Бойко И.

Объектно-ориентированные СУБД.

1. Введение

Концепция *объектно-ориентированного программирования* подразумевает, что основой управления процессом реализации программы является передача сообщений объектам. Поэтому объекты должны определяться совместно с сообщениями, на которые они должны реагировать при выполнении программы. В этом состоит главное отличие ООП от процедурного программирования, где отдельно определённые структуры данных передаются в процедуры (функции) в качестве параметров. Таким образом, объектно-ориентированная программа состоит из объектов – отдельных фрагментов кода, обрабатывающего данные, которые взаимодействуют друг с другом через определённые интерфейсы.

Объектно-ориентированный язык программирования должен обладать следующими свойствами:

1. **абстракции** – формальное о качествах или свойствах предмета путем мысленного удаления некоторых частностей или материальных объектов;
2. **инкапсуляции** – механизма, связывающего вмести код и данные, которыми он манипулирует, и защищающего их от внешних помех и некорректного использования;
3. **наследования** – процесса, с помощью которого один объект приобретает свойства другого, т.е. поддерживается иерархической классификации;
4. **полиморфизма** – свойства, позволяющего использовать один и тот же интерфейс для общего класса действий.

Разработка объектно-ориентированных программ состоит из следующих последовательных работ:

- определение основных объектов, необходимых для решения данной задачи;
- определение закрытых данных (данных состояния) для выбранных объектов;
- определение второстепенных объектов и их закрытых данных;
- определение иерархической системы классов, представляющих выбранные объекты;
- определение ключевых сообщений, которые должны обрабатывать объекты каждого класса;

- разработка последовательности выражений, которые позволяют решить поставленную задачу;
- разработка методов, обрабатывающих каждое сообщение;
- очистка проекта, то есть устранение всех вспомогательных промежуточных материалов, использовавшихся при проектировании;
- кодирование, отладка, компоновка и тестирование.

Объектно-ориентированное программирование позволяет программисту моделировать объекты определённой предметной области путем программирования их содержания и поведения в пределах класса. Конструкция «класс» обеспечивает механизм инкапсуляции для реализации абстрактных типов данных. Инкапсуляция как бы скрывает и подробности внутренней реализации типов, и внешние операции и функции, допустимые для выполнения над объектами этого типа.

2. Что такое объектно-ориентированное программирование

Элементы объектно-ориентированного программирования (ООП) появились в начале 70-х годов в языке моделирования Симула, затем получили свое развитие, и в настоящее время ООП принадлежит к числу ведущих технологий программирования.

Основная цель ООП, как и большинства других подходов к программированию – повышение эффективности разработки программ. Идеи ООП оказались плодотворными и нашли применение не только в языках программирования, но и в других областях Computer Science, например, в области разработки операционных систем.

Появление ООП было связано с тем наблюдением, что компьютерные программы представляют собой описание *действий*, выполняемых над различными *объектами*. В роли последних могут выступать, например, графические объекты,

записи в базах данных или совокупности числовых значений. В традиционных методах программирования изменение данных или правил и методов обработки часто приводило к необходимости значительного изменения программы. Всякое существенное изменения программы – это большая неприятность для программиста, так как при этом увеличивается вероятность ошибок, вследствие чего возрастает время, необходимое для «доводки» программы. Использование ООП позволяет выйти из такой ситуации с минимальными потерями, сводя необходимую модификацию программы к её расширению и дополнению. Необходимо заметить, что ООП не является панацеей от всех программистских бед, но его ценность как передовой технологии программирования несомненна. Изучение идей и методов ООП может существенно упростить разработку и отладку сложных программ.

Мы уже привыкли использовать в своих программах процедуры и функции для программирования тех сложных действий по обработке данных, которые приходится выполнять многократно. Использование подпрограмм в своё время было важным шагом на пути к увеличению эффективности программирования. Подпрограмма может иметь формальные предметы, которые при обращении к ней заменяются фактическими предметами. В этом случае есть опасность вызова подпрограммы с неправильными данными, что может привести к сбою программы и её аварийному завершению при выполнении. Поэтому естественным обобщением традиционного подхода к программированию является объединение данных и подпрограмм (процедур и функций), предназначенных для их обработки.

3. Объекты

Базовым в объектно-ориентированном программировании является понятие *объекта.* Объект имеет определённые свойства. Состояние объекта задаётся значениями его признаков. Объект «знает», как решать определённые задачи, то есть располагает методами решения. Программа, написанная с

использованием ООП, состоит из объектов, которые могут взаимодействовать между собой.

Ранее отмечалось, что программная реализация объекта представляет собой объединение данных и процедур их обработки. Переменные объектного типа называют *экземплярами* объекта. Здесь требуется уточнение – экземпляр можно лишь формально назвать переменной. Его описание даётся в предложение описания переменных, но в действительности экземпляр – нечто большее, чем обычная переменная.

В отличие от типа «запись», объектный тип содержит не только поля, описывающие данные, но также процедуры и функции, описания которых содержится в описании объекта. Эти процедуры и функции называют *методами.* Методам объекта доступны его поля. Следует отметить, что методы и их параметры определяются в описании объекта, а их реализация даётся вне этого описания, в том мест программы, которое предшествует вызову данного метода. В описании объекта фактически содержаться лишь *шаблоны* обращения к методам, которые необходимы компилятору для проверки соответствия количества параметров и их типов при обращении к методам. Вот пример описания объекта[1]:

```
Type
        Location = object
                X,Y: Integer;
                Procedure Init(InitX, InitY: Integer);
                Function GetX: Integer;
                Function GetY: Integer;
        End;
```

Здесь описывается объект, который может использоваться в дальнейшем, скажем, в графическом режиме и который предназначен для определения положения на экране произвольного графического элемента. Объект описывается с помощью зарезервированных слов

[1] Выполняется на языке Turbo Pascal, начиная с версии 5.0. Далее все примеры даны для выполнения на этом языке программирования.

object…end, между которыми находиться описание полей и методов. В нашем примере объект содержит два поля для хранения значений графических координат, а так же для описания процедуры и двух функций - это *методы* данного объекта. Процедура предназначена для задания первоначального положения объекта, а функция – для считывания его координат.

4. Инкапсуляция

Инкапсуляция является важнейшим свойством объектов, на котором строится объектно-ориентированное программирование. Инкапсуляция заключается в том, что объект скрывает в себе детали, которые несущественны для использования объекта. В традиционном подходе к программированию с использованием глобальных переменных программист не был застрахован от ошибок, связанных с использованием процедур, не предназначенных для обработки данных, связанных с этими переменными. Предположим, например, что имеется «не-ООП» программа, предназначенная для начисления заработной платы сотрудникам некой организации, а в программе имеются два массива. Один массив хранит величину заработной платы, а другой – телефонные номера сотрудников (для составления отчёта для налоговой инспекции). Что произойдёт, если программист случайно перепутает эти массивы? Очевидно, для бухгалтерии начнутся тяжёлые времена. *«Жёсткое» связание данных и процедур их обработки в одном объекте позволит избежать неприятностей такого рода.* Инкапсуляция и является средством организации доступа к данным ***только через соответствующие методы.***

В нашем примере описание объекта процедура инициализации Init и функции GetX и GetY уже не существуют как отдельные самостоятельные объекты. Это неотъемлемые части объектного типа Location. Если в программе имеется описание нескольких переменных указанного типа, то для каждой переменной резервируется своя собственная область памяти для хранения данных, а указатели на точки входа в

процедуру и функции – общие. Вызов каждого метода возможен только с помощью составного имени, явно указывающего, для обработки каких данных предназначен данный метод.

5. Наследование

Наследование – это ещё одно базовое понятие объектно-ориентированного программирования. Наследование позволяет определять новые объекты, используя свойства прежних, дополняя или изменяя их. Объект-наследник получает все поля и методы «родителя», к которым он может добавить свои собственные поля и методы или заменить («перекрыть») их своими методами. Пример описания объекта-наследника даётся ниже:

```
Tipe
        Point = object(Location)
                Visible: Boolean;
                Procedure Int(IntX, IntY: Integer);
                Procedu
                Procedu
                Functio
                Procedu
        End;
```

Наследником здесь является объект Point, описывающий графическую точку, а родителем – объект Location. Наследник не содержит описание полей и методов родителя. Имя последнего указывается в круглых скобках после слова object.

