Разработка репозитория метаданных на основе объектно-ориентированной логической модели для оценки качества данных

С.Е. Духовенский

***Аннотация* — Рассмотрен подход к организации репозитория метаданных как одного из элементов системы оценки качества данных. Для формирования репозитория предлагается использовать объектно-ориентированную модель данных, обогащенную проверками качества данных. Дано описание основных элементов репозитория такого типа, а также описан подход по привязке проверок качества данных к объектам модели данных. Предложен способ хранения рассмотренного репозитория метаданных, разработаны специальные алгоритмы для его обработки, включая алгоритм «распаковки» атрибутов класса и алгоритм определения актуальных проверок данных с учетом возможных переопределений. На основе описанных теоретических положений был реализован прототип репозитория метаданных. Указанный прототип был использован для организации проверок на примере задачи оценки качества данных операторов, осуществляющих обработку персональных данных. В сравнении с реализацией репозитория метаданных на основе физической модели данных, применение описанного в настоящем исследовании подхода отличается сокращением объема описанных атрибутов и проверок качества данных на 23% и 27% соответственно при одновременном сохранении количества реально запускаемых («реализованных») проверок. Исследуемый в статье подход может быть полезен в практических задачах анализа качества данных как потенциальный способ снижения трудозатрат на управление проверками качества.**

***Ключевые слова* — качество данных, объектно-ориентированная модель данных, репозиторий метаданных.**

I. Введение

Задачи оценки и контроля качества данных больших информационно-аналитических систем (например, в [1]) часто сопровождаются необходимостью разработки и поддержания в актуальном состоянии большого количества проверок качества. Одним из возможных вариантов организации эффективного процесса управления качеством данных является разработка специализированного репозитория, который позволяет автоматизировать часть процессов [2], [3]. Для этого репозиторий метаданных включается как самостоятельный компонент в систему контроля за качеством данных и отвечает за хранение и обработку используемых проверок качества данных.

На практике можно выделить следующие способы организации репозитория метаданных:

1) В форме исполняемых скриптов, таких как SQL запросы, python модулей или ETL джобов – здесь репозиторий метаданных выступает как хранилище готовых к выполнению проверок. В качестве примера можно привести выделенный под цели качества данных git репозиторий.
2) С привязкой к физической модели данных – в этом случае пользователь лишь описывает применяемые к физическим объектам показатели качества данных, а формирование исполняемых проверок передается другому компоненту системы качества данных. Таким образом реализовано большинство специализированных инструментов, таких как SAS, CloverDX, AWS Deequ, Great Expectation (open source) и др.
3) С привязкой к логической модели данных – схожий с предыдущим формат, но описание показателей качества данных применяется к логическим объектам. Критичным моментом для реализации данного репозитория метаданных является необходимость ведения корректной логической модели данных, а также нетривиальный алгоритм формирования исполняемых проверок вследствие необходимости их переноса в плоскость физических объектов БД.

Ввиду того, что большинство современных информационно-аналитических систем характеризуются постоянным ростом как объема данных, так и количеством обрабатываемых объектов, эффективный процесс контроля за качеством данных сопровождается пропорциональным ростом проверок качества данных и ресурсов на их поддержку.

В этой связи репозиторий в форме исполняемых скриптов представляется наименее гибким и наиболее затратным инструментом, поскольку каждая проверка требует отдельной реализации и поддержки. Репозиторий метаданных, опирающийся на физическую модель данных, частично упрощает разработку и поддержку проверок, но при этом расширение модели данных при-

Статья получена 10 марта 2024.

Духовенский С.Е., аспирант РТУ МИРЭА, Москва, Россия (dukhovenskiy.s.e@edu.mirea.ru).

водит к необходимости добавления новых проверок. При этом разработка проверок для новых объектов часто не учитывает полученные ранее наработки, вследствие чего многие промышленные инструменты предлагают создание внутренней базы шаблонов, в которой сохраняются типичные виды проверок. Тем не менее, даже при наличии такой базы разработка проверок включает поиск и привязку подходящего шаблона к объекту, а также необходимость корректировки шаблонной проверки под конкретный кейс.

Учитывая описанные особенности, представляется перспективным реализация метаданных на основе логической модели данных. К таким моделям относят иерархические, сетевые, реляционные и объектно-ориентированные модели данных (далее – ООМД). Среди указанных моделей именно ООМД обладают потенциалом к увеличению эффективности управления репозиторием метаданных за счет использования принципов наследования и композиции.

