», «дата» и так далее) атрибут может хранить ссылку на объект. Чтобы указать, на экземпляры какого класса может ссылаться

данный атрибут, необходимо задать идентификатор требуемого класса в поле REFERENCED_CLAZZ_ID. Таким образом, моделируется отношение ассоциации, тип которой можно указать в поле ASSOCIATION_TYPE. Данное поле может принимать значения: 'AS' для ассоциации в общем случае, 'AG' для агрегирования, 'CM' для композиции и 'CA' для указания, что данная ссылка является частью класса-ассоциации. Тип ассоциации накладывает ограничения на время жизни связанных объектов в соответствии со своей семантикой.

Простым заданием ссылки на объект моделируется ассоциация «один ко многим». Полюсу ассоциации, к которому принадлежит класс с атрибутом-ссылкой, соответствует кратность 0. . *. Указанный интервал кратности можно ограничить, задав верхний предел в поле MULTIPLICITY. Имя роли для данного полюса задается в поле NAME. Кратность, соответствующая противоположному полюсу ассоциации, равна 0. . 1 (или 1 в зависимости от значения поля IS_NULLABLE). Имя роли для указанного полюса может быть задано в поле REFERENCED_ROLE_NAME. Ассоциации «многие ко многим» всегда моделируются с помощью класса-ассоциации. Класс-ассоциация описывается в системе, как обычный класс, за исключением того, что он обязательно должен иметь два атрибута, ссылающихся на классы в полюсах класса-ассоциации и определенных как 'CA' в поле ASSOCIATION_TYPE.

В таблицу ATTRIBUTE_VALUE заносятся значения атрибутов. Каждое такое значение уникально определяется с помощью комбинации значений идентификатора атрибута в поле ATTRIBUTE_ID и идентификатора объекта в поле OBJECT_ID. Указанные поля являются внешними ключами к таблицам ATTRIBUTE и OBJECT соответственно и образуют составной первичный ключ к таблице ATTRIBUTE_VALE.

Все значения атрибутов простых типов хранятся в поле VALUE, определённом для универсальности как VARCHAR2(1000). Значения атрибутов-ссылок заносятся в поле REFERENCED_OBJECT_ID. Данное поле является внешним ключом к таблице OBJECT. При задании или редактировании атрибута-ссылки в поле VALUE заносится название объекта, на который указывает идентификатор REFERENCED_OBJECT_ID.

Процедуры для работы с данными

Логика работы с данными, хранящимися в описанной структуре, инкапсулирована в слое хранимых процедур [4]; прямая вставка и изменение данных в таблицах запрещены. Эти процедуры обеспечивают соблюдение требований целостности информации, и, в случае обнаружения ошибки в получаемых данных, инициируют соответствующее программное исключение. Процедуры собраны в одном PL/SQL пакете Utils. Далее для того, чтобы различать понятие «пакет» (PACKAGE) в PL/SQL и понятие «пакет», относящееся к логической структуре данных, везде, где речь идет о PL/SQL пакете, это будет указано явно.

Так как состояние PL/SQL пакета в течение сессии сохраняется, идентификатор текущей сущности (пакета, класса, атрибута и так далее) удобно хранить в соответствующей переменной PL/SQL пакета. В разработанной схеме текущий пакет задается в переменной Utils.curPackageId, по умолчанию равной идентификатору корневого пакета, класс – в переменной Utils.curClassId и т.д. При этом, например, при создании класса с помощью соответствующей процедуры, если пакет не задан явно, подразумевается, что указанный класс принадлежит текущему пакету.

Генерация первичных ключей (UUID) вынесена в отдельную функцию createUUID, которая работает через вызов системной функции SYS_GUID() и при необходимости может быть переопределена.

