Головчинер М.Н.

БАЗЫ ДАННЫХ

Основные понятия, модели данных, процесс проектирования

КУРС ЛЕКЦИЙ

Томск 2009

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Основными целями данного учебного пособия являются:

- знакомство пользователей с базовыми понятиями области технологии баз данных,
- описание классических моделей данных,
- формулировка проблем, связанных с процессом проектирования как локальных, так и распределенных по сети баз данных, и описание основных шагов (этапов) проектирования.

Отметим, что такие важные вопросы, как использование семантической модели данных “сущность-связь” (ER-модель) и языка запросов SQL, рассматриваются на практических и лабораторных занятиях в рамках изучения баз данных. Поэтому изложение искомых инструментариев входит в дополнительные к данному пособию методические материалы.

1. ВВЕДЕНИЕ

1.1. Понятие о данных как о ресурсе

Очевидно, что в большинстве областей человеческой деятельности, связанных с функционированием любой организации (предприятия, банка, учебного заведения и т.д.), принятие производственного решения основывается на анализе и использовании имеющихся ресурсов (например, финансовых, материальных, трудовых). Для того, чтобы использование ресурсов было оптимальным, ими нужно уметь эффективно управлять. Управление ресурсами в общем случае (вне зависимости от видов ресурсов) означает способность к выполнению над ними процедур **планирования, распределения, поддержки** и **сохранения, экономного расходования, правильного потребления** и **интеграции** (возможности использования в различных целях). Для успешного решения этих задач необходимо всесторонне изучить **свойства** ресурсов. Рассмотрим примеры наиболее известных ресурсов.

Финансовые ресурсы. Для реализации управленческих процедур необходима, по крайней мере, следующая информация о свойствах финансовых потоков:

- какие средства доступны,
- сколько израсходовано,
- откуда поступают,
- куда направляются.

Ответственным управляющим лицом является главный бухгалтер. Обычный инструмент для принятия решений (информационная среда, информационное хранилище) – бухгалтерская книга с информацией о денежных ресурсах и сметы, позволяющие определить, куда направлены средства, откуда они поступили, какое количество было израсходовано, сколько осталось.

Материальные ресурсы. Примером материальных ресурсов является сырье для производства изделий. Ответственный – допустим, коммерческий директор, отвечает за обеспечение производства требуемым количеством сырья нужного вида. Необходимая информация:

- какие материалы имеются в наличии,
- откуда поступают,
- куда направляются,
- сроки реализации заказов.

Примером хранилища информации о материальных ресурсах является инвентарная книга.

Трудовые (кадровые) ресурсы. Ответственный за управление – начальник отдела кадров. Хранилище – картотека личных дел, содержит следующую информацию:

- общее число сотрудников,
- о каждом сотруднике – его профессия, образование, стаж работы, должность, местонахождение рабочего места и т.д.

Просматривая картотеку, можно определить прошлые достижения сотрудников, их ценность для предприятия, возможность продвижения по службе и т.п.

Таким образом, для управления каждым из перечисленных видов ресурсов необходимо:

- определить свойства ресурса;
- выделить среду хранения;
- назначить ответственное управляющее лицо - администратора ресурса;
- реализовать вышеперечисленные процедуры управления.

Данные как ресурс. Очевидно, что информация, необходимая для управления любым из указанных ресурсов, извлекается из собранных, проанализированных и хранящихся **данных** о свойствах искомого ресурса. Возможность моделирования (представления) в памяти ЭВМ

любого хранилища (бухгалтерской и инвентарной книги, картотеки и т.п.) позволяет рассматривать сами данные как один из видов ресурсов.

Очевидно, что использование данных как ресурса предполагает:

- представление самого понятия “данные”;
- умение их **собирать** и **анализировать**;
- определение **природы** и **свойств** данных, для чего необходимо знать, **как** и **с какой целью** они применяются, где находятся, откуда поступают и т.п.; таким образом, для управления данными необходимо иметь о них как можно больше сведений;
- наличие **среды хранения** полученных сведений, которые могут надежно сохраняться только при наличии четких процедур **накопления**, **планирования** и **ведения** данных;
- возможность после сбора и организации сведений получения доступа к ресурсу данных всюду, где нужна информация, требуемая для управления другими ресурсами, то есть **интегрирование** данных;
- наличие **администратора** данных.

Исторически рассмотрение данных как ресурса стало возможным с появлением внешних запоминающих устройств и размещением на них наборов данных, организованных в виде файлов.

1.2. Файловые системы и базы данных

1.2.1. Численные и информационные прикладные системы

В истории вычислительной техники можно выделить две основных области ее использования:

- Применение вычислительной техники для выполнения **численных расчетов**, которые слишком долго или вообще невозможно производить вручную. Эта область связана, в основном, с развитием методов численного решения сложных математических задач и класса языков программирования, ориентированных на удобную запись численных алгоритмов.
- Использование средств вычислительной техники в автоматических или автоматизированных **информационных системах**. В самом широком смысле информационная система представляет собой программно-аппаратный комплекс, функции которого состоят в надежном хранении информации в памяти компьютера, выполнении специфических для данного приложения преобразований информации и/или вычислений, предоставлении пользователям удобного и легко осваиваемого интерфейса. Обычно такие системы имеют дело с большими объемами информации, и эта информация имеет достаточно сложную структуру. Классическими примерами информационных систем являются банковские системы, системы резервирования авиационных или железнодорожных билетов, мест в гостиницах и т. д.

Вторая область использования вычислительной техники возникла несколько позже первой. Это связано с тем, что на заре вычислительной техники возможности компьютеров по хранению информации были очень ограниченными. Говорить о надежном и долговременном хранении информации можно только при наличии запоминающих устройств, сохраняющих информацию после выключения электрического питания. В первых компьютерах использовались два вида устройств внешней памяти - магнитные ленты и барабаны. Емкость магнитных лент достаточно велика, но по своей физической природе они обеспечивают только **последовательный** доступ к данным. Магнитные же барабаны дают возможность **произвольного** доступа к данным, но имеют ограниченный размер.

Эти ограничения не являлись слишком существенными для чисто численных расчетов. Даже если программа должна обработать (или произвести) большой объем информации, при программировании можно продумать расположение этой информации во внешней памяти

(например, на последовательной магнитной ленте), обеспечивающее эффективное выполнение этой программы.

Но для информационных систем, в которых потребность в текущих данных определяется конечным пользователем, наличие только магнитных лент и барабанов неудовлетворительно. Представьте себе покупателя билета, который, стоя у кассы, должен дождаться полной перемотки магнитной ленты. Одним из естественных требований к таким системам является удовлетворительная средняя скорость выполнения операций.

С появлением магнитных дисков началась история систем управления данными во внешней памяти, поскольку именно эти устройства, обеспечивая удовлетворительную скорость доступа к данным в режиме **произвольной** выборки, в настоящее время обладают практически неограниченной памятью.

1.2.2. Файловые системы

Известны два подхода к организации информационных массивов: **файловая организация** и организация в виде **базы данных**. С точки зрения прикладной программы, файл - это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные. Система управления файлами берет на себя распределение внешней памяти, отображение имен файлов в соответствующие адреса во внешней памяти и обеспечение доступа к данным.

Прежде всего, файлы применяются для хранения текстовых данных: документов, текстов программ и т. д. Такие файлы обычно образуются и модифицируются с помощью различных текстовых и графических редакторов. Структура текстовых файлов обычно очень проста: это либо последовательность записей, содержащих строки текста, либо последовательность байтов, среди которых встречаются специальные символы (например, символы конца строки). Рассмотрим несколько примеров.

Пример 1. Файлы с текстами программ (исходные файлы) являются входными параметрами компиляторов, которые, в свою очередь, формируют файлы, содержащие объектные модули. С точки зрения файловой системы, как исходные, так и объектные файлы обладают абсолютно стандартной структурой - это последовательности байтов. Конечно, система программирования определяет более сложную и специфичную структуру объектного модуля. При этом **логическая структура** объектного модуля неизвестна файловой системе, эта структура поддерживается программами системы программирования.

Пример 2. Аналогично обстоит дело с файлами, формируемыми редакторами связей (компоновщиками выполняемых программ). На вход редактора связей поступают объектные модули, а на выходе строятся файлы-образы выполняемых программ (загрузочные модули). Логическая структура таких файлов остается известной только компоновщику и программе-загрузчику, являющимися компонентами операционной системы, и не известна системе управления файлами.

Пример 3. Прикладная программа, обрабатывающая собственные файлы. Очевидно, что и в этом случае логическая структура файлов известна только этой обрабатывающей программе (как результат умственной деятельности разработчика).

Таким образом, файловая организация предполагает специализацию и хранение информации, ориентированной, как правило, на одну прикладную задачу, и обеспечивается самим программистом.

На рис.1. представлена структура программы при таком подходе.

Достоинства и недостатки файловой организации

Достоинства:

- Файловые системы обычно обеспечивают хранение слабо структурированной информации, оставляя дальнейшую структуризацию обрабатывающим программам. В некоторых случаях использования файлов это даже хорошо, потому что при разработке любой

новой прикладной системы, опираясь на простые, стандартные и сравнительно дешевые средства файловой системы, можно реализовать те структуры хранения, которые наиболее естественно соответствуют специфике соответствующей прикладной области.

- Файловая организация позволяет достигнуть высокой скорости обработки.

Недостатки:

- Узкая специализация как обрабатывающих программ, так и файловых данных, что служит причиной большой **избыточности**, так как одни и те же элементы данных хранятся в разных программных системах.
- Возможность наличия **противоречивости** данных, когда для выполнения одних и тех же операций над однотипными данными, хранящихся в разных файлах, требуются разные программы.
- Частое нарушение **целостности** данных, когда логически идентичные элементы данных в разных частных файлах имеют разные типы значений (например, **Real** и **Integer**), что может привести к расхождению в отчетах, полученных с помощью ЭВМ.

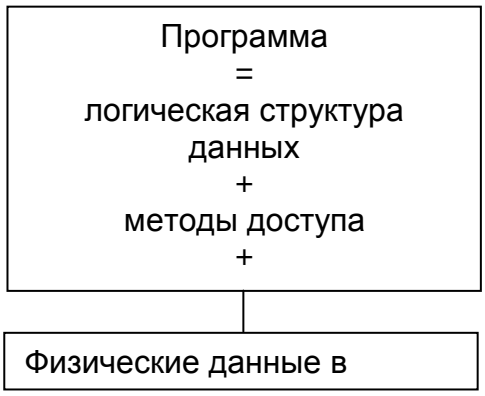


Рис.1. Структура программы при файловой организации данных

1.2.3. Файлы и информационные системы. Общее понятие о базе данных

Информационные системы главным образом ориентированы на хранение, выбор и модификацию постоянно хранимой информации, как правило, очень сложной структуры.

На начальном этапе использования вычислительной техники проблемы структуризации данных решались индивидуально в каждой информационной системе путем создания необходимых надстроек (библиотек программ) над файловыми системами, учитывающими специфику структур файлов для конкретной предметной области, причем эти индивидуальные средства управления данными составляли существенную часть информационных систем

Однако, несмотря на то, что структуры данных различны в разных информационных системах, между ними часто бывает много общего. Разумным представлялось выделить общую часть информационных систем, ответственную за управление сложно структурированными данными, в виде некоторой библиотеки программ, доступной каждой информационной системе.

Но очень скоро стало понятно, что обойтись такой общей библиотекой программ, реализующей над стандартной базовой файловой системой более сложные методы хранения данными, невозможно.

Пример. Необходимо реализовать простую информационную систему, поддерживающую учет сотрудников некоторой организации. Система должна:

- выдавать списки сотрудников в соответствии с указанными номерами отделов,
- поддерживать регистрацию перевода сотрудника из одного отдела в другой, приема на работу новых сотрудников и увольнения работающих,

- для каждого отдела должна быть предусмотрена возможность получения имени руководителя этого отдела, общей численности отдела, общей суммы выплаченной в. последний раз зарплаты и т. д.,
- для каждого сотрудника должна поддерживаться возможность выдачи полного имени сотрудника по номеру удостоверения, получения информации о текущем соответствии занимаемой должности сотрудника и о размере зарплаты.

Предположим, что мы решили реализовать эту информационную систему на основе файловой системы и пользоваться при этом одним файлом, расширив базовые возможности файловой системы за счет специальной библиотеки функций. Поскольку минимальной информационной единицей в нашем случае является сотрудник, естественно потребовать, чтобы в этом файле содержалась **одна запись** для каждого сотрудника, включающая несколько **полей**:

- номер удостоверения (**Сотр_номер**),
- полное имя сотрудника (**Сотр_имя**),
- информация о соответствии занимаемой должности (**Сотр_стат** - для простоты, «да» или «нет»),
- размер зарплаты (**Сотр_зарп**),
- номер отдела (**Сотр_отд_номер**),
- имя руководителя отдела (**Сотр_отд_рук**), поскольку мы хотим ограничиться одним файлом.

Для выполнения функций нашей информационной системы требуется возможность:

- доступа к этому файлу по **уникальному ключу** – не дублируемому в разных записях значению поля **Сотр_номер**,
- доступа к файлу по значению (возможно, и не уникальному) поля **Сотр_имя**,
- выбора всех записей с общем заданным значением поля **Сотр_отд_номер**, то есть доступ по **неуникальным ключам**.

Очевидно, для того, чтобы получить численность отдела или общий размер зарплаты, информационная система должна будет каждый раз выбирать все записи о сотрудниках отдела и подсчитывать соответствующие общие значения.

Таким образом, для реализации даже такой простой системы на базе файловой системы, во-первых, требуется создание достаточно сложной надстройки, обеспечивающей многоключевой доступ к файлам, и, во-вторых, неизбежны существенная избыточность хранения (для каждого сотрудника данного отдела повторяется имя руководителя отдела) и выполнение массовой выборки и вычислений для получения сводной информации об отделах. Кроме того, если в ходе эксплуатации системы возникнет потребность, например, выдавать списки сотрудников, получающих заданную зарплату, то придется либо полностью просматривать файл, либо реструктуризовывать (менять структуру) его, объявляя ключевым поле **Сотр_зарп** и упорядочивая записи файла по значениям этого нового ключа.

Возможным путем повышения эффективности работы данной системы была бы поддержка двух многоключевых файлов **СОТРУДНИКИ** и **ОТДЕЛЫ**: первый файл должен содержать поля **Сотр_имя**, **Сотр_номер**, **Сотр_стат**, **Сотр_зарп** и **Сотр_отд_номер**, а второй - **Отд_номер**, **Отд_рук** (имя руководителя отдела), **Сотр_зарп** (общий размер зарплаты) и **Отд_размер** (общее число сотрудников в отделе). Тогда большая часть неудобств, перечисленных в предыдущем абзаце, будет преодолена. Каждый из файлов будет содержать только не дублируемую информацию, необходимость в динамических вычислениях сводной информации не возникнет.

Однако, теперь система должна знать, что она работает с двумя информационно связанными файлами, ей должны быть известны структура и смысл каждого поля (например, что **Сотр_отд_номер** в файле **СОТРУДНИКИ** и **Отд_номер** в файле **ОТДЕЛЫ** означают одно и то же), а также понимать, что в ряде случаев изменение информации в одном файле должно вызывать модификацию второго файла, чтобы общее содержимое файлов было согласованным. Например, если на работу принимается новый сотрудник, то необходимо

добавить запись в файл **СОТРУДНИКИ**, а также соответствующим образом изменить поля **Отд_зарп** и **Отд_размер** в записи файла **ОТДЕЛЫ**, описывающей отдел этого сотрудника.

Но если информационная система поддерживает согласованное хранение информации в нескольких файлах, можно говорить о том, что она поддерживает **базу данных**. Если же некоторая вспомогательная система управления данными позволяет работать с несколькими файлами, обеспечивая их согласованность, можно назвать ее **системой управления базами данных** (СУБД). Вообще, согласованность данных является ключевым понятием баз данных. Заметим, что одно только требование **поддержки согласованности данных в нескольких файлах** не позволяет обойтись библиотекой функций: такая система должна обладать некоторыми собственными данными (**мета-данными**, данными о данных) и даже **знаниями**, определяющими **целостность** данных. Более подробно функции СУБД рассматриваются в разделе 6.

Таким образом, использование файловых систем для удовлетворения сложных интегрированных запросов, требующих обработки большого количества разнотипных данных, за приемлемое время практически невозможно. Возникает необходимость разделить программы и описания обрабатываемых ими данных, определить такую организацию хранения данных с учетом существующих связей между ними, которая позволяла бы использовать эти данные одновременно для многих приложений. Понятие **база данных** (БД) и явилось выражением этого подхода. При такой организации в состав прикладной программы включаются лишь запросы к базе данных, записанные на соответствующем языке. В запросе специфицируются (описываются) требования как о самих данных, так и о форме их представления. Искомый запрос воспринимается СУБД, которая и осуществляет поиск затребованных данных в базе и преобразование их представления. Структура прикладной программы при такой организации изображена на рис. 2.

Таким образом, БД может быть определена как:

Определение 1. **Совокупность** предназначенных для машинной обработки **интегрированных данных**, служащая для удовлетворения нужд многих пользователей.

Рис. 2. Структура программы при использовании технологии баз данных

В связи с концепцией баз данных еще раз подчеркнем два ключевых момента:

1. Информация уже не скрыта в сочетании “файл-программа”; она хранится явным образом в БД. БД ориентирована на интегрированные запросы, а не на одну задачу.

2. Возможность выделения по запросу из всех данных, хранящихся в БД, только необходимых и в требуемой форме (структуре и форматах).

Наиболее широко БД используются в управленческой деятельности благодаря следующим свойствам:

- **Скорость.** Возможный доступ к информации за требуемое время.
- **Полная доступность.** Вся информация, содержащаяся в БД, доступна для использования (с учетом, конечно, необходимости засекречивания и защиты).

- **Гибкость.** Легко вносимые изменения и дополнения в БД позволяют получать ответы на вопросы, которые ранее оставались без ответа.
- **Целостность.** Уменьшилась степень дублирования данных и ликвидирована их противоречивость; упорядочился процесс обновления и восстановления БД после сбоев; появилась возможность управления параллельного общения с БД нескольких прикладных программ.

Другими словами, под базой данных понимается некоторая унифицированная совокупность данных, совместно используемая персоналом/населением группы, предприятия, региона, страны, мира... Задача базы данных состоит в хранении всех представляющих интерес данных в одном или нескольких местах, причем таким способом, который заведомо исключает ненужную избыточность. В хорошо спроектированной базе данных избыточность данных исключается, и вероятность сохранения противоречивых данных минимизируется. Таким образом, создание баз данных преследует две основные цели: понизить избыточность данных и повысить их надежность.

Замечание. Таким образом, СУБД решают множество проблем, которые затруднительно или вообще невозможно решить при использовании файловых систем. При этом существуют приложения, для которых вполне достаточно файлов, и приложения, для которых безусловно нужны базы данных. Другими словами, современные системы управления файлами и управления базами данных представляют собой весьма совершенные инструменты, каждый из которых может быть успешно применен в соответствующей области деятельности. Но всегда необходимо помнить, что каждый инструмент приносит максимальную пользу именно в той области, для которой он создан.

КОНТРОЛЬНЫЕ ВОПРОСЫ ПО ПЕРВОМУ РАЗДЕЛУ

1. Перечислите набор общих процедур управления ресурсами.
2. Какие еще составляющие, кроме среды хранения, можно выделить в системе управления ресурсом.
3. Дайте общее определение информационной системы.
4. Какие дополнительные требования к вычислительной технике предъявляют информационные потребности по сравнению с вычислительными задачами.
5. Какова структура программы при использовании файловой системы управления информационными массивами.
6. Перечислите достоинства и недостатки файловой системы управления информационными массивами.
7. Какова структура программы при использовании технологии баз данных.
8. В чем основное функциональное отличие использования файловой системы управления информационными массивами и технологии баз данных.
9. Сформулируйте общее понятие о базе данных, как о хранилище информации. Перечислите основные преимущества в использовании баз данных.
10. Какие два ключевых момента следует отметить при переходе к технологии баз данных.

2. БАЗА ДАННЫХ КАК МОДЕЛЬ ПРЕДМЕТНОЙ ОБЛАСТИ

2.1. Понятие предметной области

Понятие предметной области является ключевым при рассмотрении базы данных как информационной системы. Интуитивный смысл этого понятия очевиден: предметная область – это некоторая сфера деятельности. Более точное определение, позволяющее сформулировать понятие **модели предметной области**, предполагает уточнение ряда других базовых понятий.

Данные. Некоторый факт, на котором основан вывод или любая другая интеллектуальная деятельность. Первичными компонентами данных являются цифры и символы или их кодированное представление в виде строки двоичных битов.

Семантика. Присвоение данным некоторых **свойств**, после чего они становятся полезными как для людей (в смысле принятия решений), так и для программ или процедур (при определении обоснованности использования). В дальнейшем слово **смысл** будем использовать как синоним "семантики".

Элемент данных. Наименьшая семантически значимая поименованная единица данных, принимающая значение (*фамилия, должность, цвет, зарплата*).

Объект. То, о чем хранятся данные (*служащий, станок, материал*).

Атрибут. Характеристика (свойство) объекта.

