Моделирование и анализ информационных систем. Т. 23, № 2 (2016), с. 93–118
Modeling and Analysis of Information Systems. Vol. 23, No 2 (2016), pp. 93–118

DOI: 10.18255/1818-1015-2016-2-93-118

УДК 004.652.6

Построение хранилища данных с динамической структурой

Артамонов Ю. Н.

получена 24 февраля 2016

Аннотация. В данной работе проведен анализ подходов к построению хранилищ данных на основе реляционных и NoSQL решений, указаны ограничения реляционного подхода для интеллектуального анализа данных. Выявлено противоречие между представлением данных в реальной предметной области и моделями представления данных в реляционном и NoSQL подходах. Выявленное противоречие связано с темпоральностью не только значений отдельных атрибутов данных, но и изменчивостью состава этих атрибутов, а также структуры связей между ними. Предложена новая логическая модель хранилища данных с динамической структурой. В основу модели положено понятие объекта как своеобразного контейнера для хранения свойств. Каждое свойство объекта включает в себя имя свойства, а также два типа значений свойства – бессылочное и ссылочное, актуальных на заданный момент времени. Ссылочное значение свойства указывает на объект, имя которого интерпретируется как значение этого свойства на заданный момент времени. Дано формальное описание модели с выделением необходимого функционала по манипулированию объектами и их свойствами (селекторы, предикаты, конструкторы), введены необходимые управляющие конструкции. Дано обоснование предложенной модели, названной OP-model, на основе проведения соответствия с логической ER моделью данных. Доказано, что любая ER модель данных может быть реализована в OP-model. В то же время указаны преимущества OP-model, связанные с возможностью изменения связей между сущностями за счет изменения ссылочных значений на определенный момент времени, отмечены потенциальные возможности по масштабируемости хранилища данных за счет уникальной идентификации каждого объекта.

Ключевые слова: NoSQL, Big Data, ER модель, базы данных, СУБД

Для цитирования: Артамонов Ю. Н., "Построение хранилища данных с динамической структурой", *Моделирование и анализ информационных систем*, **23**:2 (2016), 93–118.

Об авторах:
Артамонов Юрий Николаевич, orcid.org/0002-0007-1896-0951, канд. техн. наук,
Федеральное государственное бюждетное научное учреждение «Госметодцентр»,
ул. Люсиновская, 51, г. Москва, 117997 Россия, e-mail: junaart@mail.ru

Благодарности:
Работа выполнена в рамках государственного задания Министерства образования и науки России по проекту №2.87.2016/НМ

1. Постановка проблемы

В настоящее время разработаны и используются самые разнообразные подходы к построению хранилищ данных [1–4]. Самое широкое распространение получил реляционный подход. Однако в ряде случаев приходится сталкиваться с ограничениями

реляционных баз данных. Особенно остро эти ограничения дают о себе знать при обработке очень больших объемов данных или при очень большом потоке обращений к базе данных. В этом случае приходится решать сложные вопросы масштабирования реляционных БД, а в наиболее проблемных ситуациях вообще отказываться от реляционных решений в пользу NoSQL [5, 9]. В NoSQL за счет отсутствия строгой структуры данных и минимума связей между неделимыми частями данных (кортежами ключ-значение) существенно проще добиться их распределения по разным кластерам (масштабируемость – partition tolerance), а также существенно проще получить доступ к этим данным в любой момент времени (доступность – availability). Для объемов информации, а также интенсивности запросов к ней, характерных для современных поисковых систем, NoSQL решения оказываются основной альтернативой реляционному подходу. Однако, как неоднократно отмечено [6, 7, 10], такой подход не обеспечивает целостности данных (консистентность – consictency), поскольку здесь мы попадаем под действие CAP теоремы [6–8]: одновременно можно выполнить только любые два из трех требований (partition tolerance, availability, consictency). В условиях этой теоремы подход NoSQL в большинстве случаев – это AP-системы (Cassandra, CouchDB) [13], реляционный подход – CA-системы. Остается также альтернатива строить CP-системы – такие системы беззатратно масштабируются, а в случае потери согласованности данных становятся недоступными до момента их согласования (MongoDB, HBase, Redis).

Кроме отмеченных ограничений, следует также указать на противоречие между требованиями к доступности в любой момент времени и запросами на интеллектуальную обработку этих данных (Data Mining). Основные приоритеты AP, CA-систем – это высокая производительность (минимальное время отклика) с отсутствием избыточности данных для CA-систем (данные нормализованы) или минимумом этой избыточности для AP-систем (избыточность лишь для обеспечения доступности). В то же время приоритетными требованиями интеллектуального анализа данных являются: гибкость представления и обработки данных, оперативное получение агрегированных данных, накопление данных за продолжительный интервал времени с возможностью их одновременного анализа, допустимость избыточных данных (хранение как детализированных, так и агрегированных данных), высокая пиковая нагрузка при использовании сложных алгоритмов анализа, допустимость среднего времени отклика. Действительно пользователь системы Data Mining вынужден мириться с приемлемыми затратами времени на получение необходимого результата, но останется не удовлетворен ограниченными возможностями анализа или ограниченного доступа лишь к сегменту данных. Таким образом, системы хранения данных для их интеллектуальной обработки следует отнести к CP-системам: масштабирование необходимо для хранения и обработки больших данных (Big Data), консистентность – для получения непротиворечивых результатов анализа. Однако на текущий момент достаточно отработанными следует признать лишь технологии интеллектуального анализа данных для структурированных данных (реляционный подход), выделяя следующие классы систем: OLTP-системы (структурированная информация в рамках реляционной модели); OLAP-системы (надстройка над OLTP [1], расширяющая возможности агрегирования данных построением витрин данных, многомерных кубов данных), Data-Mining-системы (как правило, некоторая надстройка над OLAP, реализующая алгоритмы кластеризации, классификации

и т.д.). Показательным примером в данном случае являются продукты компании Pentaho: СУБД (MySQL, PostGreSQL, MS SQL и др.) – Mondrian (OLAP-система) – Weka (Data-Mining-система) [20, 21]. В то же время предпринимаются попытки распространения OLAP-технологий и для NoSQL систем. Основным ограничением в этих направлениях, по мнению автора, становится оторванность структуры представления данных в реляционном подходе от реальной модели данных, используемой в предметной области, а вслед за ней и в программных приложениях (Impedance Mismatch) [23,24,26], причем данное ограничение наследуется OLAP технологиями. Как следствие этого явления, в OLAP приходится работать с разряженными кубами, сами кубы носят временный характер (не живут долго), формируются под каждую задачу, имеются сложности с сохранением агрегированных результатов обратно в базу данных и их повторного использования. Отметим, что для NoSQL решений также можно увидеть несогласованность модели представления данных в предметной области (где данные вступают во всевозможные отношения) с упрощениями модели ключ-значение, колонка и т.д., которые, однако, отчасти могут компенсироваться гибкостью и динамичностью создаваемой структуры.

Проведем детальный анализ потери соответствия между моделью представления данных в предметной области и моделями в реляционном и NoSQL подходах. Вначале выделим основные свойства данных в реальной предметной области, а затем рассмотрим, как эти свойства отображаются в реляционном и NoSQL подходах. Априори на достаточно абстрактном уровне можно выделить следующие наиболее важные в рамках нашего анализа свойства данных в большинстве предметных областей: структурированность данных (данные образуют различные структуры в соответствии с теми отношениями, в которые они вступают), темпоральность данных (данные меняются со временем), темпоральность структуры данных (меняются не только значения, но и связи между данными). Рассмотрим эти свойства на примере данных, представленных в таблице 1. Даны множества: A – множество различных федеральных округов; B – множество различных регионов; C – множество различных городов; D – множество различных типов организаций; E – множество различных вузов. Между элементами этих множеств введем бинарное отношение R – «включает». Например, имеем: $R(\text{ФО}_1, \text{Регион}_1), R(\text{Бюджетное}, \text{ВУЗ}_1)$ и т.д. Ясно, что любое отношение между элементами указанных множеств можно представить в виде отношения «включает».

Таблица 1. Объединение данных в таблицу
Table 1. Uniting data into the table

Федеральный округ Federal District	**Регион Region**	**Город City**	**Тип организации Type of organization**	**Вуз University**
ФО1	Регион1	Город1	Бюджетное	Вуз1
ФО1	Регион1	Город1	Внебюджетное	Вуз2
ФО2	Регион2	Город2	Внебюджетное	Вуз3
ФО3	Регион3	Город3	Бюджетное	Вуз4

Поскольку отношение «включает» транзитивно, порождаются цепочки таких отношений, образующие древовидные структуры:

$$\text{Федеральный\_округ} \to \text{Регион} \to \text{Город} \to \text{Вуз}; \text{Тип\_организации} \to \text{Вуз}.$$

Каждый элемент такой цепочки в большинстве случаев можно рассматривать как некоторый объект в соответствующей предметной области. Иногда вместо цепочек предпочитают иметь дело со свойствами объектов. Например, говорят: объект «Вуз» имеет свойство с названием «Тип организации», свойство с названием «Город». Таким образом, любое свойство объекта потенциально может выступать самостоятельным объектом в древовидной структуре, полученной из различных значений этого свойства, при условии, что значение свойства измеряется не в количественных шкалах (номинальных, порядковых). При построении структуры базы данных приходится иметь дело с теми же цепочками отношений «включает». Пример подобной структуры представлен на рисунке 1.