Из методов наследника можно вызывать методы родителя. Для создания наследника не требуется иметь исходный текст объекта родителя. Объект-родитель может быть уже в составе оттранслированного модуля.

В чём привлекательность наследования? Если некий объект был уже определён и отлажен, он может быть использован и в других программах. При этом может оказаться, что новая задача отличается от предыдущей, и возникает необходимость некоторой модификации как данных, так и методов их обработки. Программисту приходится решать дилемму – создания объектов заново или использовать результаты предыдущей работы, применяя механизм наследования. Первый путь менее эффективен, так как требует дополнительных затрат времени на отладку и тестирование. Во втором случае часть этой работы оказывается выполненной, что сокращает время на разработку новой программы. Программист при этом может и не знать деталей реализации объекта-родителя.

В нашем примере к объекту, связанному с определением положения графического элемента, просто добавилось новое поле, описывающее признак видимости графической точки, и несколько новых методов, связанных с режимом отображения точки и её преобразованиями.

6. Виртуальные методы

Наследование позволяет создавать иерархические, связанные отношениями подчинения, структуры данных. Следует, однако, заметить, что при использовании этой возможности могут возникнуть проблемы. Предположим, что в нашей графической программе необходимо определить объект Circle, который является потомком другого объекта Point:

```
Type

                                    Circle  = object (point)

                                                        Radius:
```

```
Procedu
Procedu
Procedu
Procedu
End;
```

Новый объект Circle соответствует окружности. Поскольку свойства окружности отличаются от свойств точки, в объекте-наследнике придется изменять процедуры Show и Hide, которые отображают окружность и удаляют её изображение с экрана. Может оказаться, что метод Init (см. предыдущий пример) объекта Circle, унаследованный от объекта Point, также использует методы Show и Hide, впредь во время трансляции объекта Point использует ссылки на старые методы. Очевидно в объекте Circle они работать не будут. Можно, конечно, попытаться «перекрыть» метод Init. Чтобы это сделать, нам придётся полностью воспроизвести текст метода. Это усложни работу, да и не всегда возможно, поскольку исходного текста программы может не оказаться под рукой (если объект-родитель уже находиться в оттранслированном модуле).

Для решения этой проблемы используется *виртуальный метод*. Связь между виртуальным методом и вызывающими их процедурами устанавливается не во время трансляции (это называется *ранним связанием*), а во время выполнения программы *(позднее связание*.

Чтобы использовать виртуальный метод, необходимо в описании объекта после заголовка метода добавить ключевое слово virtual. Заголовки виртуальных методов родителя и наследника должны в точности совпадать. Инициализация экземпляра объекта, имеющего виртуальные методы, должна выполняться с помощью специального метода – *конструктора.* Конструктор обычно присваивает полям объекта начальные значения и выполняет другие действия по инициализации объекта. В заголовке метода-конструктора слово procedure заменяется словом constructor. Действия обратные действиям конструктора, выполняет ещё один специальный метод – *деструктор.* Он описывается словом destructor.

Конструктор выполняет действия по подготовке позднего связывания. Эти действия заключаются в создании указателя на

таблицу виртуальных методов, которая в дальнейшем используется для поиска методов. Таблица содержит адреса всех виртуальных методов. При вызове виртуального метода по его имени определяется адрес, а затем по этому адресу передается управление.

У каждого объектного типа имеется своя собственная таблица виртуальных методов, что позволяет одному и тому же оператору вызывать разные процедуры. Если имеется несколько экземпляров объектов одного типа, то недостаточно вызвать конструктор для одного из них, а затем просто скопировать этот экземпляр во все остальные. Каждый объект должен иметь свой собственный конструктор, который вызывается для каждого экземпляра. В противном случае возможен сбой в работе программы.

Заметим, что конструктор или деструктор, могут быть «пустыми», то есть не содержать операторов. Весь необходимый код в этом случае создается при трансляции ключевых слов construct и destruct.

7. Динамическое создание объектов

Переменные объектного типа могут быть *динамическими*, то есть размещаться в памяти только во время их использования. Для работы с динамическими объектами используются расширенный синтаксис процедур New и Dispose. Обе процедуры в этом случае содержат в качестве второго параметра вызов конструктора или деструктора для выделения или освобождения памяти переменной объектного типа:

```
New(P, Construct)
```

или

```
Dispose(P, Destruct)
```

Где P – указатель на переменную объектного типа, а Construct или Destruct – конструктор и деструктор этого типа.

Действие процедуры New в случае расширенного синтаксиса равносильно действию следующей пары операторов:

```
New(P);
P^.Construct;
```

Эквивалентом Dispose является следующее:

```
P^Dispose;
Dispose(P)
```

Применение расширенного синтаксиса не только улучшает читаемость исходного кода, но и генерирует более короткий и эффективный исполняемый код.

8. Полиморфизм

Полиморфизм заключается в том, что одно и то же имя может соответствовать различным действиям в зависимости от типа объекта. В тех примерах, которые рассматривались ранее, полиморфизм проявлялся в том, что метод Init действовал по-разному в зависимости от того, является объект точкой или окружностью. Полиморфизм напрямую связан с механизмом позднего связывания. Решение о том, какая операция должна быть выполнена в конкретной ситуации, принимается во время выполнения программы.

Следующий вопрос, связанный с использованием объектов, заключается в совместимости объектных типов. Полезно знать следующее. Наследник сохраняет свойства совместимости с другими объектами своего родителя. В правой части оператора присваивания вместо типов родителя можно

использовать типы наследника, но не наоборот. Таким образом, в нашем примере допустимы присваивания:

```
Var
                    Alocation : Location;
                      Apoin : Point;
    Acircle : Circle;
Alocation :=Apoint
Apoint := Acrcle;
Alocation := Acircle;
```

Дело в том, что наследник может быть более сложным объектом, содержащим поля и методы, поэтому присваиваемые значения экземпляра объекта-родителя экземпляру объекта-наследника может оставить некоторые поля неопределёнными и, следовательно, представляет потенциальную опасность. При выполнении оператора присвоения копируются только те поля данных, которые являются общими для обоих типов.

Оглавление

1. 20 лет эволюции программного обеспечения. 15

2. Реляционные базы данных. 16

3. Объектно-реляционные методы. 18

4. Объектно-ориентированные базы данных. 20

4.1 Why ODBMS? 20

4.2 Спорные моменты технологии. 22

4.3 Стандарты объектных баз данных. 25

4.4 Поставщики ООСУБД. 29

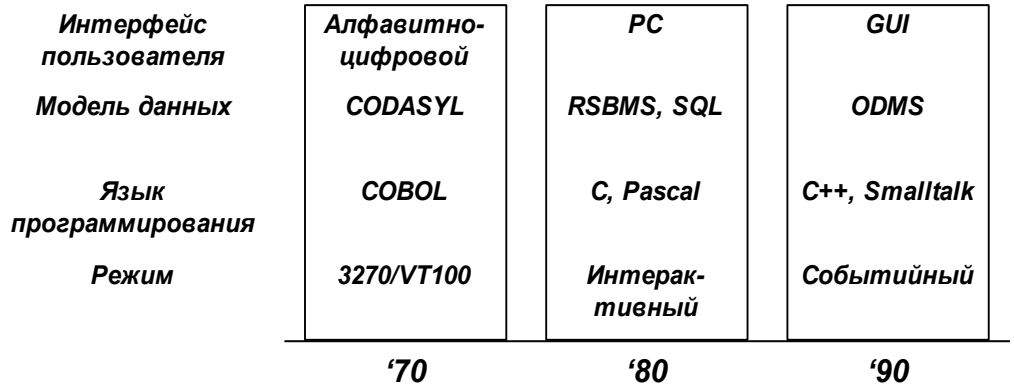
5. Заключение. 31

6. Глоссарий 33

1. 20 лет эволюции программного обеспечения.

Управление информацией всегда было основной сферой применения компьютеров и, надо думать, будет играть еще большую роль в будущем. *Системы управления базами данных*[1] *(СУБД, DBMS – Database Management System)* на протяжении всего пути развития компьютерной техники совершенствовались, поддерживая все более сложные уровни абстрактных данных, заданных пользователем, и обеспечивая взаимодействие компонентов, распределенных в глобальных сетях и постепенно интегрирующихся с телекоммуникационными системами. Позволив себе рассуждения в стиле Билла Гейтса, предположим, что результатом будет становление систем управления информацией одной из частей повседневной жизни каждого.