Заметим, что смысл объекта (данного) определяется только совокупностью его атрибутов, представленных соответствующими элементами данных. В этом смысле можно сказать, что значение элемента данных является значением атрибута объекта. На рис.3. проиллюстрирована связь между введенными выше понятиями.

Самым простым способом отображения связи элементов данных с атрибутами и соответствующими значениями является их фиксация в виде последовательности на рис.4.

На рисунке форма представления указывает на тип значений соответствующего элемента (N – целое число, S – строка символов, B – коды логических констант True и False, D – дата).

Предметная область. Совокупность объектов реального мира, рассматриваемого в рамках определенного контекста (теории, сферы деятельности, модели и т.п.).

Замечание 1. В этом определении понятие объекта трактуется достаточно широко: в качестве объектов предметной области могут рассматриваться предметы, явления, процессы. Действительно, пусть, например, требуется хранить сведения о товарах, поступивших на склад, то есть, о материальных (физических) объектах типа **ТОВАР**. Скорее всего, в состав базы данных необходимо будет включить и информацию о заказах на поставку товаров на склад, хотя **ЗАКАЗ** суть не физический объект, а процесс с атрибутами, включающими название поставляемого товара, его количество, название поставщика, срок поставки и т.д.

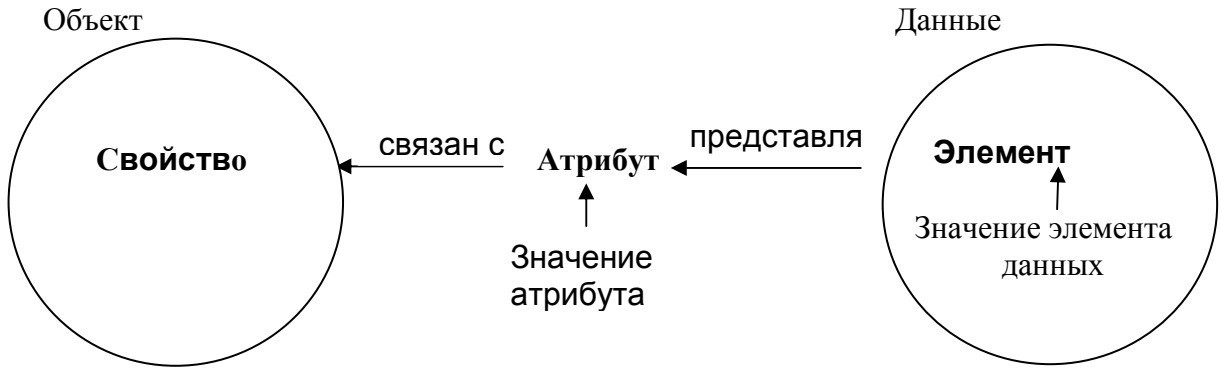


Рис. 3. Базовые понятия предметной области

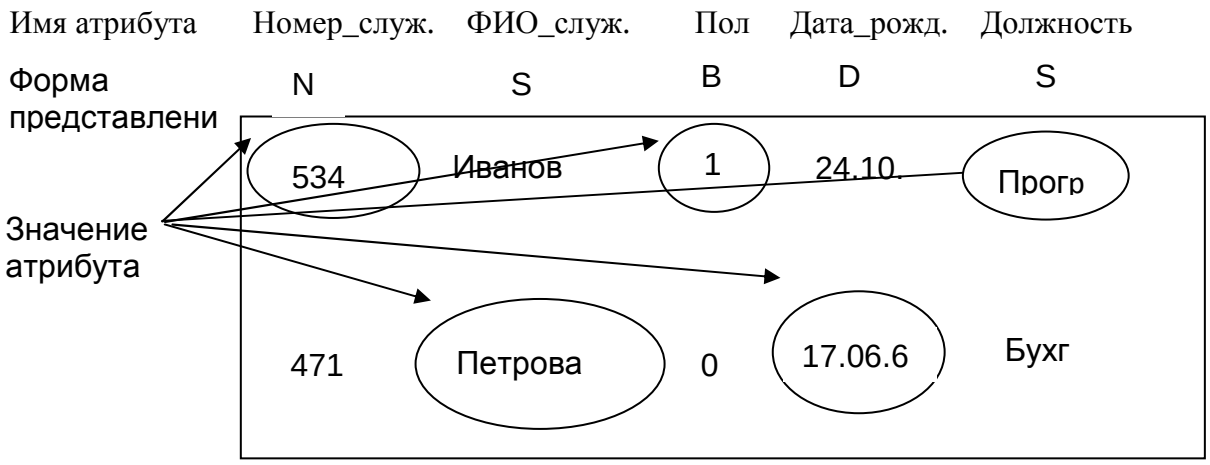


Рис.4. Связь атрибутов с элементами данных и их значениями

Замечание 2. Указание среди атрибутов объекта **ЗАКАЗ** названия товара, являющегося, очевидно, атрибутом объекта **ТОВАР,** тем самым определяет, что в данной предметной области (на складе), существует **связь** между искомыми объектами. Таким образом, объекты реального мира имеют друг с другом множество сложных связей и зависимостей, которые необходимо учитывать в информационной деятельности.

Замечание 3. Реальный мир бесконечен; в любой предметной области можно выделить бесконечное множество объектов с бесконечным количеством свойств у каждого из них и бесконечное количество связей между объектами. Очевидно, что полное и исчерпывающее описание предметной области в БД практически невозможно. Однако для решения возникающих в практической деятельности задач (проблем) полное информационное описание предметной области и не требуется. Поскольку любая решаемая задача связана с достижением некоторой цели, из предметной области всегда можно выделить лишь ограниченную совокупность (подмножество) взаимосвязанных объектов с определенными свойствами, поведение которой существенно для решения задачи. Ясно, что такие ограниченные подмножества связаны с каждой задачей, решаемой в рамках конкретной предметной области.

Определение 2. БД – это **структурная совокупность данных**, отображающих свойства **актуальных объектов** внешнего мира (рассматриваемых с определенной точки зрения).

Или, что то же

Определение 3. База данных - это совокупность описаний объектов реального мира и связей между ними, **актуальных** для **конкретной прикладной области**.

Другими словами, база данных - это некоторый набор данных, отображающий актуальные (необходимые для решения задач) данные и актуальные (значимые) связи.

2.2. Понятие системы

Дадим еще несколько определений, уточняющих отдельные понятия, уже встречавшиеся ранее.

Проблема. Объективно возникающий в деятельности человека вопрос или комплекс вопросов, решение которых представляет теоретический или практический интерес.

Проблемная ситуация. Ситуация, которая не может быть разрешена имеющимися средствами.

Цель. Некоторое **состояние**, к которому движется (или должна двигаться) совокупность взаимосвязанных объектов. Очевидно, что цель возникает при наличии проблемной ситуации.

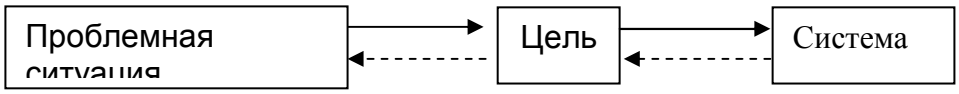
Замечание. Состояние совокупности взаимосвязанных объектов предметной области в каждый момент времени определяется совокупностью значений атрибутов объектов и характеристик связей. Изменились значения (характеристики связей) - изменилось состояние.

Проблемная среда (область). Взаимосвязанная совокупность описаний решаемых задач в рамках определенной информационной системы.

Система. Под системой будем понимать множество объектов и отношений (связей) между ними, выделенное из предметной области в соответствии с определенной целью в рамках определенного временного интервала.

В этом определении отметим два важных момента.

1. Создание системы возможно только при наличии **поставленной цели**. В этом смысле всегда актуален лозунг “Нет системы без проблемы!”. Другими словами, без четкого и полного осмысления проблемной области нельзя начинать разработку информационной системы. Упомянутый принцип можно проиллюстрировать следующей схемой:



Эту схему можно интерпретировать следующим образом: возникшая проблемная ситуация определяет постановку целей, под влиянием которых из элементов предметной области формируется система. Использование (изучение) системы позволяет достичь поставленные цели и разрешить проблемную ситуацию. Например:

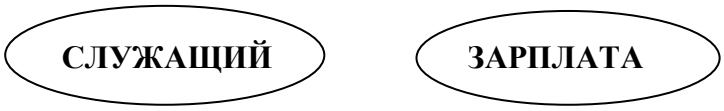
Проблемная ситуация – сложность процесса управления производством.

Цель – автоматизация процесса управления.

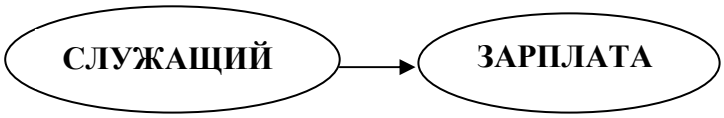
Система – база данных.

2. Система является совокупностью **взаимосвязанных** объектов. Рассмотрим два примера.

Пример 1.



Пример 2.



На диаграммах примеров изображены две совокупности объектов, один из которых соответствует множеству **служащих**, а второй – **зарплат**.

Замечание. Такие совокупности объектов носят название «**тип объекта**». Отдельные элементы этих множеств называются **экземплярами** соответствующих типов объектов. Ясно, что все экземпляры одного типа (**однотипные** объекты) характеризуются одинаковым набором атрибутов (свойств). Таким образом, встречая термин **объект**, необходимо точно представлять себе, идет речь о *типах объектов* или отдельных *экземплярах*. В теории баз данных используется эквивалентное объекту (типу объекта, экземпляру типа объекта) понятие **записи** (**типа записи**, **экземпляра типа записи**).

Очевидно, что, рассматривая совокупность несвязанных объектов Примера 1., относящихся к типам **СЛУЖАЩИЙ** и **ЗАРПЛАТА**, мы не можем определить величину зарплаты конкретных служащих и, соответственно, решить вопрос о ее (зарплате) достаточности или необходимости увеличения. Таким образом, данные могут быть поняты (осознаны) только в том случае, если между ними **установлена связь**; именно взаимосвязанные данные несут определенную **информацию**.

Согласно [1], выделим следующие группы свойств, характеризующих систему как объект исследования:

Статические свойства:

- **Целостность**. Позволяет отделить систему от окружающей среды.

- **Открытость**. Связь со средой. Наличие у системы входов (поступление информации из среды) и выходов (выдача результирующей информации в среду).
- **Внутренняя неоднородность.** Позволяет выделить в системе ее составные части.
- **Структурированность**. Наличие связей между частями системы.

Динамические свойства:

- **Функциональность**. Функции – это процессы, происходящие на выходах системы; результаты ее деятельности; продукция, ею производимая.
- **Стимулируемость**. Подверженность системы воздействиям извне и изменение ее поведения под этими воздействиями.
- **Изменчивость** со временем. Возможность изменения состава элементов, самих элементов, связей.
- **Устойчивость**. Существование в **изменяющейся среде**. Сохранение работоспособности системы при изменениях в предметной области.

Синтетические свойства:

- **Эмерджентность** (emergence – внезапное появление). Появление свойств системы как целого, отсутствующих у отдельных частей системы.
- **Неразделимость** на части. Следствие эмерджентности. Исчезновение некоторых свойств системы при выполнении операции ее декомпозиции.
- **Ингерентность** (inherent – являющийся неотъемлемой частью чего-то). Согласованность с окружающей средой, совместимость с ней.
- **Целесообразность**. Подчиненность определенной цели.

2.3. Понятие модели. Структурная модель

Основным способом изучения систем является построение ее **модели**. Существует множество определений модели. Учитывая сформулированное определение системы и перечень ее свойств, остановимся на следующем определении понятия модели [1]:

Модель есть системное отображение оригинала.

Выделим в этом определении два момента.

1. Отличительная особенность моделей от других систем состоит в их предназначении **отображать** моделируемый **оригинал**, т.е. содержать и представлять информацию об оригинале (системе).

2. Модель есть **системное** отображение оригинала. То есть, модель есть отображение:

- целевое,
- статическое или динамическое,
- ингерентное,
- проявляющаяся и развивающаяся в процессе его создания и практического использования,
- конечное, упрощенное, приближенное,
- абстрактное или реальное; абстрактность или реальность модели определяют ее **тип**.

Абстрактные (**искусственные)** модели являются результатом мыслительной деятельности. **Примеры**: математические модели (функции, системы уравнений), литературные произведения.

Реальные (**естественные**) модели. Для построения таких моделей используются материальные средства. **Примеры**: фотографии, макеты.

Таким образом, выше приведена классификация моделей по **типу**. С вопросами классификации моделей можно подробно ознакомиться в [1]. Здесь же остановимся на

описании важной для дальнейшего рассмотрения **структурной модели**, применяемой при исследовании внутреннего устройства системы.

Под **формальной структурой** понимается совокупность функциональных элементов и их отношений (связей), необходимых и достаточных для достижения системой заданных целей.

Под **связью** (**отношением**) в общем случае будем понимать упорядоченную пару типов объектов.

По **типу отношений между элементами** структуры подразделяются на:

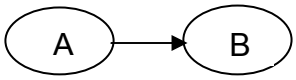
- структуры с **направленными** и **ненаправленными** связями,
- структуры с **односторонними** и **двусторонними** связями,
- структуры с **равноправными** и **неравноправными** связями,
- **иерархические** (древовидные) и **неиерархические** структуры.

Важным классом неиерархических структур являются **сетевые** структуры (**сети**) – неиерархические структуры с направленными связями.

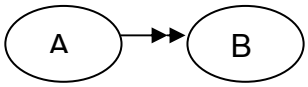
Рассмотрим виды связей, актуальные для теории баз данных.

Пусть даны два **типа объектов** A и B.

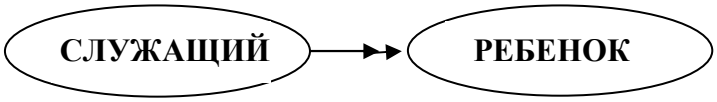
Различают следующие основные виды связей между этими объектами:

Один-к-одному (1:1). Обозначение 

Говорят, что типы A и B находятся в отношении 1:1, если в каждый момент времени каждому экземпляру типа объектов A соответствует один и только один экземпляр типа объектов B. Очевидно, что A идентифицирует B (если определен экземпляр A, то определен и экземпляр B). В качестве примера можно рассмотреть связь объектов **СЛУЖАЩИЙ** и **ЗАРПЛАТА** Примера 2.

Один-ко-многим (1:M). Обозначение 

Говорят, что типы A и B находятся в отношении 1:M, если в каждый момент времени каждому экземпляру из типа объектов A соответствует нуль, один или несколько экземпляров типа объектов B.



Очевидно, что при таком виде связи A не идентифицирует B.

В общем случае возможны четыре представления прямой и обратной связи:

1:1, 1:M, N:1, N:M.

Таким образом, **модель структуры системы** – это перечень существенных связей между элементами системы.

2.4. Модель предметной области и модель данных

Выше уже отмечалось, что для решения задач с использованием данных (как набора конкретных значений атрибутов), определяющих содержимое базы данных, сами данные должны быть полностью осмыслены и истолкованы, то есть отображать требуемую **информацию** о предметной области, что возможно только в случае определения **связей** между отдельными фактами (данными).

Таким образом, модель предметной области представляет собой *структурную модель*, причем необходимо, чтобы в этой модели информационной системы были определены **способы отображени**я сущностей, атрибутов и связей на структуры данных,

Однако, выделение объектов, их свойств и ассоциаций являются необходимым, но не достаточным условием существования модели предметной области. Для пользователей информационной системы важно, чтобы отображение объектов реального мира было **однозначным** и **непротиворечивым**. В этом случае говорят, что база данных должна удовлетворять условию **целостности**. Для того, чтобы гарантировать корректность и взаимную непротиворечивость данных, на базу данных накладываются некоторые ограничения, которые называют **ограничениями целостности,** то есть указывают условия, которым должны отвечать значения элементов данных, характеризующие объекты и связи (например, год рождения служащего не должен начинаться с 17..). В дальнейшем будут рассмотрены различные виды ограничения целостности.

Итак, **модель предметной области** – это описание структуры предметной области вместе с совокупностью связанных с ней ограничений целостности (**статическая** модель).
Динамическая модель включает, кроме того, описание **поведения** сущностей и связей каждого типа.

Учитывая сформулированные ранее определения БД и понятие модели, можно сказать, что:

Определение 4. БД – это созданная и поддерживаемая в вычислительной среде **статическая** или **динамическая модель предметной области**, представленная управляемой совокупностью именованных данных, отображающей состояния объектов и их отношений во внешнюю память ЭВМ.

Замечание. Как и всякая модель, БД отображает **определенный взгляд** на предметную область.

Однако, выделенных компонент БД как модели предметной области недостаточно для разрешения проблемных ситуаций и достижения поставленных целей. Решение задач возможно только при наличии **набора операций**, которые могут обрабатывать содержимое БД (ее элементы).

Таким образом, мы приходим к общему понятию **модели данных**, которая должна включать следующие компоненты:

- **допустимую организацию** данных,
- семантические **ограничения целостности**,
- множество **допустимых операций**.

Очевидно, что множество допустимых операций зависит от инструментария конкретной СУБД, в рамках которой реализуется модель предметной области.

Итак**, модель данных** можно определить как совокупность правил структурирования данных в базах данных, допустимых операций над ними и ограничений целостности, которым они должны удовлетворять.

Замечание. В модели данных могут учитываться не все виды ограничений целостности, например, в ней нельзя учесть результаты некорректного выполнения коллективных запросов к информационному хранилищу.

Обратим внимание на то, что понятие модели данных можно рассматривать в двух аспектах:

- как инструментарий СУБД (средства описания данных и манипулирования ими),
- как результат моделирования.

Результирующую модель обычно называют **моделью базы данных**. Заметим при этом, что функции моделей в этих аспектах существенно различаются. В настоящее время термин **модель базы данных** считается устаревшим (хотя он в литературе и встречается), под моделью данных принято понимать инструментарий СУБД, а конечным результатом моделирования в рамках выбранной СУБД являются **схемы базы данных** разных уровней (см. п.3.3).

КОНТРОЛЬНЫЕ ВОПРОСЫ ПО ВТОРОМУ РАЗДЕЛУ

1. Дайте определение следующим базовым понятиям: данные, элемент данных, атрибут, объект, предметная область.
2. Что определяет семантику объекта.
3. Сформулируйте определение базы данных, исходя из понятия предметной области.
4. Дайте определения понятиям: проблема, проблемная ситуация, цель, проблемная среда.
5. Дайте общее определение понятию системы. Приведите основные свойства системы как объекта исследования.
6. Дайте общее определение понятию модели. В чем отличительная особенность модели от других видов систем. Перечислите системные свойства модели.
7. Сформулируйте определение базы данных как модели предметной области.
8. Сформулируйте понятие модели данных. Какие составляющие должны быть определены в модели, чтобы ее можно было рассматривать как модель данных.
9. В чем отличие модели предметной области и поддерживаемой инструментарием СУБД определенной модели данных.

3. ПОНЯТИЕ О БАНКЕ ДАННЫХ

3.1. Структура банка данных

Термин “**Банк данных”** в литературе трактуется по-разному. В нашем рассмотрении под **банком данных** (БнД) будет подразумеваться информационная система, в которой база данных выступает как информационное ядро. В этой трактовке БнД можно определить как систему **языковых**, **алгоритмических**, **программных**, **организационных** и **технических средств**, обеспечивающих централизованное создание и поддержку совокупности коллективно используемых данных, а также сами данные, существующие в форме одной или нескольких баз данных. Структуру такого БнД можно представить следующей схемой (рис.5.).

Конечным пользователем (или просто **пользователем, потребителем** информации) является лицо (или коллектив), в интересах которого в БнД накапливается и хранится информация, необходимая для принятия решений (управленческого, научного, конструктивного характера и т.п.). Конечный пользователь рассматривается как непрограммирующий пользователь, который для решения своих задач может использовать БнД либо непосредственно через терминал ЭВМ, либо с помощью специалистов.

Коллектив специалистов включает категории разработчиков, ответственных за создание и ведение (поддержку, эксплуатацию) БнД.

СУБД (система управления базой данных) – сложная программная система накопления данных в БД и последующего манипулирования ими в интересах конечных пользователей. Каждой прикладной программе (ПП) или конечному пользователю СУБД возвращает только те данные из БД, которые необходимы для удовлетворения пришедшего запроса, причем в требуемой форме.

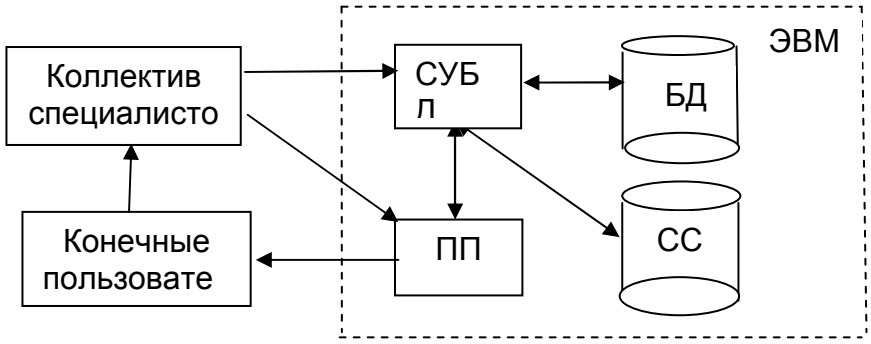


Рис. 5. Общая структура банка данных

ПП (прикладные программы) – комплекс прикладных программ (приложений), определяемых проблемной средой. Каждая из прикладных программ предназначена для решения определенной задачи (или класса задач), возникшей в ходе профессиональной деятельности конечных пользователей.

