Объектно-реляционные базы данных: прошедший этап или недооцененные возможности?

С.Д. Кузнецов
kuzloc@ispras.ru

1. Введение

Эпоха объектно-реляционных баз данных началась десять лет назад, когда в декабре 1996 года компания Informix выпустила объектно-реляционную систему управления базами данных (ОРСУБД) Informix Universal Server. Вслед за ней в 1997 г. на рынке появились ОРСУБД компаний Oracle (Oracle8) и IBM (DB2 Universal Database). В течение примерно трех лет новая технология интенсивно обсуждалась. Многим (включая меня) в то время казалось, что ОРСУБД в корне изменят способы проектирования и разработки приложений баз данных.

Однако постепенно шум вокруг ОРСУБД затих. До конца 1990-х гг. Informix, Oracle и IBM совершенствовали свои ОРСУБД. В 1999 г. появился стандарт SQL:1999 [1], в котором были зафиксированы объектные расширения языка SQL. И, наконец, после выхода в 2003 г. стандарта SQL:2003 [2], уточнившего и дополнившего SQL:1999, в сообществе баз данных окончательно перестали обсуждать объектно-реляционную технологию баз данных.

Что это означает? Оказалось ли дитя мертворожденным? Или же наш мир еще не готов к приходу нового мессии? Я не знаю, многих ли в мире сейчас это волнует, но мне кажется, что тема заслуживает обсуждения. Слишком много материальных и интеллектуальных ресурсов затратило человечество на разработку ОРСУБД, чтобы можно было позволить себе о них забыть. Слишком много полезных возможностей кроется в объектно-реляционном подходе, чтобы проектировщики и разработчики приложений баз данных могли с чистой совестью им пренебрегать.

В этой статье я хочу, прежде всего, обсудить общее понятие объектно-реляционных баз данных. Затем я коротко проанализирую основные черты и различия первых версий ОРСУБД компаний Informix, Oracle и IBM. Далее мы обсудим (снова очень кратко) предпосылки появления ОРСУБД. В следующем разделе части статьи будут рассмотрены наиболее важные аспекты языка SQL, имеющие отношение к организации объектно-

реляционных баз данных и управлению ими. Моя позиция состоит в том, что на сегодняшний день в стандарте языка SQL зафиксирована некоторая законченная модель данных, во многих отношениях близкая к реляционной модели данных, но в целом существенно от нее отличающаяся. Разработчики стандарта SQL выделяют «объектную» подмодель данных. Интересным вопросом является то, насколько эта подмодель гармонична общей модели языка. Наконец, мы рассмотрим, как реально используются ОРСУБД в настоящее время, и что препятствует их более широкому использованию. В действительности, по моему мнению, имеется несколько проблем, от решения которых зависит будущее ОРСУБД. Хочется надеяться, что эти проблемы удастся решить.

2. Понятие объектно-реляционных баз данных

Термин *объектно-реляционные базы данных* стал известен широким массам разработчиков и пользователей систем баз данных после выпуска компанией Informix в конце 1996 г. своего нового продукта Informix Universal Server (IUS). Компания Informix смогла быстро разработать IUS благодаря приобретению компании Illustra и переходу на работу в Informix ее основателя Майкла Стоунбрейкера. По сути дела, IUS был получен в результате «скрещивания» основного серверного продукта Informix Dynamic Server и СУБД Illustra.

В свою очередь, компания Illustra была основана, в основном, с целью коммерциализации известной СУБД Postgres, разработанной под руководством Майкла Стоунбрейкера в Калифорнийском университете в г. Беркли. (На основе Postgres позднее была создана СУБД PostrgreSQL, являющаяся в настоящее время одной из наиболее популярных СУБД категории Open Source.) Как известно, проект Postgres во многом представлял собой продолжение с использованием ряда новых идей (развитая система правил, поддержка темпоральных данных и т.д.) раннего проекта Майкла Стоубрейкера Ingres.

Уже в Ingres можно было заметить некоторые черты, которые в наше время многими принято относить к числу объектных (в частности, зачаточные возможности определения пользователями новых типов данных). Однако Стоунбрейкер не называл ни Ingres, ни Postgres объектно-реляционными системами. По отношению к Postgres он использовал эпитет «СУБД следующего поколения». Стоунбрейкер начал называть ОРСУБД только систему Illustra, которая, по сути, отличалась от Postgres только поддержкой языка SQL.

Тем самым, исторически IUS стало возможно именовать объектно-реляционной СУБД именно по той причине, что эта система основывалась на Illustra, которая уже называлась объектно-реляционной. Однако значит ли это, что IUS можно считать некоторой эталонной моделью ОРСУБД, а Майкла Стоунбрейкера – отцом этого термина и направления в целом? С моей точки

зрения, это было бы неверно. Во-первых, как мы покажем в разд. 4, у объектно-реляционного подхода имеются и другие предпосылки. Во-вторых, в разд. 3 мы кратко продемонстрируем, что в IUS имелись некоторые особенности (большей частью, весьма полезные), которые делают эту систему не слишком пригодной в качестве эталонной модели.

Так что же такое ОРСУБД? Что такое объектно-реляционная база данных? Если не зафиксировать какое-либо конкретное понимание этих терминов, то, очевидно, рассуждения на их тему становятся бессмысленными. Заметим, что банальное житейское толкование термина ОРСУБД как традиционной реляционной СУБД, расширенной основными объектными возможностями, является опасно упрощенным и искажающим действительность. Во-первых, имеющиеся сегодня на рынке ОРСУБД не являются «традиционными реляционными», поскольку в них не поддерживается реляционная модель данных. Они основаны на другой модели данных, представленной в стандарте языка SQL. Во-вторых, объектный мир определен в целом настолько расплывчато и нечетко, что невозможно однозначно говорить об основных объектных возможностях.

Поэтому в данной статье я не буду пытаться приводить сжатые дефиниции ОРСУБД. С моей точки зрения, сегодня под ОРСУБД следует понимать системы, которые следуют духу *Манифеста систем баз данных третьего поколения* [3] и букве стандартов SQL:1999 и SQL:2003 [1-2]. В качестве примеров реализаций ОРСУБД можно использовать Oracle (начиная с Oracle 9i) и IBM DB2 (начиная с версии 7).

3. Первые реализации

В этом разделе я кратко опишу основные особенности первых СУБД компаний Informix, Oracle и IBM, которые на рынке назывались объектно-реляционными. Целью раздела является воссоздание исторического контекста, а также выделение базовых свойств, которые впоследствии были зафиксированы в стандарте SQL и являются характеристиками современных ОРСУБД. В описании я буду придерживаться хронологического порядка.

3.1. Informix Universal Server

Итак, первой системой, выпущенной на рынок в качестве РСУБД крупной софтверной компанией, стал Informix Universal Server (IUS). Это произошло в конце декабря 1996 г. – начале января 1997 г. и является основным поводом для отмечания 10-летнего юбилея подхода. IUS было посвящено много публикаций, в том числе, очень содержательные статьи [4-5]. Однако здесь я не буду следовать порядку изложения, принятому в этих статьях, а напишу так, как мне представляется правильным сегодня.

3.1.1. Интеллектуальные большие объекты

Интеллектуальные большие объекты (*smart large objects*), поддерживаемые в IUS, позволяли снять неприятные реализационные ограничения, характерные для трактовки больших объектов (BLOB или CLOB) реляционными СУБД. Интеллектуальные большие объекты могли подвергаться журнализации и откату, что важно для сохранения целостности и доступности данных.