История развития компьютерной техники – это история непрерывного движения от языка и уровня коммуникации машины к уровню пользователя. Если первые машины требовали от пользователя оформления того, что ему нужно (то есть написания программ), в машинных кодах, то языки программирования четвертого уровня (*4GLs*) позволяли конечным пользователям, не являющимся профессиональными программистами, получать доступ к информации без детального описания каждого шага, но только с встроенными предопределенными типами



Источник: POET Technical Reference - White Paper
© POET Software, 1996

Рисунок 1

данных – например, таблицами.

Последним шагом в этом направлении стала *объектно-ориентированная технология*, радикально изменившая сферу разработки программного обеспечения уже в 1990-х годах (Рисунок 1). Объектно-ориентированный подход позволяет упаковывать данные и код для их

[1] Термины, выделенные курсивом, как правило, приведены в словарике на стр. 21

обработки вместе. Таким образом практически снимается ограничение на типы данных, позволяя работать на любом уровне абстракции.

Эволюция систем управления информацией шла параллельно этому прогрессу, начиная с низкоуровневых программ, которые, например, напрямую производили операции чтения и записи со всей памятью без ограничения доступа, лентой, цилиндрами и дорожками диска и более высокоуровневыми средствами – файловыми системами, которые оперировали с такими понятиями, как массивы, записи и индексы для повышения производительности. Базы данных в свою очередь начинали с модели записей и индексов (ISAM и др.), приобретая со временем способность восстановления после сбоев, проверки целостности данных и возможности работы нескольких пользователей одновременно. Эти ранние модели данных (CODASYL) относились скорее к уровню машинной ориентации. В дальнейшем *реляционные базы данных,* пришедшие на смену в 1980-х годах, приобрели механизм *запросов,* позволяющий пользователю указать требуемое, предоставив СУБД самой оптимальным образом найти результат, используя динамическую индексацию.

Обьектно-ориентированные СУБД (*ООСУБД*) стали разрабатываться с середины 80-х годов в основном для поддержки приложений САПР. Сложные структуры данных систем автоматизированного проектирования оказалось очень удобно оформлять в виде объектов, а технические чертежи проще хранить в базе данных, чем в файлах. Это позволяет обойтись без декомпозиции графических структур на элементы и записи их в файлы после завершения работы с чертежом, выполнения обратной операции при внесении любого изменения. Если типичные реляционные базы данных имеют связи глубиной в два уровня, то иерархическая информация чертежей САПР обычно включает порядка десяти уровней, что требует достаточно сложных операций для “сборки” результата. Объектные базы данных хорошо соответствовали подобным задачам, и эволюция многих СУБД началась именно с рынка САПР.

Между тем рынок САПР был быстро насыщен, и в начале 90-х годов производители ООСУБД обратили внимание на другие области применения, уже прочно занятые реляционными СУБД. Для этого потребовалось оснастить ООСУБД функциями оперативной обработки транзакций (OLTP), утилитами администратора баз данных (database administrator – DBA), средствами резервного копирования/восстановления и т. д. Работы в данном направлении продолжаются и сегодня, но уже можно сказать, что переход к коммерческим приложениям идет достаточно успешно.

2. Реляционные базы данных.

В *реляционных базах данных (Relational Database System, RDBS)* все данные отображаются в двумерных таблицах. База данных, таким образом, это ни что иное, как набор таблиц. RDBS и ориентированные на записи системы организованы на основе стандарта B-Tree или методе доступа,

основанном на индексации – Indexed Sequential Access Method (ISAM) и являются стандартными системами, использующимися в большинстве современных программных продуктов. Для обеспечения комбинирования таблиц для определения связей между данными, которые практически полностью отсутствуют в большинстве программных реализаций B-Tree и ISAM, используется языки, подобные *SQL* (IBM), Quel (Ingres) и RDO (Digital Equipment), причем стандартом отрасли в настоящее время стал язык SQL, поддерживаемый всеми производителями реляционных СУБД.

Оригинальная версия SQL – это интерпретируемый язык, предназначенный для выполнения операций над базами данных. Язык SQL был создан в начале 70-х как интерфейс для взаимодействия с базами данных, основанными на новой для того времени реляционной теории. Реальные приложения обычно написаны на других языках, генерирующих код на языке SQL и передающих их в СУБД в виде текста в формате ASCII. Нужно отметить также, что практически все реальные реляционные (и не только реляционные) системы помимо реализации стандарта ANSI SQL, известного сейчас в последней редакции под именем SQL2 (или SQL-92), включают в себя дополнительные расширения, например, поддержка архитектуры клиент-сервер или средства разработки приложений.

Строки таблицы составлены из полей, заранее известных базе данных. В большинстве систем нельзя добавлять новые типы данных. Каждая строка в таблице соответствует одной записи. Положение данной строки может изменяться вместе с удалением или вставкой новых строк.

Чтобы однозначно определить элемент, ему должны быть сопоставлены поле или набор полей, гарантирующих уникальность элемента внутри таблицы. Такое поле или поля называются *первичным ключом (primary key)* таблицы и часто являются числами. Если одна таблица содержит первичным ключ другой, это позволяет организовать связь между элементами разных таблиц. Это поле называется *внешним ключом (foreign key).*

Так как все поля одной таблицы должны содержать постоянное число полей заранее определенных типов, приходится создавать дополнительные таблицы, учитывающие индивидуальные особенности элементов, при помощи внешних ключей. Такой подход сильно усложняет создание сколько нибудь сложных взаимосвязей в базе данных. Желающим убедится, что это действительно так и не пожалевшим на это определенный отрезок времени, компания POET Software любезно предоставляет возможность ознакомиться с примером в своей *“белой книге”* “POET Technical Reference”. База данных рядового предприятия общепита (клиенты – Джордж Буш и Эдди Мэрфи) состоит из четырех таблиц.

Еще один крупный недостаток реляционных баз данных – это высокая трудоемкость манипулирования информацией и изменения связей.

3. Объектно-реляционные методы.

Несмотря на рассмотренные в п. 2 недостатки реляционных баз данных, они обладают рядом достоинств:

- разделение таблиц разными программами;
- развернутый “код возврата” при ошибках;
- высокая скорость обработки запросов (команда `SELECT` языка SQL; результатом выборки является таблица, которая содержит поля, удовлетворяющие заданному критерию);
- сама концепция объектных баз данных довольно сложна и