СС (словари-справочники) – вспомогательные информационные структуры, используемые СУБД для работы с БД. Управление содержимым БД осуществляется СУБД на основании точной и полной информации о данных, хранящихся в БД. Эта информация, часто называемая **метаданными**, включает описание:

- смысла (семантики) элементов данных,
- **с**пособов их использования,

- физических характеристик,
- правил и ограничений**.**

Заметим, что термин **метаданные** рассматривается с точки зрения функционирования СУБД: применительно к предметной области и проблемной среде (прикладным программам) эта информация рассматривается как соответствующие виды **знаний**. Метаданные позволяют проводить анализ требований конечных пользователей по новым данным, проектирование и программирование новых прикладных систем, сопровождение существующих приложений и документирование всех этапов развития БД. К основным функциям словарей-справочников можно отнести:

- **хранение описателей** сущностей, их атрибутов, связей, и т.д.,
- осуществление простого и эффективного **управления элементами данных при вводе** в систему как новых элементов, так и при **изменении описания** существующих,
- уменьшение **избыточности**,
- устранение **противоречивости** данных,
- **централизация управления элементами данных** с целью упрощения проектирования БД и ее расширения.
- **установление связи** между пользователями БД.

Важную роль в решении перечисленных проблем играет стандартизация имен, форматов, описаний элементов и структур данных. При автоматизированном ведении словаря-справочника решение этих проблем упрощается.

Словари-справочники обычно организуются в виде нескольких физических баз данных с логическими связями между ними.

3.2. Организационный аспект

Коллектив специалистов, обеспечивающий разработку и функционирование БнД, включает **администратора** БД, **аналитиков, системных** и **прикладных программистов.** На рис.6. показано их взаимодействие между собой и конечными пользователями.

КП – задача конечного пользователя.

Администратор базы данных. Как и каждый общезначимый ресурс, БД требует отдельного управления, причем:

- БД требует управления для обеспечения ее **повседневной эксплуатации**,
- БД развивается, отвечая изменениям в потребностях предприятия, и требуется **управление ее развитием**,
- БД и технология ее разработки и развития являются объектами высокой сложности, требующими специальных знаний, **высокого уровня квалификации** и строгой дисциплины разработки и эксплуатации.

Функция управления БД получила название «**администрирование базы данных**», а лицо, ответственное за администрирование БД**,** получило название «**Администратор базы данных»,** или **АБД.**

АБД – это специалист, имеющий представление об информационных потребностях конечных пользователей, работающий в тесном контакте с ними и отвечающий за анализ потребностей пользователей.

Классический набор функций, выполняемых АБД, включает:

- организационное и техническое **планирование** БД,
- **проектирование** БД,
- обеспечение поддержки **разработок прикладных программ**,
- управление **эксплуатацией** БД.

При этом от непосредственного управления данными отстраняются программисты, выполняющие конкретные прикладные разработки, пользователи, которые не должны изменять или даже видеть не принадлежащие им данные, и другие сотрудники.

Очевидно, что необходимость концепции администратора была осознана в период перехода от файловых структур к системам баз данных. Более подробно роль АБД на этапах планирования и проектирования рассматриваются в разделе 4.

Системные программисты занимаются созданием базового программного обеспечения. Генерируют операционную систему, в рамках которой предполагается функционирование СУБД, саму СУБД, необходимые компиляторы и обслуживающие утилиты.

Аналитики, используя знания закономерностей определенной проблемной среды, строят ее математическую модель, привлекая необходимый математический инструментарий. Основная функция аналитика – представить задачу КП в форме некоторой формальной модели (“погрузить” задачу пользователя в математическую модель его проблемной области). Конечная цель аналитика – исходное представление задачи для прикладного программиста.

Прикладной программист преобразует продукт деятельности аналитика в форму прикладной программы, предназначенной для реализации на ЭВМ.

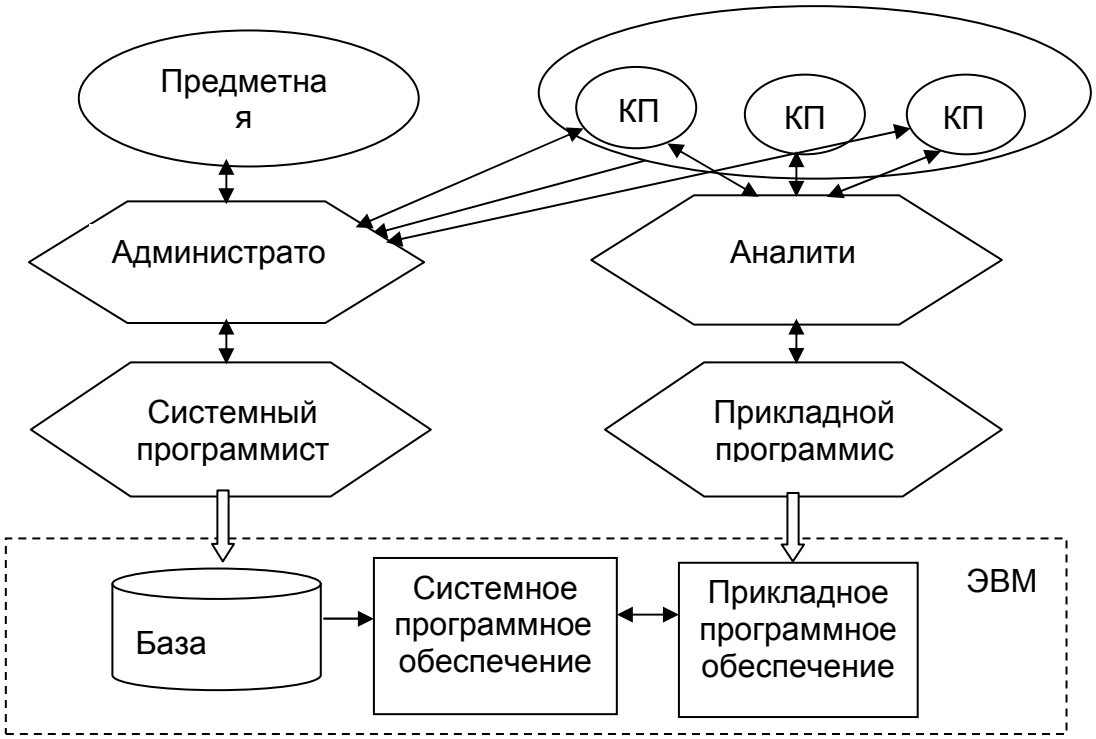


Рис. 6. Состав коллектива специалистов

3.3. Уровни представления базы данных

Итак, для конечных пользователей база данных – хранилище сведений о клиентах и счетах, договорах и поставках, дисциплинах и оценках. Однако для внутренних пользователей (коллектива специалистов) эта же система выступает в виде элементов данных, записей, файлов. Ясно, что и углы зрения на систему самих внутренних пользователей также различны. Прикладные программисты оперируют элементами данных, записями, структурными связями, не имея, как правило, представления о физической организации данных, что, в свою очередь, является прерогативой администратора и системных программистов. Иными словами, в информационной системе поддерживается несколько **уровней представления (абстракций)**. Их разновидности принято ассоциировать с понятием **архитектуры** информационной системы.

Впервые в общем виде концепция многоуровневой архитектуры базы данных была сформулирована и достаточно детально проработана в рабочем отчете группы по базам данных Американского Национального Института Стандартов ANSI/X3/SPARC в 1975 г. В отчете была предложена обобщенная архитектурная модель систем баз данных, включающая три уровня: уровень внешних схем данных, уровень концептуальной схемы данных и уровень схемы физического хранения данных.

В соответствии с этой архитектурой определялись три роли АБД: администратор концептуальной схемы, администратор внешних схем и администратор хранения данных. Эти роли в случае очень маленькой системы мог играть один человек, в большой системе для выполнения каждой роли могла назначаться группа людей, в совокупности эти группы и выполняли функции АБД.

В отчете ANSI/X3/SPARC были даны определения следующим понятиям.

1. Модели предметной области:

- **Концептуальная.** Ограниченная модель реального мира, поддерживаемая **для всех приложений** системы баз данных.
- **Внешняя** Упрощенная модель реального мира, разработанная для поддержки одного или нескольких приложений системы баз данных. Чаще всего представляет собой некоторое подмножество концептуальной модели предметной области. Эту модель можно отождествить с локальными представлениями конечных пользователей.
- **Внутренняя.** Структура хранимых данных, представляющих концептуальную модель предметной области, а так же данных, необходимых для обеспечения эффективных технологий использования ресурсов аппаратно-программной платформы рассматриваемой системы баз данных.

2. Модели данных:

- **Концептуальная.** Поддерживаемая используемой СУБД модель данных концептуального уровня архитектуры систем баз данных. В терминах этой модели описываются **концептуальные схемы** конкретных баз данных.
- **Внешняя.** Поддерживаемая используемой СУБД модель данных внешнего уровня архитектуры систем баз данных. В терминах этой модели описываются **внешние схемы** конкретных баз данных.
- **Внутренняя.** Поддерживаемая используемой СУБД модель данных внутреннего уровня архитектуры систем баз данных. В терминах этой модели описываются **внутренние схемы** конкретных баз данных.

3. Схемы:

- **Схема концептуальная.** Описание концептуальной модели предметной области средствами языка определения данными концептуального уровня архитектуры используемой СУБД.
- **Схема внешняя.** Описание внешней модели предметной области средствами языка определения данными внешнего уровня архитектуры используемой СУБД. В одной системе баз данных СУБД может одновременно поддерживаться несколько внешних схем.
- **Схема внутренняя.** Описание внутренней модели предметной области средствами языка определения хранимых данных выбранной СУБД. Иначе говоря, это описание организации базы данных в среде хранения данных используемой СУБД.

В терминах перечисленных понятий рассмотрим схему одного из ранних (70-е – 80-е годы прошлого века) подходов в определении архитектуры информационной системы (рис. 7). На этой схеме выделены следующие уровни абстракции.

ЛПП (локальные пользовательские представления) - начальный уровень абстракции; соответствует представлениям о предметной области конечных пользователей на уровне руководителей подразделений (главный бухгалтер, начальник отдела кадров и т.п.).

Инфологический (информационно-логический) – второй уровень. Представляет собой интеграцию ЛПП, соответствующую ”взгляду” на предметную область ее администратора (директора, президента фирмы, ректора вуза и т.п.). Администратор предметной области (не

путать с администратором базы данных!) обозревает все множество информационных объектов, все возможные ассоциации между ними, в то время как каждый конечный пользователь просматривает лишь ограниченный фрагмент предметной области (бухгалтерия, цех, кадры и т.п.).

Инфологическая модель предметной области - это описание структуры и поведения (динамики) предметной области в терминах, естественных и понятных пользователям разрабатываемой системы баз данных и учитывающих характер их информационных потребностей. Таким образом, ЛПП и инфологическое представление о предметной области определяет лишь информационные потребности разрабатываемой системы, отражает особенности предметной области, не затрагивая вопроса о способах отображения соответствующих данных в памяти ЭВМ. Инфологическая модель существует **независимо от реализации** системы, в частности, от выбора инструментария СУБД.

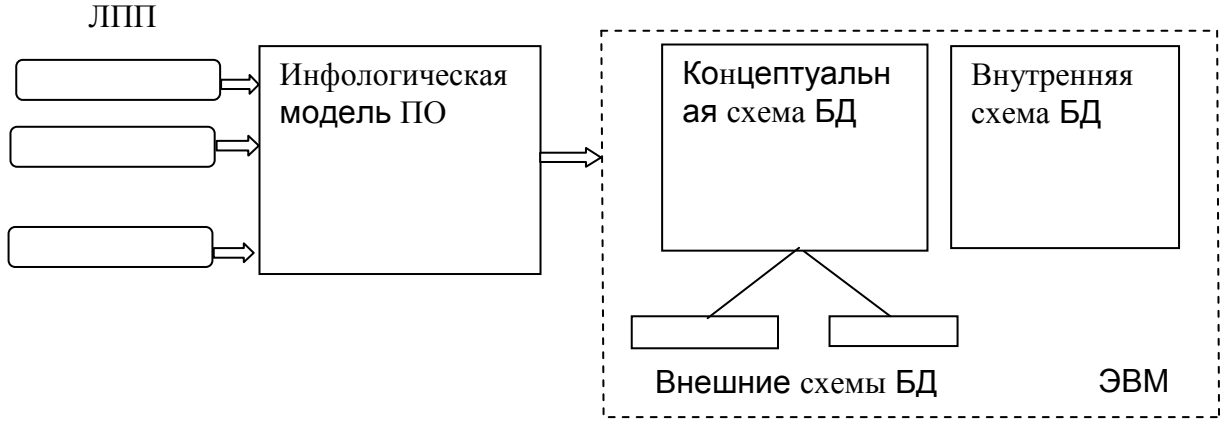


Рис. 7. Уровни представления информационной системы

Обычно это описание выражается в терминах **не отдельных объектов** ПО и связей между ними, а их **типов,** связанных с ними ограничений целостности и тех процессов ПО, которые приводят к переходу ее из одного состояния в другое.

Такое описание представляется в любом виде, допускающем однозначную интерпретацию. В простейших случаях описание создается на естественном языке, в более сложных используется также математический аппарат: таблицы, диаграммы, графы и т.п.

Поэтому инфологическое представление можно считать **бумажной** моделью, существующей в виде диаграмм и текстовых описаний. Одной из наиболее популярных моделей данных подобного рода является модель сущностей-связей (Entity-Relationship, ER-модель) П. Чена.

Концептуальный уровень абстракции соответствует представлению о логической организации данных администратора базы данных. Этот уровень абстракции очень схож с инфологическим; его отличие состоит в привязке к средствам реализации, инструментарию СУБД, то есть модель предметной области разрабатываемой системы должна быть представлена в терминах (на языке описания данных) модели данных концептуального уровня (**концептуальной модели**) выбранной конкретной СУБД. Эту стадию называют **логическим проектированием** базы данных, а ее результатом является **концептуальная схема** базы данных.

Таким образом, описание БД на концептуальном уровне задается на языке описания данных используемой СУБД в терминах и ограничениях, принятых в этой системе. Поскольку каждая СУБД поддерживает свою **модель данных** (свой инструментарий), описание единой инфологической модели предметной области в рамках разных СУБД приводит к разным (но, в идеале, эквивалентным) концептуальным схемам.

Внешние представления данных отражают взгляд на предметную область прикладных программистов.

Внешние схемы базы данных конструируются на стадии разработки приложений и их главная функция – отобразить форму представления данных для приложений.

Внутренний (физический) уровень абстракции представляет способы организации хранения данных во внешней памяти ЭВМ и отражает взгляд на разрабатываемую информационную систему администраторов и системных программистов. Очевидно, что параметры внутреннего уровня существенно влияют на эффективность эксплуатации БД (например, требуемый объем внешней памяти, время ответа системы на запрос данных).

Стадия физического проектирования базы данных заключается в выборе желаемого способа организации базы данных в среде хранения выбранной СУБД, в разработке спецификаций **внутренней схемы** средствами внутреннего уровня модели данных (ее **внутренней модели**).

Замечание. Нетрудно видеть, что в терминах трехуровневой модели этапы проектирования первого и второго уровней, представленные на рис. 7., объединяются в процесс проектирования **концептуальной модели предметной области** (**логического** проектирования).

Уже на ранней стадии разработок систем баз данных в качестве одного из основополагающих принципов построения таких систем был принят принцип **независимости** данных. В соответствии с этим принципом, в системе должны поддерживаться раздельные представления данных для пользователя («**логическое представление**») и для системных механизмов среды хранения базы данных («**физическое представление**»). Такое разделение избавляет пользователя от необходимости знания принятого способа хранения базы данных и, вместе с тем, при условии поддержки управляемого отображения между этими представлениями, позволяет динамически в процессе эксплуатации системы оптимизировать способ хранения базы данных для обеспечения более высокой производительности системы и/или более рационального использования ресурсов памяти. В соответствии с таким подходом, стали различать логический и физический архитектурные уровни системы (системные механизмы, поддерживающие такие представления данных). Связанные с ними модели данных и схемы базы данных также стали называть логическими и физическими.

Замечание. В отличие от ранних СУБД, многие современные системы не предоставляют разработчику какого-либо выбора на стадии физического проектирования. Способ хранения базы данных определяется механизмами СУБД автоматически «по умолчанию» на основе спецификаций концептуальной схемы базы данных, и внутренняя схема в явном виде в таких системах не используется.

Разновидности моделей данных будут рассмотрены в разделах 5. и 8.

КОНТРОЛЬНЫЕ ВОПРОСЫ ПО ТРЕТЬЕМУ РАЗДЕЛУ

1. Приведите схему общей структуры банка данных.
2. Приведите общую схему коллектива специалистов. Перечислите основные функции аналитиков, системных программистов, прикладных программистов.
3. Перечислите функции администратора базы данных.
4. Перечислите функции словаря-справочника.
5. Приведите схему уровней представления (абстракций) информационной системы.
6. Дайте понятие инфологической модели. В чем отличие инфологической модели от концептуальной.
7. Опишите понятия, сформулированные в отчете ANSI/SPARC:
   - Модель предметной области (концептуальная, внешняя, внутренняя).
   - Модель данных (концептуальная, внешняя, внутренняя).
   - Схема (концептуальная, внешняя, внутренняя).
8. Сформулируйте отличие в определении ANSI/SPARC понятия **модели предметной области** от понятия **модели данных** (безотносительно уровня абстракции).

4. ВОПРОСЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

4.1. Жизненный цикл информационной системы

Как и любая другая, информационная система проходит во времени свой определенный жизненный цикл. В зависимости от целей исследования, в жизненном цикле БД можно определить различные последовательности этапов. С точки зрения проектировщика и пользователя согласно [2] выделим две фазы жизненного цикла базы данных:

- **анализ** и **проектирование** – начальный (“бумажный”) этап жизни БД,
- **реализация** и **эксплуатация** системы.

Анализ и **проектирование.** Этап выполняется посредством изучения предметной области и требований, предъявляемых к создаваемой БД. На “бумажной” стадии жизни системы производится выбор:

- структур данных и стратегии их хранения в памяти ЭВМ,
- технологии обслуживания БД и взаимодействия с ней конечных пользователей,
- технических и стандартных программных средств, а также разработка оригинальных программных средств обслуживания системы.

Реализация и **эксплуатация.** Сущность реализации заключается в материализации проекта, в перенесении его в память ЭВМ. На этой стадии разрабатывается и отлаживается программное обеспечение информационной системы, создается отладочный вариант БД, разрабатываются многочисленные приложения. На стадии реализации тестируется и корректируется технология обслуживания информационной системы.

Эксплуатация начинается с наполнения системы реальной информацией. Эта стадия жизненного цикла охватывает весь комплекс действий по поддержанию функционирования информационной системы:

Очевидно, что стадия эксплуатации включает в себя разработку новых приложений, а также совершенствование и последующее развитие системы.

Методы построения моделей. Учитывая, что главной целью данного пособия является знакомство читателя с основными подходами к организации процесса проектирования базы данных как модели предметной области, а основными методами изучения реальности являются методы анализа и синтеза, согласно [1] определим **аналитический** (метод **анализа**) и **синтетический** (метод синтеза) следующим образом.

В процессе **анализа** определяется структура системы, т.е. то, как она устроена. Процедура анализа состоит в последовательном выполнении следующих трех операций:

1. Сложное целое расчленить на более мелкие части, предположительно более простые.
2. Дать полное объяснение полученным фрагментам.
3. Объединить объяснение частей в объяснение целого.

Если какая-то часть системы остается непонятной, шаги анализа осуществляются для этой части.

Первым продуктом анализа является перечень элементов системы, т.е. **модель состава** системы.

Объяснение целого – это установление его эмерджентных свойств; для этого необходимо **установить связи** между частями. Таким образом, вторым продуктом анализа является **модель структуры** системы.

В процессе **синтеза** определяется функционирование системы, т.е. ее взаимодействие со средой. Процедура **синтеза** включает последовательное выполнение трех операций:

1. Выделение большей системы (метасистемы), в которую моделируемая система входит как часть.

2. Рассмотрение состава и структуры метасистемы.

3. Объяснение роли, которую играет моделируемая система в метасистеме, через ее связи с другими частями метасистемы.

Конечным продуктом синтеза является знание связей моделируемой системы с другими частями метасистемы, т.е. модель **черного ящика**.

Очевидно, чтобы построить модель черного ящика, необходимо попутно создать модели состава и структуры метасистемы как побочные продукты.

Объединение трех моделей в единое целое позволяет сформировать модель **белого (прозрачного) ящика**, или **структурную схему системы**.

Применение описанных процедур будет рассмотрено ниже при обсуждении этапов проектирования базы данных.

4.2. Процесс проектирования

4.2.1. Организационный аспект

В роли **заказчика**, то есть основного носителя сведений о предметной области и требований об информационной системе, при проектировании выступают:

- администратор **предметной области** (АПО),
- администраторы **фрагментов предметной области**,
- коллективы **конечных пользователей**.