Постановка задачи оценки качества данных и анализ современных методов ее решения приведены в [5] и [6] соответственно. Учитывая популярность ООМД и широкие возможности их применения к различным практическим задачам, вопросам построения ООМД посвящены многочисленные научные статьи и книги. Настоящее исследование в первую очередь опиралось на идеи и рекомендации, описанные в [7]-[9]. Для оценки успешности исследуемого в статье подхода использовалась физическая модель и разработанные в [3] проверки качества данных реестра персональных данных (далее – Реестр) [4].

Целью данной работы является выработка подхода к организации репозитория метаданных на основе ООМД как одного из элементов системы оценки качества данных. Для достижения указанной цели решались следующие задачи:

- описание основных элементов и принципов работы репозитория метаданных, а также методологии привязки проверок качества данных (раздел II),
- разработка алгоритмов хранения и обработки репозитория метаданных и реализация соответствующего прототипа (раздел III),
- применение прототипа к описанию проверок данных Реестра и анализ эффективности предложенного решения (раздел IV).

II. Описание элементов репозитория и методология привязки проверок

В контексте задачи оценки качества данных репозиторий метаданных должен выполнять две основные задачи: отражать структуру анализируемых данных и предоставлять возможность управления проверками качества. Для решения этих задач предлагается использовать идеи объектно-ориентированной парадигмы и ООМД.

Как указано в [9], объектно-ориентированный подход к построению логических моделей данных обладает достаточно широкими возможностями. Поскольку настоящее исследование сфокусировано на задаче эффективного управления проверками качества, предлагаемая реализация репозитория метаданных базируется лишь на части ключевых элементов и подходов, используемых для объектно-ориентированного моделирования предметной области. Далее в разделе определяются используемые элементы и принципы.

A. Элементы репозитория и связи между ними

Ключевыми элементами репозитория метаданных являются классы, атрибуты, примитивные типы данных (число, строка, дата и т.п.) и проверки качества данных. Каждый элемент имеет идентификатор, уникальный внутри репозитория метаданных; далее без ограничения общности будем полагать, что идентификаторы представлены строкой.

Указанные элементы связаны друг с другом следующим образом.

- Атрибуты обязательно включаются в классы отношением «многие к одному». При этом класс может вообще не содержать ни одного атрибута.
- Два класса могут быть связаны друг с другом отношением «родитель – потомок».
- Атрибуты могут быть либо представлены значениями одного примитивного типа, либо являться объектами другого класса. Иначе говоря, во втором случае один класс «вкладывается» в другой посредством атрибута непримитивного типа.
- Внутри одного класса атрибут может быть представлен либо одним экземпляром, либо коллекцией одного типа.
- Проверки качества данных привязываются отношением «один к одному» либо к отдельному классу, либо к атрибуту, являющимся объектом другого класса.

Отталкиваясь от вышесказанного, можно разделить классы на следующие типы:

- абстрактные – классы, у которых есть потомки;
- сателлиты – классы, вложенные в другие классы через атрибуты непримитивных типов;
- терминальные – все прочие классы.

На рис.1 представлен пример диаграммы, отражающей классы, атрибуты и их взаимосвязи: каждый прямоугольник означает класс (идентификатор указан в заголовке) и входящие в него атрибуты, где после идентификатора атрибута указан его тип. Сплошные стрелки отражают отношение «родитель – потомок», а пунктирные – отношение «вложения». В соответствии с определенным выше, красным цветом отмечены абстрактные классы, серым – сателлиты, а зеленым – терминальные классы.

Представленный пример демонстрирует, что один класс может иметь цепочку родительских классов (Class3 является прямым потомком Class2 и непрямым потомком Class1), один сателлит может вкладываться в несколько классов (Satellite1 вложен в Class2 через атрибут Attr2 и вложен в Class3 через атрибут Attr4), а также сателлиты могут иметь собственные вложения (Satellite3 вкладывается в Satellite2 через атрибут Attr8).

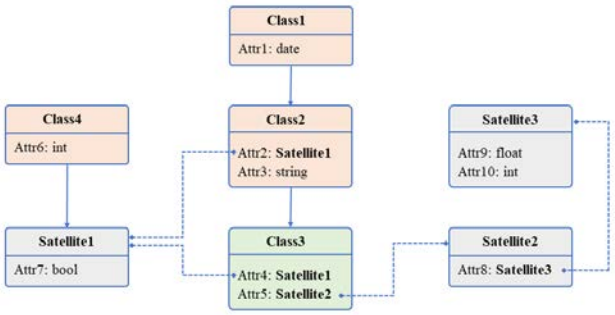

Рис. 1. Пример описания классов и атрибутов

B. Принципы наследования и композиции

В предлагаемом подходе указанные выше элементы и связи между ними дополняются следующими принципами:

1) принцип наследования: все атрибуты класса-родителя автоматически копируются в его классах-потомках;
2) принцип композиции (вложения): все атрибуты класса-сателлита автоматически копируются во всех классах, куда вкладывается этот сателлит.

