Обзор моделей данных объектно-ориентированных СУБД

А.М. Эльдарханов
033555@gmail.com

Аннотация. Объектно-ориентированные СУБД – одно из наиболее перспективных направлений развития современной теории баз данных, наряду с дедуктивными и темпоральными СУБД. Тем не менее, серьёзным препятствием к построению теоретических основ ООСУБД и внедрению действующих ООСУБД является большая разрозненность подходов и отсутствие единого стандарта как в области теории (исчисление объектов, концепции моделей данных), так и в области практики (язык запросов, API для ОО-языков…). Целью данной статьи является анализ существующих на сегодняшний день концепций формального устройства объектно-ориентированных СУБД, начиная с моделей данных и далее переходя к формальным математическим моделям (исчислениям объектов, формализациям объектных языков запросов). В завершение делается заключение о наиболее актуальных проблемах моделирования ООСУБД.

Ключевые слова: модель данных, объектно-ориентированные СУБД

1. Введение. Историческая справка.

Современная концепция объектно-ориентированных СУБД возникла как результат развития двух поколений СУБД.

Первое поколение, представленное иерархическими и сетевыми СУБД 70-ых годов, ввело такие фундаментальные элементы СУБД, как схема базы данных, язык определения данных, язык запросов, выборка (select), кортеж и другие. Модель данных в этих системах являлась деревом (иерархические СУБД) или произвольным орграфом (сетевые СУБД). Вершина графа означала семейство однотипных кортежей (аналог таблицы в реляционной БД); кортежи, как ясно из вида модели, были связаны отношением родитель-потомок, никаких ограничений на связи не налагалось – эта задача ложилась непосредственно на программиста. Важным значением этих моделей было то, что они заложили

принцип декомпозиции реальных данных на связанные между собой однотипные кортежи, который с тех пор по сути применяется практически во всех СУБД вне зависимости от их класса. Тем не менее, реализации СУБД с иерархической и сетевой моделью обладали значительным недостатком: отсутствием декларативного языка запроса к данным. Пользователи этих СУБД были вынуждены пошагово программировать процесс того, как из хранилища будут извлечены данные, вместо того чтобы просто указать в запросе декларативным образом, какие данные нужны приложению в данный момент. Революция произошла с появлением реляционных СУБД, в которых императивные языки создания запросов были заменены декларативным языком SQL. Это стало возможно благодаря появлению алгебры Кодда, формализующей соотношения между кортежами и, в частности, предоставившей концепцию нормальной формы БД. Всё это обеспечило реляционным СУБД настолько высокую эффективность создания приложений, что РСУБД по сей день сохранили безусловное лидерство на рынке СУБД.

Тем не менее, уже в середине 80-ых годов в области систем управления данными возник ряд задач, для которых и реляционная модель была недостаточно эффективна. Круг их был достаточно широк, так что в СУБД сформировалось несколько направлений, значительно различающихся целевыми проблемами и подходом к модели БД – объектно-ориентированные СУБД, дедуктивные (логические) СУБД, активные СУБД, темпоральные СУБД. Все они в совокупности образовали **третье поколение** СУБД.

Класс СУБД третьего поколения, о котором идёт речь в этой статье, – **объектно-ориентированные СУБД** – возник как реакция на неоптимальность (impedance mismatch) реляционных СУБД для применения в таких областях, как автоматизированное проектирование (computer aided design, CAD); автоматизированное производство (computer aided manufacturing, CAM); автоматизированные системы управления предприятием (ERP); системы, основанные на знаниях (knowledge-based systems), мультимедийные системы. Проблемы РСУБД, выявленные в этих областях, составляют (приблизительно) следующий список:

– Ограниченность набора простых типов и отношений между сущностями. (Последнее фактически представлено всего одним экземпляром – связь через ключ.) При наличии в предметной области сложных специфических типов данных и стандартных операций над ними их приходится эмулировать созданием составных структур над разрешёнными простыми типами данных/одномерными таблицами и написанием сложных триггеров на таблицах с этими структурами, что является прежде всего крайне громоздкой задачей.

– Отсутствие object persistence, т.е. возможности постоянного хранения в БД составного объекта как единого целого. В реляционных СУБД составные

сущности не хранятся как отдельный объект, а собираются в run-time с помощью операции JOIN из таблиц. В том числе это приводит к необходимости дополнительных вычислительных затрат на JOIN; или – если с БД работает некоторое клиентское приложение и изменяет программные объекты, агрегируемые из нескольких таблиц – к вычислительным затратам на разнесение изменений этого объекта на составляющие его таблицы.

– Отсутствие идентификации сущностей. В основе реляционных СУБД лежит ассоциативный доступ к данным, то есть доступ к записям на основе их значений (SELECT FROM… WHERE field=”value”), в отличие от адресного доступа, где обращение к элементу данных происходит по id, имени или адресу элемента. Это приводит к ряду проблем – к примеру, к несоответствию между механизмом доступа к данными с точки зрения SQL (ассоциативный механизм) и механизмом доступа к данным на носителе (адресный механизм, для которого РСУБД вынуждена выполнять преобразование в ассоциативный).

– Вычислительная неполнота. Чистый SQL, без процедурных расширений, не способен решить некоторые <разрешимые с теоретико-множественной точки зрения> задачи выборки данных из реляционной БД. Пример: пусть в БД хранится информация об отношении "родитель-ребёнок" среди некоторого множества людей. Для простоты, пусть люди задаются только своими id, а база данных задана 1 таблицей с 2 неключевыми полями: parent_id и child_id. Требуется вывести на печать всех потомков человека с данным id (т. н. задача вычисления транзитивного замыкания отношения родитель-ребёнок). Легко написать SQL-запрос, выводящий для данного id всех его потомков I поколения (детей), всех потомков II поколения (внуков) ит.д., но вывести потомков всех поколений (искомое множество) нельзя, т.к. заранее не известно нужное число шагов – JOIN'ов таблицы с собой. Даже отвлекаясь от SQL, в реляционной алгебре задача построения такого объекта, как замыкание отношения, алгоритмически неразрешима. В связи с этой проблемой в некоторые РСУБД вводят дополнительные средства, обеспечивающие работу с иерархиями (рекурсивные временные таблицы Common Table Expressions, рекурсивный оператор CONNECT BY). Их принцип действия основан на разрешении в ходе запроса создавать именованные промежуточные выборки данных с возможностью JOIN-а их с собой – что приводит к возможности вычисления результата такого запроса, если он выполняется на иерархической структуре данных.