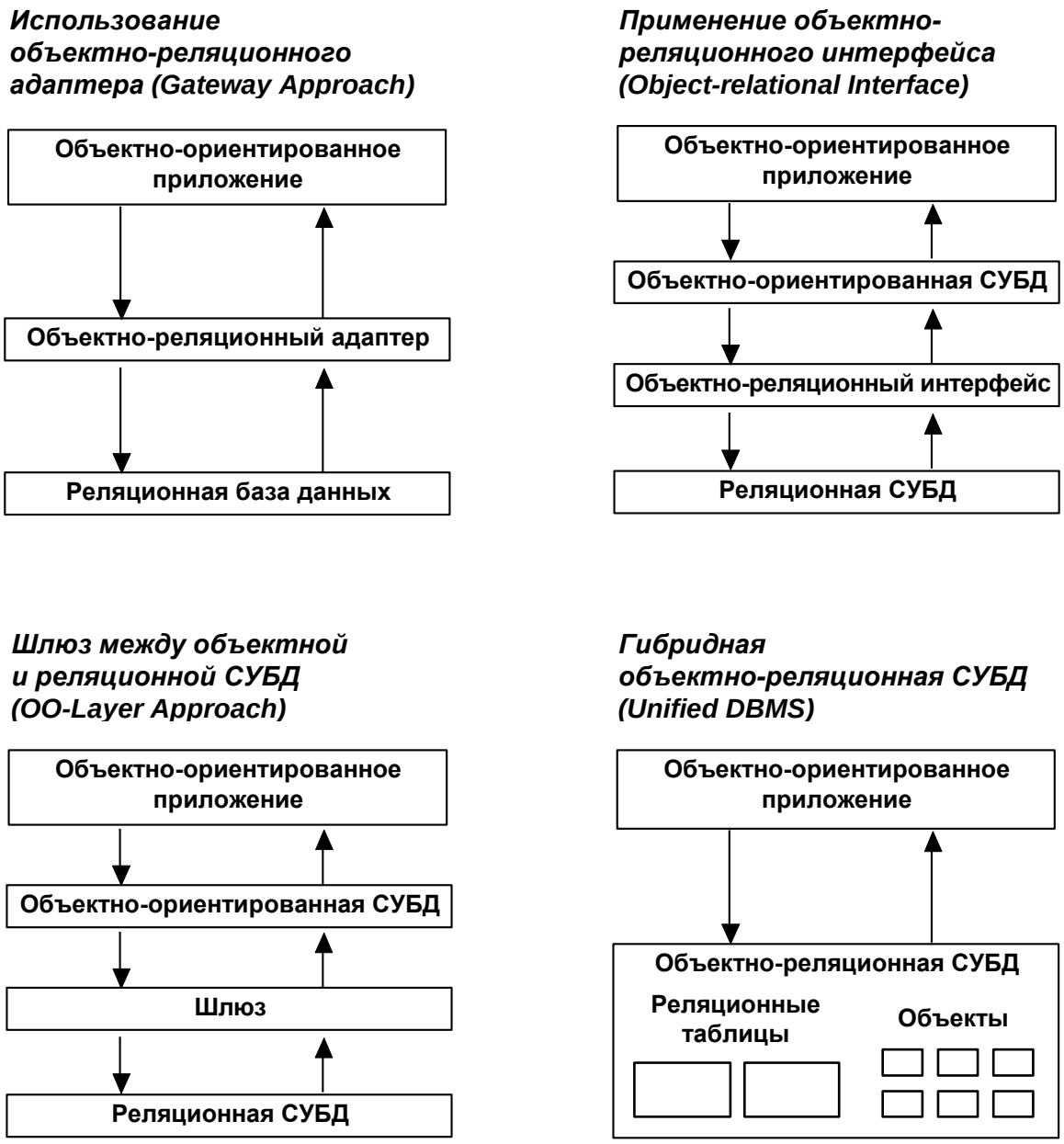


Рисунок 2 Возможные подходы к объединению объектных и реляционных БД.

требует от программистов серьезного и длительного обучения;

- относительно высокая скорость при работе с большими объемами данных.

Кроме того, во всем мире значительные средства уже инвестированы в реляционные СУБД. Многие организации не уверены, что затраты, связанные с переходом на объектные базы данных, окупятся.

Поэтому многие пользователи заинтересованы в комбинированном подходе, который бы им позволил воспользоваться достоинствами объектных баз данных, не отказываясь полностью от своих реляционных БД. Такие решения действительно существуют. Если переход от реляционной базы к объектной обходится слишком дорого, то применение последней в качестве расширения и дополнения реляционных СУБД часто является более экономичной альтернативой. Компромиссные решения позволяют соблюсти баланс между объектами и реляционными таблицами (Рисунок 2).

Объектно-реляционные адаптеры. Этот метод предполагает использование так называемого объектно-реляционного адаптера, который автоматически выделяет программные объекты и сохраняет их в реляционных базах данных. Объектно-ориентированные приложение работает как рядовой пользователь СУБД. Несмотря на некоторое снижение производительности, такой вариант позволяет программистам целиком сконцентрироваться на объектно-ориентированной разработке. Кроме того, все имеющиеся на предприятии приложения по-прежнему могут обращаться к данным, хранящимся в реляционной форме.

Некоторые объектные СУБД, например GemStone компании GemStone Systems, могут сами выполнять роль мощного объектно-реляционного адаптера, позволяя объектно-ориентированным приложениям обращаться к реляционным БД.

Объектно-реляционные адаптеры, такие как Odapter компании Hewlett-Packard для СУБД Oracle, можно с успехом использовать во многих областях, например в качестве связующего ПО, объединяющего объектно-ориентированные приложения с реляционными СУБД.

Объектно-реляционные шлюзы. При использовании такого метода пользователь взаимодействует с БД при помощи языка ООСУБД, а шлюз заменяет все объектно-ориентированные элементы этого языка на их реляционные компоненты. За это опять приходиться расплачиваться производительностью. Например, шлюз должен преобразовать объекты в набор связей, сгенерировать оригинальные идентификаторы (original identifier – OID) объектов и передать это в реляционную БД. Затем шлюз должен каждый раз, когда используется интерфейс реляционной СУБД, преобразовывать OID, найденный в базе, в соответствующий объект, сохраненный в РСУБД.

Производительность в рассмотренных двух подходах зависит от способа доступа к реляционной базе данных. Каждая РСУБД состоит из двух уровней: уровня управления данными (data manager layer) и уровня

управления носителем (storage manager layer). Первый из них обрабатывает операторы на языке SQL, а второй отображает данные в базу. Шлюз или адаптер могут взаимодействовать как с уровнем данных (то есть обращаться к РСУБД при помощи SQL), так и с уровнем носителя (вызовами процедур низкого уровня). Производительность в первом случае намного ниже (например, система OpenODB фирмы Hewlett-Packard, которая может выполнять роль шлюза, поддерживает только на высоком уровне).

Гибридные СУБД. Еще одним решением может стать создание гибридных объектно-реляционных СУБД, которые могут хранить и традиционные табличные данные, и объекты. Многие аналитики считают, что будущее за такими гибридными БД. Ведущие поставщики реляционных СУБД начинают (или планируют) добавлять к своим продуктам объектно-ориентированные средства. В частности, Sybase и Informix собираются в следующих версиях СУБД ввести поддержку объектов. Подобные разработки намерены вести и независимые фирмы. Например, компания Shores готовится оснастить объектно-ориентированными средствами СУБД Oracle8, выпуск которой намечен на конец 1996 г.

С другой стороны, производители объектных СУБД, такие как компания Object Design, сознают, что объектно-ориентированные базы данных в обозримом будущем не заменят реляционные СУБД. Это вынуждает их создавать шлюзы для поддержки реляционных и иерархических баз данных иди различного рода интерфейсы, характерным примером которых является объектно-реляционный интерфейс Ontos Integration Server фирмы Ontos, применяемый в сочетании с ее ООБД Ontos/DB.

4. Объектно-ориентированные базы данных.

4.1 Why ODBMS?

“Белыми книгами” с названием, вынесенным в заголовок, с избытком снабдит любая компания, занимающаяся объектными базами данных. Кое-что о преимуществах и недостатках объектно-ориентированных СУБД уже упоминалось выше, подведем в таком случае итог.

Объектно-ориентированные базы данных применяются с конца 1980-х для обеспечения управления базами данных приложениями, построенными в соответствии с концепцией объектно-ориентированного программирования. Объектная технология расширяет традиционную методику разработки приложений новым моделированием данных и методами программирования. Для повторного использования кода и улучшения сохранности целостности данных в объектном программировании данные и код для их обработки организованы в объекты. Таким образом, практически полностью снимаются ограничения на типы данных.

Если данные состоят из коротких, простых полей фиксированной длины (имя, адрес, баланс банковского счета), то лучшим решением будет применение реляционной базы данных. Если, однако, данные содержат вложенную структуру, динамически изменяемый размер, определяемые пользователем произвольные структуры (мультимедиа, например), представление их в табличной форме будет, как минимум, непростым. В то же время в ООСУБД каждая определенная пользователем структура – это объект, непосредственно управляемый базой данных.

В РСУБД связи управляются пользователем, создающим внешние ключи. Затем для обнаружения связей динамически во время выполнения система просматривает две (или больше) таблицы, сравнивая внешние ключи до достижения соответствия. Этот процесс, называемый объединением (join), является слабой стороной реляционной технологии. Более двух или трех уровней объединений – сигнал, чтобы искать лучшее решение. В ООСУБД пользователь просто объявляет связь, и СУБД автоматически генерирует методы управления, динамически создавая, удаляя и пересекая связи. Ссылки при этом прямые, нет необходимости в просмотре и сравнении или даже поиске индекса, который может сильно сказаться на производительности. Таким образом, применение объектной модели предпочтительнее для баз данных с большим количеством сложных связей: перекрестных ссылок, ссылок, связывающих несколько объектов с несколькими (many-to-many relationships) двунаправленными ссылками.