При описании модели данных представленные принципы позволяют декомпозировать объекты на составляющие, выделять общие атрибуты (по принципу наследования) и повторно использовать одинаковые структуры данных (по принципу композиции). Как будет показано ниже, при корректном проектировании подобный подход позволяет существенно сократить описание повторяющихся атрибутов по сравнению с описанием физической модели данных.

C. «Распаковка» атрибутов класса

Описанные выше правила позволяют построить логическую модель анализируемых данных и дополнить ее проверками данных. Поскольку непосредственное выполнение проверок осуществляется на физическом уровне, предварительно необходимо преобразовать логическую модель в физическую. Данный процесс потребует промежуточного шага, который в рамках данной статьи будет именоваться «распаковка» атрибутов класса.

Под «распаковкой» атрибутов определенного класса подразумевается определение всех атрибутов данного класса, включая:

- исходные атрибуты;
- атрибуты, полученные от всех классов-родителей по принципу наследования;
- атрибуты, полученные от всех сателлитов по принципу композиции.

Для примера на рис. 2 представлена распакованная логическая структура терминального класса Class3 со схемы на рис.1. В круглых скобках указаны названия классов, откуда были получены соответствующие атрибуты. Например, атрибут Attr1 с типом date был скопирован из родительского класса Class1, а атрибут Attr10 с типом float был получен путем двойного вложения: сначала Satellite3 был вложен в Satellite2 через Attr8, а потом последний был вложен в Class3 через Attr5.

При последующем преобразовании распакованной логической структуры в физическую структуру остаются только атрибуты примитивного типа. Для примера на рис.2 столбцы физической структуры были именованы по номерам соответствующих атрибутов из логической структуры. Тем не менее, алгоритм определения уникального наименования столбцов в физической структуре не всегда является тривиальным и остается за рамками данной статьи.

Рис. 2. Пример распакованной логической и физической структур

Для дальнейшего изложения необходимо рассмотреть вопрос идентификации атрибутов в распакованной структуре класса. Как видно из представленного на рис.2 примера, в общем случае множество полученных в результате распаковки атрибутов может содержать в себе атрибуты с одинаковым идентификатором вследствие вложения сателлита в несколько разных атрибутов: атрибуты примитивных типов Attr6 и Attr7 дважды копируются в Class3 за счет вложения в атрибуты Attr2 и Attr4.

Для уникальной идентификации подобных атрибутов достаточно использовать конкатенацию всех идентификаторов, участвующих в цепочке вложенности, сохраняя их порядок в соответствии с уровнем вложенности. Например, для атрибута Attr10 конкатенация идентификаторов примет вид «Attr5.Attr8.Attr10». Ниже по тексту такие строки будут обозначаться как *расширенный идентификатор*. Дополнительным преимущественном расширенных идентификаторов является возможность легко определить полную цепочку вложенностей для полученного в результате «распаковки» атрибута.

D. Добавление проверок качества

Реализованные на физическом уровне проверки качества данных можно условно разделить на следующие виды:

- проверки на уровне всей таблицы – например, проверка доступности,
- проверки на уровне всей таблицы с использованием столбцов в качестве параметров – например, проверка на отсутствие дубликатов в таблице на основе кортежа

столбцов, составляющих уникальный ключ,
- проверки на уровне одной строки с использованием столбцов в качестве параметров – например, проверка согласованности нескольких столбцов,
- проверки на уровне одного столбца – например, проверка на вхождение значений в заданный диапазон.

Для реализации проверок любого из перечисленных типов в исследуемом в статье репозитории метаданных предлагается добавлять проверки на уровне классов, при этом в качестве параметров проверки можно использовать атрибуты примитивных типов из распакованной структуры класса. Такие проверки также подчиняются принципам наследования и композиции:

1) Если проверка была добавлена к абстрактному классу, то эта проверка автоматически порождает копии во всех классах-потомках вплоть до терминальных классов (по принципу наследования).
2) Если проверка была добавлена к сателлиту, то эта проверка автоматически порождает копии во всех классах, куда вкладывается этот сателлит, вплоть до терминальных классов (по принципу композиции)

При этом важно отметить, что в обоих случаях порождение копии проверки не требует изменения её параметров: используемые в качестве параметров атрибуты автоматически копируются по принципам наследования и композиции вслед за копированием проверки.