– Непрозрачность интерфейса реляционной БД для приложений на ОО-языках. Множества примитивов модели данных ОО-языков программирования (объекты и литералы) и реляционной СУБД (поля, кортежи, таблицы) пересекаются лишь по представителям простых типов данных, таким как целым

числам или символьным строкам, в остальном они не совпадают. Помимо данных, сильно различаются также функциональные части.

Концепция объектно-ориентированной СУБД возникла как реализация принципов объектно-ориентированного программирования в теории баз данных и позиционировала себя как средство разрешения многих указанных выше задач. В частности, она позволила разработчикам гибко реализовывать всю специфику предметной области через механизм классов, предоставляя программисту возможность в том числе задавать собственные составные типы данных (точнее, классы, т. к. класс и тип – не во всех ООСУБД одно и то же) вместе с неотделимыми от них базовыми операциями (методами классов), как и в "обычном" ООП.

Теоретические основания ООСУБД с самого начала развивались практически независимыми группами разработчиков, в основном как математические модели и модели данных под конкретные ООСУБД. Поначалу основными представителями таких ООСУБД являлись исследовательские проекты, такие как IRIS (by Hewlett-Packard) [1] или ORION (by Microelectronics and Computer Technology Corporation) [2]; далее появились и коммерческие продукты – Gemstone [3], Jasmine [4], O2 [5]. Негативным следствием ситуации стало отсутствие единого стандарта на объектную модель данных, объектный язык запросов и т.д., что актуально и на сегодняшний день; впрочем, реляционные СУБД также проходили через эту стадию.

Некими попытками приведения ООСУБД к стандартам стали такие публикации, как "Манифест ООСУБД" [6], "Манифест систем баз данных третьего поколения" [7], "Третий манифест" [8], стандарты ODMG 1.0-3.0 (консорциум ODMG, 1991-2001, [9]) , о которых ещё пойдёт речь далее в статье. Несмотря на то, что некоторые из этих стандартов возымели успех (к примеру, ODMG 3.0 [10] был формально принят как стандарт многими вендорами, входящими в ODMG), единства всё равно не возникло. Следует отметить, что одним из основных спорных моментов в положениях этих стандартов является возможность выразить объектную модель реляционными средствами: позиции стандартов варьируются от отказа от реляционной модели (Первый манифест) до противоположной позиции, что объектная модель целиком и полностью реализуема в ОРСУБД (Третий манифест). Так или иначе, со временем происходило сближение объектной и реляционной концепций: с одной стороны, в стандарт языка SQL 1999 года [11] – SQL-3 – были включены объектные средства, позже реализованные в том числе в таких крупных представителях рынка СУБД, как Oracle и Postgres; а с другой стороны, поздние стандарты со стороны ООСУБД-вендоров, тот же ODMG 3.0, включали в том числе стандарты на объектно-реляционную привязку (mapping).

Важным этапом развития ООСУБД стала разработка к середине 2000-х гг. альтернативы механизму API ООСУБД для взаимодействия с приложениями, а именно включение средств конструирования запросов непосредственно в данный язык программирования. Это выразилось в создании таких средств, как Native Queries (для Java и C#) [12] и Microsoft LINQ [13] (для .NET).

Также в середине 2000-х гг. наблюдался рост популярности ООСУБД в связи с появлением и доступностью open-source ООСУБД, например db4o [14].

В 2006 году консорциум OMG объявил о возможности выпуска четвёртой редакции стандарта ODMG [15].

2. Общее устройство ООСУБД.

Задачу создания любой отдельно взятой ООСУБД можно рассматривать с точки зрения следующих уровней представления данных. (Деление достаточно условное, т. к. при желании каждый из уровней можно подразделить на более подробные.) Уровни указаны в порядке убывания абстракции. Первым двум из них далее посвящено по отдельной главе статьи.

Уровень формальной математической модели. На нём необходимо определить данные как формулы некоторой формальной теории, известной в математике или сконструированной самостоятельно. Вообще говоря, для построения модели ООСУБД достаточно лишь аппарата построения формул (являющегося лишь частью соответствующей ему формальной теории, наряду с аксиоматикой и правилами вывода формул), но для логического подхода к созданию ООСУБД характерно полное применение атрибутов формальной теории, в том числе логического вывода. К задаче формализации данных тесно примыкает задача формализации декларативного языка запросов: в первой задаче в виде формул строятся сами структуры данных, во второй задаче – пути обхода этих структур.

В будущем необходимо также определить, каким сущностям следующего уровня представления – **уровня модели данных**, см. следующий пункт – эти объекты соответствуют, иначе мат.модель будет изолированной "вещью в себе". Но пока формально на уровне мат.модели лишь нужно определить некую (алгебраическую?) структуру, не задумываясь о её интерпретации.

К примеру, реляционная алгебра имеет основой формальную Канторовскую теорию множеств, дополненную рядом операций над одним из её элементов – бинарными отношениями. Изначально определяются такие объекты, как множество и его элемент, затем декартово произведение множеств, затем некий предикат на этом произведении (бинарное отношение), и т. д. Интерпретациями этих объектов в реляционной модели данных являются, соответственно, поле <у таблицы> и его домен, затем кортеж и т.д.

Достоинство математической модели в её формальности – строго определены разрешённые объекты и действия над ними.