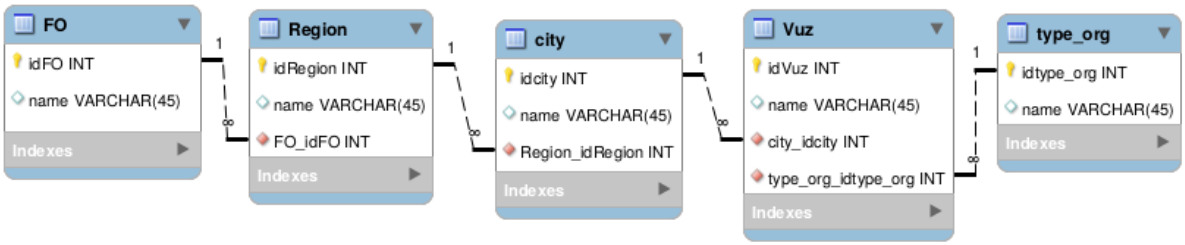

Рис. 1. Представление сущностей в реляционной базе данных
Fig 1. Presentation of the entities in a relational database

Как видно из рисунка 1, возникающие связи таблиц воспроизводят указанные выше цепочки отношений. Однако в реляционной БД после этого структура оказывается зафиксированной. В то же время, задачи, возникающие в предметных областях, приводят к постоянному дрейфу этой структуры: появляется необходимость добавления новых свойств у объектов (новых полей в соответствующие таблицы БД); выделения новых объектов (для реляционного подхода – это добавление новых таблиц БД) из значений свойств существующих объектов, которые уже получили необходимую степень уточнения. Например, в таблице «Вуз» может быть введено поле «Эффективность Вуза», которое по результатам мониторинга заполняется некоторыми значениями. После этого может появиться необходимость исследовать поле «Эффективность Вуза» как самостоятельную сущность, это потребует добавления новой таблицы в БД и соответствующей перестройки структуры. Любое действие со структурой БД усугубляется необходимостью переработки программных приложений анализа данных из БД, поскольку доработки БД автоматически не приводят к изменению функционирования программных приложений по работе с ней. Приведенные примеры требуют непротиворечивого наращивания структуры БД в согласовании с функционированием обслуживающих БД программных приложений. Однако на этом проблемы не заканчиваются. В ряде случаев в предметных областях приходится иметь дело с перестройкой структуры, т.е. изменением отношения «входит». Для реляционной базы данных такой подход означает изменение

инфологической структуры, что невозможно без существенных затрат времени на переработку как со стороны базы данных, так и со стороны обслуживающих ее программных приложений.

В NoSQL подходе следуют принципу хранить данные, как они приходят, перенося полностью или частично задачу формирования структуры данных на этап их обработки. Здесь традиционно выделяют хранилища по типу ключ-значение (key-value store), семейство колонок (bigtable store), а также документо-ориентированные БД (document store).

Системы хранения по типу ключ-значение (Redis, Reak, MemcacheDB) ориентированы в основном на производительность, доступ к данным осуществляется по ключу, часть данных, как правило, хранится в оперативной памяти [11]. Системы хранения по типу семейство колонок (Cassandra, HBase и др.) фактически индексируют строки и столбцы входной таблицы, структура фиксируется в виде совокупности столбцов, добавление новых столбцов приводит к разряженным таблицам. Анализ хранящейся информации, а также обеспечение ее целостности возлагается в основном на программные приложения, сама система реализует лишь базовые функции доступа. Поэтому говорить о соответствии модели данных в предметной области с моделью их представления в хранилище следовало бы именно для программных приложений (Hadoop, различные SQL-подобные решения под Hadoop: Hive, Impala, Shark).

Документо-ориентированные БД обычно хранят коллекции документов в некотором сериализованном формате (например, формат документов в MongoDB – это BSON), который позволяет гибко подстраивать структуру хранения документов под предметную область, изменять эту структуру в случае необходимости [12]. Однако за счет хранения в общем случае некоторой древовидной структуры осложняется доступ к необходимой информации, опять фиксируется структура данных. И хотя изменение структуры данных не становится в этом случае столь критичным для хранилища и программных приложений по работе с ним, как в реляционном подходе, все же требуется соответствующая настройка.

Следует отметить, что на потерю соответствия между моделью представления данных в предметной области и моделью в реляционном подходе указывает ряд авторов [14, 23, 24, 26]. В частности для описания предметной области в [14] рассматривается дискретная детерминированная модель (OD-модель) и отмечается, что традиционное описание такой модели осуществляется в терминах объектов и их поведения. Все это приводит авторов [14] к идее использования для моделирования нового объектно-ориентированного подхода при построении баз данных, учитывающего темпоральную природу как объектов, так и связей между ними. В цикле работ [14–19] последовательно развиваются идеи динамической информационной модели DIM, в основу которой положено представление предметной области в виде множеств объектов, свойств-атрибутов этих объектов, свойств связей объектов с другими объектами, причем на множестве объектов вводятся четыре базовых отношения: наследования, включения, истории и взаимодействия. В работах [18,19] показана как статическая полнота DIM, означающая возможность описания всех действий с фиксированной структурой OD-модели, так и динамическая полнота DIM, означающая возможность представления в DIM всех изменений во времени OD-модели. Для корректного использования DIM модели требуется приведение ее

к нормальной форме, в которой исключаются возможные противоречия, связанные с одновременным использованием на объектах отношений включения и наследования. Для исключения подобных противоречий авторы [14] вынуждены вводить ограничения определенности, однозначности, выбора, выделяя из всего множества DIM моделей DIM модели в нормальной форме (для которых соблюдаются все указанные ограничения). Отметим, что в самом описании OD-модели используются лишь понятия объекта, свойства объекта и взаимодействия объектов по времени. Понятие класса используется как один из возможных подходов (ООСУБД) к корректному представлению OD-моделей. Причем, если в реляционном подходе можно обеспечить целостность данных за счет спецификации вида связей, то в общем случае для обеспечения целостности, консистентности данных в ООСУБД возникают сложности. Кроме этого, широкий функционал по взаимодействию объектов, наследованию свойств и методов взаимодействия объектов в ООСУБД становится ограничением в масштабировании такой системы. Действительно, практика реализации NoSQL подходов в направлении масштабирования данных приводит к выводу о целесообразности наиболее простой структуры хранения данных. В то же время для ООСУБД, например, информация о структуре классов, их наследовании требует централизованного хранения. Поэтому возможные технические реализации систем, построенных на основе DIM, при успешном решении вопросов целостности данных целесообразно отнести к классу CA-систем.

На концептуальном уровне описания OD-модели можно придерживаться и другой позиции, считая первичной, наблюдаемой сущностью в предметной области признак (свойство). Именно признаки (свойства) меняются во времени. Объект или класс объектов – это производная сущность, получаемая как выделение часто встречающегося в предметной области подмножества признаков, привязанных к одному моменту времени (наблюдаемых одновременно). Объект становится лишь контейнером для хранения свойств. Каждое свойство характеризуется значением, привязанным к моменту времени. Выделяется специальное значение свойства – «пусто», также привязанное к моменту времени и означающее, что данное свойство отсутствует в этот момент времени. Объект, у которого все свойства на данный момент времени принимают значение «пусто», можно считать несуществующим в этот момент времени. Таким образом, следует говорить не о времени жизни объекта, а о динамике свойств. Как было показано, о свойствах на определенном уровне развития информационной системы часто начинают говорить как об объектах, или, наоборот, детальные характеристики некоторых объектов утрачивают интерес и их сворачивают до одного именованного свойства – имени объекта. В этом дуализме, по мнению автора, и отражается динамика связей объектов друг с другом. Для реализации идеи о наиболее простой структуре хранения данных, функционал которой минимально ограничивается заранее принятым подходом, важно моделировать именно первичные сущности OD-модели. Остальной функционал следует переносить в область надстроек над базовым функционалом.

Таким образом, на текущий момент времени сохраняется актуальность решения следующей проблемы: необходимо построить хранилище данных с динамической расширяемой структурой, которая позволяла бы проводить свое масштабирование.

2. Модель хранилища данных с динамической структурой

2.1. Концептуальное описание модели

Рассмотренные особенности реализации хранилищ данных и их ограничения, а также описанная последовательность представления данных в абстрактной предметной области позволяют предложить некоторый альтернативный подход к организации хранилищ данных, ориентированный на их интеллектуальную обработку. Данный подход сочетает в себе достоинства реляционных баз данных с их возможностью обеспечения консистентности данных в любой момент времени, а также особенности бесструктурного хранения данных, характерных в целом для NoSQL систем.

Вначале дадим концептуальное описание предлагаемого подхода. Для этого введем ряд понятий, а также опишем основные требования к функциональности.