На рис.3 представлены цепочки проверок, порождаемых добавлением проверки к самому левому элементу (на основе изображенной на рис.1 модели).

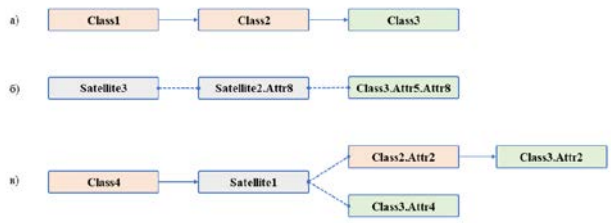

Рис. 3. Примеры цепочек порожденных проверок

Так, пример «а» описывает проверку, добавленную на уровне класса Class1: по принципу наследования сначала она порождает копию проверки на уровне класса Class2, а затем на уровне класса Class3. Пример «б» описывает проверку, добавленную на Satellite3: по принципу вложения сначала порождается проверка на уровне атрибута Attr8 в Satellite2, а затем порождается проверка на уровне атрибута Attr5 в Class3. Пример «в» изображает более сложный случай: добавленная на уровне Class4 в результате цепочки наследований и вложений порождает две (!) одинаковые проверки в Class3 – одна на уровне Attr2, вторая на уровне Attr4.

Из приведенного примера легко заметить, что одна проверка может порождать несколько копий на уровне одного класса (эти копии будут отличаться параметрами). Для различения таких случаев предлагается определять копию проверки, порожденную за счет вложения сателлита в атрибут другого класса, не на уровне этого класса, а на уровне атрибута, являющегося объектом данного сателлита (далее просто «на уровне объекта»). Например, крайняя правая копия проверки из цепочки «б» на рис.3 будет определена на уровне объекта Attr8, вложенного в объект Attr5 класса Class3.

E. Переопределение проверок качества

На практике часто возникает необходимость корректировки отдельных параметров порожденных проверок. В этой связи в репозиторий метаданных предлагается добавить возможность переопределения порожденных копий проверок. Переопределение означает добавление новой проверки поверх порожденной копии на уровне соответствующего класса или объекта. Переопределенная таким образом проверка автоматически порождает «скорректированные» копии далее по цепочке вплоть до терминальных классов.

Например, пусть на уровне Class4 для модели из рис.1 была определена проверка на диапазон значений вида «*Attr6 > 0*». Тогда на всех элементах цепочки «в» на рис.3 будут порождены соответствующие копии проверки «*Attr6 > 0*». Если на уровне Class2.Attr2 проверка будет переопределена на «*Attr6 > 1*», то на уровне Class3.Attr2 копия проверки также примет вид «*Attr6 > 1*» (прочие копии останутся без изменений).

Возможность переопределения проверок формирует дополнительную задачу, возникающую в момент трансляции проверки с уровня логической модели на физический уровень: каким образом для определенного класса или атрибута определить актуальный вариант проверки с учетом возможных переопределений? Одним из решений является построение обратной цепочки порождения проверок: двигаясь справа налево необходимо определить первый элемент, на уровне которого происходит добавление или переопределение проверки, и взять соответствующую проверку. Для примера выше, проверка диапазона значений Attr6 в объекте Class3.Attr4 определяется её самым первым видом, определенным на Class4, причём та же проверка на объекте Class3.Attr2 будет сформирована переопределенным на уровне Class2 значением (рис. 4).

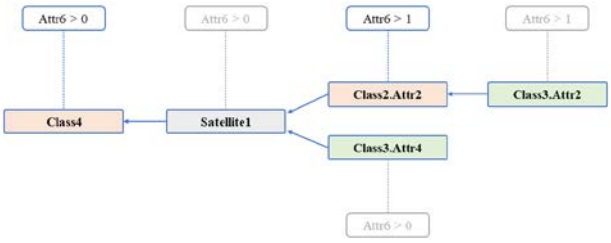

Рис. 4. Пример обратной цепочки

III. Алгоритмы хранения и обработки репозитория

В рамках настоящего исследования был разработан прототип репозитория метаданных на основе ООМД, представляющий собой Web-приложение на языке Python в связке с БД PostgreSQL. В прототипе была реализована полная функциональность представленного в

разделе II репозитория метаданных. Ниже описаны ключевые архитектурные особенности и алгоритмы, использованные при реализации данного прототипа.

A. Хранение элементов репозитория

Подчинение классов естественным образом можно попробовать представить в виде древовидной структуры данных. Тем не менее, во-первых, при описании конкретной модели данных может не оказаться единого «корневого» элемента (и его придется вводить искусственно), а во-вторых, такое хранение не обеспечивает возможность вложения одних классов в другие через атрибуты. В этой связи для реализации репозитория метаданных необходимо использовать более гибкую структуру данных – ориентированный граф.