Уровень модели данных. СУБД на уровне модели данных – это также формально-математическая модель, но в качестве её элементов берутся не математические примитивы, а примитивы некоторого языка манипулирования данными. К примеру, в случае реляционной модели это тип данных, кортеж, таблица, ключ, индекс, связь по ключу, запрос. В объектном случае, примитивами выступают класс, объект, опять тип данных, указатель, uid и прочее, а также всевозможные производные от них; соответственно, требуется определить допустимые значения примитивов и в каких соотношениях могут состоять эти примитивы – система типов, вид типизации, как соотносятся типы и классы, как устроено наследование, ит.д. На этом же уровне задаются методы классов, синтаксис языка запросов, язык управления данными. Это основной уровень разработки СУБД, большая часть статьи посвящена ему. Следует отличать разработку СУБД от разработки конкретной БД под неё; первая задача неизмеримо сложнее, т. к. вторая состоит лишь в создании конкретной реализации модели данных – т.е. схемы БД – и наполнения её данными и средствами их контроля, пользуясь уже созданным инструментарием СУБД.

Физический уровень. На этом уровне задаётся прямое взаимодействие СУБД с ЭВМ – действия с оперативной памятью, физическим хранилищем данных, процессорными инструкциями, а также всевозможные обслуживающие элементы СУБД, не относящиеся напрямую к моделированию данных – контроль совместного доступа к данным, система бэкапов, индексирование. В число задач этого этапа входит разработка механизма выполнения языка запросов (в отличие от предыдущего уровня, где задавался лишь его синтаксис), способа распределения данных по физическим хранилищам, системы кэширования запросов.

Стоит указать базовые для многих ООСУБД свойства, не относя их к какому-либо из уровней представления:

– Использование объектов. Объект – базовый элемент модели данных ООСУБД, позволяющий применять объектно-ориентированный подход при программировании этой СУБД. Он характеризуется набором своих атрибутов и методов. В отличие от реляционной модели, где в основном ведётся работа с кортежами атрибутов, атрибуты объектов принято скрывать от внешнего доступа, реализуя доступ к ним через методы. Если все поля являются скрытыми (т.е. открытыми только для методов класса), то говорят о "чистой объектной системе" [16], или, что то же самое, что объекты в такой системе являются скалярами.

Наличие объектов позволяет не только применять ООП, но и делает ООСУБД более прозрачной при доступе к ней из приложений на ОО-языках, т.к. объекты в таком случае являются примитивами как языка, так и БД, упрощая задачу создания API.

– Использование идентификаторов объектов (oid) и построение с их помощью составных объектов. Последнее означает, что атрибут объекта может быть oid-ом, то есть один объект может быть структурным подобъектом другого (или, в терминах классов, класс может иметь атрибут, принадлежащий не к простому типу данных, а к другому классу):

```
class TDept:{          // Класс "отдел компании"
char* name;            // Название отдела
TEmployee chief;       // Начальник отдела
…}
```

Такой подход, несмотря на свои преимущества (гибкость, компактность данных, простота представления иерархических структур), имеет и свои недостатки, такие как, например, однонаправленность ссылок по oid-у: из объекта класса TDept можно непосредственно адресовать его начальника, но не наоборот. Плюс, появляется необходимость сборки мусора – удаления объектов, на которые не осталось ссылок.

– Возможность перегрузки методов, то есть введения нескольких одноимённых методов с разными наборами входных параметров. Это ведёт к невозможности определения типа входных параметров в некоторых точках вызова метода на этапе компиляции. Возникшая проблема разрешается, как и в ОО-языках, введением возможности позднего связывания, т.е. определения типа входных параметров на этапе выполнения программы (в данном случае в момент вызова метода в объектном запросе к БД).

– Сохраняемость объектов (object persistence). Определение этого свойства дано выше. Для реализации сохраняемости в ООСУБД применяется следующий приём: в пуле объектов выделяется некоторое множество корневых (root) объектов, которые хранятся как единое целое (персистентны) всегда. Затем любой составной объект, в структурном графе которого есть корневые объекты, сам объявляется персистентным. Задача удаления объектов, к которым не осталось ни одного структурного пути от корневых объектов, решается, как и в случае со ссылками по oid, сборщиком мусора.

3. Уровень модели данных ООСУБД.

Представление ООБД на уровне модели данных подразделяется на **структурную** и **поведенческую** части. Первая представляет данные и их домены, вторая – функциональную составляющую, то есть функции, методы и исключения.

Структурная часть, в свою очередь, подразделяется на **уровень данных** (data level) и **уровень схемы** (schema level). Первый представляет собственно данные и соотношения между ними, второй – "искусственные" сущности, непосредственно в данные не заложенные, такие как классы, типы, роли и отношения.

Уровень данных

Наиболее универсальное описание модели данных ООБД приводится в статье [2]. В этой статье объекты считаются атомарными элементами БД и им не даётся никакого внутреннего определения.

(На первый взгляд, это отличает данную модель от модели, принятой в большинстве ОО-языков программирования, где объект содержит атрибуты и методы. Тем не менее, забегая вперёд, скажем, что на самом деле модель Беери реализует ту же самую идеологию, лишь выражая её в терминах теории графов – объекты есть вершины некоторого орграфа, а его рёбра есть отношения наподобие "быть атрибутом".)

Каждый объект характеризуется своим уникальным идентификатором (oid, object identifier). Это важнейший примитив любой ООСУБД, аналогичный понятию ключа в реляционной СУБД.

Далее, если мы хотим ввести в ООСУБД какие-либо простые типы данных и экземпляры этих типов, мы можем выделить некоторые объекты из пула объектов и поставить каждому из них в соответствие целое число, символьную строку или любое другое значение-представитель простого типа. Такие объекты назовём *объектами-значениями*[1], остальные – *абстрактными объектами*. У объектов-значений отпадает необходимость иметь oid, так как вместо этого такие объекты можно заменить на сущности, поставленные им в соответствие: ясно, что, к примеру, число "7" или строка "abcd" однозначно и полностью будут определять соответствующий им объект.