Операторы SQL имели дело не с самими большими объектами, а с их описателями. Эти описатели помещались в столбцы таблиц, передавались подпрограммам, написанным с использованием встроенного SQL и т.д. Прикладная программа, получив описатель интеллектуального большого объекта, могла работать с ним примерно так же, как и с файлом операционной системы.

3.1.2. Определение новых базовых типов данных

СУБД IUS позволяла вводить новые базовые типы данных. При этом можно использовать встроенные в IUS методы доступа и хранения, а также определять новые.

3.1.2.1. Индивидуальные типы

В IUS можно было определить новый базовый тип данных (*индивидуальный тип данных*, *distinct type*), основанный на существующем типе, но обеспечивающий автоматическое преобразование к нужному значению (за счет явного определения соответствующих функций). В очень близкой форме средства определения индивидуальных типов позже были введены в стандарт SQL.

3.1.2.2. Типы со скрытой структурой

В соответствии с идеологией IUS, потребность во введении нового базового типа данных может возникать и по причине принципиального отсутствия нужного типа. Для этого поддерживался механизм определения *типов со скрытой структурой* (*opague type*). Типы со скрытой структурой являлись абстрактными в строгом смысле этого слова. СУБД IUS была лишена какой-либо информации о внутреннем устройстве этих типов и могла манипулировать соответствующими значениями только посредством предоставленных разработчиком функций.

Чтобы определить новый тип данных со скрытой структурой, требовалось выполнить следующую последовательность действий:

- описать на языке C (или другом внешнем языке) структуру определяемых объектов;
- написать на языке C (или другом внешнем языке) вспомогательные функции, вызываемые сервером СУБД;

- зарегистрировать определяемый тип в базе данных посредством оператора CREATE OPAQUE TYPE;
- зарегистрировать вспомогательные функции посредством операторов CREATE FUNCTION и CREATE CAST;
- предоставить права доступа к определяемому типу и его вспомогательным функциям посредством оператора GRANT;
- написать требующиеся для приложения дополнительные функции, которые можно вызывать средствами SQL, и зарегистрировать их;
- если нужно, реализовать специфические для определяемого типа вторичные методы доступа (функции для работы с индексами).

После определения нового типа со скрытой структурой его можно было использовать наравне с другими базовыми типами. Для хранения, поиска и индексирования можно было использовать стандартные механизмы IUS.

3.1.2.3. Специальные методы хранения, поиска и индексации

В IUS можно было также определять новые базовые типы данных одновременно с введением специальных алгоритмов хранения, доступа и индексирования, которые отличались от стандартных алгоритмов, реализованных в сервере. Для введения нового базового типа данных с нестандартными методами доступа нужно было определить набор серверных функций, реализующий для нового типа алгоритмы доступа, просмотра, выделения памяти и т.д. Эти функции должны были быть написаны на языке C и скомпилированы в объектный формат. Далее нужно было описать новый базовый тип данных и указать функции, реализующие для этого типа алгоритмы извлечения и записи на диск значений данного типа.

Заметим, что какая-либо поддержка типов данных со скрытой структурой в других реализациях ОРСУБД или в стандарте SQL отсутствует. Это первый пример уникальных возможностей IUS, связанных, по моему мнению, с архитектурными особенностями линии СУБД Ingres-Postgres-Illustra.

3.1.3. Составные типы данных

В IUS разрешалось определять новые составные типы данных. Составные типы могли иметь структуру записи, множества, мультимножества или списка. Структуры данных могли быть вложенными, то есть значениями столбцов таблиц могут быть записи, множества или списки, состоящие в свою очередь из атомарных или составных значений.

3.1.3.1. Тип данных со структурой записи

В SQL IUS тип данных со структурой записи определялся с использованием оператора:


```
CREATE ROW TYPE <имя типа> (
        ( <имя поля> <тип поля>, . . . );
```

Определенный таким образом составной тип можно было использовать наравне с другими типами для определения столбцов таблицы. Для доступа к отдельным полям значений типа записи использовалась традиционная точечная нотация. Кроме того, типы со структурой записи можно было использовать для определения *типизированных таблиц* с помощью оператора

```
CREATE TABLE <имя таблицы> OF TYPE <имя типа>;
```

У типизированной таблицы в IUS имелся один столбец соответствующего составного типа, но в дальнейшем работа с полями значений строк этой таблицы (с точностью до синтаксических деталей) производилась точно так же, как если бы эти поля являлись столбцами таблицы.

К типам записей и таблицам в IUS мог применяться механизм наследования. Поддерживалось только одиночное наследование, то есть число супертипов любого типа или супертаблиц любой таблицы не могло быть больше единицы, но у любого типа могло существовать произвольное число подтипов.

Все функции, определенные для супертипа (обратите внимание, что в IUS не поддерживалось понятие методов, или операций определяемого пользователем типа), автоматически распространялись на все его подтипы, т.е. были применимы к любым значениям любого подтипа этого супертипа (другими словами, поддерживалась так называемая семантика включения). В то же время, для подтипов могли определяться свои функции, как новые, дополняющие функциональность, так и модифицированные, изменяющие функциональность, унаследованную от предка. В последнем случае имело место перекрытие функций и, естественно, требовалось отложенное связывание.

Значения подтипов могли использоваться почти везде, где было разрешено употребление значений их супертипов. Единственное ограничение возникало при отведении памяти и хранении значений. Здесь соответствие типов должно было быть точным.

Наследование таблиц являлось развитием концепции наследования типов. В иерархии наследования могли участвовать только типизированные таблицы, типы которых (типы записи) образуют параллельную иерархию. Однако, кроме столбцов супертаблиц, подтаблицы наследовали ограничения целостности (первичные ключи, уникальность, ссылочные ограничения), опции хранения, триггеры, индексы и методы доступа.

При определении подтаблиц некоторые свойства супертаблиц можно было переопределять. В частности, для подтаблицы можно было определить собственную опцию хранения. Отношение наследования между таблицами являлось динамическим в том смысле, что, если изменялись наследуемые характеристики супертаблиц, то это немедленно отражалось и на подтаблицах

(как на прямых наследниках, так и отделенных несколькими уровнями иерархии).

В IUS операции SELECT, UPDATE и DELETE, примененные к супертаблице, распространяются также и на все ее подтаблицы. Если требовалось ограничить запрос ровно одной супертаблицей, следовало употребить в запросе конструкцию ONLY.

3.1.3.2. Типы коллекций

В SQL IUS имелась возможность определения трех видов типов коллекций: множеств, мультимножеств и списков (иными словами, имелись конструкторы, или генераторы этих типов). Тип множества определялся с помощью конструкции:

```
CREATE TYPE <имя типа> SET <тип элементов> NOT NULL;
```

Поскольку по определению все элементы множества должны быть различны, ни в одно значение типа множества не могло входить неопределенное значение. Тип элементов множества мог задаваться именем типа для предопределенных или ранее определенных типов данных или определяться «по месту» для множеств, состоящих из элементов типа записи.

Типы мультимножеств определялись с помощью конструкции:

```
CREATE TYPE <имя типа> MULTISET <тип элементов> NOT NULL;
```

Хотя по определению среди элементов мультимножества могут содержаться дубликаты, в IUS значения типа мультимножества, как и значения-множества не могли содержать неопределенных значений.