Объект (object) – наиболее общая сущность, имеющая уникальный идентификатор (uuid_object), тип (type_object), а также имя объекта (name_object). В функциональном отношении объект представляет собой контейнер для хранения свойств.

Свойство (property) – совокупность трех атрибутов: имя свойства (name_property), значение свойства (value_property), ссылка на объект (link_object = uuid_object), чьё имя (name_object) является значением свойства. Для значения свойства (value_property) и ссылки на объект (link_object) имеется специальное значение «nil», соответствующее отсутствию значения у данного свойства.

Время (time) – имеется упорядоченное множество, каждый элемент этого множества – некоторый момент времени.

Требования к функциональности и ограничения:

1. У каждого объекта имеется множество пар, состоящих из имени объекта и момента времени, начиная с которого это имя становится актуальным. Время начала существования нового имени эквивалентно завершению существования старого имени.

2. Каждое свойство является слотом связи объекта, которому принадлежит это свойство, с другим объектом, который при необходимости может быть выделен из свойства. При выделении нового объекта B из свойства объекта A, name_property объекта A становится типом объекта B (type_object), а value_property объекта A – именем объекта B, причем в link_object объекта A заносится uuid_object объекта B.

3. Каждое значение свойства value_property – v_i, link_object – l_i относится к некоторому моменту времени t_i (time_value), который помечает эти свойства. В кортеже $\langle v_i, l_i, t_i \rangle$ каждый момент времени t_i соответствует началу актуальности указанных значений свойства v_i, l_i.

4. Свойство, для которого на данный момент времени соответствующее значение равно nil, считается несуществующим у объекта на данный момент времени.

5. Любое свойство имеет уникальный идентификатор uuid_property. У любого объекта может быть только одно одноименное название свойства.

6. Под переименованием имени свойства будем понимать одновременное прекращение существования свойства со старым именем и начало существование нового свойства с новым именем.

Базовый функционал должен быть расширен в соответствии с потребностями интеллектуального анализа данных. Однако здесь мы подробно остановимся именно на базовом функционале предложенной модели.

2.2. Формальное описание модели (базовый функционал)

Проведем формализацию предложенной концептуальной модели.

Определение 1. *Каждый объект представим упорядоченной четверкой:*

$$O = \langle i_o, t_o, P, N \rangle,$$

где i_o – уникальный идентификатор объекта O (uuid_ object); t_o – тип объекта O (type_ object); P – множество свойств объекта O: $P = \{p_i\}$; $\forall k : N = \{\langle n_{ok}, d_{ok} \rangle\}$ – множество кортежей из двух элементов, n_{ok} – имя объекта («name_ object»), d_{ok} – момент времени, который помечает имя объекта, на множестве N введено отношение линейного порядка $\prec$, такое, что

$$\forall k_1, k_2 : d_{ok_1} < d_{ok_2} \Leftrightarrow \langle n_{ok_1}, d_{ok_1} \rangle \prec \langle n_{ok_2}, d_{ok_2} \rangle$$

В дальнейшем, кроме математической нотации, будем предлагать также обозначения, используемые при технической реализации языка по манипулированию объектами и их свойствами. Так объект будем представлять списком:

$$[\text{uuid}, \text{type}, \text{Property}, \text{Name}]$$

Множество имен объекта будем также представлять списком пар:

$$\text{Name} = [[name_1, time_1], \ldots [name_n, time_n]]$$

Определение 2. *Каждое свойство объекта (каждый элемент множества P) представляет собой упорядоченный набор элементов:*

$$\forall i : p_i = \langle i_p, n, V \rangle,$$

где i_p – уникальный идентификатор свойства объекта (uuid_ property); n – название свойства (name_ property);$V = \{\langle v, l, d \rangle\}$ – множество кортежей из значений свойства для разных моментов времени, v – бессылочное значение свойства (value_ property); l – ссылка на объект, имя которого является значением свойства (link_ object), $l \in \{i_o\}$, v, l могут принимать пустое значение nil; d – момент времени (time_ value), который помечает набор $\langle v, l \rangle$; на множестве V введено отношение линейного порядка $\prec$, такое, что

$$\forall k_1, k_2 : d_{k_1} < d_{k_2} \Leftrightarrow \langle v_{k_1}, l_{k_1}, d_{k_1} \rangle \prec \langle v_{k_2}, l_{k_2}, d_{k_2} \rangle$$

Для технической реализации также введем списковую нотацию:

$$\text{Property} = [\text{Property}[0], \ldots, \text{Property}[k]]$$

$$\text{Property}[i] = [\text{uuid}_i, \text{name}_i, \text{Value}_i]$$

$$\text{Value} = [[value_1, link_1, time_1] \ldots [value_m, link_m, time_m]]$$

Пусть j-й объект представлен в виде:

$$O_j = \langle i_{oj}, t_{oj}, P_j, N_j \rangle,$$

где $P_j = \{p_k | \forall k : p_k = \langle i_{pk}, n_k, \{\langle v_n, l_n, d_n \rangle\}_k \rangle\}$, $N_j = \{\langle n_{ok}, d_{ok} \rangle\}$.

Введем функции локализации характеристик объекта, а также функции проверки свойств объекта (*селекторы и предикаты*).

1. Функция получения уникального идентификатора объекта:

   $$U_o(O_j) = i_{oj},$$

   а также обратная функция получения объекта по уникальному идентификатору:

   $$\overline{U}_o(i_{oj}) = O_j,$$

   если по данному идентификатору объект не найден, то $\overline{U}_o(i_{oj}) = nil$, т.е. данная функция является одновременно предикатом проверки наличия заданного объекта.

   В технической реализации используем точечную нотацию: $O.Uuid = O[0]$ – выдает уникальный идентификатор объекта; $Object.Uuid(value)$ – выдает объект по уникальному идентификатору value.

2. Функция получения типа объекта:

   $$T_o(O_j) = t_{oj},$$

   а также обратная функция получения множества объектов заданного типа:

   $$\overline{T}_o(t_{oj}) = \{O_k | \forall k : T_o(O_k) = t_{oj}\},$$

   если объекты данного типа отсутствуют, то $\overline{T}_o(t_{oj}) = nil$, т.е. данная функция является одновременно предикатом проверки наличия объектов заданного типа.

   Техническая реализация: $O.Type = O[1]$ – выдает тип объекта; $Object.Type(name)$ – выдает список объектов заданного типа с именем name.

3. Функция получения множества кортежей имен объекта:

   $$N_o(Q_j) = N_j.$$

   Техническая реализация: $O.Name = O[3]$ – выдает список имен объекта.

4. Функция получения имени объекта, актуального на заданный момент времени t:
$$N_{ot}(O_j, t) = n_o \Leftrightarrow \langle n_o, \max(\{d_{ok} | \forall k : d_{ok} \leq t\})\rangle \in N_j,$$
а также функция получения моментов времени для заданного имени объекта n_o:
$$\overline{N}_{ot}(O_j, n_o) = \{d_{ok} | \forall k : \langle n_o, d_{ok}\rangle \in N_j\}.$$
Техническая реализация: $O.TimeToName(t)$ - выдает имя объекта, актуального на момент времени t; $O.NameToTime(name)$ - выдает список моментов времени для заданного имени.

5. Функция получения множества свойств объекта:
$$P_o(Q_j) = P_j.$$
Техническая реализация: $O.Property = O[2]$ – выдает список свойств объекта.

6. Функция получения множества уникальных идентификаторов свойств объекта:
$$P_{ip}(Q_j) = \{i_{pk} | \forall k : p_k = \langle i_{pk}, n_{pk}, V\rangle \in P_j\},$$
если у объекта нет свойств, то $P_{ip}(Q_j) = nil$, т.е. данная функция одновременно является предикатом проверки наличия у объекта свойств.
Техническая реализация: $O.Property.Uuid$ – выдает список всех уникальных идентификаторов свойств объекта.

7. Функция получения множества имен свойств объекта:
$$P_{np}(Q_j) = \{n_k | \forall k : p_k = \langle i_{pk}, n_k, V\rangle \in P_j\}.$$
Техническая реализация: $O.Property.Names$ - выдает для всех свойств объекта список их имен.

8. Функция получения имени свойства по уникальному идентификатору свойства:
$$N_p(i_p) = n_{pk} \Leftrightarrow \langle i_p, n_{pk}, V\rangle \in P_j.$$
Техническая реализация: $O.Property.Name(uuid)$.

9. Функция получения множества значений свойства по уникальному идентификатору свойства:
$$V(i_p) = V.$$
Техническая реализация: $Property.Value(uuid)$.

10. Функция получения множества уникальных идентификаторов свойств с заданным именем:
$$I_p(n_p) = \{i_{pk} | \forall k : P_k = \langle i_{pk}, n_p, V\rangle\},$$
если свойств с таким именем не найдено, то $I_p(n_p) = nil$, т.е. данная функция одновременно является предикатом наличия свойств с заданным именем.
Техническая реализация: $Property.Uuid(name)$.

11. Функция получения уникального идентификатора объекта по уникальному идентификатору его свойства:

$$I_{op}(i_p) = i_o \Leftrightarrow i_p \in P_{ip}(\overline{U}_o(i_o)).$$

Техническая реализация: $Property.object(uuid)$.