Далее определим над объектами составные операции, такие как set, array, и tuple. (Набор составных операций для различных ООСУБД может варьироваться). Любая составная операция ставит в соответствие списку объектов ещё один объект.

array – создаёт массив однотипных объектов. Отношение однотипности задаётся снаружи операции array с помощью объявления принадлежности объектов к типам и классам, см. ниже.

set – создаёт множество однотипных объектов. То же, что и array, но порядок элементов отсутствует. Альтернативой является операция *isa*, задающая, наоборот, отношение принадлежности или инстанцирования. Традиционно в языках определения данных обозначается фигурными скобками {}.

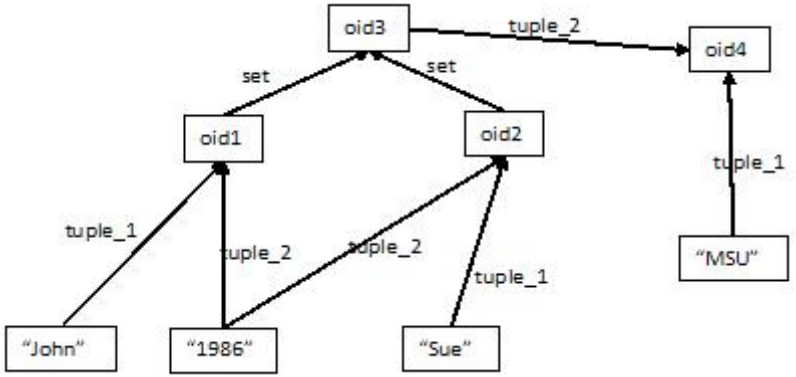
1 В ODMG 3.0 используется термин "литерал".

tuple – создаёт кортеж <возможно, разнородных> объектов. Традиционно в языках определения данных обозначается квадратными скобками [].

В зависимости от конкретной ООСУБД, можно задать и другие операции над объектами, например:

state – ставит в соответствие объекту другой объект, понимаемый как его *состояние*. К примеру, состоянием объекта oid1, адресующего работника предприятия, может служить объект oid2, являющийся результатом применения операции tuple к тройке ("Smith", "$1000", oid3). Необходимость этой операции является предметом дискуссий (дилемма "attributes vs tuple-based states"); среди её преимуществ – возможность явно разрешить важную проблему отделения задачи сравнения объектов от задачи сравнения состояний объектов, среди недостатков – возможная избыточность.

После этих соглашений базу данных можно представить в виде графа, вершинами которого являются объекты, а рёбрами – отношения вида "быть i-ым операндом операции с данным результатом":



oid1 и oid2 – идентификаторы 2 студентов, студент задаётся кортежем из имени и года рождения. Набор (set) этих 2 студентов составляет объект oid3. Объект oid4 – вуз, являющийся кортежем из объекта-названия и объекта-множества своих студентов. Номер после "tuple" означает, какой это по счёту операнд операции составления кортежа.

Уровень схемы

На уровне схемы, как уже сказано, задаются **типы**, **классы**, **отношения** и **роли**, если они есть. Всех их предлагается считать опять же неопределяемыми

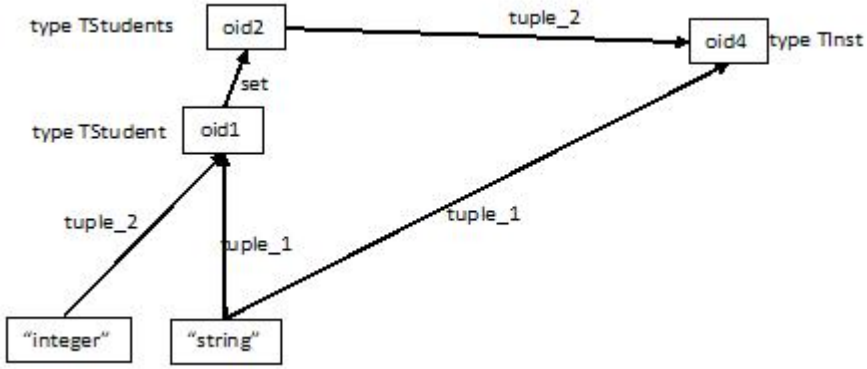
"атомами" схемы данных, то есть задаваемыми исключительно через их соотношения друг с другом и с объектами предыдущего уровня.

Как и на предыдущем уровне, схема данных понимается как некоторый (возможно несвязный) орграф, вершины которого являются указанными выше атомами, а рёбра означают операции создания составных объектов (tuple, set, array...) и прочие специфические операции, к примеру:

state – операция сопоставления состояния, аналогично *state* на уровне данных. А именно, если ООСУБД поддерживает состояния, то вершины-типы в ней мы разбиваем на две разновидности: структурные типы (объекты которых служат как состояния) и абстрактные типы. Далее стрелка *state* соединяет абстрактный тип с его типом-состоянием.

inherits (isa) – операция наследования типов или классов, о ней идёт речь ниже.

Следует заметить, что граф схемы отличается от графа объектов по использованию операций создания однородных наборов (set,array). В графе объектов все вершины-наборы создаются явно указанием всех стрелок *set* или *array* от вершины-набора к вершинам-элементам. В графе схемы же ставится единственная стрелка *set* или *array* от вершины-типа к вершине, означающей однородный набор элементов этого типа.



type TStudent = [name: string, birth_year: integer]

type TStudents={student:TStudent}

type TInst=[name: string, students: TStudents].