Управление сущностями выполнено через специализированные процедуры, каждая из которых отвечает за элементарную операцию создания, удаления, изменения сущности с соблюдением определённых требований. Так, для гарантии непротиворечивости информации при работе с пакетами требуется обеспечить выполнение следующих условий:

- уникальность имен сущностей (пакетов и классов) внутри объемлющего пакета;
- отсутствие рекурсивной вложенности пакетов;
- соблюдение границ видимости в соответствии с установленными правилами;
- каскадное удаление всех сущностей, содержащихся в удаляемом пакете.

Процедуры для работы с классами гарантируют соблюдение следующих ограничений:

- имена классов внутри пакета должны быть уникальными;
- при изменении шаблона для автоматического формирования названий объектов требуется обеспечить обновление названий всех экземпляров класса и всех значений атрибутов-ссылок, указывающих на экземпляры данного класса;

- при объявлении класса закрытым внутри пакета или при перемещении класса из одного пакета в другой, если данный класс имеет отношения с другими классами, необходимо соблюдать условия видимости в соответствии с правилами;
- отношения обобщения не должны быть рекурсивными;
- необходимо контролировать соответствие значения кратности класса количеству его экземпляров;
- при удалении класса должно происходить каскадное удаление его экземпляров, атрибутов (включая значения) и отношений, которыми связан данный класс с другими классами.

Аналогично в пакете Utils присутствует блок процедур для работы с атрибутами, отвечающий за их создание, задание названия, свойств, шаблонов проверки корректности данных (regexp), вхождение значений атрибутов в автоматически формируемое название объекта. При работе с атрибутами требованиями обеспечения целостности данных являются:

- контроль соответствия значений атрибутов задаваемым ограничениям;
- соответствие вхождения значений атрибутов в формируемое автоматически название объекта заданному порядку;
- каскадное удаление значений атрибута при удалении самого атрибута.

В отдельную группу вынесены процедуры для работы с ассоциациями. Любое отношение ассоциации определяется двумя полюсами. Для каждого полюса ассоциации должен быть указан класс, роль данного класса в ассоциации и кратность. Для описания полюса ассоциации в пакете Utils определён соответствующий тип. Для создания ассоциации необходимо передать в качестве входных параметров заполненные соответствующим образом записи для полюсов данной ассоциации. В зависимости от значений кратности в полюсах ассоциации будет создан либо атрибут-ссылка для соответствующего класса (отношение «один ко многим»), либо класс-ассоциация (отношение «многие ко многим»). Названия экземпляров класса-ассоциации автоматически формируются из значений атрибутов-ссылок, соответствующих её полюсам.

Для обеспечения целостности данных при работе с ассоциациями необходимо учитывать следующее:

- связи, описываемые ассоциацией, должны соответствовать значениям кратности, определённым для её полюсов;
- отношения агрегирования должны отвечать требованиям асимметричности (отсутствие рекурсии);
- для класса ассоциации должно быть определено в точности два атрибута-ссылки, указывающих на её полюса.

При работе с объектами условия обеспечения целостности таковы:

- количество объектов не должно противоречить значению кратности соответствующего класса;
- при создании объекта необходимо сформировать значения атрибутов, для которых определено значение по умолчанию;
- при удалении объекта-композита необходимо удалить все объекты, входящие в его состав;
- при изменении названия объекта необходимо привести в соответствие автоматически формируемые названия, имеющие вхождения атрибута-ссылки, указывающего на заданный объект.

При работе со значениями атрибутов объектов в соответствующих функциях и процедурах учтено, что:

- значение должно удовлетворять ограничениям, заданным для соответствующего атрибута;
- значение атрибута-ссылки не должно противоречить кратности, указанной для соответствующей ассоциации;
- при создании значения атрибута-ссылки в поле VALUE должно указываться название соответствующего объекта;
- пустому значению атрибута соответствует отсутствие записи в таблице ATTRIBUTE_VALUE.