Значения типа множества и мультимножества можно было использовать после предиката IN в разделе WHERE оператора SELECT. К этим значениям была применима функция CARDINALITY, возвращающая число элементов. Средствами прямого SQL можно было осуществлять доступ к значениям-множествам (и мультимножествам) только как к единому целому. Доступ к элементам множества был возможен только при использовании встраиваемого SQL IUS или языка определения хранимых процедур (SPL). Основная идея состояла в том, что множество представлялось в виде производной таблицы, каждая строка которой содержала один элемент множества. Любую операцию над множествами можно было выразить в терминах реляционных операций над производными таблицами.

Список в IUS являлся упорядоченным набором элементов, в котором допускалось наличие дубликатов. Список отличается от мультимножества тем, что для каждого элемента списка имеется порядковый номер (нумерация начинается с единицы). Определение типа списка выглядит следующим образом:

```
CREATE TYPE <имя типа> LIST <тип элементов> NOT NULL;
```


В значениях-списках, как и в значениях-множествах (и мультимножествах), запрещалось наличие элементов с неопределенным значением. Доступ к элементам значений-списков столбцов таблицы базы данных, как и к значениям-множествам и мультимножествам, обеспечивался только при использовании встраиваемого SQL или SPL.

3.1.4. Модули DataBlade

Модуль DataBlade – это функционально законченное расширение сервера IUS, рассчитанное на определенное приложение или класс приложений. С модулями DataBlade не были связаны какие-то особые возможности сервера или языковые конструкции. Такие модули строились из компонентов, описанных выше, то есть из определяемых разработчиком подпрограмм, типов и структур данных, методов доступа и других объектов, хранящихся в базах данных. Создание и сопровождение модулей DataBlade в IUS поддерживалось путем предоставления соответствующего инструментария – среды разработки, «упаковщика» модулей, утилиты регистрации, прикладного программного интерфейса.

3.2. Oracle8

Компания Oracle выпустила свою первую ОРСУБД Oracle8 в июне 1997 г. Эта система во многом базировалась на возможностях СУБД Oracle 7.3, которая, по моему мнению, была одним из наиболее удачных продуктов компании Oracle. Существенные дополнения к объектно-реляционным возможностям Oracle8 появились лишь в системе Oracle 9i (2002 г.). Здесь мы ограничимся обсуждением свойств Oracle8. Заметим, что Oracle8 было посвящено не слишком много публикаций, доступных на русском языке. Пожалуй, можно рекомендовать только [6].

3.2.1. Объектные типы и объектные таблицы

Объектные типы в Oracle8 являлись аналогом типа записи в IUS. Объектный тип данных определялся в следующем синтаксисе:

```
CREATE TYPE <имя типа> AS OBJECT
       ( <имя поля> <тип поля>, . . . );
```

Как и в IUS, для доступа к отдельным полям значений объектных типов использовалась точечная нотация.

3.2.1.1. Методы объектных типов

В Oracle 8i при определении объектного типа, помимо спецификации структуры значений этого типа, можно было определить и набор *методов* данного типа. Методы представляли собой функции, написанные на PL/SQL, Java, C или другом языке и сохраненные в базе данных или вне ее (при наличии регистрации в базе данных).

Методы объектного типа разбивались на три категории: методы-члены, статические методы и методы сравнения.

Методы-члены вызывались в нотации:

```
<имя_объекта>.<имя_метода> ( список_параметров ).
```

У них имелся неявный параметр SELF, и они могли обращаться к значениям атрибутов объекта. Под термином *объект* здесь понимается строка объектной таблицы. Как правило, *именем объекта* являлось имя объектной таблицы или ее псевдонима, используемые в операторе выборки SQL.

Статические методы вызывались в нотации

```
<имя_объектного_типа>.<имя_метода> ( список_параметров )
```

и не могли обращаться к значениям атрибутов конкретных объектов.

Методы сравнения служили для сравнения экземпляров объектов. Сравнение объектов могло производиться с помощью методов вида MAP или вида ORDER. Метод типа MAP принимает в качестве параметра объект некоторого объектного типа, а возвращает значение одного из встроенных типов, которое может использоваться в операциях сравнения и сортировки. Таким образом, этот метод выполняет отображение объекта на значение одного из встроенных типов. Метод типа ORDER сравнивает два объекта и возвращает *–1*, если первый объект меньше второго, *0*, если объекты равны, и *1*, если первый объект больше второго.

Если при определении объектного типа метод сравнения не задавался, то объекты этого типа можно было сравнивать только на равенство/неравенство, причем объекты не должны были содержать элементов типа LOB. Тогда объекты считались равными в том и только в том случае, когда все их элементы не содержали неопределенных значений, и значения соответствующих элементов совпадали.

У каждого объектного типа имелся определяемый системой *метод-конструктор*, который создавал новый объект этого типа (строку типизированной таблицы) и присваивал начальные значения его атрибутам. Метод-конструктор – это функция, которая возвращает объект данного типа. Имя метода-конструктора совпадает с именем объектного типа, имена и типы параметров конструктора – с именами и типами атрибутов объектного типа.

3.2.1.2. Объектные таблицы, ссылочные типы

Объектной таблицей в Oracle8 называлась таблица, строки которой имеют объектный тип. Синтаксис определения был близок к синтаксису определения типизированных таблиц в IUS.

В Oracle8 не поддерживалось наследование (ни типов, ни таблиц), однако уже в Oracle8i (начало 1998 г.) появилась возможность наследования таблиц почти

в той же форме, как это делалось в IUS, но без поддержки параллельной иерархии наследования объектных типов.

В Oracle8 строки объектных таблиц назывались *строчными объектами* (*row objects*). Для всех строчных объектов обеспечивалась возможность уникальной идентификации. Использовалось понятие *объектного идентификатора*, и столбец любой таблицы можно было определить на встроенном типе REF. Значения этого столбца являлись своего рода указателями на строчные объекты той же самой или другой объектной таблицы.

В качестве значений типа REF использовались уникальные идентификаторы строк таблиц ROWID, которые в Oracle почти напрямую адресуют строки во внешней памяти. Следует отметить, что в Oracle (в отличие от других СУБД, в частности, DB2) исторически допускалось использование ROWID на уровне пользователя. В соответствии с идеологией Oracle, в любой таблице (в том числе, объектной) присутствует неявный столбец, содержащий ROWID каждой строки.

Считается, что REF-значение, указывающее на некоторый строчный объект, и сам этот строчный объект имеют разные, хотя и связные типы данных. Система не брала на себя ответственность за согласование ссылок; за это отвечали пользователи. В SQL Oracle8 поддерживалась встроенная функция REF, позволяющая получить значение объектного идентификатора указанной строки. Наличие в объектной таблице столбцов ссылочного типа позволяло в ряде случаев упрощать формулировку запросов (в духе путевых выражений ODMG). Кроме того, как утверждает компания Oracle, при явном использовании объектных идентификаторов (т.е. ROWID) удается повысить эффективности выполнения ряда запросов.

3.2.2. Типы коллекций в Oracle8

В Oracle8 поддерживались две разновидности типов коллекций: *табличные типы* (*table types*) и *типы массивов* (*array types*). Значениями каждого типа коллекции являлись коллекции (таблицы или массивы) элементов одного и того же типа (*типа элемента*). Тип элемента мог быть встроенным или объектным типом, но не типом коллекции.

3.2.2.1. Табличный тип

Табличный тип создавался конструкцией следующего вида:

```
CREATE TYPE <имя типа> AS TABLE OF <тип элемента>;
```

После этого можно было определить таблицу, типом одного из столбцов которой являлся данный табличный тип. В действительности для хранения табличных значений этого столбца создавалась одна дополнительная «вложенная» таблица, строки которой связывались с соответствующими строками внешней таблицы с помощью объектных идентификаторов. Такая

организация позволяла выбирать связанные данные из внешней и вложенной таблиц без использования явной операции соединения и достаточно эффективно за счет поддержки явных связей между строками.