Диаграмма типов для БД студентов и вузов, упомянутой выше, и её задание на гипотетическом DDL

Типы и классы: Существенной чертой ООСУБД является выделение разницы между типами и классами. В терминах графа схемы разница следующая: вначале в пуле объектов выделяются все однородные наборы, состав которых задан заранее (к примеру, integer, или перечислимый тип, или множество структурно созданных объектов с одной и той же структурой). В графе схемы на каждый такой набор создаётся вершина, называемая **типом**, затем такие вершины-типы соединяются стрелками структурных операторов, если какие-то из этих типов структурно образованы из других. В противоположность этому, **классом** называют вершину, обозначающую абстрактный, заранее не заданный однородный набор объектов данного типа, что реализуется введением ещё одной особой разновидности стрелки, задающей принадлежность класса к типу.

Иными словами, для вершины-типа заранее можно указать её расширение (список объектов данного типа), а для вершины-класса нельзя.

Соотношение между примитивами тип и класс является достаточно спорным вопросом в различных ООСУБД. Существуют как ООСУБД, в схемах которых используется только один из этих примитивов (к примеру, в системах, написанных на CLOS, часто есть только классы и нет типов), так и ООСУБД, поддерживающие оба примитива, тем более что последнего требует ODMG.

Отношения (есть не во всех моделях ООСУБД): Отношение с точки зрения математической логики – это функция, отображающая некоторое множество однотипных объектов на множество {false,true}.

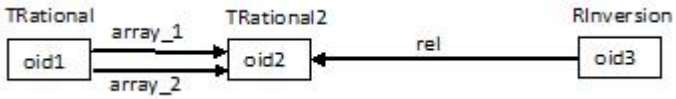
В ООСУБД отношения задаются классами. Класс определяет некоторое множество однотипных объектов. Все объекты, которые входят в данный класс (потенциально могут существовать другие объекты этого же типа с другими значениями атрибутов, не входящие в класс), формируют отношение так же, как формируют отношения кортежи, находящиеся в таблицах РСУБД. Таким образом, в ООСУБД класс задает некоторое отношение так же, как в РСУБД отношение задается таблицей.

Распространённый пример отношения – бинарное отношение, задаваемое на декартовом произведении 2 множеств и означающее, что если для данной пары объектов оно истинно, то эти 2 объекта состоят в некотором "физическом" отношении, например отношении родства между объектами-людьми, а если ложно – то не состоят. Тем самым для задания отношения достаточно на области определения отношения задать подмножество, на котором и только на котором функция истинна.

В ООСУБД для задания отношения между двумя объектами могут применяться ссылки по oid. Если объект o_employee значением своего атрибута works_in_dept имеет ссылку на объект o_dept, то можно говорить что этот объект служащего o_employee находится в отношении works_in_dept с объектом отдела o_dept, т.е. служащий o_employee работает в отделе o_dept. При этом нужно обратить внимание на то, что ссылки в ООСУБД однонаправлены и в отличие от РСУБД нельзя говорить о том что отдел o_dept находится в симметричном отношении "содержит_работника" к служащему o_employee. Это связанно с техническими сложностями реализации в ООСУБД поиска всех объектов, ссылающиеся на некоторый заданный объект. При необходимости такой функционал может быть реализован аналогично тому, как реализован поиск по индексу в современных РСУБД. В качестве примера такого расширения трактовки ссылок можно привести механизм запросов интегрированных в язык программирования (LINQ) разработки компании Microsoft. Этот механизм позволяет найти всех работников заданного отдела:

```
from emp in employees where emp.works_in_dept == o_dept select emp;
```

В графе схемы ООСУБД отношения задаются так же, как и классы, то есть введением вершин-отношений и стрелок *rel*, указывающих на вершину-тип.




```
RInversion i1 = TRational2.Create ((2,5),(5,2));
RInversion i2 = TRational2.Create ((1,3),(3,1));
```

Отношение обратности двух рациональных чисел и добавление в него пар (2/5, 5/2) и (1/3, 3), в предположении наличия C++-подобного языка определения данных и класса TRational

Тем самым введение отношений выглядит избыточным, дублирующим функциональность классов, но при введении стандартных операций над отношениями позволяет фактически создавать в рассматриваемом "графовом фреймворке" реляционные модели данных.

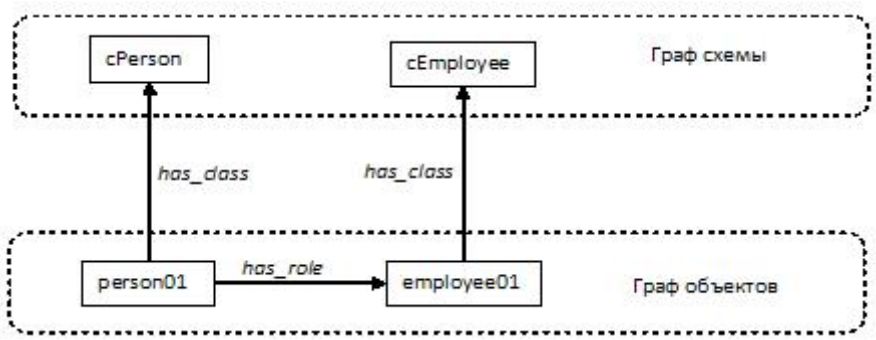
Роли (есть не во всех моделях ООСУБД): Основная задача ролей – реализовать возможность динамического наследования.

Рассмотрим распространённую проблему [18]: пусть в ООБД есть классы "работник" и "человек" – cEmployee и cPerson, и в некоторый момент человек, занесённый в БД как объект класса cPerson, был взят на работу, т.е. возникла необходимость <временно> придать ему поля и методы класса cEmployee, а в будущем, возможно, лишить его присвоенных атрибутов. Это может быть ре-

ализовано созданием нового класса, унаследованного от двух названных (если имеется возможность множественного наследования и в классах нет одинаковых полей наподобие "имя"), либо созданием cEmployee как унаследованного от cPerson. Данные способы представляются неудобными: первый из них нарушает принцип замещения Лисков ("представитель типа должен безопасно заменяться на представителя унаследованного типа"), т.к. классы cEmployee и cPerson лишаются естественного отношения вложенности, второй же приводит к необходимости удалить прежний объект и пересоздать его как объект нового класса.