В качестве вершин графа выступают объекты, которые характеризуются названием, уникальным идентификатором и одним из следующих типов: класс, атрибут, примитивный тип данных.

Ребра графов связывают только пары вершин определенных типов:

- однонаправленное отношение «Потомок → Родитель» связывает класс-потомок с непосредственным классом-родителем,
- однонаправленное отношение «Атрибут → Класс» связывает атрибут с классом, которому он принадлежит,
- однонаправленное отношение «Атрибут → Тип» связывает атрибут либо с примитивным классом, либо с классом, объектом которого является этот атрибут.

Представление репозитория в таком виде обладает некоторой избыточностью, но при этом позволяет реализовать относительно простые алгоритмы, требующиеся для выполнения ключевых операций.

B. Алгоритм «распаковки» атрибутов класса

Как описано в разделе II, распакованная структура класса необходима для реализации возможности переопределения проверок, а также в момент переноса логической модели на физический уровень.

Алгоритм «распаковки» получает на вход идентификатор класса C_0 и возвращает S_{res} – множество расширенных идентификаторов атрибутов.

Шаг 0. Инициализируем множество $S_{res} = \emptyset$.

Шаг 1. Стартуя с класса C_0 и продвигаясь по цепочке от потомка к родителю (вплоть до класса без родителей – всего n родителей), определяем для класса C_0 идентификаторы всех его родительских классов C_i, где $i = 1..n$.

Шаг 2. Для каждого C_i определяем множество идентификаторов его атрибутов S_i, где $i = 0..n$.

Шаг 3. Конструируем множество $S_{all} = \cup S_i$,, где $i = 0..n$.

Шаг 4. В цикле для каждого идентификатора A из множества S_{all} выполняем следующие действия:

1) определяем для атрибута с идентификатором A соответствующий ему тип атрибута – обозначим его идентификатор как T;
2) если T является идентификатором примитивного класса, то добавляем A в S_{res} и переходим к следующей итерации на шаге 4;
3) в противном случае рекурсивно вызываем алгоритм для идентификатора класса T и получаем на выходе множество расширенных идентификаторов S^*; затем дополняем множество S_{res} результатом CONCAT(A, A^*) для всех A^* из множества S^*, где CONCAT() – функция конкатенации ее аргументов.

Шаг 6. Возвращаем множество S_{res}.

Важным условием для корректной работы данного алгоритма является отсутствие в графе циклов – на практике это означает, что атрибут определенного класса не может быть объектом данного класса или объектом всех его потомков любого уровня.

C. Хранение привязки проверок

Описание проверок качества данных в прототипе репозитория реализовано в двух вариантах:

1) первичное определение проверки на уровне класса,
2) переопределение порожденной копии на уровне класса или объекта.

Первичное определение проверки реализуется за счет хранения простой связки идентификатора проверки и идентификатора вершины графа типа класс. При этом если проверка содержит в себе параметры в виде атрибутов, то данные параметры тоже достаточно связать с идентификаторами необходимых вершин графа с типом атрибут.

В случае переопределения такая простая связка не сработает в ситуации переопределения проверки на уровне объекта. Здесь идентификатор проверки привязывается к расширенному идентификатору атрибута, дополненному идентификатором класса. При этом параметры проверки, относящиеся к атрибутам, также связываются с расширенным идентификаторами соответствующих атрибутов.

D. Алгоритм определения актуальных проверок

Как было описано в разделе II, при переносе проверок на физический уровень необходимо решать задачу определения актуального варианта проверки с учетом возможных переопределений. Для решения данной задачи был реализован алгоритм, который выполняется на основе распакованных классов.

Алгоритм определения актуальной проверки получает на вход три параметра: идентификатор проверки R, идентификатор класса C_0 и, при наличии, расширенный идентификатор атрибута в виде A = CONCAT(A_1, A_2, … A_n), где n – количество параметров функции конкатенации. На выходе алгоритм возвращает описание проверки или пустоту, если это описание не определено.

Шаг 0. Инициализируем две новые переменные:

$C^* = C_0$ и $A^* = A$.

Шаг 1. Полагаем, что A^* представлено в виде CONCAT(A_1^*, A_2^*, … A_m^*), где $0 \le m \le n$.