Не все действия, описываемые приведенными процедурами, являются атомарными. Здесь под атомарностью понимается, что операция является наименьшим, неделимым блоком алгоритма изменения данных. Так, например, мы не можем создать только объект, для которого определен атрибут, не допускающий пустых значений. В данном случае атомарной операцией будет являться создание объекта и задание значения указанного атрибута. В связи с этим проверку некоторых ограничений необходимо откладывать до завершения выполнения определенного блока процедур. Задать указанный блок можно с помощью специальных процедур beginBlock() и endBlock().

По умолчанию вызов любой процедуры для модификации данных считается атомарным, то есть проверка информации на непротиворечивость выполняется перед завершением работы

процедуры. Если вызов процедуры производится после вызова Utils.beginBlock(), то непосредственная модификация и проверка целостности данных откладывается до вызова завершающей процедуры Utils.endBlock(). Т.о., вызовы процедур, помещенные между Utils.beginBlock() и Utils.endBlock(), образуют атомарный блок.

Заключение

В работе рассмотрена методика универсальной (не привязанной к определенной предметной области) организации структуры данных в реляционной СУБД, основанная на метамодели языка UML. Указанный подход может быть полезен при решении широкого спектра прикладных задач, которые связаны со сбором первичной информации в блок транзакционных данных. При использовании данного метода создание логической модели предметной области основывается на формировании метаданных об описываемых объектах и связях между ними. Анализ сформированных метаданных позволяет упростить решение ряда задач, в том числе:

- разработки CASE средств для формирования модели предметной области;
- наложения сложных структурных ограничений на модель предметной области;
- автоматизации формирования форм ввода и отображения информации при разработке типовых программных приложений для работы с данными;
- организации синхронизации данных в распределённых, в т.ч. иерархических, системах с переменным набором описываемых объектов и их характеристик;
- ведения журналов изменения как самих данных, так и их структуры.

Благодаря уникальности идентификаторов описание объекта на верхнем уровне иерархической информационной системы будет представлять собой композицию всех субмоделей в нижележащих уровнях иерархии, построенной на основе базовой структуры. Это существенным образом облегчает построение распределённых структур хранения данных и повышает уровень их согласованности и непротиворечивости, т.к. все ограничения заложены на уровне единой объектной модели.

Данный подход наиболее эффективен, когда набор регистрируемых параметров невелик (десятки–сотни для каждого типа объекта), но требуется возможность динамического изменения структуры описываемых объектов при гарантированном сохранении целостности. Напротив, хранение потоковых данных, таких, как данные реального времени и т.д., т.е. когда структура данных фиксирована, а объёмы данных большие, целесообразно построение нормализованных структур. В целом опыт использования подобной структуры данных для практических применений [5] показал, что надёжность и производительность вполне достаточны для практического применения, а удобство в настройке создаёт существенные преимущества перед другими вариантами построения модели хранения данных.

Список литературы

1. Энциклопедия PLM. Екатеринбург: Азия, 2008. 448 с.

2. Шерстюк Ю.М. Основы метауправления функциональностью в информационных системах. СПб.: СПИИРАН, 2000. 156 с.

3. Язык UML. Руководство пользователя / Г. Буч, Дж. Рамбо, А. Джекобсон. М.: ДМ К Пресс; СПб: Питер, 2004.

4. Oracle PL/SQL для профессионалов 3-е изд. / С. Фейерштейн, Б. Прибыл. СПб: Питер, 2003.

5. Решетников И.С. Информационная система поддержки принятия решений в многоуровневой структуре на примере организации капитального ремонта нефтегазовой компании // Вестник Казанского государственного технического университета им. А.Н. Туполева. 2007. № 3. С. 45–48.

ONTOLOGICAL MODEL OF A TRANSACTIONAL DATA BLOCK IN A RELATIONAL DBMS

I.S. Reshetnikov, P.N. Kosnyrev

The paper considers a technique to organize physical data structure in a relational DBMS. The technique is based on the subset of the Unified Modeling Language (UML) metamodel for class diagrams.

Keywords: metamodel, package, class, attribute, object, relation, generalization, association, aggregation, composition.