Указанная проблема называется задачей реализации динамического наследования и может быть разрешена следующим образом. Среди объектов-представителей классов выделим некоторые и назовём их **ролями**. Далее введём в граф объектов особый тип стрелки *has_role* между обычным объектом и объектом-ролью. Смысл такой конструкции – обычному объекту присваивается временная принадлежность к объектам данного класса, возможность "сыграть роль" объекта данного класса.

Возвращаясь к задаче о cPerson и cEmployee, роль можно назначить так:



Далее при необходимости для данного объекта присваиваются или отнимаются роли.

Поведенческий аспект.

Помимо структурной части, хранящей собственно данные, модель всякой ООБД также обладает т.н. поведенческой частью, у которой нет аналога в реляционной модели. Основными элементами поведенческой части являются: метаклассы, классовые атрибуты, операция наследования, методы и функции. [17]

Метаклассы. Для введения метаклассов предлагается несколько изменить структуру данных, переместив вершины-классы из графа схемы в граф объектов (сделав их абстрактными объектами). Для каждого класса, по определению класса выше, при этом необходимо ввести ещё одну вершину, обозначающую набор объектов данного класса, и соединить их стрелкой *set*. Это даёт возможность производить структурные операции над классами, работая с ними как с объектами. Для организации коллекций объектов-классов в граф схемы вводятся новые вершины, называемые **метаклассами**, с которыми объекты-классы можно связывать стрелкой *has_class*.

Классовые атрибуты. Объектные модели с метаклассами позволяют реализовать известные по ООП **классовые атрибуты**, то есть возможность наличия у класса атрибута, значение которого одно и то же для всех объектов данного класса. Более точно, так как классы теперь являются объектами, классовый атрибут есть не что иное, как вершина-объект (обычный), соединённая стрелкой структурной операции с вершиной-классом.

Наследование. Выше указывалось, что существует стрелка *inherits* между типами или классами, обозначающая наследование. В данной объектной модели **наследованием типов** называют операцию создания одного типа из другого с помощью структурного добавления вершин. Это объясняет отождествление в [17] операций *inherits* и *isa*, так как такой подход есть реализация вложенности типов: представители типа-потомка гарантированно имеют в своём составе те же поля, что и представители типа-предка (и, может быть, ещё какие-то), так что могут использоваться вместо представителей типа-предка согласно принципу замещения Лисков.

TEmployee=[name:string, salary:integer]
TEmployeeChief *inherits* TEmployee *add* dept:TDept

Наследование классов есть наследование соответствующих им типов плюс требование унаследовать классовые атрибуты на графе объектов (если модель с метаклассами и классовыми атрибутами), а также унаследовать методы родительского класса.

Методы и функции. Это фундаментальные составляющие ООСУБД, неясным может вначале показаться их место в графе объектов и графе схемы, так как они не являются данными или доменами. Существует подход, при котором они считаются не входящими в объектную модель и внешним образом манипулируют данными. Такая практика используется в ООСУБД, в которых для модификации данных используется непосредственно язык программирования БД, а не язык управления данными, например Caché [18], db4o (хотя они огра-

ниченно поддерживают также управление данными через SQL). Этот подход гораздо проще прочих, но он никак не принимает во внимание целостность данных и контроль целостности.

Существует возможность включить методы и функции в объектную модель, для чего применяется тот факт, что любая математическая функция представима на теоретико-множественном языке как выделенное множество пар вида [входная переменная, выходная переменная]. Иными словами, функция есть некое отношение, а отношения входят в объектную модель. Следует заметить, что представление в виде отношения для функций нескольких переменных неоднозначно: например, функцию, возвращающую для данных двух родителей множество их детей, можно представить как минимум 2 способами.

- Тернарное отношение: (o_person, o_person, {o_person}).
- Вложение бинарных отношений: (o_person, (o_person, {o_person})).

Для полного включения функций в объектную модель следует добавить в неё структурные операции над функциями, хорошо известные из функционального программирования: композиция, apply-to-all и другие (см. ниже о функциональном подходе в математической модели).

Остаётся вопрос, как с помощью такого подхода реализовано включение методов классов (а не абстрактных функций) в граф ООСУБД, но это скорее технический вопрос, решение которого схоже с решением задачи включения отношений, метаклассов ит.д. в объектную модель и поэтому не будет здесь приводиться.

4. Уровень математической модели ООСУБД.

Как уже сказано выше, задача формализации данных ООСУБД схожа с задачей формализации декларативного языка определения данных: обе они предполагают построение объектов или запросов к ним в виде формул некоторой формальной теории. Существует два подхода к созданию такой формальной теории – функциональный и логический (дедуктивный).

При функциональном подходе из всей формальной теории задаются только структурные правила создания формул и, быть может, оценки их истинности, то есть семантика теории. Логический вывод не применяется. Подход получил своё название от функционального программирования, основой которого является построение результата работы программы как результата последовательного вычисления некоторой математической функции, что схоже с последовательным построением формул.

В [17] выделяется два основных функциональных подхода: алгебра select-project-join (SPJ) и многосортное исчисление предикатов высшего порядка (называемое у него полным исчислением объектов).

Основой функционального подхода является задание объектов, базовых функций над ними и структурных операций над функциями. В алгебре SPJ (идеологию которой использует SQL) объектами являются: атомы объектной модели, наборы однородных элементов (set, array), кортежи. Базовыми функциями являются:

- разнообразные элементарные операции над числами и строками (сумма, конкатенация итд);
- σ (pred)(set) – предикат pred на множестве set (aka выборка);
- p (fun)(set) – применение одноместной функции fun к каждому элементу множества set (aka apply-to-all из ФП).