Обратите внимание на очень специальную природу табличного типа в Oracle. Фактически, здесь неразрывно связаны логика (логично было бы считать, что значениями табличного типа являются мультимножества значений соответствующего объектного типа) и исторические особенности реализации СУБД Oracle.

3.2.2.2. Тип массива

Вторая, менее привязанная к особенностям реализации, разновидность типов коллекций в Oracle8 называлась VARRAY, что вполне соответствует стандарту SQL:1999 и означает *массив переменного размера* (*varying-length array*). При определении типа массива указываются имя типа, тип элементов и максимальное число элементов, которые может содержать значение определяемого типа. В отличие от значений табличного типа, для элементов значений типа массива поддерживается порядок. Для определения типа массива использовалась следующая синтаксическая конструкция:

```
CREATE TYPE <имя типа> AS VARRAY (<литеральное натуральное число>)
                    OF <тип элемента>;
```

Как и в случае с множествами (и мультимножествами) в IUS, на уровне прямого SQL Oracle8 можно было работать только с массивами целиком. Однако существовала возможность привести в запросе (или другом операторе SQL) тип массива к табличному типу.

Как уже говорилось, в Oracle8 отсутствовала поддержка наследования, и запрещалось определение типов коллекций с использованием ранее определенных типов коллекций. Соответствующие возможности появились только в Oracle 9i (2002 г.).

3.3. DB2 Universal Database

Компания IBM последней, после Informix и Oracle, объявила свой объектно-реляционный продукт (в конце 1998 г.) и назвала его DB2 Universal Database (UDB). Однако, как стало ясно немного позже, базовые идеи объектно-реляционных расширений были заложены еще в продукте компании IBM DB2 for Common Servers, выпущенном в 1995 г. Эти идеи описаны в статье [7]. Насколько мне известно, первой публикацией на русском языке, посвященной непосредственно DB2 Universal Database, является [8].

3.3.1. Базовые идеи объектно-реляционных расширений DB2

Традиционные возможности языка SQL, связанные с типами данных и поиском, недостаточны для нового поколения мультимедийных приложений баз данных. Требования таких приложений настолько различны, что не могут

быть удовлетворены любым набором предопределенных расширений языка. Требуются средства определения пользователями новых типов данных и функций над ними.

Другим требованием современных приложений баз данных является возможность хранения правил, делающих данные более активными и позволяющих системе баз данных автоматически выполнять проверки корректности данных и автоматизировать многие бизнес-процедуры. Активизация данных дает возможность приложениям совместно использовать не только сами данные, но и поведение данных.

Обе указанные возможности позволяют повысить ценность хранимых данных за счет расширения их семантического содержимого. Тенденция к повышению роли семантического содержимого хранимых данных является наиболее важной в современном управлении базами данных. В соответствии с этой тенденцией реляционные системы баз данных расширяются в двух направлениях:

- добавлении *объектной инфраструктуры* к самой системе баз данных в виде поддержки определяемых пользователями типов данных, функций и правил;
- построении поверх этой инфраструктуры *реляционных расширителей*, которые поддерживают специализированные приложения, такие как выборка изображений, развитый текстовый поиск, географические приложения.

Система, которая обеспечивает объектную инфраструктуру и набор реляционных расширителей, называется *объектно-реляционной*.

3.3.1.1. Объектная инфраструктура

В DB2 поддерживались три типа данных для хранения больших объектов: BLOB для бинарных объектов, CLOB для символьных строк и DBCLOB для строк, в которых используются двухбайтовые наборы символов. Для каждого из этих типов данных можно было хранить объекты объемом до 2 Гбт. Подсистема восстановления DB2 давала возможность создателю таблицы выключать обычную журнализацию изменений столбцов с большими объектами. При выключенной журнализации по отношению к таким столбцам гарантировалась согласованность транзакций, но столбец не мог участвовать в процедуре прямого восстановления.

Пользователи DB2 имели возможность создания собственных функций с использованием языков C, C++ и Basic. Вызовы *определяемых пользователями функций* (*User-Defined Functions*, UDF) могли использоваться в любом выражении SQL, где предполагалось наличие скалярного значения. В SQL DB2 можно было перегружать имя функции, т.е. определять несколько функций с одним именем и разными типами параметров. При обработке вызова функции DB2 вызывала функцию, сигнатура которой строго соответствовала типам аргументов вызова.

В DB2 определяемые пользователями типы данных назывались *индивидуальными типами* (*distinct type*). В каждом из индивидуальных типов использовалось общее представление одного из встроенных типов (называемых базовыми типами), но мог иметься собственный набор допустимых операций. При создании индивидуального типа DB2 генерировала функции, преобразующие значение индивидуального типа в значение его базового типа и наоборот.

В DB2 существовали две категории активных данных – *ограничения* и *триггеры*. Ограничения являются декларативными утверждениями, истинность которых контролируется системой. Триггеры – это автоматические действия, которые срабатывают каждый раз при возникновении определенного события или условия.

Большие объекты, определяемые пользователями типы и функции, ограничения и триггеры представляли в отдельности мощные возможности. Но истинная объектно-реляционная мощность DB2 происходила из синергии этих возможностей.

3.3.1.2. Реляционные расширители

На основе представленной выше объектной инфраструктуры можно было создавать *реляционные расширители* (*extender*) для поддержки конкретных прикладных областей. Еще до выпуска DB2 UDB поддерживались расширители Text Extender для быстрого контекстного поиска в больших текстовых документах; Image Extender для хранения и выборки изображений, представленных в разных форматах; Video Extender для хранения и выборки видео-последовательностей; Audio Extender для хранения и выборки аудио-клипов и т.д.

Все эти возможности были доступны в DB2 for Common Servers с июля 1995 г. В течение следующих двух лет в лабораториях IBM производилась интенсивная разработка дополнительных объектно-реляционных средств, включая абстрактные типы данных с наследованием; UDF с телом, написанным на SQL; UDF, результатом которых являются таблицы; возможность обращаться со строкой таблицы как с объектом, возможно содержащим ссылки на другие строки-объекты; дополнительные расширители для таких специализированных типов данных, как временные ряды и географические данные.

3.3.2. Дополнительные объектно-реляционные возможности DB2 UDB

3.3.2.1. Структура, поведение, наследование

Для некоторых типов данных имеет смысл трактовать объект как именованный набор атрибутов. Этот вид определяемого пользователями типа в терминологии IBM называется *структурным типом.* Механизм

структурных типов с самого начала включал не только возможность определения типов со структурой, но и средства определения иерархий типов и иерархий таблиц. Кроме того, допускалось использование «*ссылок*» для определения связей между объектами и навигации от одного объекта к другому.

С самого начала для определения поведения структурного типа можно было использовать UDF. Однако в DB2 Version 7 появилась возможность определения *методов* объектов. Методы могут быть написаны на SQL или на внешнем языке, например, на Java или C. Но методы всегда ассоциируются с некоторым структурным типом, и их вызовы производятся в синтаксисе, отличном от синтаксиса вызова UDF.

3.3.2.2. Создание иерархий типизированных таблиц

Иерархии таблиц используются для хранения объектов и для обеспечения возможности изменять эти объекты на уровне SQL. С концептуальной точки зрения, в иерархии таблиц отражается соответствующая иерархия типов. Однако типы, подобно библиотекам классов, могут использоваться в различных контекстах, поэтому не все типы из иерархии типов обязаны присутствовать в соответствующей иерархии таблиц. Единственным требованием является то, что при образовании иерархии таблиц должно использоваться подмножество ассоциированной иерархии типов.