АПО открыта перспектива всей организации (предприятия, банка, вуза, фирмы и т.д.) и он, как правило, совместно с руководителями подразделений (фрагментов ПО) решает вопрос о необходимых ресурсах для эффективного функционирования организации. Поскольку целью создания базы данных, очевидно, является создание информационного хранилища для разрабатываемой автоматизированной информационной системы организации, деятельность АПО в конечном счете направлена на обеспечение адекватности проекта базы данных интегральным информационным потребностям приложений (прикладных программ), реализующих соответствующие **бизнес-процессы**.

Замечание. Под **бизнес-процессом** можно понимать связанную совокупность бизнес-функций, в ходе выполнения которой потребляются определенные ресурсы и создается продукт (предмет, услуга и т.д.), представляющий ценность для потребителя.

Общение с конечными пользователями позволяет учесть в разрабатываемой модели специфику ПО, проблемы низшего звена организации-заказчика.

Группу проектировщиков возглавляет **администратор базы данных** (АБД) – специалист по информационным системам. Учитывая, что АБД может не быть специалистом в ПО, ему в помощь организуется группа **аналитиков** (консультантов), стыкующих (согласующих) работу разработчиков и конечных пользователей. Естественным представляется включение в группу проектировщиков **системных программистов** и **разработчиков приложений** (см. п.3.2).

Администратор баз данных реализует процессы **детального планирования** и **проектирования**. Он осуществляет анализ и синтез данных для каждой создаваемой базы данных. АБД необходимо, чтобы администраторы-заказчики подробно рассмотрели каждую БД с целью сделать ее как можно стабильнее. Все многообразие задач, выполнение которых возлагается на АБД, можно разделить в соответствии с этапами жизненного цикла БД (для полноты перечислим все этапы жизненного цикла, не ограничиваясь только этапом проектирования).

Анализ и проектирование:

- работа с заказчиками для установки реальных целей и требований к прикладным программам и базам данных,
- управление процессами **логического** и **физического проектирования**,

- выбор связанного с БД **программного обеспечения и оборудования**,
- **долгосрочное планирование**, в том числе в определении перспектив расширения БД.

Реализация:

- реализация проекта инструментальными средствами выбранной СУБД,
- создание отладочного варианта БД,
- разработка и отладка программного обеспечения информационной системы,
- разработка приложений,
- тестирование и коррекция технологии обслуживания информационной системы.

Эксплуатация и использование:

- управление процессами **включения новых данных** в базу и внесения изменений,
- разработка и контроль действий, гарантирующих **сохранение целостности** БД, включая процедуры ее копирования и восстановления после сбоев,
- организация **защиты** БД с помощью механизмов управления доступом и средств СУБД,
- введение стандартов на содержимое и использование БД,
- сопровождение специальных средств программного обеспечения для работы с БД (**словари-справочники данных, языки запросов**),
- проведение консультаций пользователей БД.

С другой стороны, рассмотренные задачи можно разделить на два класса:

- **административные** и **технические,**
- **прикладные** и **системные**.

АБД, таким образом, является лицом, ответственным за достоверность и полноту данных, содержащихся в БД, их согласованность, а также за соблюдение регламента работ по актуализации БД.

4.2.2. Задачи и структура процесса проектирования

Проектирование базы данных - одна из наиболее сложных и ответственных задач, связанных с созданием информационной системы. В результате ее решения должны быть определены содержание БД, эффективный для всех ее будущих пользователей способ организации данных и инструментальные средства управления данными. Основная цель процесса проектирования БД состоит в получении такого проекта, который удовлетворяет следующим требованиям:

- **Корректность** схемы БД, а именно:
  - каждому объекту ПО соответствуют данные в памяти ЭВМ,
  - каждому процессу ПО соответствуют адекватные процедуры обработки данных.
- Обеспечение **целостности**:
  - формулировка ограничений,
  - описание правил применения ограничений,
  - описание правил обработки данных при нарушении ограничений целостности.
- **Защита данных**:
  - защита от разрушений при сбоях оборудования (**восстанавливаемость** данных),
  - от некорректных обновлений (**согласованность** методов модификации данных),
  - от несанкционированного доступа (**безопасность** данных).
- Обеспечение **ограничений на аппаратные** и **программные** ресурсы вычислительной системы.
- **Эффективность функционирования**;
  - эффективность использования ресурсов,
  - обеспечение требований ко времени реакции системы на запросы и обновление БД.

- **Простота** и **удобство** в эксплуатации.
- **Гибкость** – возможность развития и последующей адаптации системы к изменениям в ПО и к новым потребностям пользователей.

Удовлетворение первых 4-х требований обязательно для принятия проекта.

В структуре процесса проектирования выделим следующие этапы:

Этап 1. Формулировка и анализ требований, включающий:

Шаг 1. Сбор и анализ априорной информации о ПО**,**

Шаг 2. Анализ и синтез инфологической модели ПО.

Этап 2. Проектирование концептуальной схемы.

Этап 3. Физическое проектирование.

На каждом из этапов решаются следующие основные задачи.

Этап 1.

Шаг 1. Устанавливаются цели организации, определяются специфические требования к БД, вытекающие из этих целей или сформулированные непосредственно управляющим персоналом организации. Эти требования документируются в форме, доступной как конечному пользователю, так и проектировщику БД.

Специфические цели и требования к БД необходимо определить на самом высоком уровне организации-заказчика (уровень АПО). Собранные и документированные требования должны включать ограничения, обусловливающие:

- безопасность,
- надежность,
- уровень достигнутой технологии,
- политические и бюрократические ограничения.

Должна быть также оценена политика организации в отношении персонала, управленческой деятельности и перспектив роста организации.

Шаг 2. Описываются и анализируются разнообразные информационные требования пользователей и производится синтез первоначального проекта базы данных. Результатом этого этапа является «бумажное» высокоуровневое представление требований в виде, например, ER-модели (модели **сущность-связь**»). На этом этапе осуществляется:

- определение сущностей,
- определение атрибутов сущностей,
- идентификация ключевых атрибутов сущностей,
- определение связей между сущностями.

Требования каждого пользователя анализируются и представляются в некоторой общей форме. Объекты и события, ассоциированные с представлениями каждого пользователя, моделируются множеством сущностей, атрибутов и связей между сущностями. Эти требования, представленные, например, набором **локальных** ER-моделей, анализируются и сливаются в единое **глобальное** представление. Очевидно, что в процессе анализа должны быть выявлены и ликвидированы противоречивые и избыточные данные (более подробно этот этап рассмотрен в 4.2.3.).

Этап 2. После построения первоначальной информационной структуры следует ее уточнение и совершенствование. Главная цель – создание СУБД-ориентированной концептуальной схемы с использованием в качестве исходных данных результатов инфологического проектирования и требований обработки данных.

На данном этапе проектируются также многочисленные приложения. Результатом проектирования программного обеспечения являются:

- интерфейсы приложений,
- функциональные характеристики приложений,
- наборы возможных запросов к БД,

Построенная схема может быть оценена количественно с помощью таких характеристик, как:

- число обращений к логическим записям каждого приложения,

- объем обрабатываемых в каждом приложении данных,
- общий объем хранимых данных.

Эти оценки помогают приблизительно определить эффективность функционирования физической БД в терминах затрачиваемого на обработку времени и требуемой физической памяти.

Этап 3. Рассмотрению вопросов физического проектирования базы данных посвящена отдельная глава данного пособия. Здесь лишь отметим, что параметры внутренней (физической) модели БД принципиальным образом влияют на эксплуатационные характеристики информационной системы.

Процесс проектирования хорошо структурирован, так как каждый его этап завершается четко определенным результатом, и допускает итеративное повторение в случае, если полученный результат не соответствует требованиям заказчиков или сформулированным ограничениям, либо если возникают дополнительные требования. В общем случае это позволяет проектировщику пересматривать проектные решения с любого предыдущего этапа. Схема метода нисходящего (**каскадного**) проектирования с последовательными итерациями представлена на рис. 8.

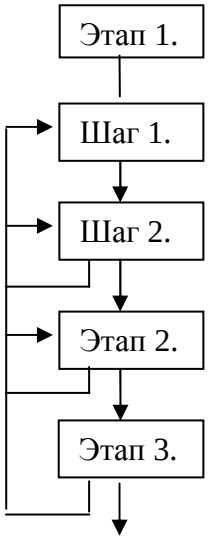


Рис. 8. Прямой и итеративный варианты в каскадной методологии

4.2.3. Формулирование и анализ требований. Инфологическое проектирование

Шаг 1. Сбор и анализ априорной информации о ПО

Проектирование базы данных – это процесс разработки ее структуры в соответствии с информационными требованиями пользователей, поэтому он включает в себя опрос будущих пользователей для того, чтобы понять и задокументировать их требования. АБД следует выяснить следующие вопросы:

- что представляют собой требования пользователей; в какой форме они могут быть выражены;
- как требования пользователей могут быть преобразованы в эффективную структуру базы данных;
- какие данные используются разными приложениями; смогут ли приложения совместно использовать какие-либо из этих данных;
- сможет ли новая система объединить существующие приложения или их необходимо будет кардинально переделывать для совместной работы с новой системой;
- кто будет вводить данные в базу и в какой форме; как часто будут изменяться данные;

- достаточно ли будет для ПО одной базы или потребуется несколько баз данных с различными структурами;
- какая информация является наиболее чувствительной к скорости ее извлечения и изменения;
- как часто и каким образом структура базы данных должна перестраиваться в соответствии с новыми и/или изменяющимися требованиями.

Согласно [3], на этом шаге осуществляется:

1. Определение сферы применения.

2. Сбор информации об использовании данных.

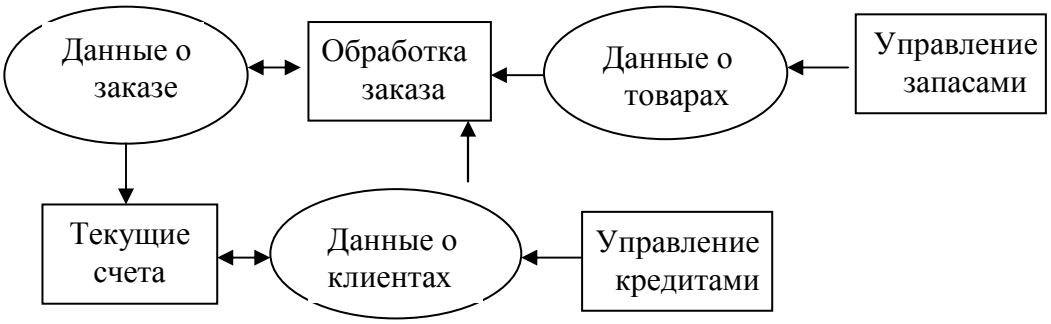
3. Преобразование инфомационных требований.

Рассмотрим каждый из этих пунктов подробнее.

Пункт 1. Определение сферы применения баз данных как в настоящем, так и в будущем

Лучшим источником такой информации является **информационная схема** организации.

Пример. Фрагмент информационной схемы и системных зависимостей.



Используя подобную схему в качестве основы, можно определить бизнес-функции и бизнес-процессы, которые следует рассмотреть в рамках проекта. Если информационная схема в организации отсутствует или эта схема не содержит диаграммы зависимостей данных и задач, определение сферы применения проекта возлагается на АБД.

Другой фактор, который необходимо принимать во внимание при определении сферы применения – возможные в будущем изменения в деятельности организации. Если обнаружится, что будущие изменения могут оказать воздействие на БД, сфера применения проекта должна быть расширена с тем, чтобы учесть влияние возможных изменений.

Пункт 2. Сбор информации об использовании данных.

Для полноты обсуждения данного пункта заметим, что инфомацию об использовании данных можно разделить на два вида [3]:

1. информация, связанная с производственными функциями,
2. информация, связанная с функциями управления.

Производственные функции организации. Собранная инфомация о производственных функциях и об использовании данных является основной исходной инфомацией процесса проектирования базы данных. Поэтому достоверность и полнота собранной информации являются чрезвычайно важными факторами.

Для каждой производственной функции у **руководителей подразделений** выявляется:

- наименование работы,
- выполняемая функция,
- цель выполняемой работы.

Функции управления. На этом шаг определяются:

- функции контроля и планирования и требуемые для выполнения этих функций данные,
- предположения о возможных изменениях в деятельности организации в будущем.

Инфомация об управляющих функциях должна быть определена путем консультаций с высшим руоводством – **АПО**. В этих обсуждениях АБД должен получить общее представление о следующих вопросах:

- основные компоненты деятельности и их взаимодействие друг с другом, то есть вопросы взаимодействия различных фрагментов ПО,
- внутренняя среда, включающая:
- структуру организации,
- правила и политику, определяющие повседневную деятельность,
- внешняя среда, прямо или косвенно влияющая на деятельность организации (законодательные органы, рынки сбыта и т.п.),
- информация, требуемая для планирования деятельности,
- виды информации, используемой для контроля и оценки функционирования,
- предполагаемые изменения, которые могут влиять на род или сферу деятельности, либо на способы ведения деятельности.

Пример. Выявленная информация о производственных и управленческих функциях может быть задокументирована (представлена) в следующем виде.

Наименование работы	Функция	Цель
Комплектация заказов [1)]	Комплектация товаров со склада соответствии с заказом	Выполнение заказов клиент
Планирование запасов товаров[1)]	Покупка товаров	Подержка запаса товаров
Управление запасами товаров [2)]	Определение оптималь- ного количества товар и времени для их закупки	Минимизация капиталловложения при создании запаса товаров
Прием заказов [1)]	Выписка заказа	Прием заказа клиента

[1)] – основная (производственная) функция , [2)] – функция управления.

Идентификация производственных задач и задач управления. На этом шаге определяются связи между выявленными функциями и необходимыми для их реализации данными.

Здесь функцию (в [3] **задачу**)определим как самый низкий уровень деятельности, который многократно использует уникальный набор данных.

Выделяются следующие характеристики функции:

- Это уникальная единица деятельности, состоящая из набора последовательно выполняющихся шагов (**операций**),
- все шаги направлены на достижение одной и той же цели,
- на каждом шаге создается или используется один и тот же (уникальный) набор данных.

Связь между функцией и данными можно определить как уникальную связь, возникающую между элементами данных при их использовании.

Пункт 3. Преобразование информационных требований

Процесс преобразования информации, собранной во время собеседований, в форму, используемую при дальнейшем анализе, включает следующие основные шаги:

1. Составление списка всех используемых и создаваемых элементов данных.
2. Определение всех производственных функций, их характеристик и используемых данных.

3. Определение всех функций управления, их характеристик и используемых данных.

4. Составление списка всех явных и неявных правил и линий поведения в управлении деятельностью организации.

5. Составление списка возможных будущих изменений и путей их влияния на деятельность организации.

Пример. Список элементов данных может выглядеть следующим образом.

Идентификатор	Наименование	Определение
1	Номер заказа	Однозначно определяет каждый заказ
2	Заказанное количество	Определяет количество одного вида товара, заказанного одним клиентом

Сформированный список элементов данных служит основанием для создания **словаря элементов данных**.

Первичные **метаданные** (описания и определения данных, то есть данные о данных) необходимы для построения и уточнения инфологической модели, используемой в дальнейшем в качестве общей основы для сбора и анализа других метаданных. Без такой основы собранные метаданные едва ли будут достаточно надежными: количество метаданных может оказаться больше, чем необходимо, а их структура может оказаться недостаточно эффективной для выявления ошибок.

Предполагается, что приложения определяются независимо друг от друга либо аналитиками, либо специалистами ПО. Каждое подразделение решает свои задачи **вне** каких-либо внешних ограничений. Это означает, что каждая отдельная функция обработки данных будет определена в рамках локального представления данных (см. уровень ЛПП на рис.7) без учета представлений пользователей других приложений.

Таким образом, во время общего анализа требований должны быть определены элементы данных и их связи, которые будут уточняться на последующих этапах проектирования. Здесь же должны быть, по крайней мере, обнаружены все **конфликтные ситуации**. Дело в том, что представления пользователей о данных и их обработке не всегда могут служить прообразом окончательной структуры базы данных; они лишь устанавливают формальную основу для накопления метаданных. Интеграция представлений осуществляется на следующих этапах проектирования.

Итак, первый шаг предполагает:

- **сбор данных** (результатов измерений, наблюдений, отчетов, документов научно-технического, технико-экономического, бухгалтерского и т.п. характера, опроса экспертов – специалистов в данной предметной области),
- **выявление перечня задач**, которые могут быть решены с использованием разрабатываемой БД,
- **содержательный анализ** собранной информации.

Целью содержательного анализа собранной информации является:

- **устранение дублирования** данных,
- **выявление и устранение противоречивости** данных и неоднозначности их определений и описаний,
- выявление и формулирование **правил поведения и принятия решений** в соответствии с описаниями ситуаций,
- выявление **возможных изменений** в будущем и их влияния на правила поведения и т.п.

Другими словами, на этом этапе происходит “погружение” АБД в предметную область и проблемную среду. Именно на этом этапе формулируются цели создания БД.

Выходом первого этапа очевидным образом является:

- описание **целей** создания БД,

- описание **входной** и **выходной информации**,
- **модель состава** системы.

Содержательный анализ собранной информации позволяет определить возможные **динамические** свойства создаваемой модели.

Шаг 2. Анализ данных и синтез инфологической модели

Общая схема построения **инфологической** модели предметной области с учетом целей конечных пользователей и информационных требований прилоложений изображена на рис. 9.

В этой схеме предусмотрено построение инфологической модели путем объединения информационного описания предметной области (**ПО-информация**) и информационных требований прикладных программ (**ПП-информация**). Следовательно, в инфологической модели объединяются два представления:

- **объективно** существующей ПО,
- **субъективных** информационных требований прикладных программистов, выраженных в запросах к базе данных, запланированных в приложениях.

ПО-информация отображает объекты (процессы, явления, предметы) реального мира как составные части ПО (элементы ПО), их существенные свойства, а также взаимосвязи между этими элементами. Эта ПО-информация не связана ни с конкретными приложениями, ни с конкретным способом обработки, а описывает естественные концептуальные связи всех данных в БД.

ПП-информация описывает данные и связи, используемые в приложениях.

В общем случае в методологии построения инфологических моделей ПО-информация должна использоваться для построения первоначальной инфологической модели данных, а ПП-информация – для совершенствования последней с целью повышения эффективности обработки данных.

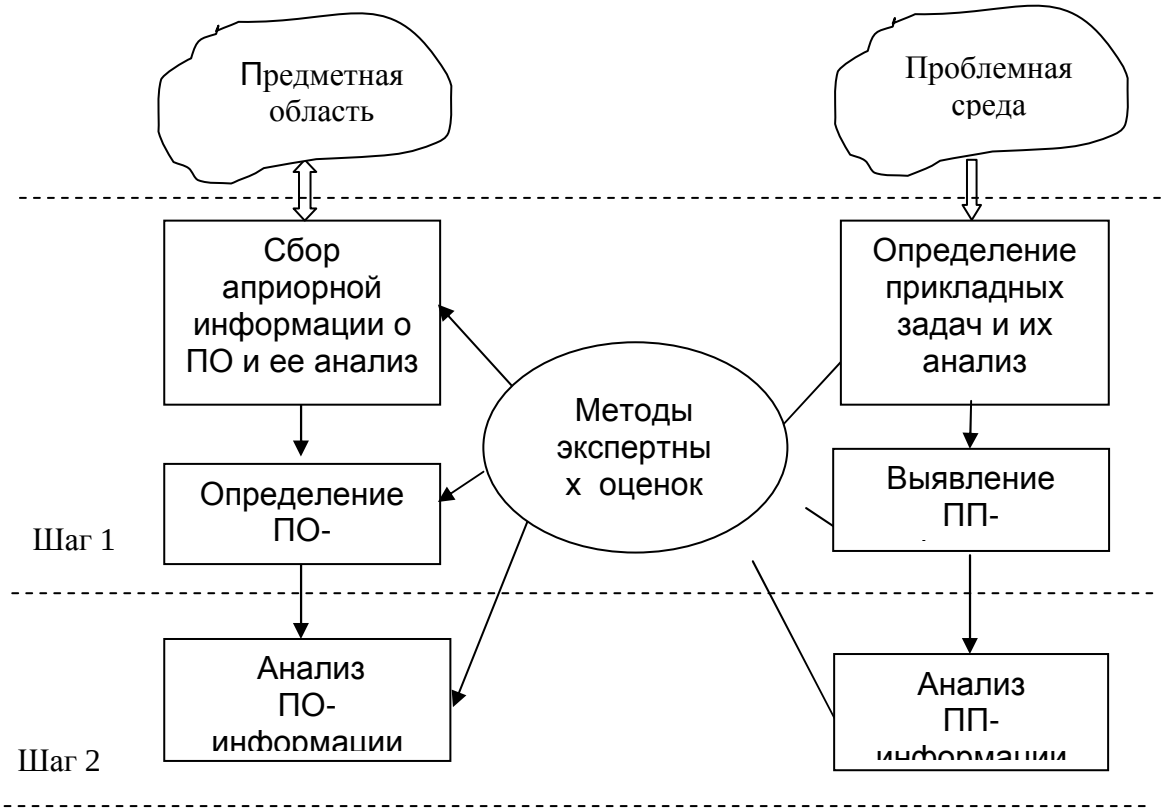


Рис. 9. Общая схема инфологического проектирования

Без включения ПО-информации созданная БД обеспечивает, в лучшем случае, лишь объединение нескольких пользователей, занятых формированием собственных файлов.