Структурные операции – композиция, кортеж, агрегация (на вход получает однородный набор и бинарную функцию f, например сложение, на выходе выдаёт результат $f(x_1,f(x_2,f(x_3...)...)))$, операция if-then-else и т. д.

Приведём несколько примеров правильно построенных запросов в алгебре SPJ.

Каждый из них допускает 2 формы записи: выражение на некотором SQL-подобном языке и выражение на языке лямбда-исчисления. Кроме того, добавим ещё, забегая вперёд, запись в виде формулы на языке полного исчисления, чтобы провести сравнение. Очевидно, третья запись не есть альтернативная форма записи двух предыдущих, т. к. полное исчисление устроено иначе, чем SPJ-алгебра.

Пусть база данных задана так (пример из [17], но без состояний):

```
o_type o_person = [name:string, b-date:date]
o_type o_empl inherits o_person add [sal:int, children:{c_persons}]
o_type o_dept=[dname:string, budget:int, mgr:c_empls, employees:{c_empls}]
class c_persons: o_person
class c_empls: o_empl
class c_depts: o_dept
…
//Заполнение базы
dept1: o_dept = (какой-то вызов конструктора)
dept2: o_dept = (какой-то вызов конструктора)
…
```

depts: {dept1,dept2,...}

Запрос 1: Найти всех начальников отделов, бюджет которых <отделов> более 1000.

SQL: select mgr from depts where budget>1000
Лямбда-исчисление: p (mgr)(σ (budget>1000)(depts))
Формула: d.mgr: d depts & d.budget>1000

Запрос 2: Для каждого отдела, вывести его название, имена сотрудников с >3 детьми, и имена их детей.

SQL: select dname, (select name, (select name from children)
from employees
where count(children)>3)
Лямбда-исчисление: p(dname, p(name, p(name)(children))
(σ (aggr(+,p(f_1,children))>3)(employees)))(depts)
Формула: e.dname, {d.name, {c.name: c d.children}:
count(d.children)>3}:d e.employees

(где f_1 – базовая функция, принимающая на вход string и на выходе выдающая тождественную 1; допустим, она объявлена выше)

Видно, что полное исчисление объектов отличается от алгебры SPJ в первую очередь тем, что оно допускает операцию принадлежности к типам, то есть является многосортным, а также что оно является исчислением предикатов высшего (выше первого) порядка, так как допускает квантификаторы не только над атомарными объектами. Это достаточно неудобная черта, так как известно, что теории высшего порядка не аксиоматизируемы (не существует набора формул такого, что все остальные правильно построенные формулы из них выводятся), что в свою очередь ведёт к большой вычислительной сложности задачи проверки истинности формул (считаем, что задана семантика формул, то есть отображение множества формул на {true,false}), в то время как в случае аксиоматизируемой разрешимой теории для проверки истинности формулы достаточно было бы проверить её на разрешимость. Для сравнения, реляционная модель основана на исчислении предикатов первого порядка, допускающем аксиоматизацию.

Дедуктивный подход. Этот подход представляет целое направление в современной теории баз данных, причём, что характерно, объектно-ориентированные системы применяют некоторые дедуктивные механизмы – к

примеру, в создании языков запросов (т. н. языки, основанные на правилах) или в задачах контроля целостности.

Общая идея этого подхода – представление запросов СУБД в виде формул некоторой формальной теории, получаемых с помощью логического вывода в этой теории. Логический вывод над множеством формул представляет из себя заранее выделенный набор формул, называемых аксиомами, и правил (схем) вывода, позволяющим ставить в соответствие набору формул ещё одну формулу, называемую синтаксическим следствием из данных формул. За аксиомы обычно берутся базовые предикаты принадлежности литералов к простым типам и заранее определённые отношения. Пример правила [16]:

$$\mathrm{MOTHER\_OF}(X,Y),\ \mathrm{MOTHER\_OF}(Y,Z) \rightarrow \mathrm{GRANDMOTHER\_OF}(X,Z)$$

Здесь MOTHER_OF и GRANDMOTHER_OF – имена предикатов, означающих соответствующие формы родства.

Ясно, что если в логической БД будет задана принадлежность некоторых людей к предикатам MOTHER_OF и GRANDMOTHER_OF в виде аксиом, то база будет иметь возможность обрабатывать запросы вида, к примеру, SELECT * FROM persons p WHERE GRANDMOTHER_OF(p,p0), вычисляя истинность предикатов для разных p с помощью или аксиом непосредственной принадлежности человека к GRANDMOTHER_OF(·,p0), или данного правила вывода.

Значительным плюсом такого подхода является поддержка рекурсивных запросов. Часто приводимый пример – запись транзитивного замыкания.

Пусть есть граф и на множестве его вершин есть отношение child (вторая вершина есть прямой потомок первой), требуется определить его транзитивное замыкание, то есть отношение path_exists (вторая вершина достижима из первой). Для этого можно использовать пару дедуктивных аксиом:

$$\mathrm{child}(x,y) \rightarrow \mathrm{path\_exists}(x,y)$$

$$\mathrm{path\_exists}(x,z),\ \mathrm{child}(z,y) \rightarrow \mathrm{path\_exists}(x,y).$$

В такой аксиоматике формула path_exists (x,y) будет разрешима тогда и только тогда, когда y достижим из x, что и требовалось от БД.

Кроме того, представление БД в дедуктивной форме позволяет единообразно, в виде аксиом, задавать ограничения целостности (например, в виде предикатов принадлежности)

Тем не менее, в применении логического подхода к моделированию ООСУБД есть несколько трудностей. Одна из основных проблем связана с наличием в ООСУБД, возможно, неразрешимой рекурсии в правиле, к примеру если при вычислении тела правила необходимо перебрать элементы отношения, модифицируемого в выводе правила, например:

$$p_1(X,Y) \rightarrow p_2(X,\{Y\})$$

Это правило реализует операцию group by по первому аргументу: для каждого фиксированного объекта X берутся все объекты Y, которые удовлетворяют в паре с этим X отношению p_1, и объединяются в <зависящее от X> множество {Y}, после чего пара из этого X и соответствующего ему объекта-множества {Y} включается в новое отношение p_2, которое следует назвать отношением группировки. Но если положить $p_1=p_2$, то получается, что для обхода множества пар {X,Y} в том числе необходимо знать, входит ли создаваемая пара в отношение, а это неизвестно, т.к. правило ещё не сработало.