Шаг 2. Поочередно проверяем условия:

1) если $m = 0$ и к классу C^* привязана проверка R, то останавливаем алгоритм и возвращаем описание привязанной проверки,
2) если $m > 0$ и к атрибуту A^* класса C^* привязана проверка R, то останавливаем алгоритм и возвращаем описание привязанной проверки.

Шаг 3. Пытаемся для класса C^* определить его родительский класс – обозначим его как C_p.

Шаг 4. Поочередно проверяем следующие условия:

1) если $m = 0$ и C_p не было определено, то останавливаем алгоритм и возвращаем пустоту,
2) если $m = 0$ и C_p было определено, то полагаем $C^* = C_p$ и переходим к шагу 2,
3) если $m > 0$ и в классе C_p присутствует атрибут A^*, то полагаем $C^* = C_p$ и переходим к шагу 2,
4) иначе для идентификатора A_1^* определяем идентификатор его типа – обозначим как T. Далее полагаем $C^* = C_p$ и A^* = CONCAT(A_2^*, … A_m^*) – и переходим к шагу 1.

Представленный алгоритм работает из предположения, что в качестве параметра A передается расширенный идентификатор атрибута, являющимся объектом другого класса.

IV. Пример реализации репозитория метаданных

Описанный выше подход к организации репозитория метаданных был опробован для реализации проверок качества данных Реестра, разработанных ранее в [3].

В рамках текущего исследования указанная физическая модель реестра была преобразована в классы следующих типов: 2 абстрактных класса («Общая» и «Дополнительная»), 3 сателлита («Приказ», «ОбрДанных» и «ТрПередача») и 3 терминальных класса, соответствующих трем исходным таблицам (рис. 5). Общее количество атрибутов составило 30, что на 23% меньше количества атрибутов в физической модели из [3].

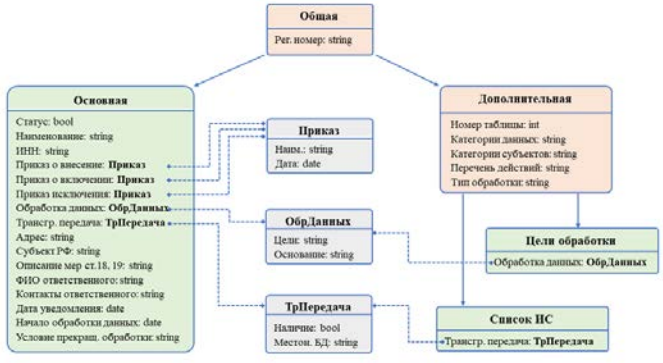

Рис. 5. Модель данных Реестра в репозитории метаданных

Разработанный репозиторий был наполнен проверками качества, описанными в [3] – результат привязки представлен в Табл. I ниже. Проверки, описанные в последней строке таблицы, не могут быть привязаны к конкретному классу, потому они требуют отдельной реализации. Итоговое количество проверок в репозитории составило 43, что на 27% меньше исходных проверок на основе физической модели данных.

Таким образом, реализация репозитория метаданных на основе ООМД в задаче анализа качества данных Реестра привела к сокращению описания атрибутов и проверок качества данных относительно используемых в [3] объектов. Такое сокращение связано с группировкой повторяющихся атрибутов в таблицах «Цели обработки» и «Список ИС» в абстрактном классе «Дополнительная», выделением трех схожих атрибутов «Приказ…» в отдельный сателлит и повторным использованием части пересекающихся атрибутов за счет сателлитов «ТрПередача» и «ОбрДанных». Важно отметить, что при этом количество реально запускаемых («реализованных») проверок не уменьшилось – за счет проверок, порожденных по принципам наследования и композиции.

Таблица I. Проверки качества данных Реестра в репозитории метаданных

Класс «Общая таблица» • Дубликаты по «Рег. номер» • Полнота по «Рег. номер»
Класс «Основная таблица» • Дубликаты по "ИНН" • Дубликаты по кортежу: "Наименование", "Адрес" • Полнота по "Приказ о внесении" • Полнота по "Приказ об исключении" • Полнота по "Наименование" • Полнота по "ИНН" • Полнота по "Адрес" • Полнота по "Дата уведомления" • Полнота по "Субъект РФ" • Полнота по "Описание мер, предусмотренных ст. 18.1 и 19 Закона" • Полнота по "ФИО ответственного за обработку данных" • Полнота по "Контакты ответственного" • Полнота по "Начало обработки данных" • Полнота по "Срок или условие прекращения обработки ПДн" • Допустимый диапазон по "ИНН" • Допустимый диапазон по "Наличие шифровальных средств" • Допустимый диапазон по "Приказ о включении"."Дата" • Допустимый диапазон по "Начало обработки данных" • Согласованность по "ФИО ответственного" и "Контакты" • Согласованность по "Дата уведомления" и "Приказ о включении"."Дата" • Согласованность по "Начало обработки данных" и "Приказ о включении"."Дата" • (отключение) Полнота по "Обработка данных"."Цели" • (отключение) Полнота по "Обработка данных"."Основание"
Класс «Дополнительная таблица» • (переопределение) Дубликаты по кортежу: "Рег. номер", "Номер таблицы" • Полнота по "Номер таблицы" • Полнота по "Категории данных" • Полнота по "Категории субъектов" • Полнота по "Перечень действий"