12. Функция получения уникальных идентификаторов свойств объекта, актуальных на заданный момент времени:

$$P_{ot}(O_j, t) = \{i_{pk} | \forall k : \langle i_{pk}, n_k, \{\langle v_n, l_n, \max(\{d_l | \forall l : d_l \leq t\})\rangle |$$
$$v_n \neq \text{nil} \vee l_n \neq \text{nil}\}\rangle \in P_j\}.$$

Техническая реализация: $O.Property.Uuid(t)$.

13. Функция получения множества всех моментов времени для заданного свойства:

$$D_p(i_p) = \{d_l | \forall l : p_k = \langle i_p, n_{pk}, \{\langle v_{pk}, l_{pk}, d_l\rangle\}\rangle\}.$$

Техническая реализация: $Property.Time(uuid)$.

14. Функция получения множества всех бессылочных значений заданного свойства:

$$V_p(i_p) = \{v_l | \forall l : p_k = \langle i_p, n_{pk}, \{\langle v_l, l_{pk}, d_{pk}\rangle\}\rangle\}.$$

Техническая реализация: $Property.Value.Values(uuid)$.

15. Функция получения множества всех ссылочных значений заданного свойства:

$$L_p(i_p) = \{l_l | \forall l : p_k = \langle i_p, n_{pk}, \{\langle v_{pk}, l_l, d_{pk}\rangle\}\rangle\}.$$

Техническая реализация: $Property.Value.Links(uuid)$.

16. Функция получения бессылочного значения свойства на заданный момент времени:

$$V_{pt}(i_p, t) = v \Leftrightarrow p_k = \langle i_p, n_{pk}, \{\langle v, l_{pk}, \max(\{d_l | \forall l : d_l \leq t\})\rangle\}\rangle,$$

если $V_{pt}(i_p, t) = nil$, то это означает, что у данного свойства на текущий момент времени отсутствует бессылочное значение, таким образом, данная функция одновременно является предикатом проверки наличия бессылочного значения у заданного свойства на заданный момент времени.

Техническая реализация: $Property.Value.Values.Time(uuid, t)$.

17. Функция получения ссылочного значения свойства на заданный момент времени:

$$L_{pt}(i_p, t) = l \Leftrightarrow p_k = \langle i_p, n_{pk}, \{\langle v_{pk}, l, \max(\{d_l | \forall l : d_l \leq t\})\rangle\}\rangle,$$

если $L_{pt}(i_p, t) = nil$, то это означает, что у данного свойства на текущий момент времени отсутствует ссылочное значение, таким образом, данная функция одновременно является предикатом проверки наличия ссылочного значения у заданного свойства на заданный момент времени.

Техническая реализация: $Property.Value.Links.Time(uuid, t)$.

Введем функции добавления и изменения объектов (*конструкторы*). Заметим, что изменение объектов приводит к изменению их контекста, которое невозможно без побочных эффектов – операторов присваивания (обозначим такой оператор символом <-).

1. Функция создания нового объекта Q_k:

$$A_o(i_o, t_o, n_o, d_o) = \langle i_o, t_o, \varnothing, \{\langle n_o, d_o\rangle\}\rangle$$

$$Q_j\text{<-}A_o(i_o, t_o, n_o, d_o).$$

Техническая реализация: $Object.Create(uuid, type, name, time)$.

2. Функция добавления нового свойства в заданный объект:

$$A_p(O_j, i_p, n_p) = \langle U_o(Q_j), T_o(Q_j), P_o(Q_j) \cup \{\langle i_p, n_p, \varnothing\rangle\}, N_o(Q_j)\rangle$$

$$Q_j\text{<-}A_p(O_j, i_p, n_p).$$

Техническая реализация: $O.Property.Add(uuid, name)$.

3. Функция изменения имени объекта на определенный момент времени:

$$CN_o(Q_j, n_o, d) = \langle U_o(Q_j), T_o(Q_j), P_o(Q_j), N_o(Q_j) \cup \{\langle n_o, d\rangle\}\rangle$$

$$Q_j\text{<-}CN_o(Q_j, n_o, d)$$

Техническая реализация: $O.Name.Change(name, time)$.

4. Функция изменения значения бессылочного свойства на определенный момент времени:

$$CPV(Q_j, i_p, v_p, d_p) = \langle U_o(Q_j), T_o(Q_j), (P_o(Q_j) \setminus \{\langle i_p, N_p(i_p), V(i_p)\rangle\})$$

$$\cup\{\langle i_p, N_p(i_p), V(i_p) \cup \{\langle v_p, L_{pt}(i_p, d_p), d_p\rangle\}\rangle\}, N_o(Q_j)\rangle$$

$$Q_j\text{<-}CPV(Q_j, i_p, v_p, d_p).$$

Техническая реализация: $Property.Value.Change(uuid, value, time)$.

5. Функция изменения значения ссылочного свойства на определенный момент времени:

$$CPL(Q_j, i_p, l_p, d_p) = \langle U_o(Q_j), T_o(Q_j), (P_o(Q_j) \setminus \{\langle i_p, N_p(i_p), V(i_p)\rangle\})$$

$$\cup\{\langle i_p, N_p(i_p), V(i_p) \cup \{\langle V_{pt}(i_p, d_p), l_p, d_p\rangle\}\rangle\}, N_o(Q_j)\rangle$$

$$Q_j\text{<-}CPL(Q_j, i_p, v_p, d_p).$$

Техническая реализация: $Property.Link.Change(object, uuid, value, time)$.

6. Функция выделения нового объекта Q_k из заданного на определенный момент времени t свойства i_p:

$$E_o(Q_j, i_p, t, d_o) = A_o(i_o, N_p(i_p), V_{pt}(i_p, t), d_o).$$

$$Q_k\text{<-}E_o(Q_j, i_p, t, d_o)$$

Техническая реализация: $Allocation(object, uuid, time1, time2)$.

Заметим, что использование предложенных функций без операторов присваивания не создает побочных эффектов, что соответствует функциональной парадигме программирования. Это может оказаться важным достоинством при построении объектов на лету в интеллектуальном анализе данных. Для манипулирования объектами и свойствами базовый функционал целесообразно расширить рядом полезных общепринятых управляющих конструкций [22], сохраняя при этом функциональный подход. Вместе с введенными селекторами, предикатами и конструкторами весь этот функционал можно рассматривать как язык по манипулированию данными хранилища на нижнем уровне. В дальнейшем можно создавать высокоуровневые абстракции над операторами данного языка, расширяя базовый функционал предложенной модели.

1. Предикат проверки наличия элемента в множестве:

$$a \notin A \Rightarrow mem(a, A) = nil,$$

$$a \in A \Rightarrow mem(a, A) = t$$

Техническая реализация: $mem(x, Set)$.

2. Функция редукции множества:

$$red(L) = \{l_i | \forall l_i \in L, l_i \neq nil\}$$

Техническая реализация: $red(Set)$.

3. Отображающий функционал:

$$map(f(x_1, x_2, \ldots, x_n), A_1 = \{a_{11}, a_{12}, \ldots, a_{1m}\}, A_2 = \{a_{21}, a_{22}, \ldots, a_{2m}\}, \ldots$$
$$A_n = \{a_{n1}, a_{n2}, \ldots, a_{nm}\}) = \{f(a_{11}, a_{21}, \ldots, a_{n1}), f(a_{12}, a_{22}, \ldots, a_{n2}), \ldots,$$
$$f(a_{1m}, a_{2m}, \ldots, a_{nm})\}$$

Техническая реализация: $map([x1, .., xn], f(x1, ..., xn), A1, ..., An)$ – указываем имена переменных, используемых в отображении, отображаемую функцию, а также множества, откуда переменные принимают значения.

4. Блочный оператор:

$$block(command_1; command_2; \ldots; command_n),$$

где $command_1, command_2, \ldots, command_n$ – блок команд (в дальнейшем $block$), выполняемых последовательно. Оператор возвращает результат последней команды $command_n$.

Техническая реализация: $block(command1; ...; command_n)$.

5. Оператор локального контекста:

$$let(command_1; command_2; \ldots; command_n),$$

аналогично блочному оператору – перечень команд, выполняемых последовательно с возможным присваиванием переменным локальных значений, видимых только внутри $let()$. Оператор возвращает результат последней команды $command_n$.

Техническая реализация: $let(command1; ...; command_n)$.

6. Условный оператор:

$$condition \neq nil \Rightarrow if(condition, block_1, block_2) = block_1$$

$$condition = nil \Rightarrow if(condition, block_1, block_2) = block_2$$

Техническая реализация: $if(condition, \{block1\}, \{block2\})$.

Продемонстрируем использование введенных управляющих конструкций.