Методом борьбы с этой проблемой является введение введение т. н. ограничений стратификации на правила, запрещающих предикату в выводе правила зависеть от себя самого в теле правила.

5. Выводы и основные проблемы.

Представленная структурная модель данных является представлением на языке теории графов целого класса структурных объектных моделей и достаточно точно реализует как основные принципы ООП (oids, структурные объекты, персистентность объектов, инкапсуляцию, наследование, полиморфизм), так и абстракции группировки данных – абстрактные множества, типы, классы. Модель состоит из двух уровней, структурного и поведенческого, первый из которых есть усложнённая версия реляционной модели данных, второй представляет агрегирующие объекты – типы, классы, отношения, функции. Право агрегирующих объектов в данной модели участвовать в тех же операциях, что и индивидуальные объекты, даёт в качестве математической формализации этой структурной модели достаточно выразительную систему – полное объектное исчисление, являющееся разновидностью логики второго порядка, но одновременно это право (наряду с разнообразием базовых элементов ООСУБД и связей между ними) приводит к проблеме невозможности создать единообразное средство проверки корректности объекта теории (чем бы он ни был – формулой структурного типа, предикатом, правилом или чем-либо другим), выраженное на языке этой теории. Иными словами, можно сказать, что ограничения на корректность объектов теории являются пока на сегодняшний день внешними средствами по отношению к теории, в основном выраженными в форме указания конкретных классов некорректных объектов с обозначением пути разрешения проблемы.

Кроме указанной (кратко называемой в [17] "higher-orderness problem") важной является также проблема реализации в данной модели сочетания функционального и логического программирования. Первое представляет аппарат для работы с функциями как с данными, второе – с отношениями как с данными (операции с отношениями также предоставляет, очевидно, реляционное исчисление; известно, что реляционная СУБД является частным случаем логической). Оба аппарата необходимы, т.к. хотя функции и представимы, как сказано выше, в виде отношений, но это представление неудобно. Полной формализации включения функций и отношений в объектную модель пока не создано.

Список литературы

[1] D. H. Fishman et al. Overview of the Iris DBMS. In Object-oriented concepts, databases, and applications, ACM Press. New York, NY, 1989, 219-250

[2] Won Kim, Jorge F. Garza, Nathaniel Ballou, Darrell Woelk. Architecture of the ORION Next-Generation Database System // IEEE Trans. Knowledge and Data Eng.- 2, N 1.- 1990.- 109-124

[3] Tomothy Andrews, Craig Harris. Combining Language and Database Advances in an Object-Oriented Development Environment GemStone Object-Oriented DBMS // Proc. OOPCLA'87, Orlando, Fla, USA, Oct. 4-8, 1987.- 430-440

[4] Hiroshi Ishikawa et al. An Object-Oriented Database System Jasmine: Implementation, Application, and Extension. IEEE Transactions on Knowledge and Data Engineering, Volume 8 Issue 2, April 1996, 285 - 304

[5] Christophe Lecluse, Philippe Richard, Fernando Velez. O2, an Object-Oriented Data Model // Proc. ACM SIGMOD Int. Conf. Manag. Data, Chicago, Ill, USA, June 1-3, 1988, ACM SIGMOD Record.- 17, N 3.- 1988.- 424-433

[6] Malcolm Atkinson, Francois Bancilhon, David DeWitt, Klaus Dittrich, David Maier, and Stanley Zdonik: “The Object-Oriented Database System Manifesto”, Proc. 1st International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan (1989). New York, N.Y.: Elsevier Science (1990). (Имеется русский перевод: М. Аткинсон и др. “Манифест систем объектно-ориентированных баз данных”, *СУБД,* No. 4, 1995, http://citforum.ru/database/classics/oo_manifesto/)

[7] 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, 1996, http://citforum.ru/database/classics/manifest/)

[8] Hugh Darwen and C. J. Date: *The Third Manifesto. ACM SIGMOD Record 24,* No. 1 (March 1995). (Имеется русский перевод: Х. Дарвин, К. Дейт. “Третий манифест”, *СУБД,* No. 1, 1996, http://citforum.ru/database/classics/third_manifesto/)

[9] Стандарты ODMG. http://www.odbms.org/ODMG/OG/

[10] “The Object Data Standard: ODMG 3.0”. Edited by R.G.G. Cattel, Douglas K. Barry. Morgan Kauffmann Publishers, 2000
[11] Jim Melton, Alan R. Symon. “SQL:1999. Understanding Relational Language Components”. Morgan Kaufmann Publishers, 2002
[12] William R. Cook and Carl Rosenberger. Native Queries for Persistent Objects. February 01, 2006, http://drdobbs.com/windows/184406432
[13] Elisa Flasko. Introducing LINQ to Relational Data. Microsoft Data Platform Development Technical Article, January 2008, http://msdn.microsoft.com/en-us/library/cc161164.aspx
[14] db4o: http://www.db4o.com/
[15] 4th Generation Standard for Object Databases on its Way, NEWS 2/18/2006, http://www.odbms.org/About/News/20060218.aspx
[16] К. Дж. Дейт – Введение в системы баз данных, 8-е издание.: Пер. с англ. – М.: Издательский дом "Вильяме", 2005
[17] C.Beeri – A formal approach to object-oriented databases. – Data & Knowledge Engineering, vol 5 (1990), p. 353-382
[18] InterSystems Caché, http://www.intersystems.com/cache/