При определении *типизированной таблицы* (аналог типизированной таблицы IUS или объектной таблицы Oracle8) нужно явно специфицировать дополнительный столбец, в котором будут храниться значения объектных идентификаторов (OID) каждого объекта (строки), содержащейся в таблице. Значения OID в DB2 могут как генерироваться системой (в DB2 это совсем не RowID, как в Oracle8, а некоторые абстрактные идентификаторы), так и задаваться пользователями при занесении строки в типизированную таблицу. Столбец OID создается как первый столбец таблицы, и за ним следуют столбцы, соответствующие атрибутам типа таблицы. Объектные идентификаторы используются в операциях *разыменования* (*dereference*), т.е. обеспечивают возможность добраться до строки по ссылке и преобразовать ее в экземпляр (значение) структурного типа.

В иерархии таблиц поддерживается семантика включения, т.е. все строки подтаблиц доступны через запрос, адресуемый к их супертаблице. Естественно, запрос к супертаблице может выдать только значения столбцов, определенных в этой супертаблице. Однако имеется возможность формулировки запроса к супертаблице с внешним объединением строк всех подтаблиц. Имеется и обратная возможность. Используя в запросе ключевое слово ONLY, можно ограничить оператор выборки набором строк, непосредственно принадлежащих супертаблице. Как и в Oracle, в запросах на DB2 SQL можно использовать путевые выражения, но не в «точечной», а в «стрелочной» нотации.

Заметим, что когда-то разработчики DB2 UDB собирались включить в систему и средства работы с типами коллекций, однако, насколько известно автору, этого не произошло до сих пор.

4. Коротко о предпосылках ОРСУБД

До появления 10 лет назад ОРСУБД ведущих компаний термин «объектно-реляционная СУБД» связывался с системами Postges-Illustra и UniSQL, разработанными под руководством Майкла Стоунбрейкера и Вона Кима соответственно. Про первое направление уже достаточно говорилось выше, поскольку оно лежит в основе IUS. Повторим лишь, что до основания компании Illustra сам Стоунбрейкер не использовал этого термина. Например, в [9] говорилось, что в модели данных Postges сочетаются объектные и расширенные реляционные черты, но авторы не берутся присвоить ей какое-либо отдельное название.

С другой стороны, Вон Ким сразу стал называть СУБД UniSQL «единой реляционной и объектно-ориентированной системой баз данных» [10]. В соответствии с подходом UniSQL, в ОРСУБД должны поддерживаться следующие возможности [11]:

- n-мерное объектно-ориентированное моделирование;
- двухмерное реляционное моделирование;
- наследование;
- инкапсуляция;
- постоянство существования объектов (*object persistence*);
- композиция классов;
- полиморфизм;
- навигационный доступ к объектам;
- реляционный доступ (соединения);
- непроцедурный доступ через запросы;
- интерфейсы для традиционных языков третьего поколения;
- интерфейсы для объектных языков третьего поколения;
- интерфейсы для языков четвертого поколения;
- независимое от языков хранение данных;
- независимость служб баз данных от файловых систем;
- поддержка оперативных служб СУБД.

Как отмечалось в [11], единая модель данных UniSQL опирается на следующее наблюдение. Если взять объектно-ориентированную модель данных и удалить из нее понятия наследования и инкапсуляции, то останется сетевая модель данных[*]. Далее, если удалить из нее понятия указателей и коллекций, то останется реляционная модель данных. В этом наблюдении

[*] В смысле CODASYL, см., например, [12].

имеется доля разумного смысла с учетом недостаточной точности определения объектно-ориентированной модели данных. В следующем разделе будет обсуждаться подход к сочетанию реляционных и объектных свойств в модели данных, представленной в современном стандарте языка SQL, и я приведу собственную точку зрения по этому поводу.

Пожалуй, наиболее существенные различия в подходах Майкла Стоунбрейкера и Вона Кима состояли в выборе отправной точки для построения объектно-ориентированных систем баз данных и в акцентах, которые делались при выполнении этой работы. Стоунбрейкер двигался от реляционной модели данных и от собственных реализаций СУБД, основанных на этой модели. Для него было наиболее важно сохранить в объектно-реляционной системе возможности реляционных систем, добавив, по мере возможности, средства, свойственные объектным системам. Понятно, что этот подход оказался привлекательным для компаний, производивших SQL-ориентированные СУБД, поскольку позволял им наращивать функциональные возможности своих систем без потребности в их коренной перестройке.

Вон Ким до создания СУБД UniSQL, в частности, активно занимался проектированием и разработкой чисто объектно-ориентированных СУБД. Под его руководством было разработано семейство ООСУБД Orion [13]. Поэтому для него было естественно подходить к решению проблемы единой объектно-реляционной модели данных с позиций объектно-ориентированного мира. Как говорится в [10], ядро UniSQL проектировалось с целью полной поддержки надмножества базовой объектной модели (Core Object Model, COM) консорциума Object Management Group[**]. Реляционные свойства обеспечивались, фактически, путем специального использования объектных возможностей системы. По пути UniSQL впоследствии пошли многие компании, производившие не SQL-ориентированные (в частности, объектно-ориентированные) СУБД, для обеспечения поддержки языка SQL. Тем не менее, подход Вона Кима не оказал существенного влияния на «основной поток» ведущих коммерческих СУБД.

С методологической точки зрения на развитие основного объектно-реляционного подхода и соответствующих средств, специфицированных в стандарте языка SQL, важнейшее влияние оказал *Манифест систем баз данных третьего поколения*, опубликованный группой авторов под очевидным руководством Майкла Стоунбрейкера в 1990 г. [3]. В этом документе постулировались три основных принципа систем следующего поколения:

[**] Модель данных COM лежала в основе архитектуры CORBA и являлась в начале 1990-х гг. единственной объектной моделью данных, для которой существовала независимая от производителей спецификация. Впоследствии на модели COM во многом основывалась стандартная объектная модель данных ODMG (Object Data Management Group) [15].

- помимо традиционных услуг по управлению данными, СУБД третьего поколения должны обеспечивать поддержку более богатых структур объектов и правил;
- СУБД третьего поколения должны включить в себя СУБД второго поколения;
- СУБД третьего поколения должны быть открыты для других подсистем.

Эти принципы развивались в тринадцати технических предложениях, включающих обеспечение развитой системы типов с поддержкой наследования и инкапсуляции. Если внимательно посмотреть на стандарты SQL:1999 и SQL:2003 [1,2], а также на возможности современных версий СУБД DB2 и Oracle, то можно увидеть отражение в них всех принципов и предложений *Манифеста систем баз данных третьего поколения*. Другой вопрос, насколько эти принципы и предложения можно считать объектными? Мы обсудим этот вопрос в контексте языка SQL в следующем разделе.

5. ОРСУБД и стандарт языка SQL

В процессе стандартизации языка баз данных SQL можно выделить, по моему мнению, три основных этапа. Первый этап начался в середине 1980-х гг. и завершился принятием международного стандарта SQL/89, наиболее важным достижением которого явилась языковая спецификация базовых средств запросов и механизмов ссылочной целостности. Результатом второго этапа явилось принятие стандарта SQL/92, в котором были полностью специфицированы средства определения и эволюции схемы базы данных, поддержки ссылочной целостности, манипулирования данными, связывания SQL с языками программирования и ряд других возможностей, позволяющих использовать стандарт SQL для реализации полнофункциональной РСУБД.