• Полнота по "Тип обработки"
Класс «Цели обработки»
• Дубликаты по «Обработка данных».«Цели»
Класс «Список ИС»
• Дубликаты по «Категории субъектов»
Класс «Приказ»
• Полнота по "Наименование" • Полнота по "Дата"
Класс «ОбрДанных»
• Полнота по "Цели" • Полнота по "Основание"
Класс «ТрПередача»
• Полнота по "Наличие" • Полнота по "Местонахождение БД" • Допустимый диапазон по "Наличие"
Смежные проверки
• Ссылочная целостность по таблицам "Основная", "Цели обработки" и "Список ИС" • Согласованность заполнения "Обработка данных" между таблицами "Основная" и "Список ИС" • Согласованность заполнения "Трансграничная передача" между таблицами "Основная" и "Цели обработки"

V. Заключение

Применение описываемого в статье подхода к реализации репозитория метаданных на основе ООМД имеет ряд преимуществ и недостатков. Как было показано, основным преимуществом предложенного репозитория является сокращение описания атрибутов и проверок качества данных – это наблюдается даже на примере небольшой модели Реестра, состоящей из 3 физических таблиц. Применение исследуемого подхода к более широкой физической модели данных потенциально может приводить к еще большему сокращению описания объектов.

Также можно отметить следующие вытекающие из вышесказанного преимущества:

- сокращение трудозатрат на разработку проверок для новых таблиц,
- гибкое управление разработанными проверками за счет возможности исправить проверку в одной точке,
- исключение «человеческой» ошибки, когда аналитик забывает добавить уже известные проверки на новые объекты.

Тем не менее, описанные преимущества компенсируются перечнем недостатков:

- необходимость разработки специализированного ПО для управления репозиторием,
- использование затратных алгоритмов обработки, включая «распаковку» классов и определение финального перечня проверок,
- необходимость «мэппинга» классов и их атрибутов на физическую модель для запуска проверок,
- высокая зависимость от корректного разбиения данных на классы и атрибуты.

Последний пункт стоит рассмотреть отдельно. В простейшем случае можно в качестве репозитория использовать физическую модель: очевидно, что тогда сокращение описания атрибутов будет нулевым, а привязка проверок к логической модели будет в точности повторять разработку на основе физической модели. Другая крайность, когда репозиторий перегружен избыточным использованием сателлитов и абстрактных классов, с одной стороны, может привести к значительному сокращению описания атрибутов, но с другой – усложнить привязку проверок, поскольку потребует их частого переопределения. Более того, поскольку в реальных ситуациях предметная область может активно изменяться и/или дополняться новым объектами, плохо структурированный репозиторий метаданных может усложнить добавление новых артефактов.

С учетом вышесказанного применение репозитория метаданных на основе ООМД для оценки качества данных может быть оправдано в ситуациях, когда сопутствующее формирование классов и атрибутов выполняется аналитиками, которые хорошо ориентируются в предметной области, понимают текущие требования к качеству данных и представляют будущее развитие репозитория. Также использование подхода может быть полезно в IT-ландшафте, где уже используют ООМД для обработки данных, например для построения хранилищ данных и реализации процессов ETL. В этом случае вопросы корректного представления ООМД, как и вопросы её «приземления» на физическую модель, скорее всего, уже решены – и необходимо будет лишь добавить возможность привязки проверок качества к готовой модели.

Библиография

[1] В.П. Лось, Е.В. Никульчев, П.Ю. Пушкин, А.М. Русаков. Информационно-аналитическая система мониторинга выполнения операторами персональных данных требований законодательства // *Проблемы информационной безопасности. Компьютерные системы*, № 3, с. 16-23, 2020.

[2] А.А. Ильин. Автоматизированная технология проектирования модели данных при построении информационно-аналитической системы // *Вестник российских университетов. Математика*, т. 13, №. 1, с. 89-90, 2008.