Далее в результате **анализа** собранной информации (в виде первоначальной инфологической модели и выявленной ПП-информации) определяются объекты ПО, их свойства и взаимосвязи, которые необходимо смоделировать в базе данных.

Таким образом, в результате сбора и анализа информации о ПО осуществляется:

- идентификация функциональной деятельности предметной области,
- идентификация объектов, которые осуществляют эту функциональную деятельность,
- идентификация всех сущностей и взаимосвязей между ними.
- идентификация характеристик этих сущностей.
- идентификация взаимосвязей между сущностями и их типов.

На основе результатов анализа проводится окончательный **синтез структуры** инфологической модели, то есть формируется **структурная модель** БД.

Очевидно, что на этом шаге проектирования реализуется свойство **эмерджентности** создаваемой модели.

Итак, начальной стадией проектирования системы базы данных является разработка модели предметной области, которая базируется на анализе информационных потребностей будущих пользователей разрабатываемой системы.

Рассмотрим основные подходы к созданию инфологической модели предметной области.

Функциональный подход к проектированию БД

Этот метод является наиболее распространенным. Он реализует принцип «от задач» и применяется в том случае, когда известны функции некоторой группы лиц и/или комплекса задач, для обслуживания информационных потребностей которых создается рассматриваемая БД.

Предметный подход к проектированию БД

Предметный подход к проектированию БД применяется в тех случаях, когда у разработчиков есть четкое представление о самой ПО и о том, какую именно информацию они хотели бы хранить в БД, а структура запросов не определена или определена не полностью. Тогда основное внимание уделяется исследованию ПО и наиболее адекватному ее отображению в БД с учетом самого широкого спектра информационных запросов к ней.

Проектирование с использованием метода «сущность-связь»

Этот метод является комбинацией двух предыдущих и обладает достоинствами обоих. Этап инфологического проектирования начинается с моделирования ПО. Проектировщик разбивает ее на ряд локальных областей, каждая из которых (в идеале) включает в себя информацию, достаточную для обеспечения запросов отдельной группы будущих пользователей или решения отдельной задачи (подзадачи). Каждое локальное представление моделируется отдельно, а затем выполняется их объединение.

Выбор локального представления зависит от масштабов ПО. Обычно ПО разбивается на локальные области таким образом, чтобы каждая из них соответствовала отдельному внешнему приложению.

Заметим, что выбор сущностей (т.е. объектов, о которых в системе будет накапливаться информация) в отдельных случаях может оказаться затруднительным, поскольку порция информации может быть представлена как сущность, атрибут или связь. Например, информацию об ЭКЗАМЕНЕ можно оформить как:

- отдельную сущность,
- как атрибут сущности СЕССИЯ,
- как связь между сущностями СТУДЕНТ, ПРЕПОДАВАТЕЛЬ, ДИСЦИПЛИНА.

Поэтому рекомендуется проработать несколько вариантов, учитывая необходимость совместного использования данных разными пользователями и возможности дальнейшего развития схемы БД.

Для каждой сущности выбираются атрибуты, которые делятся на два типа: **идентифицирующие** и **описательные**. Идентифицирующие атрибуты входят в состав **ключа** (или ключей) и позволяют однозначно распознавать экземпляры сущности. Ключ должен включать в свой состав минимально необходимое для идентификации количество атрибутов. Остальные атрибуты называются описательными и заключают в себе интересующие свойства сущности.

Спецификация атрибута состоит из его названия, указания типа данных и описания ограничений целостности - множества значений, которые может принимать данный атрибут.

Далее осуществляется спецификация связей: выявляются связи между сущностями внутри локального представления. Каждая связь именуется. Кроме спецификации связей типа «сущность-сущность», выполняется спецификация связей типа «сущность-атрибут» и «атрибут-атрибут» для отношений между атрибутами, которые относятся к одной и той же сущности или к одной и той же связи типа «сущность-сущность».

При объединении проектировщик может формировать конструкции, производные по отношению к тем, которые были использованы в локальных представлениях. Целью введения подобных абстракций является:

- объединение в единое целое фрагментарных представлений о различных свойствах одного и того же объекта;
- введение абстрактных понятий, удобных для решения задач системы, установление их связи с более конкретными понятиями, использованными в модели;
- образование классов и подклассов подобных объектов (например, класс «изделие» и подклассы типов изделий, производимых на данном предприятии).

При небольшом количестве локальных областей объединение выполняется за один шаг. В противном случае объединение обычно выполняется в несколько этапов.

При объединении представлений используют 3 основные концепции:

- **идентичность**. Два или более элементов модели идентичны, если они имеют одинаковое семантическое значение.
- **агрегация**. Позволяет рассматривать связь между элементами как новый элемент. Например, связь между сущностями СТУДЕНТ, ДИСЦИПЛИНА, ПРЕПОДАВАТЕЛЬ, ОЦЕНКА может быть представлена агрегированным элементом: сущностью ЭКЗАМЕН с атрибутами ФАМИЛИЯ-СТУДЕНТА, НАЗВАНИЕ-ДИСЦИПЛИНЫ, ФАМИЛИЯ-ПРЕПОДАВАТЕЛЯ, КОД-ОЦЕНКИ.
- **обобщение**. Подчеркивает общую природу объектов, позволяет образовывать многоуровневую иерархию обобщений. Например, в объединяемых представлениях присутствуют следующие сущности:

  ГРУЗОВЫЕ АВТОМОБИЛИ РОССИЙСКОГО ПРОИЭВОДСТВА
  ЛЕГКОВЫЕ АВТОМОБИЛИ РОССИЙСКОГО ПРОИЭВОДСТВА
  КОМПЛЕКТУЮЩИЕ РОССИЙСКОГО ПРОИЭВОДСТВА
  ИМПОРТНЫЕ КОМПЛЕКТУЮЩИЕ

Их можно объединить следующим образом (см. рис. 10). На этапе объединения необходимо выявить и устранить все противоречия. Например, одинаковые названия семантически различных объектов или связей или несогласованные ограничения целостности на одни и те же атрибуты в разных приложениях. Устранение противоречий вызывает необходимость возврата к этапу моделирования локальных представлений с целью внесения в них соответствующих изменений.

По завершении объединения результаты проектирования являют собой концептуальную инфологическую модель предметной области. Модели локальных представлений - это внешние инфологические модели.

Замечание. Большой объем информации, получаемый как непосредственно от экспертов, так и из документальных источников с помощью экспертов, страдает большой долей

субъективности. Это может привести не только к противоречивости хранимых данных в БД, но и, что самое важное, к принятию неадекватных решений. Поэтому необходимо в общем случае с целью повышения объективности этого вида информации использовать **методы экспертных оценок** (простейшим из которых является, например, **анкетирование**).

4.2.4. Общая схема логического (концептуального) проектирования

Напомним, что мы рассматриваем концептуальную схему как результат представления инфологической модели предметной области в терминах инструментария (концептуальной модели данных) конкретной СУБД.

На этапе логического проектирования разрабатывается логическая структура БД, соответствующая логической модели ПО. Решение этой задачи существенно зависит от модели данных, поддерживаемой выбранной СУБД. Результатом выполнения этого этапа являются схемы БД концептуального и внешнего уровней архитектуры, составленные на языках определения данных, поддерживаемых данной СУБД.

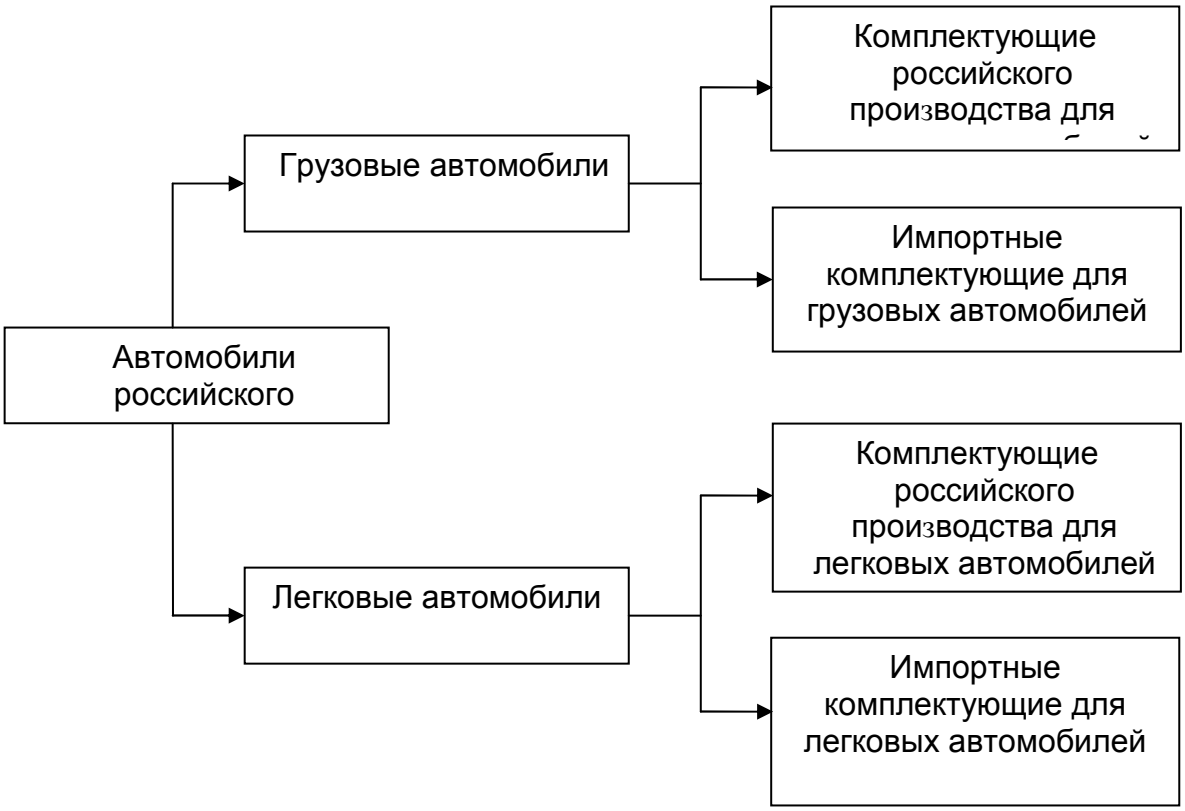


Рис.10. Пример объединения сущностей

Общая структура процесса логического проектирования (включая этап инфологического проектирования) представлена схемой на рис.11.

На основании этой схемы можно определить следующие основные этапы процесса концептуального проектирования.

Этап 1. Обзор предметной области

- Определение целей создания информационной системы.
- Определение специфических требований к БД, вытекающих из этих целей и/или сформулированных персоналом.

Этап 2.

- Определение (идентификация) сущностей (объектов).
- Определение атрибутов сущностей.
- Идентификация ключевых атрибутов сущностей.
- Определение связей между сущностями.
- Формирование инфологической модели.

Формирование и анализ требований (инфологическое проектирование)

Выбор СУБД

Проектирование реализации

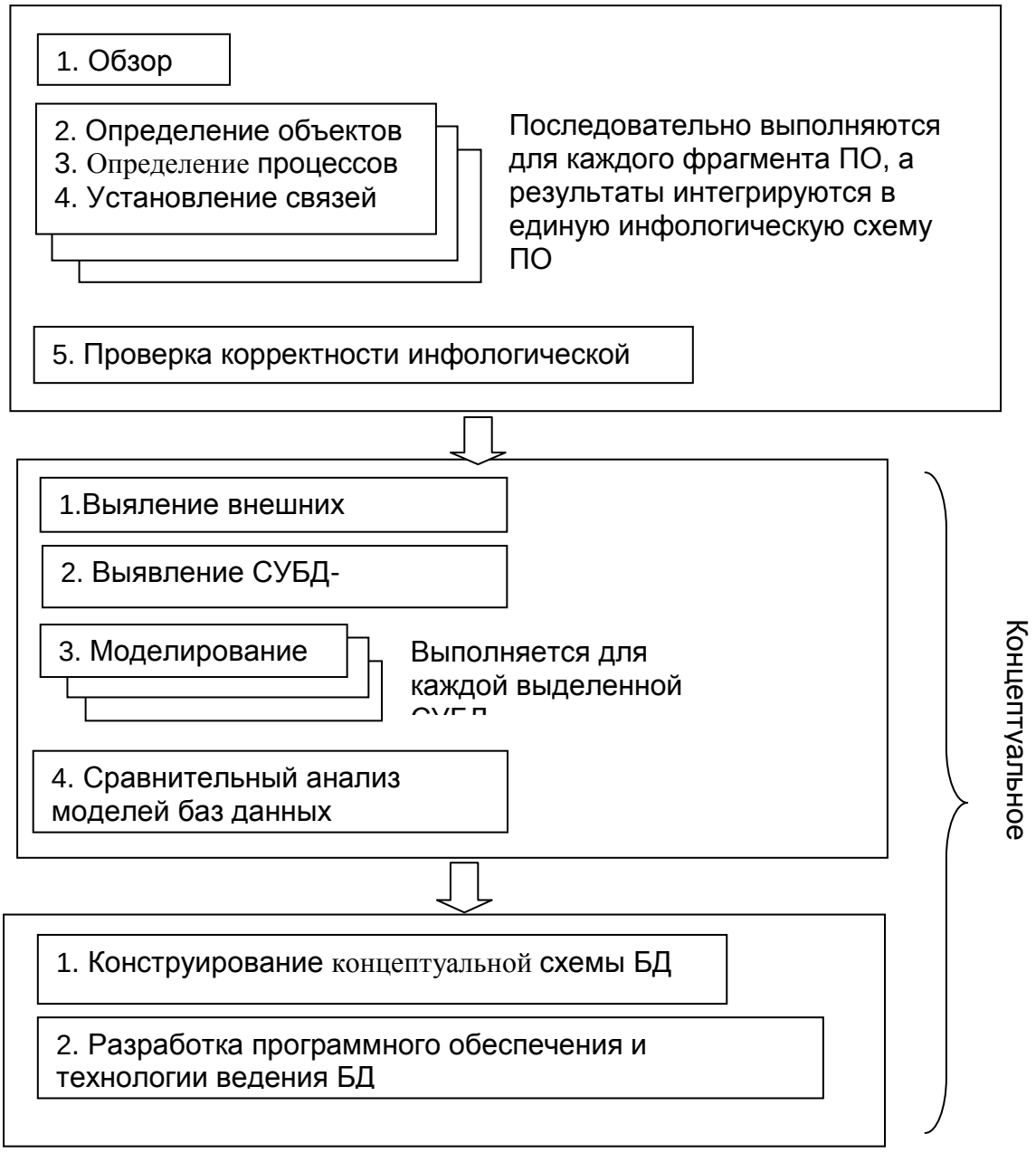


Рис.11. Структура процесса концептуального проектирования

Этап 3. Проектирование реализации

- Преобразование инфологической схемы в концептуальную.
- Установка (формулировка) ограничений целостности.
- Обеспечение средств защиты.
- Разработка (проектирование) приложений.

Несколько комментариев к схеме.

К обследованию ПО:

1. Ясно, что инфологическое описание ориентировано на некоторую **идеальную среду** реализации, то есть отражает, **что** заказчик хочет отразить в информационной системе, но не **как** эти желания будут отражены.

2. Идентификация ключевых атрибутов сущностей – это процесс выделения, так называемых, **ключей** логических записей БД, среди которых различают **первичные** и **вторичные** ключи (неключевые атрибуты).

Первичный ключ – это атрибут (или совокупность атрибутов), значение которого единственным (уникальным) образом идентифицирует экземпляр записи во множестве однотипных записей. Примеры первичных ключей:

- **номер_зачетной_книжки** типа записи **СТУДЕНТ** (пример простого первичного ключа, состоящего из одного атрибута),
- **номер_рейса** + **дата_вылета** типа записи **СВЕДЕНИЯ_О_РЕЙСАХ** (пример составного первичного ключа, образованного несколькими атрибутами); знак "+" здесь означает операцию конкатенации.

Вторичный ключ идентифицирует не уникальный экземпляр записи, а все экземпляры, имеющие определенные свойства. Например, все студенты, закончившие сессию без троек.

К выбору СУБД:

1. Каждая конкретная среда реализации имеет **внешние ограничения**, из которых можно выделить:

1. технические, определяемые конфигурацией вычислительной системы, параметрами функционирования ее компонентов, надежностью их работы и т.д.,

- **программные**, определяемые операционной системой и языками программирования,
- **организационные**, определяемые сроками разработки, имеющимися кадровыми ресурсами, возможностями переподготовки специалистов и т.д.

2. Моделирование БД здесь предполагает преобразование инфологической схемы (модели) в концептуальную схему (модель) БД, поддерживаемую СУБД. Очевидно, что искомое преобразование может быть реализовано средствами нескольких **СУБД-претендентов**, способных функционировать в рамках сформулированных внешних ограничений. Если получено несколько приемлемых моделей БД, они подлежат **сравнительному анализу**. Если приемлемых моделей не оказалось, необходимо либо скорректировать требования к информационной системе, либо начать разработку собственной СУБД.

3. Сравнительный анализ моделей БД производится путем оценки целого ряда факторов, из которых упомянем следующие:

- требуемые объемы основной и дисковой памяти,
- трудоемкость разработки программных средств окружения СУБД,
- трудоемкость реализации приложений,
- затраты на обучение персонала,
- стоимость эксплуатации системы,
- возможность совмещения разработки БД с ранее выполненными программными продуктами, прогнозируемые сроки реализации.

К проектированию реализаций:

1. Конструирование концептуальной схемы БД предполагает окончательное уточнение всех параметров логической организации БД и формирование ее описания на языке описания данных выбранной СУБД, то есть преобразование инфологической модели в концептуальную схему.

2. Уточнение (детализация) концептуальной схемы на основе выявленных дополнительных требований прикладных программ.

3. Установка (формулировка) ограничений целостности. Данный шаг предполагает выработку правил, которые будут устанавливать и поддерживать целостность данных. Будучи определенными, такие правила поддерживаются либо автоматически (в клиент-серверных СУБД) сервером баз данных; либо их поддержку приходится возлагать на пользовательское приложение (в локальных СУБД). Эти правила включают:

- определение типа данных,
- создание полей, опирающихся на домены,
- установка значений по умолчанию,
- определение ограничений целостности,
- определение проверочных условий.

4. Обеспечение защиты предполагает решение вопросов надежности данных и, при необходимости, сохранения секретности информации. Для этого необходимо определить:

- кто будет иметь права (и какие) на использование базы данных,
- кто будет иметь права на модификацию, вставку и удаление данных,
- нужно ли делать различие в правах доступа,
- каким образом обеспечить общий режим защиты информации,
- и т.п.

5. Разработка (проектирование) приложений.

На рис.12. представлена общая схема процесса проектирования, учитывающая этап физического проектирования, организацию информационных потоков и совокупность внешних требований.

Этап физического проектирования заключается в увязке логической структуры БД и физической среды хранения с целью наиболее эффективного размещения данных, т.е. отображении логической структуры БД в структуру хранения. Решается вопрос размещения хранимых данных в пространстве памяти, выбора эффективных методов доступа к различным компонентам «физической» БД. Принятые на этом этапе решения оказывают определяющее влияние на производительность системы. Более подробно вопросы физического проектирования будут рассмотрены в разделе 8.

В заключение отметим, что фактически проектирование БД имеет итерационный характер. В процессе функционирования системы становится возможным изменение ее реальных характеристик, выявление «узких» мест. И если система не отвечает предъявляемым к ней требованиям, то обычно она подвергается реорганизации, т.е. модификации первоначально созданного проекта.

КОНТРОЛЬНЫЕ ВОПРОСЫ ПО ЧЕТВЕРТОМУ РАЗДЕЛУ

1. Перечислите и охарактеризуйте основные этапы жизненного цикла информационной системы.
2. Приведите схему организации централизованного планирования.
3. Перечислите функции администратора базы данных в реализации процессов планирования и проектирования.
4. Приведите общую схему инфологического проектирования. Дайте понятие ПО- и ПП-информации и поясните смысл их использовании для процесса проектирования.
5. Приведите общую схему концептуального проектирования.
6. Опишите этапы концептуального проектирования.

7. Приведите общую схему процесса проектирования.
8. В чем состоит процесс идентификации ключевых атрибутов?
9. Какие виды внешних ограничений вам известны?
10. Свяжите системные свойства модели данных с каждым из этапов проектирования. На каком этапе какие свойства реализуются?

5. МОДЕЛИ ДАННЫХ

Исторически поддерживаемые СУБД модели данных, описанные в литературе, традиционно разбивают на **реляционные** и **навигационные** (**иерархические** и **сетевые**). Следует заметить, что эта классификация весьма условна, поскольку каждая конкретная СУБД поддерживает собственную оригинальную модель данных. Однако, рассмотрение каждой из классических видов моделей данных в «чистом виде» представляется достаточно полезным.

Доказано, что все классические модели данных эквивалентны в том смысле, что любая из трех видов концептуальных моделей может быть преобразована в концептуальную модель любого другого вида, адекватно представляющую предметную область.