С точки зрения основной темы данной статьи для нас основной интерес представляет третий этап, важным промежуточным финишем которого явилось появление стандарта SQL:1999, развитого и уточненного в стандарте SQL:2003. Сейчас (в феврале 2007 г.) трудно сказать, закончился ли этот этап и происходит ли новый этап, связанный с интеграцией в SQL средств управления XML-данными. Ответ на этот вопрос можно будет дать только после принятия следующего стандарта.

В этой статье в стандартах SQL:1999 и SQL:2003 (на которые далее я буду обобщенно ссылаться, как на *стандарт SQL*) меня, главным образом, интересуют система типов SQL, возможности определения типов пользователями и средства определения и использования так называемых *типизированных таблиц (typed tables)*, составляющие, по мнению разработчиков стандарта, *объектную модель SQL*.

5.1. SQL как модель данных

Конечно, любой вариант стандарта языка SQL можно было трактовать как определение модели данных в соответствии с критериями Эдгара Кодда [16]. Для полноты картины напомню, что по Кодду модель данных представляет собой некий абстрактный эталон, которому должны соответствовать все реализации, претендующие на соответствие общей модели. В спецификации модели данных Кодд выделял структурную, целостную и манипуляционную составляющие, определяющие, соответственно:

- структуры данных, которые могут существовать в базах данных, отвечающих модели;
- базовые ограничения целостности, определение и поддержка которых должны обеспечиваться в системах баз данных, поддерживающих эту модель;
- средства манипулирования данными, задающие критерий выразительной мощности любого конкретного языка баз данных, который может поддерживаться в соответствующих СУБД.

Для реляционной модели данных в структурной части модели говорится, что единственной «родовой» структурой данных является *n*-арное отношение, атрибуты которого содержат атомарные значения. В целостной части требуется поддержка *целостности сущности* (наличие у любого отношения хотя бы одного возможного ключа) и *целостности ссылок* (недопущение внешних ключей, содержащих не неопределенное значение, для которого отсутствует совпадающее с ним значение возможного ключа в отношении, в которое ведет ссылка). Наконец, в манипуляционной составляющей реляционной модели данных специфицировались эквивалентные механизмы реляционной алгебры и реляционного исчисления.

Достаточно очевидно, что:

- любой из перечисленных выше стандартов языка SQL определяет некоторую модель данных;
- с появлением каждого следующего варианта стандарта языка SQL соответствующая модель данных дополняется и уточняется;
- наконец (и это самое важное!), модель данных, определяемая языком SQL, никогда (!) не была и не является реляционной.

Действительно, начиная со стандарта SQL/89 было понятно, что SQL-ориентированная база данных состоит из таблиц, в столбцах которых могут содержаться данные типов, специфицированных в стандарте. Говорилось, что в таблицах *могут* определяться первичный и возможный ключи, и в этом случае СУБД должна поддерживать уникальность их значений. Некоторым образом требовалась поддержка ссылочной целостности. Наконец, в качестве

«абстрактного» механизма манипулирования данными выступали средства самого языка SQL.

С другой стороны, только в стандарте SQL/92 были явно введены конструкции определения базовых таблиц, а тем самым, и возможности определения первичных и возможных ключей. Только в стандарте SQL:1999 был специфицирован полный набор параметризуемых, генерируемых и определяемых пользователями типов данных, значения которых могут присутствовать в столбцах таблиц SQL-ориентированных баз данных. По мере развития стандарта расширялись и уточнялись средства поддержки ссылочной целостности и манипулирования данными и т.д.

Наконец, начиная с SQL/89, определение хотя бы одного возможного ключа в таблице было необязательным, так что в таких таблицах могли присутствовать строки-дубликаты, они не являлись множествами строк; следовательно, они не были аналогами отношений реляционной модели данных. В заголовках таблиц присутствовал неявный порядок имен столбцов, следовательно, они не являлись аналогами заголовков отношений реляционной модели данных. В столбцах таблиц, наряду с типизированными значениями, могли храниться нетипизированные неопределенные значения, вызывающие необходимость в использовании трехзначной логики и приводящие к иногда парадоксальным ситуациям [17].

Трактовка стандарта SQL как спецификации некоторой особой модели данных (модели данных SQL) стала действительно актуальной после появления стандарта SQL:1999. До этого достаточно было говорить, что язык SQL во многом следует реляционной модели данных, но с некоторыми отклонениями, которые следует иметь в виду. Однако появление «объектных расширений» в SQL:1999 («объектность» этих расширений мы обсудим в следующем подразделе), в особенности, полной системы типов, во-первых, делает законченной собственную модель SQL, во-вторых, еще больше отдаляет модель SQL от классической реляционной модели данных Кодда и, в третьих, создает потребность в сопоставлении модели SQL с объектной моделью ODMG [15] и «истинно реляционной» моделью Кристофера Дейта и Хью Дарвена [18][*].

Понимание отличий модели данных SQL от реляционной модели данных очень важно для проектировщиков, администраторов и разработчиков приложений SQL-ориентированных баз данных. Похоже, что сегодняшняя

[*] Замечу, что некоторое сопоставление этих моделей делается в [18] и вышедшем в 2006 г. новым изданием этой книги [19]. Однако авторы этих книг не трактуют стандарт SQL как модель данных и вообще относятся к SQL подчеркнуто и категорически отрицательно. Поэтому, с моей точки зрения, сравнения, приводимые в [18, 19] не вполне объективны. Так что потребность в объективном сопоставлении понятий, возможностей и недостатков этих трех моделей данных остается.

модель данных SQL включает классическую реляционную модель данных [16], хотя расходится в части механизма типов данных с «истинно реляционной» моделью [18]. Очевидным преимуществом классической реляционной модели данных является ее собственная простота и следующая из нее простота баз данных и их приложений. Поэтому при использовании SQL нужно явно отдавать себе отчет в реальной потребности использования возможностей, выходящих за пределы классической реляционной модели данных.

5.2. Объектная подмодель SQL

В самом стандарте языка SQL, естественно, вообще ничего не говорится о моделях данных. Однако главный редактор стандарта SQL Джим Мелтон в [1] утверждает, что объектная подмодель SQL включает два основных отличительных набора механизмов: средства определения и использования *структурных типов данных* (*User Defined Type – UDT*) и *типизированных таблиц* (*typed table*).

Первый компонент позволяет определять новые типы данных, которые могут быть гораздо более сложными, чем встроенные типы данных языка SQL. При определении структурного UDT требуется специфицировать не только содержащиеся в нем элементы данных, но и семантику типа данных, т.е. его поведение на основе интерфейса вызовов методов. UDT являются полностью инкапсулированными. Доступ к значениями UDT возможен только через методы этого типа.

Для каждого атрибута UDT автоматически определяются два метода: *наблюдатель* (*observer*) и мутатор (*mutator*). Метод-наблюдатель применятся к значению UDT и возвращает значение соответствующего атрибута этого структурного значения. Метод-мутатор действует над местоположением (столбцом строки некоторой таблицы, переменной или параметром) значения UDT и заменяет это структурное значение новым значением с указанным значением соответствующего атрибута. Синтаксические правила вызова методов-наблюдателя и мутатора таковы, что позволяют манипулировать с атрибутами значений UDT в привычной «точечной» нотации.

Для определения структурных UDT поддерживается механизм одиночного наследования. При определении подтипа можно добавлять к структуре супертипа новые атрибуты, специфицировать новые методы или изменять реализацию методов супертипа (с сохранением сигнатуры). Соответственно, поддерживаются полиморфизм по включению (значение любого UDT является значением любого своего супертипа) и отложенное связывание.