В отличие от реляционных, ООСУБД полностью поддерживают объектно-ориентированные языки программирования. Разработчики, применяющие C++ или Smalltalk, имеют дело с одним набором правил (позволяющих использовать такие преимущества объектной технологии, как *наследование, инкапсуляция* и *полиморфизм*). Разработчик не должен прибегать к трансляции объектной модели в реляционную и обратно. Прикладные программы обращаются и функционируют с объектами, сохраненными в базе данных, которая использует стандартную объектно-ориентированную семантику языка и операции. Напротив, реляционная база данных требует, чтобы разработчик транслировал объектную модель к поддерживаемой модели данных и включил подпрограммы, чтобы обеспечить это отображение во время выполнения. Следствием являются дополнительные усилия при разработке и уменьшение эффективности.

И, наконец, ООСУБД подходят (опять же без трансляций между объектной и реляционной моделями) для организации распределенных вычислений. Традиционные базы данных (в том числе и реляционные и некоторые объектные) построены вокруг центрального сервера, выполняющего все операции над базой. По существу, эта модель мало отличается от мэйнфреймовой организации 60-х годов с центральной ЭВМ – мэйнфреймом (mainframe), выполняющей все вычисления, и пассивных терминалов. Такая архитектура имеет ряд недостатков, главным из которых является вопрос масштабируемости. В настоящее время рабочие станции (клиенты) имеют вычислительную мощность порядка 30 - 50 % мощности сервера базы данных, то есть большая часть вычислительных ресурсов распределена среди клиентов. Поэтому все больше приложений,

и в первую очередь базы данных и средства принятия решений, работают в распределенных средах, в которых объекты (объектные программные компоненты) распределены по многим рабочим станциям и серверам и где любой пользователь может получить доступ к любому объекту. Благодаря стандартам межкомпонентного взаимодействия (об этом позже) все эти фрагменты кода комбинируются друг с другом независимо от аппаратного, программного обеспечения, операционных систем, сетей, компиляторов, языков программирования, различных средств организации запросов и формирования отчетов и динамически изменяются при манипулировании объектами без потери работоспособности.

4.2 Спорные моменты технологии.

Все ООСУБД по определению поддерживают сохранение и разделение объектов. Но, когда дело доходит до практической разработки приложений на разных ООСУБД, проявляется множество отличий в реализации поддержки трех характеристик:

- Целостность;
- Масштабируемость;
- Отказоустойчивость.

Отметим, что ООБД не требуют многих из тех внутренних функций и механизмов, которые столь привычны и необходимы в реляционных БД. Например, при небольшом числе пользователей, длинных транзакциях и незначительной загрузке сервера объектные СУБД не нуждаются в поддержке сложных механизмов резервного копирования/восстановления (исторически сложилось так, что первые ООБД проектировались для поддержки небольших рабочих групп – порядка десяти человек – и не были приспособлены для обслуживания сотен пользователей). Тем не менее технология БД определенно созрела для крупных проектов.

Для иллюстрации первой категории рассмотрим механизм кэширования объектов. Большинство объектных СУБД помещают код приложения непосредственно в то же адресное пространство, где работает сама СУБД. Благодаря этому достигается повышение производительности часто в 10-100 раз по сравнению с раздельными адресными пространствами. Но при такой модели объект с ошибкой может повредить объекты и разрушить базу данных.

Существуют два подхода к организации реакции СУБД для предотвращения потери данных. Большинство систем передают приложению указатели на объекты, и рано или поздно такие указатели обязательно становятся неверными. Так, они всегда неправильны после перехода объекта к другому пользователю (например, после перемещения на другой сервер). Если программист, разрабатывающий приложение, пунктуален, то ошибки не возникает. Если же приложение попытается применить указатель в неподходящий для этого момент, то в лучшем случае произойдет крах системы, в худшем – будет утеряна информация в середине другого объекта и нарушится целостность базы данных.

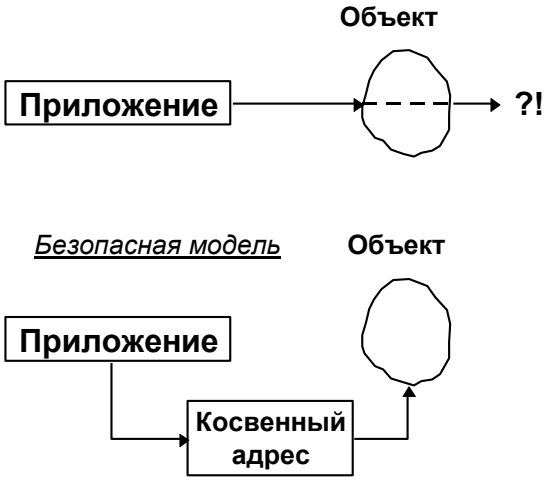


Рисунок 3 Прямая и косвенная адресации.

Есть метод, лучший, чем использование прямых указателей (Рисунок 3). СУБД добавляет дополнительный указатель и при необходимости, если объект перемещается, система может автоматически разрешить ситуацию (перезагрузить, если это необходимо, объект) без возникновения конфликтной ситуации.

Существует еще одна причина для применения косвенной адресации: благодаря этому можно отслеживать частоту вызовов объектов для организации эффективного механизма свопинга.

Это необходимо для реализации уже второго необходимого свойства баз данных – масштабируемости. Опять следует упомянуть организацию распределенных компонентов. Классическая схема клиент-сервер, где основная нагрузка приходится на клиента (такая архитектура называется еще “толстый клиент-тонкий сервер”), лучше справляется с этой задачей, чем мэйнфреймовая структура, однако ее все равно нельзя масштабировать до уровня предприятия. Благодаря *многозвенной архитектуре клиент-сервер (N-Tier architecture)* происходит равномерное распределение вычислительной нагрузки между сервером и конечным пользователем. Нагрузка распределяется по трем и более звеньям, обеспечивающим дополнительную вычислительную мощность. К чему же еще ведет такая практика? “Архитектура клиент-сервер, еще совсем недавно считавшаяся сложной средой, постепенно превратилась в исключительно сложную среду. Почему? Благодаря ускоренному переходу к использованию систем клиент-сервер нескольких звеньев” (PC Magazine). Разработчикам приходится расплачиваться дополнительными сложностями, большими затратами времени и множеством проблем, связанных с интеграцией. Оставим очередное упоминание распределенных компонентов на этой не лишенной оптимизма ноте.

Третье необходимое качество базы данных – это отказоустойчивость. Именно это свойство отличает программный продукт от “прилады”. Существуют несколько способов обеспечения отказоустойчивости:

- резервное копирование и восстановление;
- распределение компонентов;
- независимость компонентов;
- копирование.

Руководствуясь первым принципом, программист определяет потенциально опасные участки кода и вставляет в программу некоторые действия, соответствующие началу транзакции – сохранение информации, необходимой для восстановления после сбоя, и окончанию транзакции – восстановление или, в случае невозможности, принятие каких-то других мер, например, отправка сообщения администратору. В современных СУБД этот механизм обеспечивает восстановление в случае возникновения практически любой ошибки системы, приложения или компьютера, хотя, конечно, нельзя говорить об идеальной защите от сбоев.

В мэйнфреймовой архитектуре единственным источником сбоев была центральная ЭВМ. При переходе к распределенной многозвенной организации ошибки могут вызывать не только компьютеры, включенные в сеть, но и коммуникационные каналы. В многозвенной архитектуре при сбое одного из звеньев без специальных мер результаты работы других окажутся бесполезными. Поэтому при разработке распределенных систем обеспечивается принципиально более высокий уровень обеспечения отказоустойчивости. Назовем обязательные для современных распределенных СУБД свойства:

- прозрачный доступ ко всем объектам независимо от их местоположения, благодаря чему пользователю доступны все сервисы СУБД и может производиться перераспределение компонентов без нежелательных последствий.
- так называемый “трехфазный монитор транзакций” (third-party transaction monitor), благодаря которому транзакция выполняется не в два, а в три этапа – сначала посылается запрос о готовности к транзакции.