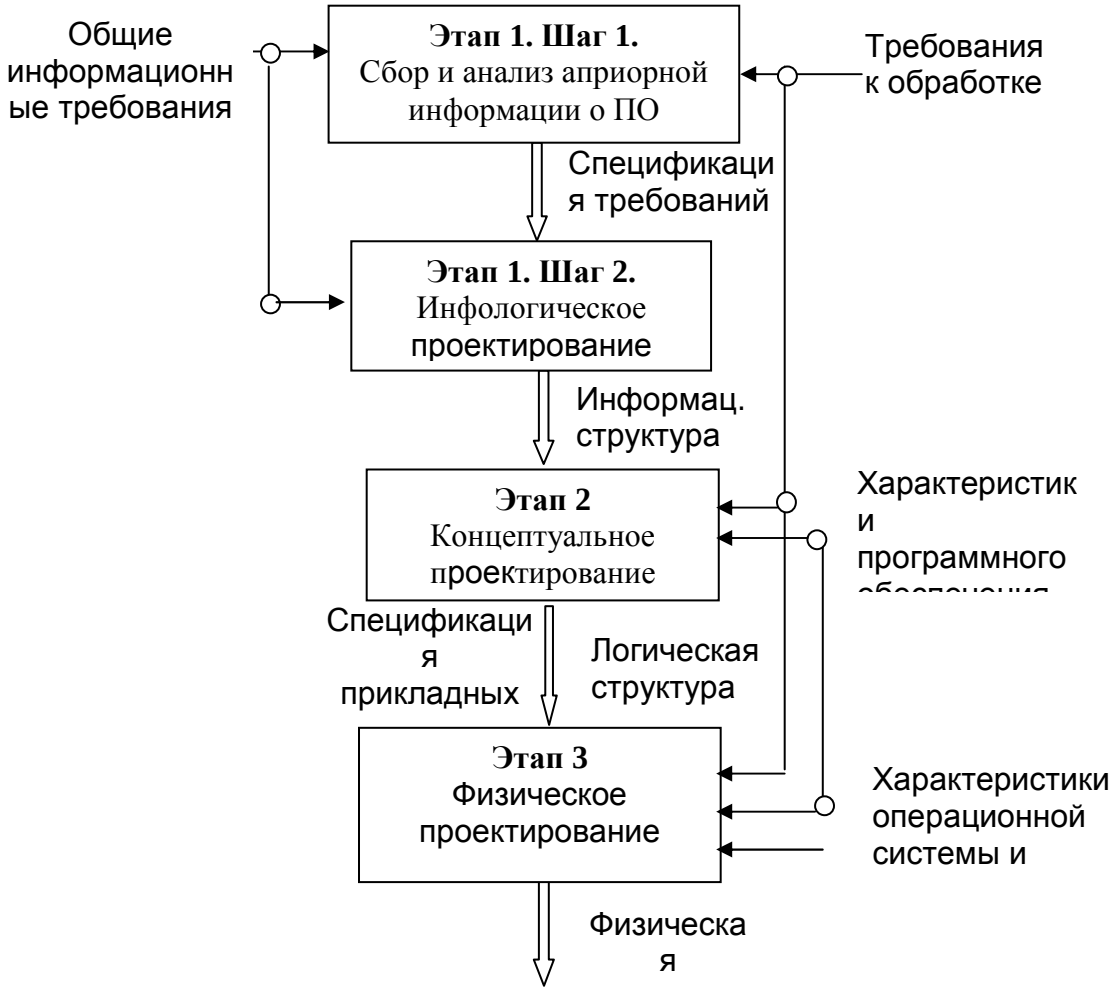


Рис.12. Основные этапы проектирования БД

5.1. Реляционная модель данных

5.1.1. Базовые понятия

Реляционная модель данных была предложена Э.Коддом в 1970 г.; первые промышленные реляционные СУБД начали появляться в конце 1970-х – начале 1980-х гг.

В основе реляционной модели данных лежит математическое понятие **теоретико-множественного отношения**. Тем самым, в основе данной модели – строго обоснованная математическая теория.

Основными понятиями реляционных баз данных являются:

- тип данных,
- домен,
- кортеж,
- отношение,
- атрибут,
- схема отношения,
- первичный ключ.

Тип данных. Понятие тип данных в реляционной модели данных полностью адекватно понятию типа данных в языках программирования. Обычно в современных реляционных БД допускается хранение символьных, числовых данных, битовых строк, а также специальных «темпоральных» данных (дата, время, временной интервал).

Домен. В самом общем виде домен определяется заданием некоторого базового типа данных, к которому относятся элементы домена, и способа определения принадлежности произвольного элемента типа данных домену. Например, домен **Имена** очевидным образом определяется на базовом типе строк символов, но в число его значений могут входить только те строки, которые могут изображать имя (в частности, такие строки не могут начинаться с мягкого знака). Обычно такой способ определения принадлежности элемента типа данных домену сводится к заданию некоторого логического выражения, применяемого к элементам данных. Если вычисление этого логического выражения дает результат **True**, то элемент данных является элементом домена. Таким образом, правильной интуитивной трактовкой понятия домена является понимание домена как допустимого потенциального множества значений элементов данных определенного типа.

Следует отметить также семантическую нагрузку понятия домена: данные считаются сравнимыми только в том случае, когда они относятся к одному домену.

Декартовым произведением k доменов (D_1, D_2, … , D_k), которое обозначается

$$(D_1 \times D_2 \times \dots \times D_k),$$

называется множество всех **кортежей** вида (v_1, v_2, … , v_k) длины k таких, что

$$v_1 \in D_1, v_2 \in D_2, \dots , v_k \in D_k .$$

Пример. Пусть $D_1 = \{1, 2, 3\}$, $D_2 = \{a, b, c, d\}$.

Тогда

$$D_1 \times D_2 = \{ (1,a), (1,b), (1,c) (1,d), (2,a), (2,b), (2,c), (2,d), (3,a), (3,b), (3,c), (3,d) \}.$$

Или в матричном виде

$$D_1 \times D_2 = \begin{Vmatrix} 1,a & 1,b & 1,c & 1,d \\ 2,a & 2,b & 2,c & 2,d \\ 3,a & 3,b & 3,c & 3,d \end{Vmatrix}$$

Отношением называется произвольное конечное подмножество декартова произведения одного или нескольких доменов.

Примеры математических отношений:

1). { (1,a), (2,b), (3,c), (1,d) }

2). {0} – пустое подмножество

Кортежи (1,a), (2,b), (3,c), (1,d) являются элементами отношения (1).

В практике реляционных баз данных отношение представляется в виде двумерной таблицы, в которой **строки** соответствуют **кортежам**, а **столбцы** – **доменам**. Для примера 1) реляционный эквивалент математического отношения выглядит следующим образом:

1	a
2	b
3	c
1	d

Учитывая, что реляционное отношение – это элемент модели данных, определим смысл для строк и столбцов двумерной таблицы:

a) строка – это совокупность значений элементов данных, характеризующих конкретный объект (набор требуемых сведений об объекте), то есть строка представляет **кортеж отношения** (более точное определение понятия кортежа дано ниже) ;

b) столбец – значения, принимающие соответствующей характеристикой (свойством) объектов, сведения о которых включены в таблицу; заметим, что тип и диапазон этих значений определяется соответствующим доменом.

Естественно рассматривать реляционное отношение как модель совокупности **однотипных** объектов, то есть объектов предметной области, у которых на этапе проектирования выделено для изучения множество одних и тех же свойств. Учитывая, что для выполнения операций над объектом он и все его свойства должны быть поименованы, снабдим именами таблицу (реляционное отношение) в целом и все ее столбцы в отдельности. Тогда имя отношения выступает в роли **имени типа** объекта, а имена столбцов – **имен** его **свойств** (**атрибутов**). Таким образом, **атрибут** представляет некоторое **свойство** объекта.

Схема отношения представляется:

1. перечнем имен отношения и атрибутов в виде $R(A_1, A_2, \dots, A_k)$, где R – имя отношения; $A_1, A_2, \dots, A_k$ – имена атрибутов;

2. описанием доменов для каждого атрибута; заметим, что различные атрибуты могут быть определены на одном домене, например, дата рождения и дата поступления на работу;

3. условиями ограничения целостности, задаваемыми на доменах; например, дата рождения не должна определять возраст более 200 лет, а дата поступления на работу – меньше минимального, установленного трудовым законодательством.

В общем случае схема отношения - это именованное множество пар (**имя атрибута, имя домена). Степень**, или «**арность**» **схемы отношения**, - мощность этого множества. Если все атрибуты одного отношения определены на разных доменах, резонно использовать для именования атрибутов имена соответствующих доменов (не забывая, конечно, о том, что это является всего лишь удобным способом именования и не устраняет различия между понятиями домена и атрибута).

Кортеж, соответствующий данной схеме отношения, - это множество пар (**имя атрибута**, **значение)**, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения. **Значение** является допустимым значением домена данного атрибута (или типа данных, если понятие домена не поддерживается). Тем самым, **степень**, или «**арность**» **кортежа**, то есть число элементов в нем, совпадает с «арностью» соответствующей схемы отношения.

Таким образом, можно сказать, что отношение - это множество кортежей, соответствующих **экземпляру схемы отношения** (его **телу**).

Схема БД (в структурном смысле) - это набор именованных схем отношений. В целом **реляционная база данных** представляется набором взаимосвязанных отношений различного смыслового наполнения, а ее **схема** – списком имен отношений, ее образующих, и совокупностью схем каждого из этих отношений.

Легко заметить, что отношение является отражением некоторой сущности реального мира и с точки зрения обработки данных представляет собой таблицу. В локальных базах данных

каждая таблица размещается в отдельном файле, то с точки зрения размещения данных для локальных баз данных отношение можно отождествлять с файлом.

Не учитывая пункты 2. и 3., схему отношения можно ассоциировать с пустой таблицей (то есть с ее заголовком), а экземпляр схемы – с заполненной таблицей, отображающей состояние отношения в текущий момент времени, причем строками заполненной таблицы являются кортежи отношения-экземпляра, а имена атрибутов именуют столбцы этой таблицы. Поэтому иногда говорят «столбец таблицы», имея в виду «атрибут отношения». Этой терминологии придерживаются в большинстве коммерческих реляционных СУБД.

Домен же представляется неким обобщенным типом, который может быть источником для типов полей в записи. Таким образом, следующие тройки терминов являются эквивалентными:

- отношение, таблица, файл (для локальных баз данных)
- кортеж, строка, запись
- атрибут, столбец, поле.

Соотношения между базовыми понятиями реляционной представлены на рис.13.

Атрибут (или набор атрибутов), который может быть использован для однозначной идентификации конкретного кортежа (строки, записи), является **первичным ключом** (см..п.4.2.4.) Для каждого отношения, по крайней мере, полный набор его атрибутов обладает этим свойством. Однако при формальном определении первичного ключа требуется обеспечение его «минимальности», то есть, в набор атрибутов первичного ключа не должны входить такие атрибуты, которые можно отбросить без ущерба для основного свойства - однозначного определения кортежей. Это значит, что если из первичного ключа исключить произвольный атрибут, оставшихся атрибутов будет недостаточно для однозначной идентификации отдельных кортежей. Понятие первичного ключа является исключительно важным в связи с понятием **целостности базы данных** (см. п. 5.1.3.).

Замечание. Во многих СУБД имеется возможность помимо первичного определять еще ряд **уникальных** ключей. Отличие уникального ключа от первичного состоит в том, что уникальный ключ не является главным идентификатором записи и на него не может ссылаться внешний ключ другой таблицы. Его главная задача - гарантировать уникальность значения поля.

Фундаментальные свойства отношений (таблиц). Определенные таким образом таблицы обладают следующими свойствами:

1. **Однородность столбцов.** Для всех столбцов таблицы выполняется следующее требование: элементы столбца принимают значения на одном домене.

2. **Отсутствие кортежей-дубликатов.** В таблице нет двух одинаковых строк. Это свойство следует из определения отношения как множества кортежей. В классической теории множеств, по определению, каждое множество состоит из различных элементов. Именно из этого свойства вытекает необходимость наличия у каждого отношения первичного ключа.

3. **Отсутствие упорядоченности кортежей.** В операциях с такой таблицей ее строки и столбцы могут просматриваться в любом порядке и в любой последовательности безотносительно к их информационному содержанию и смыслу. Свойство отсутствия упорядоченности кортежей отношения также является следствием определения отношения-экземпляра как множества кортежей. Отсутствие требования к поддержанию порядка на множестве кортежей отношения дает дополнительную гибкость СУБД при хранении баз данных во внешней памяти и при выполнении запросов к базе данных. Это не противоречит тому, что при формулировании запроса к БД можно потребовать сортировки результирующей таблицы в соответствии со значениями некоторых столбцов. Такой результат - это вообще говоря, не отношение, а некоторый упорядоченный список кортежей.

4. **Отсутствие упорядоченности атрибутов**. Атрибуты отношений не упорядочены, поскольку по определению схема отношения есть множество пар (имя атрибута, имя домена). Для ссылки на значение атрибута в кортеже отношения всегда используется имя атрибута. Это свойство теоретически позволяет, например, модифицировать схемы существующих

отношений не только путем добавления новых атрибутов, но и путем удаления существующих атрибутов. Однако в большинстве существующих систем такая возможность не допускается, и хотя упорядоченность набора атрибутов отношения явно не требуется, часто в качестве неявного порядка атрибутов используется их порядок в определении схемы отношения.

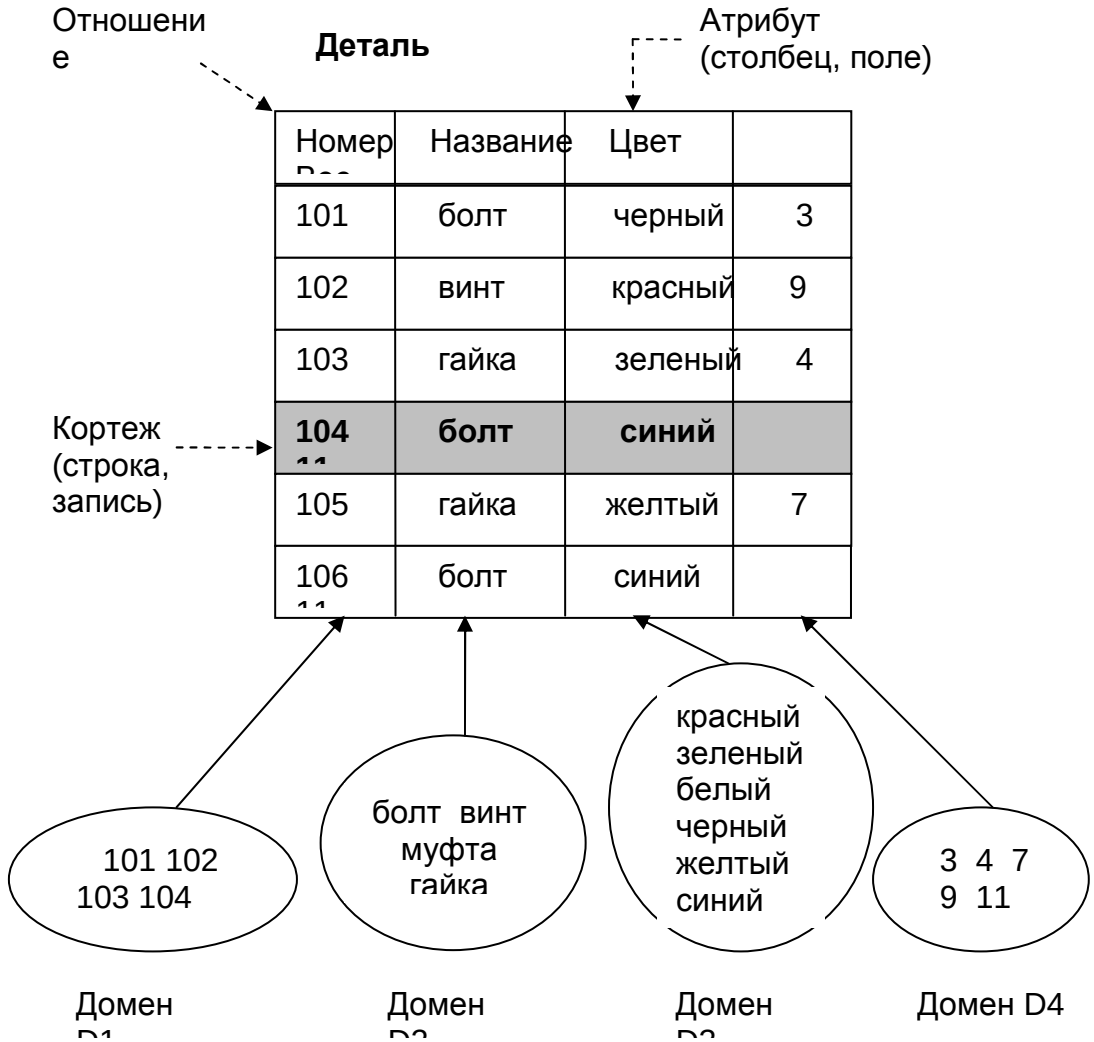


Рис.13. Базовые понятия реляционной модели и соотношения между ними

5. **Атомарность значений атрибутов**. Значения всех атрибутов являются атомарными. Это следует из определения домена как потенциального множества значений простого типа данных, т.е. среди значений домена не могут содержаться множества значений (в свою очередь являющиеся отношениями). Принято говорить, что в реляционных базах данных допускаются только **нормализованные** отношения (см. следующий пункт).

5.1.2. Принципы нормализации

Для простоты будем считать, что проблема проектирования реляционной базы данных состоит в обоснованном принятии решений о том, из каких отношений должна состоять БД и какие атрибуты должны быть у этих отношений, опустив обсуждение вопросов, связанных с условиями ограничения целостности и защиты, то есть проектирование реляционных БД включает следующие основные шаги:

1) определение информационных потребностей базы данных,

2) анализ объектов реального мира, которые необходимо смоделировать в базе данных; формирование из этих объектов сущностей и списков характеристик этих сущностей,

3) установка в соответствие сущностям и характеристикам отношений (таблиц) и атрибутов (столбцов),

4) определение первичных ключей - атрибутов, которые уникальным образом идентифицируют каждый объект,

5) установка связей между объектами (таблицами и столбцами).

После определения отношений, атрибутов и связей между отношениями следует посмотреть на проектируемую базу данных в целом и проанализировать ее, используя **правила нормализации**, с целью устранения логических ошибок. Важность нормализации состоит в том, что она позволяет разбить большие отношения, как правило, содержащие большую избыточность информации, на более мелкие логические единицы, группирующие только данные, объединенные «по природе». После применения правил нормализации логически единые группы данных располагаются не более чем в одной таблице. Это дает следующие преимущества:

- данные легко обновлять или удалять,
- исключается возможность рассогласования копий данных,
- уменьшается возможность введения некорректных данных.

Рассмотрим классический подход, при котором весь процесс проектирования производится в терминах реляционной модели данных методом последовательных приближений к удовлетворительному набору схем отношений. Исходной точкой является представление предметной области в виде одного или нескольких отношений, а на каждом шаге проектирования строится некоторый набор схем отношений, обладающих лучшими свойствами. Таким образом, процесс проектирования включает **процесс нормализации схем отношений**.

В теории реляционных баз данных обычно выделяется следующая последовательность **нормальных форм**:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF) (правильнее было бы считать эту нормальную форму третьей, однако по историческим причинам третья ступень оказалась занятой к моменту изобретения BCNF, из-за чего она и получила нестандартное название);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

Важно отметить, что нормальные формы обладают следующими важными свойствами:

- при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются,
- каждая следующая нормальная форма в некотором смысле улучшает свойства предыдущей.

В основе классического процесса проектирования лежит метод нормализации, который опирается на декомпозицию (разложение) отношения, находящегося в предыдущей нормальной форме, в два или более отношения, удовлетворяющих требованиям следующей нормальной формы.

Замечание. Нормализованные отношения обладают некоторыми ограничениями (не любую информацию удобно представлять в виде плоских таблиц), но существенно упрощают манипулирование данными.

Наиболее важные на практике нормальные формы отношений основываются на фундаментальном в теории реляционных баз данных понятии **функциональной зависимости**.

Пусть $R(A_1, A_2, \ldots, A_n)$ схема отношения R, а X и Y – подмножества множества $\{A_1, A_2, \ldots, A_n\}$. Говорят, что “X функционально определяет Y” или “Y функционально зависит от X” и обозначают это как X → Y, если в любом отношении R’, являющимся текущим значением R, не могут содержаться два кортежа, компоненты которых совпадают по всем атрибутам, принадлежащим множеству X, и не совпадают по одному и более атрибутам, принадлежащим множеству Y.

Таким образом, термин **функциональная зависимость** означает, что атрибут A_j отношения R функционально зависит от атрибута A_i того же отношения, если в каждый момент времени каждому значению атрибута A_i соответствует не более чем одно значение атрибута A_j, связанного с A_i в отношении R. Или еще проще: функциональная зависимость A_j от A_i означает, что если в любой момент времени известно значение A_i, то можно однозначно получить и значение A_j.

Очевидно, что атрибут может функционально зависеть не от какого-либо одного атрибута, а от целой группы атрибутов.