1. Проверка наличия объекта заданного типа «type» с заданным именем «name» на данный момент времени «d»:

$$Test_o(type, name, d) = mem(name, map(N_{ot}(x, d), \overline{T}_o(type))).$$

2. Получение уникального идентификатора свойства объекта Q по имени его свойства «name»:

$$Get_{name}(Q, name) = red(map(if(mem(name, \{N_p(x)\}), x, nil), P_{ip}(Q))).$$

3. Выделение объектов из некоторого свойства для всех объектов, которые имеют это свойство, также, кроме создания объектов, обновляется ссылочное значение свойства объектов, от которых были созданы новые объекты (предполагаем, что заданное свойство имеет название «name», также осуществляется проверка - если объект такого типа с заданным на данный момент времени «d» именем уже существует, то он не создается):

$$G(name, d) = map(if(Test_o(name, V_{pt}(Get_{name}(x, name), d), d), nil,$$
$$x\text{<-}CPL(x, Get_{name}(x, name), U_o(E_o(x, Get_{name}(x, name), d, d)), d)).$$
$$map(\overline{U}_o(x), (map(I_{op}(x), I_p(name)))))$$

Прокомментируем составляющие функции $G(name, d)$:

$I_p(name)$ – получаем множество уникальных идентификаторов свойств с заданным именем;

$map(I_{op}(x), I_p(name))$ – получаем множество уникальных идентификаторов объектов, у которых имеется свойство с именем «name»;

$A = map(\overline{U}_o(x), (map(I_{op}(x), I_p(name))))$ – получаем множество объектов, у которых имеется свойство с именем «name»;

$V_{pt}(Get_{name}(x, name), d)$ – получаем значение свойства с именем «name» у объекта $x \in A$ на заданный момент времени d;

$Test_o(name, V_{pt}(Get_{name}(x, name), d), d)$ – проверяем, есть ли объекты типа «name» с именем, соответствующим значению свойства «name» у объекта $x \in A$ на заданный момент времени d, если это условие не выполнено, то выполняется блок кода по созданию нового объекта и модификации ссылочного значения у объекта x:

$$x\text{<-}CPL(x, Get_{name}(x, name), U_o(E_o(x, Get_{name}(x, name), d, d)), d),$$

где $E_o(x, Get_{name}(x, name), d, d)$ – выделяем новый объект из свойства с именем «name» объекта $x \in A$ на момент времени d;

$x\text{<-}CPL(x, Get_{name}(x, name), U_o(E_o(x, Get_{name}(x, name), d, d)), d)$ – создаем новый контекст у объекта x, в котором модифицируем ссылочное значение свойства с именем «name» объекта $x \in A$ на uuid созданного с помощью $E_o(x, Get_{name}(x, name), d, d)$ объекта.

Для удобства в дальнейшем будем ссылаться на описанную модель хранилища данных как на OP-model (object-property-model).

3. Обоснование некоторых свойств модели

Для обоснования свойств предложенной модели покажем соответствие и имеющиеся различия OP-model и логической ER-модели данных [2, 3, 25].

Лемма 1. *Любая сущность ER-модели данных может быть реализована в OP-model.*

Доказательство. В данном случае возможны два варианта, представленные на рисунке 2:

1. Первичный ключ сущности (Entity_1) является простым. Подобный пример представлен на рисунке 2 (а). Такая сущность соответствует множеству объектов в OP-model. Каждый экземпляр сущности отображается в соответствующий объект OP-model. Все объекты этого множества имеют одинаковый тип – имя сущности. Атрибут (Attribute_1) – первичный ключ задает имя объекта, все остальные атрибуты задают свойства объекта.

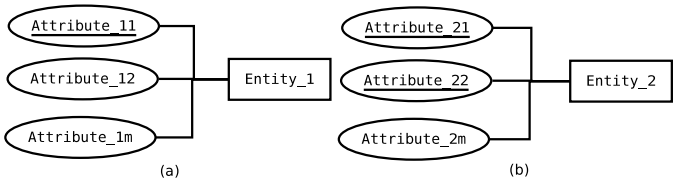

Рис. 2. Одна сущность
Fig.2 One entity

Обозначим:

$Entity_j.Name$ – имя j-й сущности (Entity_j, j=1,2), соответствует типу объекта;

$Attribute_{jk}.Name$ – имя k-го атрибута j-й сущности;

$Attribute_{jk}.Value$ – значение k-го атрибута j-й сущности;

d_1 – момент времени начала существования сущности и ее свойств;

m – количество атрибутов сущности, а также количество свойств объектов соответствующего типа;

n – количество экземпляров сущности, а также количество объектов соответствующего типа;

$uuid_{oi}$ – уникальный идентификатор i-го объекта;

$uuid_{pik}$ – уникальный идентификатор k-го свойства i-го объекта.

Тогда множество объектов OP-model строится следующим образом:

$$\{\langle uuid_{oi}, Entity_1.Name, P_i, \langle Attribute_{11}.Value, d_1\rangle\rangle_{i=1..n}\}$$

$$\forall i : P_i = \{\langle uuid_{pik}, Attribute_{1k}.Name, \{\langle Attribute_{1k}.Value, nil, d_1\rangle\}\rangle_{k=1..m,k\neq 1}\}$$

2. Имеется одна сущность (Entity_2), первичный ключ которой является составным – например, состоит из атрибутов Attribut_21,Attribute_22 (рисунок 2 (b)). В этом случае отображение строится аналогично предыдущему случаю, за исключением имени объекта. Имя объекта можно построить соединением частей составного ключа, кроме этого, каждая часть составного первичного ключа добавляется также как отдельное свойство объекта.

$$\{\langle uuid_{oi}, Entity_2.Name, P_i, \langle Attribute_{21}.Value : Attribute_{22}.Value, d_1\rangle\rangle_{i=1..n}\}$$

$$\forall i : P_i = \{\langle uuid_{pik}, Attribute_{2k}.Name, \{\langle Attribute_{2k}.Value, nil, d_1\rangle\}\rangle_{k=1..m}\}$$

Заметим, что если рассматривать статическую OP модель (значение d_1 не меняется у имени объектов и их свойств), то построенное отображение экземпляров сущностей ER модели в объекты OP модели является биективным, т.е. возможно взаимно однозначное отображение объектов OP модели в сущности и экземпляры сущностей ER модели. □

Лемма 2. *Рекурсивное отношение ER-модели данных может быть реализовано в OP-model.*

Доказательство. Возможные виды рекурсивных отношений в ER-модели показаны на рисунке 3.

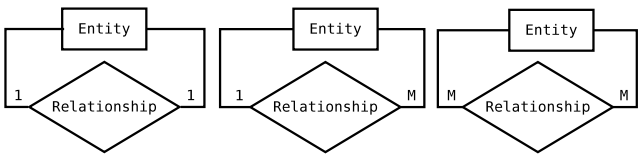

Рис. 3. Рекурсивные связи
Fig 3. Recursive links

Заметим, что в базовом функционале OP-model отсутствует механизм фиксации вида отношения – один к одному, один ко многим, многие ко многим. Данное разграничение вида отношений может быть перенесено на программные приложения по взаимодействию с хранилищем. Поэтому для доказательства достаточно рассмотреть только один вид отношений: многие ко многим, считая остальные виды частными случаями, получаемыми из этого отношения наложением определенных ограничений (как правило, для обеспечения целостности данных).

Для реализации рекурсивного отношения многие ко многим в OP-model в объекты, соответствующие экземплярам сущности, кроме свойств – атрибутов сущности, добавляется свойство с именем типа объектов. Причем ссылочное значение этого свойства указывает на uuid объекта, отвечающего структуре отношения:

$$\{\langle uuid_{oi}, Entity.Name, P_i, \langle Attribute_1.Value, d_1\rangle\rangle_{i=1..n}\}$$

$$\forall i : P_i = \{\langle uuid_{pik}, Attribute_k.Name, \{\langle Attribute_k.Value, nil, d_1\rangle\}\rangle_{k=1..m}\} \cup$$

$$\{\langle uuid_{pi(m+1)}, Entity.Name, \{\langle nil, uuid_o, d_1\rangle | T_o(\overline{U}_o(uuid_o)) = Entity.Name\}\rangle\}.$$

□

Лемма 3. *Связи сущностей ER-модели данных могут отображаться в связи объектов в OP-model.*

Доказательство. На рисунках 4, 5, 6 показаны возможные виды связей между сущностями в ER модели данных. Кроме этого, на рисунках показана модальность связей: двойная линия связи – «должен» (каждый экземпляр сущности имеет связь), одинарная линия связи – «возможно» (некоторые экземпляры сущности имеют связь). Как уже было отмечено, в базовом функционале OP-model не предусмотрено механизмов контроля вида отношений. Также в OP-model не предусмотрено специальных механизмов контроля модальности связей. Однако введение ограничений на создание новых объектов определенного типа, добавления и изменения свойств объектов позволяют реализовать все рассмотренные виды связей ER модели данных (причем реализация этих ограничений может быть вынесена за рамки базового функционала OP-model).

Реализация связей один к одному.