Что произойдет, если один из компонентов выйдет из строя? Система, созданная в соответствии только с вышеизложенными доводами, приостановит работу всех пользователей и прервет все транзакции. Поэтому важно такое свойство СУБД, как независимость компонентов.

При сетевом сбое сеть разделяется на части, компоненты каждой из которых не могут сообщаться с компонентами другой части. Для того, чтобы сохранить возможность работы внутри каждой такой части, необходимо дублирование критически важной информации внутри каждого сегмента. Современные системы позволяют администратору базы данных динамически определять сегменты сети, варьируя таким образом уровень надежности всей системы в целом.

И, наконец, о копировании (replication) данных. Простейшим способом является добавление к каждому (основному) серверу резервного. После каждой операции основной сервер передает измененные данные резервному, который автоматически включается в случае выхода из строя основного. Естественно, такая схема не лишена недостатков. Во-первых, это приводит к значительным накладным расходам при дублировании данных, что не только сказывается на производительности, но и само по себе является потенциальным источником сбоев. Во-вторых, в случае сбоя, повлекшего за собой разрыв соединения между двумя серверами, каждый

из них должен будет работать в своем сегменте сети в качестве основного сервера, причем изменения, сделанные на серверах за время работы в таком режиме, будет невозможно синхронизовать даже после восстановления работоспособности сети.

Более совершенным является подход, когда создается необходимое (подбираемое в соответствии с требуемым уровнем надежности) число копий в сегменте. Таким образом увеличивается доступность копий и даже (при распределении нагрузки между серверами) повышается скорость чтения. Проблема невозможности обновления данных несколькими серверами одновременно в случае их взаимной недоступности решается за счет разрешения проведения модификаций только в одном из сегментов, например имеющем наибольшее число пользователей. При хорошо настроенной схеме кэширования затраты на накладные расходы при дублировании модифицированных данных близки к нулю.

4.3 Стандарты объектных баз данных.

Для обеспечения переносимости приложений (приложение может работать на разных СУБД) и совместимости с СУБД (может взаимодействовать с разными СУБД), естественно, необходима выработка стандартов. Сразу заметим, что установление стандартов лишает производителя в некоторой степени свободы в принятии решений и увеличивает стоимость продукта за счет лицензионных отчислений и больше не будем обсуждать целесообразность (прямо скажем, очевидную) стандартизации.

В области объектных СУБД в настоящее время выработаны стандарты для:

- объектной модели;
- языка описания объектов;
- языка организации запросов (*Object Query Language – OQL*);
- “связующего” языка (C++ и, конечно же, Smalltalk);
- администрирования;
- обмена (импорт/экспорт);
- интерфейсов инструментария и др.

Хотя у Microsoft и свое мнение на этот счет, организацией, выработавшей наиболее используемые на сегодня и устоявшиеся стандарты, является консорциум поставщиков ООСУБД *ODMG* (ООСУБД), которого поддерживают практически все действующие лица отрасли. В сотрудничестве с *OMG*, ANSI, ISO и другими организациями был создан стандарт ODMG-93. Этот стандарт включает в себя средства для построения законченного приложения, которое будет работать (после перекомпиляции) в любой совместимой с этой спецификацией ООСУБД. В книгу ODMG-93 входят следующие разделы:

- Язык определения объектов (Object Definition Language – ODL);

- Язык объектных запросов (Object Query Language – OQL);
- Связывание с C++;
- Связывание со Smalltalk.

ODL. В качестве языка определения объектов (ODL) ODMG был выбран существующий язык IDL (Interface Definition Language – язык описания интерфейсов), который был дополнен такими необходимыми для объектных БД свойствами, как определение коллекций, двунаправленных связей типа “многие-ко-многим”, ключей и др. В сочетании со средствами языка IDL определения *атрибутов* и операций, это позволяет определять практически любые объекты. Все дополнения реализованы в виде доопределения методов, что обеспечивает совместимость со стандартами OMG, например стандартом CORBA.

Рисунок 4 показывает работоспособную схему для построения приложения на стандартных языках программирования, в процессе которой автоматически генерируются *метаданные*, заголовочные файлы и методы. Приведем также пример на языке ODL из “белой книги” компании Objectivity, который иллюстрирует связи типа “один-ко-многим”, объявленные между преподавателем и студентами:

```
interface professor : employee {
  attribute string <32> name;
  unique attribute lang unsigned ssn;
  relationship dept works_in inverse faculty;
```

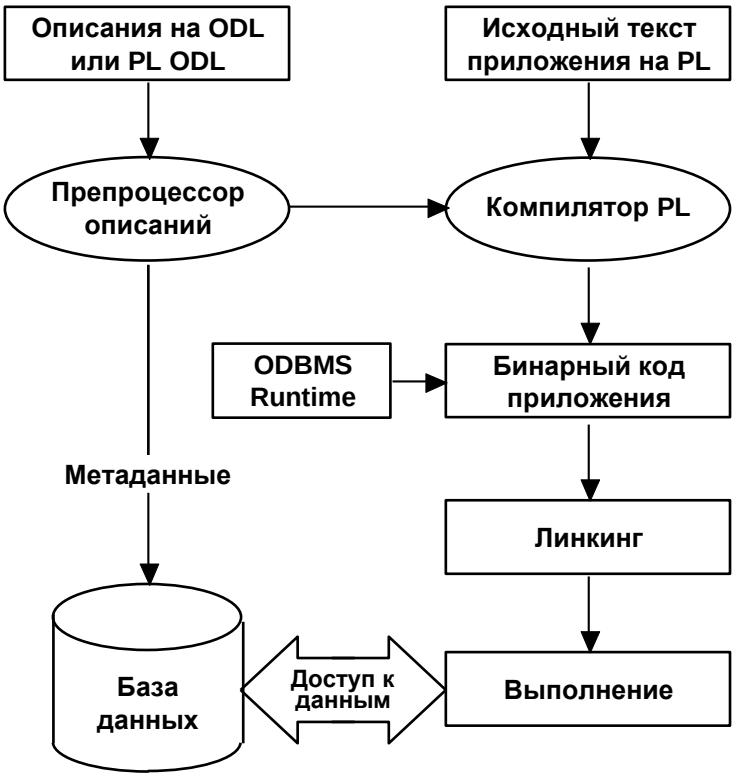


Рисунок 4 Схема использования ODL для построения приложения.

```
  relationship set <section> teaches inverse taught_by;
  . . . operations . . .
  {
interface section : class {
  . . . taught_by: professor . . . ;
  . . .
  }
```

OQL. За основу языка OQL была взята команда `SELECT` языка SQL2 (или SQL-92) и добавлены возможность направлять запрос к объекту или коллекции объектов и возможность вызывать методы в рамках одного запроса. Данные, полученные в результате запроса, могут быть скалярными (включая кортежи), объектами или коллекциями объектов. Некоторые примеры на языке OQL (тот же источник):

- `Select x from x in faculty where x.salary > x.dept.chair.salary`
- `sort s in (select struct (name: x.name, s:x.ssn) from x in faculty where for all y in x.advisees:y.age<25) by s.name`
- `Chair.salary`
- `Students except TAs`
- `list (1,2) + list (count (jse.advisees), 1+2)`
- `exists x in faculty [1:n]: x.spouse.age<25`

C++. Спецификация ODMG-93 позволяет программистам легко использовать объекты в то время как ООСУБД прозрачным образом управляет ими. При определении стандарта члены ODMG руководствовались следующими принципами:

- **Использование стандартных компиляторов** обеспечивается тем, что все расширения реализуются средствами языка – библиотеками классов и перегрузкой операторов.
- **Определение временных экземпляров (Transient Instance) и экземпляров, создаваемых на длительный срок (Transient Instance) при помощи оператора `new()`.** При перегрузке оператора `new()` оба типа экземпляров могут создаваться от одного класса, который может существовать продолжительное время.
- **Обеспечение устойчивости через стандартный механизм наследования;** пользователь может определять экземпляры временные и рассчитанные на продолжительное использование средствами оригинальной версии языка.
- **Использование специального механизма указателей (Smart Pointers).** Связи между объектами объявляются при помощи шаблона `Ref<>` и перегрузки оператора `->`; это позволяет использовать специальные указатели (контролируемые системой; см., например, *идентичность* в словарике (стр. **Ошибка! Закладка не определена.**) и упоминание косвенной адресации (стр. **Ошибка! Закладка не определена.**) как обычные.

```
class Professor: Employee {
    long ssn;
    char* name;
    int age;
    Ref<Department>dept inverse faculty;
    Set<Section> teaches inverse taught_by;
    . . .
    void grant_tenure()
    void assign_course(section)
    }
. . .
Ref<Professor>prof;
. . .
prof = new(db, Professor);
prof->name="Smith";
prof->age+prof->age+1;
```

На этом, пожалуй, чувство благодарности компании Objectivity в значительной мере ослабеет, так как примеров на языке Smalltalk найти не удалось.