[3] С.Е. Духовенский, П.Ю. Пушкин, Е.В. Никульчев. Методика оценки качества данных реестра операторов персональных данных // *International Journal of Open Information Technologies*, т. 12, № 1, с. 129-136, 2024.

[4] *Реестр операторов, осуществляющих обработку персональных данных* [Электронный ресурс] URL: https://pd.rkn.gov.ru/operators-registry/operators-list/

[5] P. Oliveira, F. Rodrigues, P.R. Henriques. A formal definition of data quality problems // In *Proceedings of the 2005 International Conference on Information Quality*, MIT, 2005.

[6] J. Wang, Y. Liu, P. Li, Z. Lin, S. Sindakis, S. Aggarwal. Overview of data quality: examining the dimensions, antecedents, and impacts of data quality // *Journal of the Knowledge Economy*, 2023 https://doi.org/10.1007/s13132-022-01096-6.

[7] Е.П. Емельченков, В.И. Мунерман, Д.В. Мунерман, Т.А. Самойлова. Объектно-ориентированный подход к разработке моделей данных // *Современные информационные технологии и ИТ-образование*, т. 16, № 3, с. 564-574, 2020.

[8] L. Zhao, S.A. Roberts. An Object-Oriented Data Model for Database Modelling, Implementation and Access // *The Computer Journal*, vol. 31, no. 2, pp. 116-124, 1988.

[9] Д. Харрингтон. *Проектирование объектно-ориентированных баз данных*. Litres, 2022.

Implementation of Metadata Repository based on Object-Oriented Model for Data Quality Assessment

S. E. Dukhovenskiy

***Abstract* — The paper considers the approach to organizing a metadata repository as one of the elements of the data quality assessment system. An object-oriented data model enriched with data quality checks is proposed for repository formation. A description of the key repository elements is given, along with an approach to linking data quality checks to objects in the data model. The article provides a method for storing the discussed metadata repository and special algorithms for its processing, including the "unpacking" algorithm for class attributes and the algorithm for determining the relevant data checks considering possible overrides.**

Based on the described theoretical propositions, a prototype of the metadata repository was implemented. The prototype was used to organize checks for assessing the data quality of personal data operator's registry. In comparison with the implementation of a metadata repository based on a physical data model, the application of the approach described in this research results in a reduction of attribute and data quality check description by 23% and 27%, respectively, while maintaining the same quantity of executed checks.

The investigated approach can be useful in practical tasks related to data quality analysis as a potential way to reduce the workload of data quality check management.

***Key words* — data quality, object-oriented data model, metadata repository.**

REFERENCES

[1] V.P. Los, E.V. Nikulchev, P.Y. Pushkin, A.M. Rusakov, "Information and analytical system for monitoring the compliance of personal data operators with the requirements of the legislation," *Problems of information security. Computer systems*, no. 3, pp. 16-23, 2020. (in Rus)

[2] A. A. Ilyin, "Automated technology for designing a data model when building an information and analytical system ," *Bulletin of Russian Universities. Mathematics*, vol. 13, no. 1, pp. 89-90, 2008. (in Rus)

[3] S.E. Dukhovenskiy, P.Y. Pushkin, E.V. Nikulchev, "The data quality assessment technique of personal data operators registry," *International Journal of Open Information Technologies*, vol. 12, no. 1, pp. 129-136, 2024. (in Rus)

[4] *Personal data operators register*, https://pd.rkn.gov.ru/operators-registry/operators-list/ (in Rus)

[5] P. Oliveira, F. Rodrigues, P. R. Henriques, "A formal definition of data quality problems" in *Proceedings of the 2005 International Conference on Information Quality*, MIT, 2005.

[6] J. Wang, Y. Liu, P. Li, Z. Lin, S. Sindakis, S. Aggarwal, "Overview of data quality: examining the dimensions, antecedents, and impacts of data quality," *Journal of the Knowledge Economy*, 2023 https://doi.org/10.1007/s13132-022-01096-6.

[7] E.P. Emelchenkov, V.I. Munerman, D.V. Munerman, T.A. Samoilova, "The object oriented approach to designing data models," *Modern Information Technologies and IT-education*, vol. 16, no. 3, pp. 564-574, 2020. (in Rus)

[8] L. Zhao, S.A. Roberts, "An Object-Oriented Data Model for Database Modelling, Implementation and Access," *The Computer Journal*, vol. 31, no. 2, pp. 116-124, 1988.

[9] D. Harrington, *Designing Object-Oriented Databases*, Litres, 2022.

About Authors

S. E. Dukhovenskiy, graduate student, MIREA – Russian Technological University, Moscow, Russia (dukhovenskiy.s.e@edu.mirea.ru).