Второй компонент – типизированные таблицы – позволяет определять таблицы, строки которых являются *экземплярами* (или *значениями*) UDT , с которым явно ассоциируется таблица. При определении типизированной таблицы можно объявить ее подтаблицей некоторой другой типизированной таблицы. Супертаблица должна быть ассоциирована со структурным типом,

являющимся непосредственным супертипом определяемой подтаблицы. Каждый столбец указанной супертаблицы наследуется подтаблицей; наследуются и характеристики столбцов супертаблицы – значения по умолчанию, ограничения целостности и т.д. Эти столбцы называются *унаследованными столбцами* подтаблицы, и они соответствуют атрибутам UDT подтаблицы, унаследованным от UDT супертаблицы. Кроме того, подтаблица будет содержать по одному столбцу для каждого собственного атрибута ассоциированного структурного типа. Такие столбцы подтаблицы называются *заново определенными.*

В определении максимальной супертаблицы (не являющейся подтаблицей какой-либо другой супертаблицы) должна присутствовать спецификация «*самоссылающегося*» (*self -referencing*) столбца, и самоссылающийся столбец, определенный в максимальной супертаблице, наследуется любой ее подтаблицей. Эта спецификация не может входить в определение подтаблицы. Значения самоссылающегося столбца относятся к *ссылочному типу*, ассоциированному с UDT, на котором определена данная типизированная таблица. Значения ссылочного типа, или *ссылочные значения* могут генерироваться одним из трех способов (указываемых при определении UDT):

- задаваться приложением как значения некоторого встроенного типа SQL каждый раз при сохранении экземпляра структурного типа как строки типизированной таблицы;
- порождаться из одного или нескольких атрибутов UDT;
- автоматически генерироваться системой.

Любой ссылочный тип можно использовать для объявления типа любого местоположения, в том числе, столбца другой типизированной или обычной таблицы. Поскольку любое действующее ссылочное значение является уникальным идентификатором соответствующей строки типизированной таблицы, в объектной подмодели SQL оно считается аналогом объектного идентификатора в обычных объектных моделях (в частности, в модели ODMG), и в SQL-запросах можно использовать «стрелочную» нотацию для перехода от одного «объекта» к другому по ссылочному значению. При использовании этой возможности SQL-запросы почти неотличимы синтаксически от запросов на языке OQL [15].

Эта синтаксическая близость может привести к впечатлению о близости самих объектных моделей SQL и ODMG. Обманчивость этого впечатления обосновывается в [20]. Я не буду говорить об этом в данной статье, хотя некоторые утверждения в [20] уже кажутся мне слишком категоричными. Но здесь мне хочется затронуть другой вопрос: насколько гармонирует объектная подмодель SQL с общей моделью данных SQL?

Заметим, что средства определения структурных UDT можно считать частью стандарта SQL, относящегося к системе типов данных. В стандарте SQL поддерживаются:

- параметризуемые типы точных и приближенных чисел;
- параметризуемые типы символьных и битовых строк;
- темпоральные типы данных (дата-время, интервал);
- генерируемые типы коллекций (множества, мультимножества, массивы);
- анонимные строчные типы;
- индивидуальные и структурные UDT;
- ссылочные типы.*

При определении типа коллекции можно использовать любой тип элементов, включая типы коллекций и UDT. Элементы анонимных строчных типов и атрибуты структурных UDT могут также иметь любой тип данных. Для значений любого типа данных определена операция сравнения по равенству, так что с точки зрения модели данных SQL осмысленными являются столбцы, определенные на любом типе данных (по крайней мере, работают наиболее важные в модели SQL, как и в реляционной модели, операции экви- и естественного соединения). Так что система типов SQL вполне гармонична модели данных SQL. Более того, я не стал бы считать, что в системе типов SQL имеются «объектные расширения»: подобные средства свойственны любому полнотиповому языку, не обязательно объектно-ориентированному.

Вообще говоря, нет ничего особенно «объектного» и в типизированных таблицах. По сути дела, объявляется таблица с двумя столбцами, один из которых определен на структурном UDT, а другой содержит уникальные идентификаторы (суррогатные, если они генерируются системой). По сути дела, «объектность» в стандарте SQL появляется, когда мы начинаем считать строки типизированной таблицы объектами, а идентификаторы этих строк объектными идентификаторами. В этом случае на свет выходит «стрелочная нотация», заменяющая эквисоединения. Но мы должны отдавать себе отчет, что это всего лишь синтаксическая замена: чтобы добраться от строки одной таблицы до строки другой таблицы по ссылочному значению, требуется, чтобы это ссылочное значение хранилось в обеих строках, т.е. переход от одной строки к другой делается, фактически, путем эквисоединения.

Таким образом, «объектные расширения» SQL не выводят язык за пределы общей модели SQL. При этом обеспечиваются функциональные возможности, которые специалисты из объектного мира считают «объектными» (см., например, разд. 4 относительно подхода Вона Кима к организации UniSQL).

6. Проблемы и перспективы ОРСУБД

Как отмечалось во введении, в последние годы про ОРСУБД говорят и пишут очень мало. Я обсуждал вопросы использования расширенных возможностей

* В этом списке умышленно отсутствует тип данных XML, поскольку соответствующие аспекты языка SQL в данной статье не рассматривается.

SQL со многими разработчиками приложений, администраторами баз данных, сотрудниками ведущих компаний-производителей СУБД и пришел к следующим выводам.

Для большинства практических разработчиков приложений расширенные возможности SQL ограничиваются средствами серверного программирования с использованием хранимых процедур и функций. Мне не удалось найти пример недавно созданного независимыми разработчиками приложения баз данных, для которого в базе данных определялся бы специальный структурный UDT. Тем более, мне неизвестны базы данных, содержащие типизированные таблицы.

Казалось бы, естественно использовать расширенные возможности ОРСУБД самими компаниями, производящими эти системы, для включения в них новых функциональных возможностей (как это и происходило 10 лет назад). Однако и таких примеров крайне мало. Например, я был очень обрадован, узнав, что специалисты компании Oracle воспользовались объектными расширениями SQL для реализации в Oracle 10g функциональных возможностей управления многомерными данными, которые ранее поддерживались отдельным продуктом Oracle Express Server [21].

Почему же пользователи ОРСУБД и разработчики приложений пренебрегают развитыми средствами, которые обеспечиваются в этих системах? Можно было бы подумать, что людей из «реляционного» мира SQL отпугивают новые «объектные» возможности. Но в разд. 5 мы видели, что «объектные» расширения SQL совсем не являются объектными и не меняют модель данных SQL. Похоже, что причина кроется совсем в другом, а именно, в излишне обширном наборе возможностей.

Расширенные возможности SQL, в особенности, средства серверного программирования, обеспечивающие возможности определения UDT, хранимых процедур и функций, триггеров и т.д. позволяют переносить на сервер баз данных все бо́льшую часть логики приложений. Это в значительной степени противоречит учению Эвгара Кодда, в котором обосновывалась целесообразность независимости базы данных от приложений. Независимость базы данных от приложений часто выглядит очень привлекательной идеей, но для ее применения разумно отказаться от многих расширений SQL.

Кроме того, даже если не принимать во внимание упомянутую идею, при проектировании приложения базы данных имеется три альтернативы: можно реализовать логику приложения на стороне клиента, на сервере приложений и на сервере баз данных. Очевидно, что каждая альтернатива имеет право на жизнь, и каждая из них может оказаться выигрышной в конкретной ситуации. Однако отсутствует методология, не говоря уже про технологию, выбора наиболее выгодного проектного решения. Для расширения области использования ОРСУБД требуются дополнительные исследования и разработки в этом направлении.