Smalltalk. ODMG-93 поддерживает ту же объектную модель для Smalltalk, что и для C++, IDL и запросы на языке OQL; это позволяет разделять один и тот же объект пользователям C++ и Smalltalk. Спецификация поддерживает типы (возможны бестиповые поля) и синтаксис оригинальной версии Smalltalk.

Взаимодействие с другими стандартами. Многие стандарты совместимы с объектными базами данных, например STEP, CFI, TINA-C, ISO ODP, ANSI X3H7, OpenGIS и др. Сейчас они могут напрямую взаимодействовать с любой стандартной ООСУБД, хотя в некоторые из них и были внесены изменения для обеспечения совместимости. Два других стандарта заслуживают более детального описания – *OMG* и *SQL*.

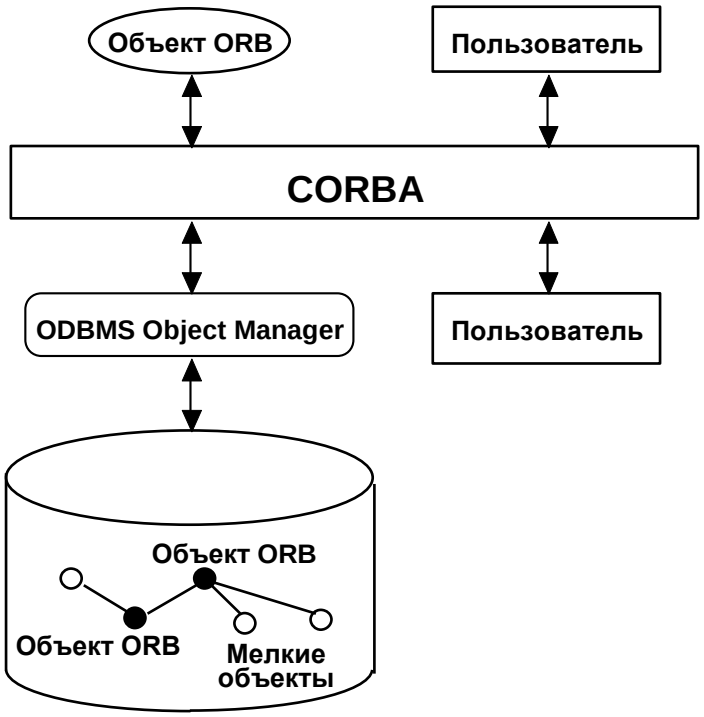


Рисунок 5 ООСУБД, построенная на основе стандартов ODMG во взаимодействии с CORBA.

Стандарты *OMG*. Первым результатом деятельности OMG стало утверждение (OMG не создает стандартов, а принимает одну из существующих реализаций) Архитектуры Брокера Объектных Запросов (Common

Object Request Broker Architecture – *CORBA*) – средства диспетчеризации запросов между объектами и пользователями; в дальнейшем были добавлены некоторые сервисы. Интерфейс ODMG сейчас полностью адаптирован к спецификации Persistence Object Service консорциума OMG, что позволяет пользователям систем, основанных на архитектуре CORBA, пользоваться преимуществами от ООСУБД, которые могут содержать объекты, отвечающие стандарту OMG и используемые так же, как и любые другие ("мелкие") объекты спецификации OMG (Рисунок 5). Объекты OMG в свою очередь доступны через интерфейс ODMG.

Язык *SQL*. Из-за распространенности SQL был заложен в основу *OQL*, который был дополнен средствами поддержки объектной модели. В настоящее время разрабатывается версия языка SQL, известная под названием SQL3, в которой будут реализована поддержка объектов и SQL будет приведен в соответствие современным понятиям о полноценном языке программирования. В отличие от ODMG, в SQL не планируется привязка к ODL, а также C++ и Smalltalk, которые важны для пользователей ООСУБД. Несмотря на это, возможности SQL3 в организации запросов совпадают с возможностями OQL. Когда SQL3 будет готов (разработки ведутся сейчас на ранней стадии обсуждения основных вопросов относительно объектной модели), ODMG, вероятно, дополнит его, как это уже сделано для C++ и Smalltalk.

4.4 Поставщики ООСУБД.

Список современных коммерческих объектно-ориентированных систем включает в себя следующие продукты:

- **Objectivity/DB** компании **Objectivity, Inc.** (последняя версия – 2.1) идеально, по заявлениям фирмы, подходит для приложений, которые работают в распределенных средах, требуют гибкой модификации данных, организации сложных связей, а также нуждаются в высокой производительности и работы с

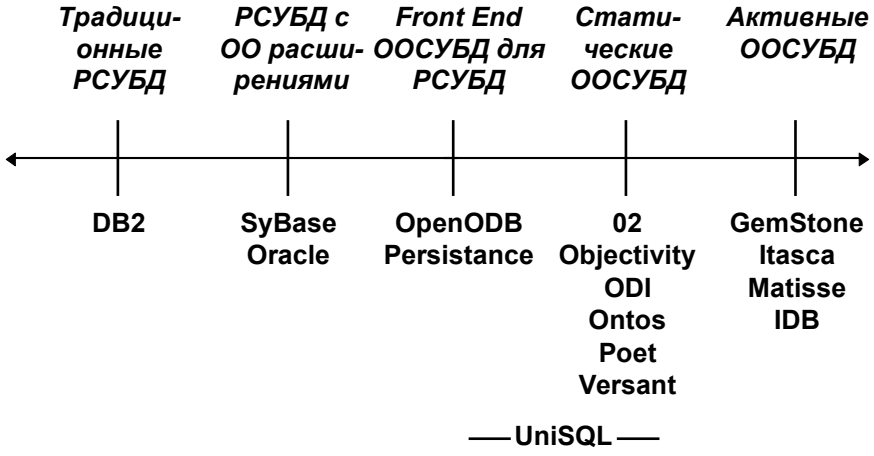


Рисунок 6 Современный рынок СУБД.

большими объемами данных. Вероятно, все компании, производящие ООСУБД, ставят своей целью сложить такое впечатление относительно собственных разработок у читателей распространяемых ими документов (хотя некоторые и делают это в более деликатной форме). Более содержательно, Objectivity обеспечила интеграцию инструментария СУБД и разработки приложений с такими средствами программирования, как SoftBench и C++ SoftBench. Благодаря интегрированному графическому интерфейсу разработки схемы БД и инструментам отладки и анализа упрощается задание модели базы данных и, соответственно, разработки приложений для Objectivity/DB.

- СУБД **GemStone** корпорации **GemStone Systems, Inc.** известна в последней редакции под номером 5.0. GemStone традиционно сосредоточена на рынке Smalltalk (хотя не так давно и была выпущена версия для C++) и имеет заказчиков, способных продемонстрировать на производстве крупномасштабные, целевые применения ее продуктов. К сожалению, списком этих заказчиков объем информации, которую компания хочет донести до интересующихся (WWW), ограничивается.
- **ONTOS Corp.,** разработчик СУБД **ONTOS** (кто бы подумал), по традиции занимается развитием сервера объектно-ориентированной СУБД, но в последнее время придает особое значение своим Службам Интеграции Объектов (Object Integration Services).
- Построенная на основе реляционной СУБД AllBase, система **OpenODB** фирмы **Hewlett-Packard** также, как и Objectivity/DB, интегрирована с системой SoftBench и существует в версии для C++. Благодаря глубокой интеграции, SoftBench распознает файлы приложений OpenODB для установки оптимальной конфигурации, может создавать базы данных формата OpenODB из своей интегрированной среды, обеспечивает оперативную помощь из среды разработки и т. д.
- **Object Design Inc.** со своей СУБД **ObjectStore** занимает лидирующее положение в отрасли, осуществляя около 33% поставок на рынке объектно-ориентированных СУБД и последняя модернизация системы (клиент языка SQL и шлюз к реляционной СУБД) должны только укрепить положение фирмы. Object Design поддерживает версии своей СУБД как для C++, так и для Smalltalk.
- **Versant Object Technology, Inc.** (СУБД **Versant**) проводит двойную стратегию, предлагая средство обеспечения объектно-ориентированной СУБД высокого класса для телекоммуникаций и инструментальные средства Smalltalk для более общих случаев разработки приложений. Используя разработанный фирмой интерфейс VERSANT Smalltalk Language Interface, СУБД

совместима как с версией языка Smalltalk компании ParcPlace-Digitalk, так и с Visual Age for Smalltalk корпорации IBM.