Два или более атрибута называются взаимно независимыми, если ни один из этих атрибутов не является функционально зависимым от других атрибутов.

Единственный способ определения существования функциональных зависимостей для схемы отношения R заключается в тщательном анализе семантики атрибутов этого отношения. В этом смысле зависимости являются фактически отображением связей, существующих в реальном мире.

Наличие тех или иных зависимостей между атрибутами в схеме определяет **степень** ее **нормализации**.

С практической точки зрения, достаточно трех первых форм - следует учитывать время, необходимое системе для «соединения» таблиц при отображении их на экране. Поэтому мы ограничимся изучением процесса приведения отношений к первым трем формам.

Этот процесс включает:

- устранение повторяющихся групп (приведение к 1НФ)
- удаление частично зависимых атрибутов (приведение к 2НФ)
- удаление транзитивно зависимых атрибутов (приведение к 3НФ).

Первая нормальная форма

Назовем **простым** атрибут, значения которого неделимы.

Назовем **сложным** атрибут, значения которого можно представить конкатенацией нескольких значений других атрибутов.

Пример. Пусть задано следующее отношение R1:

R1

Таб_номер	Фио	Оклад	Комната	Телефон	Дети
				Имя	
211 10	Иванов Л.А.	1500	12	616	Саша Женя 7

Замечание. Здесь и далее жирным шрифтом в примерах схем отношений выделяются первичные ключи.

Отношение называется **нормализованным** или приведенным к **первой нормальной форме** (**1НФ**), если все его атрибуты являются атомарными (простыми); другими словами, если на пересечении строки и столбца находится значение только одного элемента данных. Преобразование ненормализованной формы в 1НФ осуществляется, как правило, увеличением размера отношения и изменению его первичного ключа. После приведения к 1НФ отношение R1 принимает следующий вид:

R2

Таб_номер	**Имя_ребенка**	Возраст_ребенка	Фио	Оклад	Комната	Телефон
211	Саша	10	Иванов Л.А	1500	12	616
211	Женя	7	Иванов Л.А	1500	12	616
211	Лена	3	Иванов Л.А	1500	12	616

Очевидно, что в нормализованном отношении R2 все неключевые атрибуты функционально зависят от ключа.

Замечание. При определении первичного ключа в отношении R2 предполагается, что в семье нет детей с одинаковым именем, в противном случае в состав ключа необходимо было бы включить атрибут **Возраст_ребенка (**вероятность одинакового имени у близнецов при нормальных родителях равна нулю).

Вторая нормальная форма

Говорят, что неключевой атрибут **функционально полно** зависит от **составного ключа,** если он функционально зависит от ключа, но не находится в функциональной зависимости ни от какой части составного ключа. Очевидно, что в отношении R2 атрибуты ФИО, Оклад, Комната, Телефон не находятся в полной функциональной зависимости от ключа отношения, поскольку они функционально зависят от части ключа **Таб_номер**. Наличие неполной функциональной зависимости приводит к высокой степени дублирования информации в отношении, а следовательно, к необходимости при изменении значения атрибута, (например, Оклад) вносить изменения в большое количество записей. Более того, данные о сотрудниках, не имеющих детей, естественным образом в отношение включены быть не могут. Таким образом, для отношения, находящегося в 1НФ, могут потребоваться дальнейшие преобразования.

Говорят, что отношение находится во **второй нормальной форме (2НФ),** если оно находится в 1НФ и все неключевые атрибуты функционально полно зависят от составного ключа.

Замечание. Проблема функциональной полноты рассматривается только в контексте составного ключа. Если первичный ключ включает только один атрибут, очевидно, эта проблема снимается.

Для искомого приведения 1НФ во 2НФ необходимо:

1) построить проекцию 1НФ, исключив атрибуты, которые не находятся в полной функциональной зависимости от составного ключа;

2) построить дополнительно одну или несколько проекций на часть составного ключа и атрибуты, функционально зависящие от этой части ключа.

Для нашего примера отношение R2 должно быть преобразовано в два отношения R3 и R4, оба находящиеся во 2НФ:

R3

Таб_номер	**Имя_ребенка**	Возраст_ребенка
211	Саша	10
211	Женя	7
211	Лена	3

R4

Таб_номер	Фио	Оклад	Комната	Телефон
211	Иванов Л.А.	1500	12	616

Третья нормальная форма

Пусть X, Y, и Z – три атрибута некоторого отношения. При этом

$$X \to Y \quad \text{и} \quad Y \to Z,$$

$$Z \not\to Y \qquad Y \not\to X.$$

но обратное соответствие отсутствует, то есть или Тогда говорят, что Z **транзитивно зависит** от X. Например, в отношении R4 транзитивной является зависимость

Таб_номер → Комната → Телефон

Хранение в одном отношении атрибутов, находящихся в транзитивной зависимости от ключа, порождает ряд неудобств. Действительно, номер телефона есть характеристика комнаты, поэтому сведения о телефоне будут многократно дублироваться в записях для всех сотрудников, располагающихся в оной комнате. Более того, изменение телефона в одной комнате приводит к корректировке многих записей БД. Другая проблема, связанная с возможным нарушением целостности БД, возникает в случае, когда комнату покидает единственный находящийся в ней сотрудник; очевидно, при этом, что всякая связь с этой комнатой теряется.

Говорят, что отношение находится в **третьей нормальной форме (3НФ)**, если оно находится во 2НФ и каждый неключевой атрибут нетранзитивно зависит от ключевого атрибута.

Для преобразования 2НФ в 3НФ необходимо построить несколько проекций. Для нашего примера отношение R4 должно быть приведено к двум отношениям R5 и R6:

R5

Таб_номер	Фио	Оклад	Комната
211	Иванов Л.А.	1500	12

R6

Комната	Телефон
12	616

Очевидно, что в процессе приведения отношений в 2НФ и 3НФ число отношений в схеме БД увеличивается. Однако, всегда сохраняется возможность получить исходные отношения, используя операции соединения. С другой стороны, появление новых отношений порождает проблемы поддержания семантической целостности.

Выше было отмечено, что на практике третья нормальная форма схем отношений является достаточной в большинстве случаев и приведением к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается. Однако иногда полезно продолжить процесс нормализации. Нормальные формы более высоких порядков подробно рассмотрены в [3].

Замечание. Необходимо четко понимать, что разбиение информации на более мелкие единицы с одной стороны, способствует повышению надежности и непротиворечивости базы данных, а с другой стороны, снижает ее производительность, так как требуются дополнительные затраты процессорного времени (серверного или машины пользователя) на обратное «соединение» таблиц, например, при представлении информации на экране. Иногда для достижения требуемой производительности нужно сделать отход от канонической нормализации, ясно осознавая, что необходимо обеспечить меры по предотвращению противоречивости данных. Поэтому всякое решение о необходимости того или иного действия по нормализации можно принимать, только тщательно проанализировав предметную область и класс поставленной задачи. Может потребоваться несколько итераций для достижения состояния, которое будет желаемым компромиссом.

Таким образом, в целом можно отметить, что хорошо спроектированная БД:

- ***удовлетворяет всем требованиям пользователей к содержимому базы данных***. Перед проектированием базы необходимо провести обширные исследования требований пользователей к функционированию базы данных,

- ***гарантирует непротиворечивость и целостность данных***. При проектировании таблиц нужно определить их атрибуты и некоторые правила, ограничивающие возможность ввода пользователем неверных значений. Для контроля данных перед непосредственной записью их в таблицу СУБД должна осуществлять вызов правил модели данных и тем самым гарантировать сохранение целостности информации,
- ***обеспечивает естественное, легкое для восприятия структурирование информации.*** Качественное построение базы позволяет делать запросы к базе более «прозрачными» и легкими для понимания; следовательно, снижается вероятность внесения некорректных данных и улучшается качество сопровождения базы,
- ***удовлетворяет требованиям пользователей к производительности базы данных***. При больших объемах информации вопросы сохранения производительности начинают играть главную роль, сразу «высвечивая» все недочеты этапа проектирования.

5.1.3. Целостность сущности и ссылок

В реляционной модели данных фиксируются два базовых требования целостности, которые должны поддерживаться в любой реляционной СУБД.

1. Требование **целостности сущностей**. Объекту или сущности реального мира в реляционных БД соответствуют кортежи отношений. Требование состоит в том, что любой кортеж любого отношения должен быть отличим от любого другого кортежа этого отношения, т.е. другими словами, любое отношение должно обладать первичным ключом. Это требование автоматически удовлетворяется, если в системе не нарушаются базовые свойства отношений.

2. Требованием **целостности по ссылкам**. Мы видели, что результатом процесса нормализации отношений является представление сложных сущностей реального мира в реляционной БД в виде нескольких кортежей нескольких отношений.

Пример. В реляционной базе требуется представить сущности **ОТДЕЛ** с атрибутами **Отд_номер** (номер отдела), **Отд_кол** (количество сотрудников) и **СОТРУДНИКИ** (набор сотрудников). Для каждого сотрудника нужно хранить **Сотр_номер** (номер сотрудника), Сотр_имя (имя сотрудника), **Сотр_зарп** (заработная плата сотрудника) и **Сотр_отд_номер** (номер отдела, где работает сотрудник). При правильном проектировании (в соответствии с принципами нормализации) соответствующей БД в ней появятся два отношения:

ОТДЕЛ (Отд_номер, Отд_Кол**)** (первичный ключ - **Отд_номер**)

СОТРУДНИКИ (Сотр_номер, Сотр_имя**,** Сотр_зарп**,** Сотр_отд_номер**)** (первичный ключ **Сотр_номер**)

Заметим, что атрибут Сотр_отд_номер появляется в отношении **СОТРУДНИКИ** не потому, что номер отдела является собственным свойством сотрудника, а лишь для того, чтобы иметь возможность восстановить при необходимости полную сущность **ОТДЕЛ**. Значение атрибута Сотр_отд_номер в любом кортеже отношения **СОТРУДНИКИ** должно соответствовать значению атрибута Отд_номер в некотором кортеже отношения **ОТДЕЛ**. Атрибут такого рода называется **внешним ключом**, поскольку его значения однозначно характеризуют сущности, представленные кортежами некоторого другого отношения (то есть задают значения их первичного ключа). Другими словами, для поддержания ссылочной целостности данных многие СУБД поддерживают механизм внешних ключей.

Говорят, что отношение, в котором определен внешний ключ, **ссылается** на соответствующее отношение, в котором такой же атрибут является первичным ключом; при этом отношение, на первичный ключ которого ссылается внешний ключ другого отношения,

называется *master-отношением*, или **главным** отношением; а отношение, от которого исходит ссылка, называется *detail-отношением*, или **подчиненным** отношением.

Требование **целостности по ссылкам**, или **требование внешнего ключа**, состоит в том, что для каждого значения внешнего ключа, появляющегося в подчиненном отношении, в главном отношении должен найтись кортеж с таким же значением первичного ключа, либо значение внешнего ключа должно быть полностью неопределенным (иметь значение NULL. т.е. ни на что не указывать). Для нашего примера это означает, что если для сотрудника указан номер отдела, то этот отдел должен существовать.

Ясно, что для соблюдения целостности сущности достаточно гарантировать отсутствие в любом отношении кортежей с одним и тем же значением первичного ключа. С целостностью по ссылкам дела обстоят сложнее. Понятно, что при обновлении подчиненного отношения (вставке новых кортежей или модификации значения внешнего ключа в существующих кортежах) достаточно следить за тем, чтобы не появлялись некорректные значения внешнего ключа. Проблемы возникают при удалении или изменении кортежа из главного отношения.

СУБД имеет возможность автоматически генерировать ошибку в случаях, когда делается попытка:

- вставки в подчиненное отношение записи, для внешнего ключа которой не существует соответствия в главном отношении (например, там нет еще записи с таким первичным ключом);
- удаления из главного отношения записи, на первичный ключ которой имеется хотя бы одна ссылка из подчиненных отношений;
- изменения первичного ключа записи главного отношения, на которую имеется хотя бы одна ссылка из подчиненных таблиц.

Можно выделить следующие подходы, поддерживающие целостность по ссылкам.

1. Запрещается производить удаление и изменение кортежей, на которые существуют ссылки (то есть, сначала нужно либо удалить ссылающиеся кортежи из подчиненных таблиц, либо соответствующим образом изменить значения их внешнего ключа).
2. При удалении кортежа из главного отношения во всех ссылающихся на него кортежах всех подчиненных отношений значение внешнего ключа автоматически становится неопределенным.
3. При удалении кортежа из главного отношения из подчиненных отношений автоматически удаляются все ссылающиеся кортежи.
4. Распространить всякие изменения в первичном ключе главного отношения на подчиненные отношения, а именно, при изменении в главном отношении первичного ключа записи в подчиненных отношениях должны быть изменены все внешние ключи записей, ссылающихся на изменяемую.

Для выбора подхода необходимо анализировать требования конкретной прикладной области.

5.1.4. Манипулирование данными в реляционных моделях

Согласно [2], реляционная модель состоит из трех частей, описывающих разные аспекты реляционного подхода: **структурной** части, **манипуляционной** части и **целостной** части.

В **структурной части** модели фиксируется, что единственной структурой данных, используемой в реляционных БД, является нормализованное **n-арное** отношение.

Целостная компонента была рассмотрена в предыдущем пункте.

В **манипуляционной** составляющей реляционной модели определяются два базовых механизма манипулирования реляционными данными:

1. Основанная на теории множеств **реляционная алгебра**.

2. Базирующееся на математической логике (точнее, на исчислении предикатов первого порядка) **реляционное исчисление**. В свою очередь, обычно рассматриваются два вида реляционного исчисления - исчисление доменов и исчисление предикатов.

Все эти механизмы обладают одним важным свойством: они замкнуты относительно понятия отношения. Это означает, что выражения реляционной алгебры и формулы реляционного исчисления определяются над отношениями реляционных БД и результатами вычислений также являются отношения. Как следствие, любое выражение или формула могут интерпретироваться как отношение, что позволяет использовать их в других выражениях или формулах. Как мы увидим, алгебра и исчисление обладают большой выразительной мощностью: очень сложные запросы к базе данных могут быть выражены с помощью одного выражения реляционной алгебры или одной формулы реляционного исчисления. По этой причине именно эти механизмы включены в реляционную модель данных.

И еще одно важное замечание. Основные достоинства реляционной модели заключаются не только в простоте ее представления, но и в возможности «разрезать» и «склеивать» отношения, то есть в возможности манипулировать частями отношений, формируя новые отношения. Заметим, что схема отношения может рассматриваться как описание типа переменной с именем, совпадающим с именем отношения, а экземпляры отношения – как значения этой переменной. Другими словами, операндами операций манипулирования отношениями могут выступать переменные.

5.1.4.1. Операции реляционной алгебры

Алгеброй отношений называют систему операций манипулирования отношениями, каждый оператор которой в качестве операндов имеет одно или более отношений и образует новое отношение по заранее обусловленному правилу.

Для определения языка реляционной алгебры используют следующие основные операции, которые можно объединить в две группы:

I. Традиционные операции над множествами: **объединение**, **пересечение**, **вычитание**, **прямое (декартово) произведение**.

II. Специальные реляционные операции: **проекция, выборка**, **соединение**, **деление**.

III. Кроме того, в состав алгебры включается операция **присваивания** и операция **переименования атрибутов**, дающая возможность корректно сформировать заголовок (схему) результирующего отношения.

- Операция **присваивания** позволяет сохранить результат вычисления реляционного выражения в существующем отношении БД. Поскольку результатом любой реляционной операции (кроме операции присваивания) является некоторое отношение, можно образовывать реляционные выражения, в которых вместо отношения-операнда некоторой реляционной операции находится вложенное реляционное выражение.
- Операция **переименования** производит отношение, тело которого совпадает с телом операнда, но имена атрибутов изменены. Такая операция дает возможность корректно сформировать заголовок (схему) результирующего отношения.

Традиционные операции над множествами

Особенности теоретико-множественных операций реляционной алгебры

Несмотря на то, что в основе теоретико-множественной части реляционной алгебры лежит классическая теория множеств, соответствующие операции реляционной алгебры обладают некоторыми особенностями. Начнем с операции объединения (все, что будет говориться по поводу объединения, переносится на операции пересечения и взятия разности). Смысл операции объединения в реляционной алгебре в целом остается теоретико-множественным. Но если в теории множеств операция объединения осмысленна для любых двух множеств-

операндов, то в случае реляционной алгебры результатом операции объединения должно являться отношение. Если допустить в реляционной алгебре возможность теоретико-множественного объединения произвольных двух отношений (с разными схемами), то, конечно, результатом операции будет множество, но это множество, состоящее из разнотипных кортежей, т.е. оно не является отношением. Если исходить из требования замкнутости реляционной алгебры относительно понятия отношения, то такая операция объединения является бессмысленной. Все эти соображения приводят к появлению понятия **совместимости отношений по типу:** два отношения совместимы по типу в том и только в том случае, если они обладают одинаковыми заголовками. объединению: два отношения совместимы по объединению в том и только в том случае, когда обладают одинаковыми заголовками. Точнее, будем говорить, что два отношения совместимы по типу, если:

1. Каждое из них имеет одно и то же множество имен атрибутов.
2. Атрибуты с одинаковыми именами в обоих отношениях определены на одном и том же домене.

Если два отношения совместимы по объединению, то при обычном выполнении над ними операций объединения, пересечения и взятия разности результатом операции является отношение с корректно определенным заголовком, совпадающим с заголовком каждого из отношений-операндов. Напомним, что если два отношения «почти» совместимы по объединению, т.е. совместимы во всем, кроме имен атрибутов, то до выполнения операции типа соединения эти отношения можно сделать полностью совместимыми по объединению путем применения операции **переименования.**

Другие проблемы связаны с операцией взятия декартова (или прямого) произведения двух отношений. В теории множеств прямое произведение может быть получено для любых двух множеств, и элементами результирующего множества являются пары, составленные из элементов первого и второго множеств. Поскольку отношения являются множествами, то и для любых двух отношений возможно получение прямого произведения. Но результат не будет отношением! Элементами результата будут являться не кортежи, а пары кортежей. Поэтому в реляционной алгебре используется специализированная форма операции взятия прямого произведения - **расширенное** прямое произведение отношений. При взятии расширенного прямого произведения двух отношений элементом результирующего отношения является кортеж, являющийся конкатенацией (или слиянием) одного кортежа первого отношения и одного кортежа второго отношения. Но теперь возникает второй вопрос - как получить корректно сформированный заголовок отношения-результата? Очевидно, что проблемой может быть именование атрибутов результирующего отношения, если отношения-операнды обладают одноименными атрибутами. Эти соображения приводят к появлению понятия **совместимости по взятию расширенного прямого произведения**. Два отношения совместимы по взятию прямого произведения в том и только в том случае, если множества имен атрибутов этих отношений не пересекаются. Любые два отношения могут быть сделаны совместимыми по взятию прямого произведения путем применения операции переименования к одному из этих отношений. Следует заметить, что операция взятия прямого произведения не является слишком осмысленной на практике. Во-первых, мощность ее результата очень велика даже при допустимых мощностях операндов, а во-вторых, результат операции не более информативен, чем взятые в совокупности операнды. Как мы увидим немного ниже, основной смысл включения операции расширенного прямого произведения в состав реляционной алгебры состоит в том, что на ее основе определяется действительно полезная операция соединения. По поводу теоретико-множественных операций реляционной алгебры следует еще заметить, что

- все четыре операции являются **ассоциативными**. Т.е., если обозначить через OP любую из четырех операций, то

$$(A\ OP\ B)\ OP\ C = A\ OP\ (B\ OP\ C)$$

(A, B и C - отношения, обладающие свойствами, требуемыми для корректного выполнения соответствующей операции),

- все операции, кроме взятия разности, являются коммутативными, то есть,

A OP B = B OP A.

1. **Объединение**. Объединением двух совместимых по типу отношений S и P

(S UNION P)

называется отношение с тем же заголовком, что и у исходных отношений, и с телом, состоящим из множества всех кортежей, принадлежащих S или P, или обоим отношениям.

Пример 1. Пусть заданы два отношения

S

Номер	Фио	Город
01	Иванов	Томск
02	Петров	Томск

P

Номер	Фио	Город
01	Иванов	Томск
03	Сидоров	Кемерово

Пусть отношение S содержит сведения о поставщиках, проживающих в Томске, а отношение P – о поставщиках одного товара, например, «Труба». Тогда отношение

S UNION P

будет содержать сведения о поставщиках, проживающих в Томске, либо поставляющих товар «Труба»:

Номер	Фио	Город
01	Иванов	Томск
02	Петров	Томск
03	Сидоров	Кемерово

2. **Пересечение**. Пересечением двух совместимых по типу отношений S и P

(S INTERSECT P)

называется отношение с тем же заголовком, что и у исходных отношений, и с телом, состоящим из множества кортежей, принадлежащих одновременно обоим отношениям.

Результирующее отношение в условиях Примера 1. включает сведения о поставщиках из Томска, поставляющих товар «Труба»:

Номер	Фио	Город
01	Иванов	Томск

3. **Вычитание**. Вычитанием двух совместимых по типу отношений S и P

(S MINUS P)

называется отношение с тем же заголовком, что и у исходных отношений, и с телом, состоящим из множества кортежей, принадлежащих отношению S и не принадлежащих отношению P.