Другой проблемой, не связанной с возможностями серверной реализации логики приложений, является использование в базах данных типов коллекций. Поддержка в стандарте SQL типов мультимножеств, элементами которых могут быть значения анонимных строчных типов, обеспечивает теперь возможность определения вложенных таблиц с произвольным (теоретически, неограниченным) уровнем вложенности. Поскольку все значения, хранимые в базе данных, продолжают оставаться строго типизированными, такая возможность не противоречит базовому требованию первой нормальной формы, унаследованному из реляционной модели данных, но, по существу, обеспечивает подход к прямому моделированию иерархических структур.

Разработчики приложений баз данных в течение многих лет порицали SQL за отсутствие прямой поддержки иерархий. Теперь эта возможность появилась и многих поставила в тупик. В каких случаях действительно выгодно пользоваться вложенными таблицами? Нужно ли разработчикам приложений понимать, что на уровне хранения данных в реализации SQL-ориентированных СУБД, скорее всего, физическая вложенность поддерживаться не будет? В действительности, проблема выбора способа представления иерархических данных свойственна не только современному SQL, аналогичные проблемы возникают, например, и при проектировании баз XML-данных. Здесь тоже требуются исследования с целью выработки обоснованных рекомендаций и методологий проектирования*.

Другими словами, аскетичность подхода Кодда, выраженная в классической реляционной модели данных, оказывается пока понятнее и полезнее богатства современной модели данных SQL (думаю, что это равно применимо к объектной модели ODMG и «истинно реляционной» модели Дейта и Дарвена). Для полноценного использования этого богатства требуются понятные и обоснованные методологии.

Я думаю, что накопленный за 10 лет багаж объектных расширений, специфицированных в стандарте SQL и реализованных в передовых продуктах компаний IBM и Oracle, не должен бесславно пропасть. Это было бы нерационально, учитывая громадный объем человеческого труда, затраченного за создание этих расширений. Мне хочется надеяться, что с помощью исследовательского сообщества баз данных удастся решить упомянутые выше и другие проблемы и привлечь разработчиков к полноценному использованию возможностей ОРСУБД.

* Замечу, что здесь не могут выручить средства концептуального моделирования баз данных, такие как E/R-диаграммы или диаграммы классов UML, поскольку в них также допускаются различные альтернативы моделирования иерархических данных.

7. Заключение

Появление объектно-реляционного подхода к организации баз данных и СУБД явилось важным шагом на пути развития технологии баз данных. За 10 лет, прошедших с момента выхода первой коммерческой ОРСУБД, объектно-реляционный подход, с одной стороны, приобрел зрелость, утвердился в спецификациях стандарта SQL, но, с другой стороны, так и не завоевал широкого распространения среди разработчиков приложений баз данных.

В этой статье, посвященной юбилею ОРСУБД, были обсуждены история подхода, его современное состояние и проблемы, препятствующие его широкому использованию

Литература

[1] Jim Melton. Advanced SQL:1999. Understanding Object-Relational and Other Advanced Features. Morgan Kaufmann Publishers, 2003

[2] Сергей Кузнецов. Наиболее интересные новшества в стандарте SQL:2003. http://www.citforum.ru/database/sql/sql2003/, 2004

[3] M. Stonebraker, L. Rowe, B. Lindsay, J. Gray, M. Carey, M. Brodie, Ph. Bernstein, D. Beech. Third-Generation Data Base System Manifesto. Proc. IFIP WG 2.6 Conf. on Object-Oriented Databases, July 1990, ACM SIGMOD Record, 19, No. 3 (September 1990). (Имеется русский перевод: Стоунбрейкер М. и др. “Системы баз данных третьего поколения: Манифест”, СУБД, No. 2, 1995, http://old.osp.ru/dbms/1995/02/23.htm)

[4] В.Галатенко, А.Гвоздев. Типы и структуры данных в Informix Universal Server. Jet Info/Информационный бюллетень, N 12-13, 1997, http://www.jetinfo.ru/1997/12-13/1/article1.12-13.1997.html

[5] А. Грачев. Объектно-реляционная СУБД Informix Universal Server. СУБД, N 1-2, 1998, http://old.osp.ru/dbms/1998/01-02/013.htm

[6] В. Пржиялковский. Новые одежды знакомых СУБД: Объектная реальность, данная нам. СУБД, No. 2, 1995, http://old.osp.ru/dbms/1997/04/88.htm

[7] Donald Chamberlin. Anatomy of an Object-Relational Database, DB2 Magazine, Winter 1996, Vol. 1, No. 1. (Имеется русский перевод: Д. Чемберлин. “Анатомия объектно-реляционных баз данных”, СУБД, No. 1-2, 1998, http://old.osp.ru/dbms/1998/01-02/003.htm)

[8] Н. Игнатович. База данных DB2 Universal Database. Материалы конференции «Корпоративные Базы Данных'1999», http://www.citforum.ru/seminars/cbd99/db2.shtml

[9] Michael Stonebraker, Lawrence A. Rowe, Michael Hirohama: The Implementation of Postgres. IEEE Trans. Knowl. Data Eng. 2 (1), 1990

[10] Won Kim. UniSQL/X unified relational and object-oriented database system. ACM SIGMOD Record, 23 , No. 2 (June 1994)

[11] Albert D 'Andrea and Phil Janus. UniSQL's next-generation object-relational database management system**.** ACM SIGMOD Record, 25 , No. 3 (September 1996)

[12] М.Р. Когаловский. Энциклопедия технологий баз данных. – М.: Финансы и статистика, 2002.

[13] Won Kim, Jorge F. Garza, Nathaniel Ballou, Darrell Woelk. Architecture of the ORION Next-Generation Database System. IEEE Trans. Knowledge and Data Eng., 2 (1), 1990

[14] OMG Core Object Model. http://www.objs.com/x3h7/omgcore.htm

[15] С.Д. Кузнецов. Три манифеста баз данных: ретроспектива и перспективы. Базы данных и информационные технологии XXI века. Материалы международной научной конференции. Москва, РГГУ, 2004, http://www.citforum.ru/database/articles/manifests/

[16] E. F. Codd: A Relational Model of Data for Large Shared Data Banks, CACM 13, No. 6 (June 1970). Republished in “Milestones of Research”, CACM 26, No. 1 (January 1982). (Имеется русский перевод: Кодд Е. А. “Реляционная модель для больших совместно используемых банков данных”, СУБД, No. 1, 1995, http://old.osp.ru/dbms/1995/01/01.htm)

[17] Сергей Кузнецов. Дубликаты, неопределенные значения, первичные и возможные ключи и другие экзотические прелести языка SQL. СУБД, No. 3, 1997, http://www.citforum.ru/database/articles/art_5.shtml

[18] К. Дейт, Хью Дарвен. Основы будущих систем баз данных. Третий манифест. – М: Янус-К, 2004

[19] C.J. Date and Hugh Darwen. Databases, Types, and the Relational Model. The Third Manifesto. Addison Wesley; 3 edition (February 24, 2006)

[20] Сергей Кузнецов. Объектны» ли объектные расширения языка SQL? http://www.citforum.ru/database/articles/sql_odmg/, 2005 г.

[21] Bud Endress, Hemant Verma. Leveraging Business Intelligence Tools with the OLAP option to the Oracle10g Database. http://www.oracle.com/technology/products/bi/olap/40261_leveragingtools.pdf