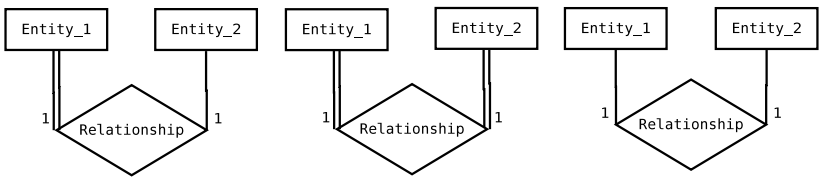

Рис. 4. Виды связей сущности один к одному
Fig 4. Types of links for Entity one to one

1. Пусть экземпляры сущности «Entity_1» при простом первичном ключе «Attribute_11» отображаются в объекты OP-model:

$$\{\langle uuid_{oi}, Entity_1.Name, P_i, \langle Attribute_{11}.Value, d_1\rangle\rangle_{i=1..n}\}$$

$$P_i = \{\langle uuid_{pk}, Attribute_{1k}.Name, \{\langle Attribute_{1k}.Value, nil, d_1\rangle\}\rangle_{k=1..m}\}.$$

Для доказательства примем соглашение, что $F(O_{ji})$ - внешняя функция, задающая наличие связи один к одному для объекта (принимает значение t при наличии связи и nil при ее отсутствии), соответствующего i-му экземпляру j-й сущности. В этом случае объекты, соответствующие экземплярам «Entity_2», можно выделить из значений первичного ключа экземпляров «Entity_1» следующим образом:

- выделить множество A объектов, типа «Entity_1.Name»: A<-$\overline{T}_0(Entity_1.Name)$;
- выделить подмножество B объектов из A, для которых внешняя функция $F(..)$ указывает наличие связи: B<-$red(map(if(F(x), x, nil), A))$;
- для каждого объекта из B создать объект типа «Entity_2.Name» со ссылкой на созданный объект в B:

$$C\text{<-}map(Get_{name}(x, Attribute_{11}.Name), B)$$

$$D\text{<-}map(A_o(gen_{uuid}, Attribute_{11}.Name, V_{pt}(x, d), d), C)$$

$$map(x_1\text{<-}x_2, B, map(CPL(x_1, x_2, x_3, d), B, C, map(U_o(x), D)))$$

Все это удобнее записать, используя введенные обозначения функций при технической реализации.

```
A <- Object.Type(Entity1.Name)
B <- red(map([x], if(F(x), x, nil), A))
C <- map([x],GetName(x, Attribute11.Name), B)
D <- map([x], Object.Create(GenUuid, Attribute11.Name,
                  Property.Value.Values.Time(x,d),d), C)
 map([x,y], block(x <- y), B,
                  map([x1,x2,x3],
                  Property.Link.Change(x1,x2,x3,d), B, C,
                        map([z], z.Uuid,D)))
```

2. Пусть экземпляры сущности «Entity_1» при составном первичном ключе «Attribute_11», «Attribute_12» отображаются в объекты OP-model:

$$\{\langle uuid_{oi}, Entity_1.Name, P_i, \langle Attribute_{11}.Value : Attribute_{12}.Value, d_1\rangle\rangle_{i=1..n}\}$$

$$P_i = \{\langle uuid_{pk}, Attribute_{1k}.Name, \{\langle Attribute_{1k}.Value, nil, d_1\rangle\}\rangle_{k=1..m}\}$$

В этом случае порядок действий может быть аналогичен предыдущему пункту, отличие состоит в том, что объекты выделяются из каждого атрибута составного первичного ключа «Entity_1»:

```
A <- Object.Type(Entity1.Name)
B <- red(map([x], if(F(x), x, nil), A))
C <- map([x],GetName(x, Attribute11.Name), B)
D <- map([x],GetName(x, Attribute12.Name), B)
E <- map([x], Object.Create(GenUuid, Attribute11.Name,
                  Property.Value.Values.Time(x,d),d), C)
F <- map([x],Object.Create(GenUuid, Attribute12.Name,
                  Property.Value.Values.Time(x,d),d), D)
```

```
map([x,y], block(x <- y), B,
                  map([x1,x2,x3],
                  Property.Link.Change(x1,x2,x3,d),
                  B, C, map([z], z.Uuid,E)))
map([x,y], block(x <- y), B,
                  map([x1,x2,x3],
                  Property.Link.Change(x1,x2,x3,d),
                  B, D, map([z], z.Uuid,F)))
```

Модальность связей: «Entity_1» – «должен»; «Entity_2» – «возможно». В дальнейшем разрешается добавлять новые объекты в множество объектов типа «Entity_2», а также запрещается добавлять объекты «Entity_1» без выделения нового объекта из значения первичного ключа и добавления его в множество «Entity_2».

Модальность связей: «Entity_1» – «должен»; «Entity_2» – «должен».

Порядок действий совпадает с предыдущим случаем. Однако после выделения объектов запрещается добавлять новые объекты в множество объектов типа «Entity_2», а также запрещается добавлять объекты «Entity_1» без выделения нового объекта из значения первичного ключа и добавления его в множество «Entity_2».

Модальность связей: «Entity_1» – «возможно»; «Entity_2» – «возможно».

Порядок действий совпадает с предыдущими случаями. После выполнения действий разрешается добавлять новые объекты в множество объектов типа «Entity_2» и «Entity_1».

Реализация связей один ко многим.

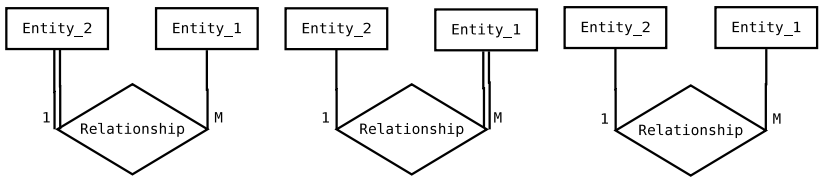

Рис. 5. Виды связей сущности один ко многим
Fig 5. Types of links for Entity one to many

Для доказательства аналогично предыдущему случаю примем соглашение о задании внешней функции $F(O_{ji})$, определяющей наличие связи для объекта (принимает значение t при наличии связи и nil при ее отсутствии). В отличие от связи один к одному в данном случае разные объекты сущности «Entity_1» могут быть связаны с одинаковым объектом сущности «Entity_2». Поэтому объекты сущности «Entity_2» должны создаваться в одном экземпляре.

1. В случае простого первичного ключа такая связь может быть реализована следующим образом (вводится вспомогательная функция $NameUuid$, которая для заданного множества объектов A и заданного имени $name$ на момент времени d формирует подмножество уникальных идентификаторов объектов, которые в данный момент времени имеют имя $name$):

```
A <- Object.Type(Entity1.Name)
B <- red(map([x], if(F(x),x,nil), A))
NameUuid(A,name,d) =
```

```
red(map([x],
        if(mem(name, x.TimeToName(d)),
        x.Uuid, nil), A))
C <- map([x], GetName(x,Attribute11.Name),B)
D <- Object.Type(Attribute11.Name)
map([x],
if(TestO(Attribute11.Name,
     Property.Value.Values.Time(x,d),d),
     map([y],
     block(Object.Uuid(Property.Object(x)) <-
               Property.Link.Change(
               Object.Uuid(Property.Object(x)),x,y,d)),
     NameUuid(D, Property.Value.Values.Time(x,d))),
     block(G <- Object.Create(GenUuid,Attribute11.Name,
               Property.Value.Values.Time(x,d),d),
               Object.Uuid(Property.Object(x)) <-
               Property.Link.Change(
               Object.Uuid(Property.Object(x)),
               x,G.Uuid,d))), C)
```

2. Для реализации отношения один ко многим с составным первичным ключом «Attribute_11.Name» и «Attribute_12.Name» порядок действий полностью совпадает с предыдущим пунктом, но указанная там процедура выполняется отдельно для каждого атрибута «Attribute_11.Name» и «Attribute_12.Name».

Для реализации модальности связей необходимо ввести следующие ограничения.

Модальность связей: «Entity_1» – «возможно»; «Entity_2» – «должен».

Разрешается добавлять новые объекты в множество объектов типа «Entity_1», а также запрещается непосредственно добавлять объекты типа «Entity_2». Объекты типа «Entity_2» можно добавлять лишь путем их выделения из объектов типа «Entity_1».

Модальность связей: «Entity_1» – «должен»; «Entity_2» – «возможно».

После выделения объектов типа «Entity_2» из свойства объектов «Entity_1» разрешается добавление новых объектов типа «Entity_2», запрещается добавление новых объектов типа «Entity_1» без выделения новых объектов типа «Entity_2» из соответствующих свойств.

Модальность связей: «Entity_1» – «возможно»; «Entity_2» – «возможно».

После выделения объектов типа «Entity_2» из свойства объектов «Entity_1» разрешается добавление как новых объектов типа «Entity_1», так и типа «Entity_2».