- СУБД **UniSQL** компании **UniSQL Inc.** – хорошо устоявшаяся система, позволяющая пользователям осуществлять запросы и модификацию базы при помощи разработанного компанией языка SQL/X (подобные языки, носящие условное название Object SQL, разработаны и некоторыми другими поставщиками). Вся БД UniSQL может состоять одновременно из связей в локальных РСУБД и классов в локальных объектных базах UniSQL. Благодаря механизму каталогов, СУБД передает запросы и модификации данных в локальные базы данных и, обработав (перевод в другой формат, группирование, сортировка и т. д.) полученный от них результат, возвращает его пользователю.

Кроме того ООСУБД предлагают: Object Database, Inc. (Object Database), Itasca Systems Inc. (Itasca) O2 Technology (O2) и некоторые другие компании.

5. Заключение.

В 1996 г. наметился заметный сдвиг в области освоения объектных СУБД. Уже существуют примеры практического их использования крупными биржами, банками, страховыми компаниями, а также в сфере производства и телекоммуникаций, где базам данных, содержащим гигабайты информации, приходится обслуживать сотни пользователей. Они оказались хорошей альтернативой в тех случаях, когда применение реляционных БД вынуждало строить сложную схему с чрезмерно большим числом межтабличных связей.

Благодаря значительному прогрессу в развитии объектной технологии, за последние пять лет производителям удалось довести свои ООСУБД до такого уровня, что они стали вполне отвечать реальным требованиям рынка.

Несмотря на то, что технология объектных СУБД созрела для крупных проектов, для действительно массового ее распространения необходим специальный инструментарий.

В настоящий момент ощущается настоятельная потребность в интеграции ООСУБД с существующими инструментальными средствами. Разработчики уже сегодня могли бы продуктивно использовать версии Visual Basic, Power Builder, Forte или Delphi, поддерживающие ООСУБД. Большинство продуктов для создания приложений в той или иной мере являются объектно-ориентированными, но работают по-прежнему с реляционными БД. Специалисты считают, что партнерство производителей ООСУБД и средств программирования способно привести к появлению столь необходимого инструментария.

Эксперты уже неоднократно объявляли наступающий год “годом объектных баз данных”, однако сейчас все говорит о том, что 1997 г. действительно имеет шансы наконец им стать. Основными стимулами

растущего интереса к ООСУБД аналитики считают расширение применения мультителиа-приложений и новых средств, улучшающих их стыкуемость с существующими базами данных.

6. Глоссарий

4GL (4th Generation Language) – Язык программирования четвертого поколения ♦Язык программирования, при создании которого используются языки программирования третьего уровня (3GL) – процедурные языки типа C и Pascal. 4GL проще в использовании, чем 3GL, им обычно отдают предпочтение при составлении программ обслуживания баз данных и применяют в сочетании с соответствующими средствами разработки.

Blob (Binary Large Object) – Двоичный большой объект, блоб. ♦Длинный линейный блок данных (например, цифровое изображение или видеоклип), который наиболее подходит для хранения в *ООСУБД*.

CORBA (Common Object Request Broker Architecture) Архитектура брокера объектных запросов ♦Стандарт взаимодействия распределенных компонентов, разработанный *OMG*.

DBMS (Database Management System) – Система управления базами данных, *СУБД*

N - звенная архитектура (N-Tier Model) ♦*Архитектура клиент-сервер*, в которой применяются средства разбиения программ или распределенные объекты для разделения вычислительной нагрузки среди такого количества серверов приложений, которое необходимо при имеющемся уровне нагрузки. При многозвенной модели системы количество возможных клиентских мест значительно больше, чем при использовании двухзвенной модели. См также *middleware*.

ODBMS (Object Database Management System) – Объектно-ориентированная *СУБД – ООСУБД*. ♦СУБД, хранящая данные и взаимосвязи между ее элементами непосредственно в самой базе данных в виде объектов, содержащих, как правило, алгоритмы обработки этих данных.

ODMG (Object Database Management Group) ♦Консорциум производителей объектных баз данных для выработки стандартов (ODMG-93, ODMG-95).

OMG (Open Management Group) ♦Консорциум поставщиков в сфере объектной технологии для выработки стандартов межкомпонентного взаимодействия. Объединяет практически всех ведущих производителей (более чем 500); членство Microsoft, видимо, лишь условно.

OQL (Object Query Language) Язык объектных запросов ♦Разработанный консорциумом *ODMG* язык описания запросов, за основу которого был принят *SQL*-92.

RDBMS (Relational Database Management System) – Реляционная *СУБД – СУБД*, хранящая взаимосвязи между элементами в виде двумерных таблиц и использующая для запросов язык *SQL*.

SQL (Structured Query Language) – Язык структурированных запросов ♦Интерпретируемый язык, описывающий операции (создание, обработка и извлечение) над реляционными базами данных.

Архитектура клиент-сервер (Client-server architecture) ♦Архитектура, обеспечивающая распределение нагрузки между клиентом и сервером. Обычно эти функции выполняют два разных компьютера, объединенных при помощи сети.

Атрибуты (Attributes) ♦Видимая за пределами объекта информация о состоянии этого объекта.

“Белая книга” (White Paper) ♦Официальное издание.

Гибриды (Hybrids) ♦1. Средства связи между мирами объектных и реляционных баз данных, включая базы данных, которые хранят информацию в реляционной форме, но используют объектные буферные средства. См. также *объектно-реляционные методы* 2. *СУБД*, которые могут хранить и табличные данные, и объекты. Этого определения я старался придерживаться.

Идентичность (Identity) ♦Возможность получения уникального адреса объекта независимо от его местоположения и *атрибутов*.

Инкапсуляция (Encapsulation) ♦Объединение данных и кода в один

модуль – объект, доступ к которому может осуществляться только через строго определенный интерфейс.

Метаданные (Metadata) ♦Данные, являющиеся описанием других данных (например, схема базы данных по отношению к ее содержимому).

Наследование (Inheritance) ♦Механизм, благодаря которому определения класса распространяется на классы, лежащие ниже его в иерархии обобщения классов. Это позволяет многократно изменять определения, внося по мере необходимости изменения, связанные со специализацией.

Объектно-реляционные методы (Object-relational Approaches) ♦Подходы, позволяющие воспользоваться преимуществами объектных баз данных, не отказываясь полностью от реляционных БД.

Отображение (Mapping) ♦Процесс установления связей между приложениями, построенными вокруг объектно-ориентированных и реляционных баз данных.

Полиморфизм (Polymorphism) ♦Способность объектов различных классов и самих классов удовлетворять одним и тем же *протоколам* или отдельным сообщениям, выполняя при этом различные действия, предписываемые их собственными методами.

Промежуточное обеспечение (Middleware) ♦ПО, служащее посредником между клиентом и сервером, например, для предоставления общих интерфейсов. Следуя традиции, и я тоже напишу, что промежуточное ПО – это слэш в термине “клиент/сервер”.

Протокол (Protocol) ♦Набор сообщений, на которые может ответить класс (протокол класса) или его объекты (протокол объекта). Протокол определяется заданными методами. Все объекты одного класса отвечают одному протоколу.

СУБД – Система Управления Базами Данных. ♦Лежащая в основе базы данных прикладная программа, выполняющая операции над хранимой информацией.

Транзакция (Transaction) – обработка запроса ♦Выполнение элементарной целостной операции над данными, в течение которой база данных находится в неустойчивом состоянии.

ООСУБД (ODBMS)– Объектно-Ориентированная Система Управления Базами Данных.