Реализация связей многие ко многим

Данный вид связи может быть сведен к двухстороннему использованию отношения один ко многим. Для этого в объекты типа «Entity_1» добавляются свойства, соответствующие компонентам составного первичного ключа сущности «Entity_2» (или одно свойство в случае простого первичного ключа). Аналогично в объекты типа «Entity_2» добавляются свойства, соответствующие компонентам составного первичного ключа сущности «Entity_1» (или одно свойство в случае простого первичного ключа). В дальнейшем в ссылочные значения добавленных свойств объектов типа «Entity_1», «Entity_2» в двухстороннем порядке добавляются ссылки

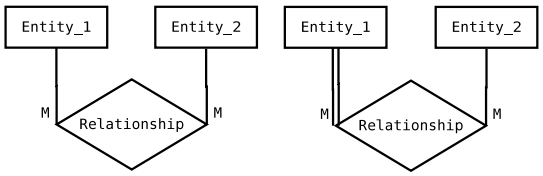

Рис. 6. Виды связей сущности многие ко многим
Fig 6. Types of links for Entity many to many

на объекты типа «Entity_2», «Entity_1» соответственно, согласно реализуемому отношению многие ко многим.

Модальность связей: «Entity_1» – «возможно»; «Entity_2» – «возможно».

Разрешается добавление новых объектов как типа «Entity_1», так и типа «Entity_2».

Модальность связей: «Entity_1» – «необходимо»; «Entity_2» – «возможно».

Разрешается добавление новых объектов типа «Entity_2», однако при добавлении новых объектов типа «Entity_1» требуется добавлять ссылочные значения соответствующих свойств в объекты типа «Entity_2» на добавленный объект типа «Entity_1».

□

Теорема 1. *Любая ER модель данных отображается в OP-model.*

Доказательство. Для доказательства используем принцип математической индукции: предположим, что n сущностей с возможными связями между ними уже реализованы в OP-model. Докажем, что, исходя из этого предположения, добавление еще одной сущности с возможными связями ее с другими сущностями из n уже существующих также реализуемо в OP-model. Возможны следующие варианты:

1. Добавление новой сущности без связей сводится к случаям, рассмотренным в лемме 1;

2. Добавление рекурсивной связи для некоторой сущности сводится к случаям, рассмотренным в лемме 2;

3. Добавление новой сущности A и ее связи с уже существующей сущностью B в отношении один к одному, один ко многим, многие ко многим сводятся к случаям, рассмотренным в лемме 3.

Таким образом, исходя из предположения реализуемости ER модели данных в OP-model для n сущностей, следует ее реализуемость для $n+1$ сущностей, а поскольку реализуемость ER модели данных в OP-model для одной или двух связанных сущностей была доказана в леммах 1, 2, 3, то тем самым доказана реализуемость ER модели данных в OP-model для любого количества сущностей и связей между ними. □

Заметим, что после отображения сущностей и связей этих сущностей в ER модели в объекты и связи между объектами в OP-model сохраняются виды связей по фактическим данным. Например, если таблица 1 связана с таблицей 2 в отношении один ко многим, но в текущем наполнении этих таблиц всем строкам таблицы 1 соответствует только одна строка таблицы 2 (фактическое наполнение пока отвечает отношению один к одному), то информация, что это была связь один ко многим

в OP-model нигде не хранится. Соответственно в дальнейшем контроль за выполнением данного ограничения в OP-model будет отсутствовать. Все это позволяет сделать вывод, что отображение связей ER модели в связи OP-model не является биективным за счет отсутствия в базовом функционале OP-model механизмов фиксации вида связей, модальности связей. По умолчанию в OP-model любая связь имеет минимальные ограничения и может превратиться со временем в связь многие ко многим с модальностью в оба направления «возможно». Аналогичная ситуация наблюдается в NoSQL подходе. Как было отмечено, возможность введения данных ограничений вынесена за рамки базового функционала OP-model и связана с решением вопросов целостности данных. Решение этих вопросов в свою очередь будет зависеть от принятого подхода к хранению данных: централизованное, децентрализованное.

Проследим также на различия между ER моделью и OP-model по возможности учета темпоральности: сущностей; атрибутов сущностей; связей между сущностями. Для этого введем требуемые определения.

Определение 3. *Темпоральность сущностей ER модели данных – это возможность изменения состава сущностей во времени.*

Определение 4. *Темпоральность атрибутов сущности ER модели данных – это возможность изменения состава атрибутов сущности во времени.*

Определение 5. *Темпоральность связей между сущностями ER модели данных – это возможность изменения вида, модальности связей между сущностями, а также возможность возникновения новых связей, исчезновения уже существующих связей во времени.*

Акцентируем внимание на требуемых сопоставлениях между ER моделью и OP-model:

- сущность ER модели (в реляционном подходе – это таблица) соответствует типу объекта в OP-model;

- атрибут сущности ER модели (в реляционном подходе – столбец таблицы) соответствует свойству объекта OP-model;

- экземпляр сущности Entity в ER модели (в реляционном подходе – строка таблицы) соответствует объекту OP-model типа Entity;

- связь между сущностями Entity 1, Entity 2 по атрибуту первичный ключ Attr1 ER модели (в реляционном подходе – связь между таблицами в отношении 1:1, 1:M, M:M) соответствует связи объектов типа Entity 1 с объектами типа Entity 2 по свойству с именем Attr1 у объектов типа Entity2.

Теорема 2. *В рамках сопоставления ER модели и OP-model в предложенной OP-model поддерживается темпоральность сущностей, атрибутов сущности ER модели данных, темпоральность связей между сущностями ER модели данных.*

Доказательство. Поддержка темпоральности на уровне сущностей ER модели данных связана с возможностью добавления новых таблиц в базу данных без связей их с уже существующими таблицами. Такая возможность может быть реализована в рамках реляционного подхода. Однако возникает рассогласование между SQL-запросами к базе данных и программными приложениями, обслуживающими ее

(Impedance Mismatch). Программные приложения приходится перерабатывать. В OP-model темпоральность на уровне сущностей ER модели поддерживается возможностью добавления объектов нового типа.

Поддержка темпоральности на уровне атрибутов сущности ER модели данных связана с возможностью создания таблиц с меняющимся во времени составом столбцов или поддержания для каждой строки таблицы своего перечня столбцов. Такая возможность очевидно отсутствует в рамках реляционного подхода. В OP-model такая возможность поддерживается:

- для исключения необходимого атрибута на заданный момент времени достаточно записать в ссылочное и бессылочное значения заданного свойства «пусто» (nil);

- для добавления необходимого атрибута на заданный момент времени d_p необходимо выполнить функцию добавления нового свойства у выбранного объекта $A_p(Q_j, i_p, n_p)$, а также добавить бессылочное значение этого свойства с помощью $CPV(Q_j, i_p, v_p, d_p)$;

- поскольку в OP-model строки таблицы хранятся отдельно в виде объектов, существует возможность поддержки для каждого объекта своего состава атрибутов-свойств.

Поддержка темпоральности на уровне связей между сущностями ER модели данных означает возможность менять тип и модальность связей между таблицами базы данных во времени, а также добавлять или исключать эти связи. Для реляционного подхода изменение типа или модальности связи приводит к нарушению целостности данных. Если такие изменения выполняются, то они приводят к необходимости существенной переработки программных приложений по работе с базой данных. Кроме того, такие изменения являются необратимыми, т.е. информация, что раньше эта связь имела другой тип или модальность, утрачивается. Добавление или исключение связей, как правило, также приводят к нарушению целостности данных и утрачиванию информации о структуре базы данных до момента изменения.

В рамках предложенной OP-model поддержка темпоральности на уровне связей между сущностями ER модели данных осуществляется за счет отсутствия ограничения на поддержания такой целостности данных в базовом функционале OP-model (как было показано, любая связь со временем может стать связью многие ко многим с модальностью «возможно» с обоих концов связи), причем за счет фиксации моментов времени в модели данных существует возможность сохранения истории о виде и модальности связи фактических данных в любой момент времени. □

Как было отмечено, реализация ER модели данных в рамках реляционного подхода приводит к проблемам масштабирования при кластерном, облачном хранении данных. В OP-model каждый объект имеет свой глобальный уникальный идентификатор (uuid). Это позволяет хранить объекты одного типа на разных компьютерах и повышает масштабируемость хранилища данных. Также следует отметить преднамеренную избыточность: значение каждого свойства может храниться в бессылочном и ссылочном виде. В соответствии с принятым протоколом можно реализовать разные схемы согласования этих значений, например, 1) всегда пытаться получать ссылочное значение, а при недоступности узла выдавать бессылочное значение, при

каждом успешном получении ссылочного значения синхронизировать бессылочное значение; 2) всегда выдавать бессылочное значение и с определенным регламентом проводить синхронизацию этого значения со ссылочным. Все это позволит повысить доступность системы.

4. Направления расширения базового функционала OP-model

Базовый функционал OP-model целесообразно расширить в направлении интеллектуального анализа данных. Для этого введем следующее понятие. *Макет* – совокупность свойств объектов, используемых для иерархического древовидного представления данных. Такое представление целесообразно строить на лету. Фактически макет реализует динамическое построение интересующих структур данных, существующих на указанный момент времени, его использование наряду с возможностью древовидной визуализации данных позволяет проводить сводную аналитику.

Для реализации макета на вход подается структура макета в виде последовательности свойств объектов в выбранном множестве. На примере таблицы 1 формируется последовательность: (<перечень свойств вуза, ссылочное/бессылочное свойство вуза – город>,<перечень свойств города, ссылочное/бессылочное свойство города – регион>, <перечень свойств региона, ссылочное/бессылочное свойство региона – федеральный округ>, <перечень свойств федерального округа>). В указанной последовательности каждый последний элемент каждого кортежа является свойством, по которому формируется структура. В этом случае итоговую древовидную структуру формируют путем последовательного уточнения – сериализации (например, в формате JSON): получают перечень всех вузов, отсортированных по значению последнего свойства – город, сериализуют объекты по последнему свойству (раскладывают вузы по городам), получают перечень свойств всех городов, отсортированных по значению последнего свойства – регион, сериализуют города по регионам и т.д. В итоге получается свернутая древовидная структура с сохранением в ней uuid соответствующих объектов и uuid их свойств. Полученная структура может быть передана внешним программным приложениям для дальнейшего анализа. Основные направления анализа здесь отчасти совпадают с задачами OLAP технологий – получение сводной аналитики, однако в отличие от OLAP технологий появляется возможность непротиворечивого развития этой структуры, в частности для каждого узла дерева, который представляет либо свойство, либо объект исходного хранилища, можно добавлять новые свойства. При реализации взаимодействия программных приложений анализа данных с сервером исходного хранилища по известным uuid объектов и их свойств новые свойства могут быть непротиворечиво добавлены в хранилище: если объект уже существовал, то новое свойство просто добавляется, если новое свойство добавлено в древовидную структуру для исходного uuid свойства, то в хранилище из этого свойства выделяется новый объект с добавлением к нему нового свойства.

В заключение отметим, что предложенную модель хранилища данных за счет ее свойства унификации структуры, учитывающей динамику данных и их связей,

можно развивать в разных направлениях, используя как традиционный реляционный подход, так и объектно-ориентированный подход, или NoSQL решения.

Список литературы / References

[1] Барсегян А. А., *Технологии анализа данных: Data Mining, Visual Mining, Text Mining, OLAP*, БХВ-Петербург, СПб, 2007; [Barsegjan A. A., *Tehnologii analiza dannyh: Data Mining, Visual Mining, Text Mining, OLAP*, BHV-Peterburg, SPb, 2007, 384 pp., (in Russian).]

[2] Дейт К.Дж., *Введение в системы баз данных*, Вильямс, М., 2001, 1072 с.; [Dejt K.Dzh., *Vvedenie v sistemy baz dannyh*, Viljams, M., 2001, (in Russian).]

[3] Мартин Дж., *Организация баз данных в вычислительных системах*, Мир, М., 1980, 665 с.; [Martin J., *Computer data-base organization*, IBM Systems Research Institute, New Jersey, 1977, 665 pp., (in Russian).]

[4] Коннолли Т., *Базы данных: проектирование, реализация и сопровождение: Теория и практика*, Вильямс, М., 2003, 1440 с.; [Konnolli T., *Bazy dannyh: proektirovanie, realizacija i soprovozhdenie: Teorija i praktika*, Viljams, M., 2003, 1440 pp., (in Russian).]

[5] *List Of NoSQL Databases*, http://nosql-database.org/.

[6] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg, "Communication-efficient leader election and consensus with limited link synchrony", *The Proceedings of the International Symposium on Principles of Distributed Computing (PODC)*, 2004, 328–337.

[7] Herlihy M. Shavit N., "The topological structure of asynchronous computability", *Journal of the ACM*, **46**:6 (1999), 858–923.

[8] Haifeng Y. Amin V., "The costs and limits of availability for replicated services", *ACM Transactions on Computer Systems*, **24**:1 (2006), 70–113.

[9] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, Ramana Yerneni, "Pnuts: Yahoo!'s hosted data serving platform", *PVLDB*, **1**:2 (2008), 1277–1288.

[10] Swati Ahirrao Rajesh Ingle, "Scalable transactions in cloud data stores", *Journal of Cloud Computing: Advances, Systems and Applications*, **4**:21 (2015), 1–14.

[11] *In-memory data structure store Redis*, http://redis.io/.

[12] *MongoDB Professional with Cloud Manager*, https://www.mongodb.org/.

[13] *A Database for the Web CouchDB*, http://couchdb.apache.org/.

[14] Писаренко Д.С., Рублев В.С., "Объектная СУБД Динамическая информационная модель DIM и ее основные концепции", *Моделирование и анализ информационных систем*, **16**:1 (2009), 62–91; [Pisarenko D.S., Rublev V.S., "Object DBMS DIM and its main concepts", *Modeling and Analysis of Information Systems*, **16**:1 (2009), 62–91, (in Russian).]

[15] Рублев В.С., "Язык объектных запросов динамической информационной модели DIM", *Моделирование и анализ информационных систем*, **17**:3 (2010), 144–161; [Rublev V.S., "The object query language of the dynamic information model DIM", *Modeling and Analysis of Information Systems*, **17**:3 (2010), 144–161, (in Russian).]

[16] Рублев В.С., "Отношение истории и динамика схем баз данных СУБД DIM", *Моделирование и анализ информационных систем*, **19**:2 (2012), 97–108; [Roublev V.S., "Evolution of DBMS DIM Database Schemes", *Modeling and Analysis of Information Systems*, **19**:2 (2012), 97–108, (in Russian).]

[17] Антонов Д.В., Рублев В.С., "Эффективность доступа к данным в СУБД DIM", *Моделирование и анализ информационных систем*, **22**:2 (2015), 158–175; [Antonov D.V., Roublev V.S., "Access Efficiency to Data in DIM DBMS", *Modeling and Analysis of Information Systems*, **22**:2 (2015), 158–175, (in Russian).]

[18] Петров А.Н., Рублев В.С., "Полнота динамики значений свойств данных в СУБД DIM", *Моделирование и анализ информационных систем*, **22**:2 (2015), 259–277. [Petrov A.N., Roublev V.S., "Completeness of the Dynamics of the Attributes Values of Data in the Database DIM", *Modeling and Analysis of Information Systems*, **22**:2 (2015), 259–277, (in Russian)].

[19] Roublev V.S., "Static completeness of the dynamic information model", *Automatic control and computer sciences*, **49**:3 (2015), 167–176.

[20] *A Comprehensive Data Integration and Business Analytics Platform*, http://www.pentaho.com/.

[21] *Data Mining Software in Java*, http://www.cs.waikato.ac.nz/ml/weka/.

[22] Doug H., *Let Over Lambda*, 2010, 384 pp.

[23] Alexandros B., "An Efficient Database Storage Structure for Large Dynamic Objects", *Proceedings, IEEE Data Engineering Conference, Phoenix, Arizona*, 1992, 301–308.

[24] Полтавцев А.А., "Динамические структуры в реляционных базах данных", *Программные продукты и системы*, **2**:110 (2015), 95–97; [Poltavtsev A.A., "Dynamic structures in relation databases", *Software & Systems*, **2**:110 (2015), 95–97, (in Russian).]

[25] Цикритзис Д., Лоховски Ф., *Модели данных*, Финансы и статистика, М., 1985, 168 с.; [Tsikritzis D., Lokhovski F., *Modeli dannykh*, Finansy i statistika, M., 1985, 168 pp., (in Russian).]

[26] Калиниченко Л.А., *Методы и средства интеграции неоднородных баз данных*, Наука, М., 1983, 424 с.; [Kalinichenko L.A., *Metody i sredstva integratsii neodnorodnykh baz dannykh*, Nauka, M., 1983, 424 pp., (in Russian).]

Artamonov Yu. N., "Building a Data Store with the Dynamic Structure", *Modeling and Analysis of Information Systems*, **23**:2 (2016), 93–118.

DOI: 10.18255/1818-1015-2016-2-93-118

Abstract. This article presents the analysis of approaches to data warehouse construction based on relational and NoSQL solutions and lists the limitations of the relational approach to data mining. The contradiction between data presentation in the real subject domain and the model of data presentation in the relational and NoSQL approaches is revealed. The revealed contradiction is related to the temporality of the values of individual data attributes, the variability of the composition of these attributes, and structure of connections between them. A new logical model of the data warehouse with dynamic structure is proposed. The model is based on the concept of the object as a container for properties storage. Each property of the object includes the property name and two property values - without reference and with reference, that are relevant at a given time. The reference property value points to an object whose name is interpreted as the value of the property at a given time. A formal description of the model with allocation of the necessary functionality to manipulate objects and their properties (selectors, predicates, constructors) is given and the necessary control structures are introduced. Substantiation of the proposed model, called an OP-model is given on the basis of compliance with the logical ER data model. It is proved that any ER data model can be implemented in the OP-model. At the same time, the advantages of the OP-model are indicated, they are associated with the possibility of changing connections between entities due to changes in the reference value at a particular time. The potential for scalability of data warehouse due to the unique identification of each object is noted.

Keywords: NoSQL, Big Data, ER model, Databases, DBMS

On the authors:
Artamonov Yuri Nikolaevich, orcid.org/0000-0002-7246-8329, PhD,
Federal state budgetary scientific institution «Gosmetodcentr»,
Lyusinovskaya str., 51, Moscow, 117997 Russia, e-mail: junaart@mail.ru

Acknowledgments: This work was supported by the state task of the Ministry of Education and Science of the Russian under the project №2.87.2016/HM