Результирующее отношение в условиях Примера 1. включает сведения о поставщиках из Томска, не поставляющих товар «Труба»:

Номер	Фио	Город
02	Петров	Томск

Очевидно, что отношение (**P MINUS S**) содержит сведения о поставщиках товара «Труба», не проживающих в Томске.

4. **Декартово произведение**. Декартово произведение двух **не имеющих общих имен** отношений S и P **(S TIMES P)** определяется как отношение с заголовком, который представляет собой сцепление исходных заголовков, и телом, состоящим из множества всех кортежей таких, что каждый кортеж получается конкатенацией некоторого кортежа

отношения S с некоторым кортежем отношения P. Очевидно, что количество кортежей результирующего отношения равно произведению количества кортежей отношений S и P.

Пример 2. Пусть отношения S (поставщики) и P (товары) имеют вид:

S

Ном_пос
S100
S101
S102

P

Ном_тов
P20
P21

В результирующем отношении (**S TIMES P**) фиксируется тот факт, что каждый из поставщиков поставляет все товары:

Ном_пост	**Ном_тов**
S100	P20
S100	P21
S101	P20
S101	P21
S102	P20
S102	P21

Специальные реляционные операции

1. **Выборка** (сокращенное название операции - □□□□ Θ (тета) - выборка):

S WHERE XΘY, где Θ ={<, ≤, =, ≠, ≥,>}.

Θ - выборкой из отношения S по атрибутам X и Y называется отношение, имеющее тот же заголовок, что и отношение S, и тело, содержащее множество всех кортежей отношения S таких, для которых проверка условия «XΘY» дает значение «истина». Ясно,что атрибуты X и Y должны быть определены на одном и том же домене, а операция на этом домене должна иметь смысл.

На практике эта операция используется, как правило, в модификации, когда вместо X или Y указано скалярное значение, чаще всего в форме

S WHERE XΘC, где C - литеральная константа.

В результате выполнения этого оператора получается «горизонтальное» подмножество исходного отношения.

Пример 3. Пусть задано отношение S (поставщики)

S

Номер	Фио	Город
S01	Иванов	Томск
S02	Петров	Томск
S03	Сидоров	Кемерово
S04	Кузнецов	Барнаул
S05	Быков	Омск

Результатом выполнения операции **S WHERE Город = «Томск»**, будет отношение

Номе	Фио	Город
S01	Иванов	Томск
S02	Петров	Томск

В указанной форме оператор выборки включает только простое сравнение. Однако, благодаря тождествам:

1. S WHERE Q1 AND Q2 ≡ (S WHERE Q1) INTERSECT (S WHERE Q2)
2. S WHERE Q1 OR Q2 ≡ (S WHERE Q1) UNION (S WHERE Q2)
3. S WHERE NOT Q ≡ S MINUS (S WHERE Q)

оператор выборки можно расширить до формы, в которой условие в выражении WHERE будет содержать произвольное число логических сочетаний простых сравнений.

2. **Проекция**. Проекцией отношения S по атрибутам X_1, X_2, …, X_n и обозначаемой как

$$\mathbf{S[X_1, X_2, \ldots, X_n]},$$

называется отношение с заголовком $\{X_1, X_2, \ldots, X_n\}$и телом, содержащим множество всех кортежей $\{X_1{:}x_1, X_2{:}x_2, \ldots, X_n{:}x_n\}$ таких, для которых в отношении S значение атрибута X_1 равно x_1, атрибута X_2 равно x_2, …, атрибута X_n равно x_n.

Пример 4. Результатом выполнения операции **S[Фио]** для отношения S предыдущего примера будет отношение

Фио
Иванов
Петров
Сидоров
Кузнецов
Быков

Таким образом, операция проекции строит «вертикальное» подмножество искомого отношения, то есть, подмножество, получаемое исключением всех атрибутов, не указанных в операторе, с последующим удалением дублируемых кортежей.

Замечание. Некорректное применение операции проекции может привести к потере информации; для нашего примера такая ситуация могла бы возникнуть, если в отношении S хранились бы сведения о двух разных поставщиках с одинаковой фамилией.

3. **Естественное (экви) соединение**. Пусть отношения S и P имеют заголовки$\{X_1, X_2, \ldots, Xm, Y_1, Y_2, \ldots, Yn\}$ и $\{Y_1, Y_2, \ldots, Y_n, Z_1, Z_2, \ldots, Zp$, соответственно, причем атрибуты $\{Y_1, Y_2, \ldots, Yn\}$по именам и доменам в обоих отношениях совпадают.

Рассмотрим три совокупности $\{X_1, X_2, \ldots, Xm\}$, $\{Y_1, Y_2, \ldots, Yn\}$ и $\{Z_1, Z_2, \ldots, Zp\}$ как три составных атрибута {X}, {Y} и {Z}. Тогда естественным соединением отношений S и P

S JOIN P

называется отношение с заголовком {X, Y, Z} и телом, содержащим множество всех кортежей {X:x, Y:y, Z:z} таких, для которых в отношении S значение атрибута X равно x, атрибута Y равно y, а в отношении P значение атрибута Y равно y, а атрибута Z равно z.

Замечание. При отсутствии общих атрибутов у исходных отношений S и P естественное соединение эквивалентно декартовому произведению.

Пример 5. Пусть заданы два отношения S (поставщики) и P (детали)

S

Ном_пост	Фио	Город
S01	Иванов	Томск
S02	Петров	Томск
S03	Сидоров	Кемерово
S04	Кузнецов	Барнаул
S05	Быков	Омск

P

Ном_дет	Назв_дет	Цвет	Вес	Город
P01	Вентиль	Белый	200	Томск
P02	Колено	Черный	500	Томск

P03	Втулка	Желтый	50	Кемерово
P04	Кран	Белый	600	Барнаул
P05	Смеситель	Белый	700	Барнаул
P06	Труба	Черный	1000	Омск

Результатом **(S JOIN P)** соединения этих двух отношений будет отношение

Ном_пост	Фио	Город	**Ном_дет**	Назв_дет	Цвет	Вес
S01	Иванов	Томск	P01	Вентиль	Белый	200
S01	Иванов	Томск	P02	Колено	Черный	500
S02	Петров	Томск	P01	Вентиль	Белый	200
S02	Петров	Томск	P02	Колено	Черный	500
S03	Сидоров	Кемерово	P03	Втулка	Желтый	50
S04	Кузнецов	Барнаул	P04	Кран	Белый	600
S04	Кузнецов	Барнаул	P05	Смеситель	Желтый	700
S05	Быков	Омск	P06	Труба	Черный	1000

Очевидно, что соединение может производиться не по одному, а по нескольким атрибутам.

Пример 6. Пусть заданы два отношения

R1

A	B	C
a1	b1	c1
a1	b2	c2
a2	b1	c1
a3	b2	c2

R2

B	C	D
b1	c1	d1
b1	c1	d2
b2	c2	d2

Результатом выполнения операции **(R1 JOIN R2)** будет являться отношение

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d2
a2	b1	c1	d1
a2	b1	c1	d2
a3	b2	c2	d2

Операция естественного соединения применяется к паре отношений R1 и R2, обладающих (возможно составным) общим атрибутом S (т.е. атрибутом с одним и тем же именем и определенным на одном и том же домене). Пусть AB обозначает объединение заголовков отношений R1 и R2. Тогда естественное соединение R1 и R2 - это спроецированный на AB результат эквисоединения R1 и R2 по A.S и B.S. Если вспомнить определение внешнего ключа отношения, то должно стать понятно, что основной смысл операции естественного соединения - возможность восстановления сложной сущности, декомопозированной (расчлененной) по причине требования первой нормальной формы. Операция естественного соединения не включается прямо в состав набора операций реляционной алгебры, но имеет очень важное практическое значение.

4. **Θ-соединение** (**условное соединение**). Это операция соединения двух отношений на основе некоторых условий, отличных от эквивалентности.

Пусть отношения S и P не имеют общих имен атрибутов и условие Θ определяется аналогично операции выборки. Тогда Θ-соединением отношения S по атрибуту X с отношением P по атрибуту Y называется результат вычисления выражения

(S TIMES P) WHERE XΘY,

то есть, результирующее отношение строится путем выполнения операции выборки по условию для декартова произведения исходных отношений.

Очевидно, что атрибуты X и Y должны быть определены на одном и том же домене и операция Θ должна иметь на этом домене смысл.

Пример 7. Пусть имеются два отношения S (потоки студенческих групп) и P (аудитории).

S

Ном_потока	Кол_студ
001	100
002	30

P

Ном_ауд	Вмест_ауд
302	140
128	50
336	20
432	110

Результатом выполнения операции Θ-соединения отношений S и P по условию

ВМЕСТ_АУД > КОЛ_СТУД

(вместимость аудитории должна быть больше количества студентов в потоке) будет отношение

Ном_потока	Кол_студ	Ном_ауд	Вмест_ауд
001	100	302	140
001	100	432	110
002	30	302	140
002	30	128	50
002	30	432	110

5. **Деление**. Пусть заголовки отношений S (делимое) и P (делитель) имеют вид{X_1, X_2, …, Xm, Y_1, Y_2, … , Yn}и {Y_1, Y_2, … , Yn},соответственно, причем атрибуты {Y_1, Y_2, … , Yn}в обоих отношениях представлены одинаковыми именами и определены на одних и тех же доменах. Будем рассматривать выражения{X_1, X_2, …, Xm} и {Y_1, Y_2, … , Yn}как два составных атрибута {X} и {Y}. Тогда результатом деления отношения S на отношение P, обозначаемое как

S DEVIDEBY P,

называется отношение с заголовком {X} и телом, содержащим множество всех кортежей{X:x} таких, что для каждого xЄX существует множество кортежей {X:x, Y:y} отношения S таких, у которых множество составляющих {Y:y} включает **все** кортежи {Y:y} отношения P. Или нестрого: результат содержит такие X-значения из отношения S, для которых соответствующие Y-значения включают **все** Y-значения из отношения P.

Пример 8. Пусть заданы следующие отношения

SP

Ном_пост	Ном_дет
S1	P1
S1	P2
S1	P3

S1	P4
S1	P5
S1	P6
S2	P1
S2	P2
S2	P4
S3	P2
S4	P2
S4	P4
S4	P5

P1

Ном_дет
P1

P2

Ном_дет
P2
P4

P3

Ном_дет
P1
P2
P3
P4
P5

Результатами выполнения операций деления будут следующие отношения:

SP DIVIDEBY P1:

Ном_пост
S1
S2

SP DEVIDEBY P2:

Ном_пост
S1
S2
S4

SP DEVIDEBY P3:

Ном_пост
S1

Пример использования операций реляционной алгебры
Пусть заданы отношения S (поставщики), P (детали) и SP (поставки):

S

Ном_пост	Фио	Город
S01	Иванов	Томск
S02	Петров	Томск
S03	Сидоров	Кемерово

S04	Кузнецов	Барнаул
S05	Быков	Омск

P

Ном_дет	Назв_дет	Цвет	Вес	Город
P01	Вентиль	Белый	200	Томск
P02	Колено	Черный	500	Томск
P03	Втулка	Желтый	50	Кемерово
P04	Кран	Белый	600	Барнаул
P05	Смеситель	Желтый	700	Барнаул
P06	Труба	Черный	1000	Омск

SP

Ном_пост	**Ном_дет**	Количество
S01	P01	300
S01	P02	200
S01	P03	400
S01	P04	200
S01	P05	100
S01	P06	100
S02	P01	300
S02	P02	400
S03	P02	200
S04	P02	200
S04	P04	300
S04	P05	400

Запросы к базе данных, включающей отношения S, P, и SP, могут быть реализованы следующими цепочками операций реляционной алгебры.

Запрос 1. Получить имена поставщиков, поставляющих деталь «Колено»

(((SP JOIN P) WHERE Назв_дет = «Колено») JOIN S) [Фио]

Запрос 2. Получить имена поставщиков, поставляющих все детали.

(((SP[Ном_пост, Ном_дет]) DEVIDEBY (P[Ном_дет])) JOIN S) [Фио]

Запрос 3. Получить имена поставщиков, которые не поставляют деталь «Кран».

(S[Ном_пост,Фио]MINUS(((SP JOIN P) WHERE Назв_дет=«Кран»)JOIN S) [Ном_пост,ФИО]) [Фио]

Учитывая, что каждый экземпляр схемы отношения можно представить как значение некоторой переменной, выполнение этого запроса может быть реализовано следующей иллюстративной программой:

T1 = **S[Ном_пост, Фио]**;
T2 = **SP JOIN P**;
T3 = **T2 WHERE Назв_дет = «Кран»**;
T4 = **T3 JOIN S**;
T5 = **T4[Ном_пост, Фио]**;
T6= **T1 MINUS T5**;
T7 = **T6[Фио].**

Замечание. Если посмотреть на общий обзор реляционных операций, то видно, что домены атрибутов результирующего отношения однозначно определяются доменами отношений-результатов. Однако с именами атрибутов результата не всегда бывает все так просто. Например, представим себе, что у отношений-операндов операции прямого произведения имеются одноименные атрибуты с одинаковыми доменами. Каким должен был

бы быть заголовок результирующего отношения? Поскольку заголовок - это множество, в нем не должны содержаться одинаковые элементы. Но и потерять атрибут в результате недопустимо. А это значит, что в этом случае вообще невозможно корректно выполнить операцию прямого произведения. Аналогичные проблемы могут возникать и в случаях других двуместных операций. Для их разрешения в состав операций реляционной алгебры вводится операция **переименования**. Ее следует применять в любом случае, когда возникает конфликт именования атрибутов в отношениях-операндах одной реляционной операции. Тогда к одному из операндов сначала применяется операция переименования, а затем основная операция выполняется уже безо всяких проблем.

5.1.4.2. Реляционное исчисление

Реляционная алгебра и реляционное исчисление представляют собой два альтернативных подхода. Принципиальное различие между ними состоит в следующем: **алгебра** представляет в явном виде набор операций, которые можно использовать, чтобы сообщить системе, как из определенных в базе данных отношений **построить** необходимое отношение, в то время как **исчисление** представляет собой просто систему обозначение для **определения** необходимого отношения в терминах имеющихся отношений.

Пример. Пусть к базе данных, содержащей отношения S, P и SP (точнее, к СУБД, в рамках которой реализована данная база данных) поступил запрос:

«Получить номера и города поставщиков, поставляющих деталь «Смеситель».

Реализация запроса в рамках аппарата реляционной алгебры могла бы включать следующие шаги:

П1. Образовать **естественное соединение** отношений SP и P по атрибуту **Ном_дет**.

П2. Выбрать из результата этого соединения кортежи, у которых **Назв_дет= «Смеситель».**

П3. Образовать естественное соединение результирующего отношения П2 и отношения S по атрибуту **Ном_пост**.

П4. Спроецировать результирующее отношение по атрибутам **Ном_пост** и **Город**.

В терминах реляционного исчисления формулировка искомого выражения могла бы выглядеть следующим образом:

“Получить атрибуты **Ном_пост** и **Город** для таких поставщиков отношения S, для которых существует поставка в отношении SP детали со значением атрибута **Ном_дет** таким, что значение атрибута **Назв_дет** в соответствующих кортежах отношения P равно «**Смеситель**».

Реляционное исчисление основано на разделе математической логики, который называется **исчислением** (или **логикой**) **предикатов**. Существуют две версии реляционного исчисления – **исчисление кортежей** и **исчисление доменов**.

В исчислении кортежей основным является понятие **переменной кортежа**, то есть, переменной, в качестве допустимых значения которой выступают кортежи некоторого отношения. В исчислении доменов основным является понятие **переменной домена**. Обе эти версии реляционного исчисления эквивалентны. Рассмотрим в самом общем виде механизм использования аппарата исчисления кортежей.

Рассмотрим основные понятия данного инструментария:

- **переменная кортежа**,
- **правильно построенная формула**,
- **выражение исчисления кортежей**.

Пусть **переменная кортежа** определяется (описывается) выражением [3]

RANGE OF T IS $V_1, V_2, \dots, V_n$,

где

T – переменная кортежа,

$\mathbf{V}_i$ (i =1, …, n) – имя отношения либо выражение исчисления кортежей.

Пусть каждое из V_i (i =1, … , n) является отношением, причем все отношения должны быть совместимыми по типу. Тогда областью значений переменной кортежа T является множество кортежей объединения отношений V_i (i =1, … , n).

Примеры определения переменной кортежа:

RANGE OF TS IS S;

RANGE OF TP IS P;

RANGE OF T1 IS TS WHERE S.Город = «Томск».

Замечание. В третьем примере значениями переменной T1 являются все кортежи отношения S, для которых выполняется указанное в определении условие.

Правильно построенная формула (**ППФ**). Понятие ППФ является одним из центральных в логике предикатов, следовательно, и в реляционном исчислении. ППФ служат для выражения условий, накладываемых на кортежные переменные.

Базовым при построении ППФ является определение **атома**. Применительно к исчислению кортежей выделяют два типа атомов:

Тип 1. **T.X Θ U.Y**, где T,U – переменные-кортежи, X,Y – атрибуты, Θ = {<, ≤, =, ≠, ≥,>}.

Тип 2. **C Θ T.X**, **T.X Θ C**, где C – литеральная константа.

Таким образом, основой ППФ являются простые сравнения, представляющие собой операции сравнения скалярных значений (значений атрибутов переменных или литерально заданных констант). Например, конструкция **S.Город =«Томск»** является простым сравнением. По определению, простое сравнение является ППФ. Итак,

Правило 1. Каждый атом есть ППФ.

Более сложные варианты ППФ строятся с помощью логических связок **NOT**, **AND**, **OR** и **IF** ... **THEN**.

Правило 2 (введение логических формул).

Если φ и ψ – суть ППФ, то

- **NOT φ**,
- **φ AND ψ**,
- **φ OR ψ**,
- **IF Q THEN φ** , (где Q – логическая формула),
- **(φ)**,

тоже суть ППФ.

Наконец, допускается построение ППФ с помощью кванторов.

Правило 3 (введение кванторов).

Обозначим через **EXISTS** квантор **c*уществования*** и через **FORALL** – квантор ***всеобщности***. Пусть φ есть ППФ и T – переменная. Тогда

- **EXISTS T(φ)**,
- **FORALL T(φ)** - суть ППФ.

Здесь первая формула означает:

«Существует по крайней мере одно такое значение переменной T (по крайней мере один кортеж соответствующего отношения), на котором результат вычисления ППФ φ равен булевому значению **True**».

Трактовка второй формулы очевидна:

«Для всех значений переменной T (на всех кортежах соответствующего отношения) вычисление значения ППФ φ дает значение **True**.»

Переменные, входящие в ППФ, могут быть **свободными** или **связанными**. Все переменные, входящие в ППФ, при построении которой не использовались кванторы, являются свободными. Фактически, это означает, что если для какого-то набора значений свободных кортежных переменных при вычислении ППФ получено значение **истина**, то эти значения кортежных переменных могут входить в результирующее отношение. Если же имя

переменной T использовано сразу после квантора при построении ППФ вида EXISTS или FORALL, то в этой ППФ и во всех ППФ, построенных с ее участием, T - это связанная переменная. Это означает, что такая переменная не видна за пределами минимальной ППФ, связавшей эту переменную. При вычислении значения такой ППФ используется не одно значение связанной переменной, а вся ее область определения.

Пусть **СОТР1** и **СОТР2** - две кортежные переменные, определенные на отношении **СОТРУДНИКИ**. Тогда, ППФ

EXISTS СОТР2 (СОТР1.Сотр_зарп > СОТР2.Сотр_зарп)

для текущего кортежа переменной **СОТР1** принимает значение **истина** в том и только в том случае, если во всем отношении **СОТРУДНИКИ** найдется хотя бы один кортеж (связанный с переменной **СОТР2**) такой, что значение его атрибута **СОТР_ЗАРП** удовлетворяет внутреннему условию сравнения. В свою очередь, ППФ

FORALL СОТР2 (СОТР1.Сотр_зарп > СОТР2.Сотр_зарп)

для текущего кортежа переменной **СОТР1** принимает значение **истина** в том и только в том случае, если для всех кортежей отношения **СОТРУДНИКИ** (связанных с переменной **СОТР2**) значения атрибута **Сотр_зарп** удовлетворяет условию сравнения.

На самом деле, правильнее говорить не о свободных и связанных переменных, а о свободных и связанных **вхождениях** переменных. Легко видеть, что если переменная T является связанной в ППФ φ, то во всех ППФ, включающих φ, может использоваться имя переменной T, которая может быть свободной или связанной, но в любом случае не имеет никакого отношения к вхождению переменной T в ППФ φ.

Пример. Пусть ППФ F1 и F2 имеют следующий вид:

F1 - **EXISTS СОТР2 (СОТР1.Сотр_отд_ном = СОТР2.Сотр_отд_ном)**

F2 - **FORALL СОТР2 (СОТР1.Сотр_зарп > СОТР2.Сотр_зарп)**

Очевидно, что в ППФ (F1 AND F2) два связанных вхождения переменной **СОТР2** имеют совершенно разный смысл.

Сформулируем данные понятия более точно. Под **экземпляром** переменной кортежа в ППФ будем понимать наличие **имени** переменной:

a) в строке **T.X**,

b) в строках **EXISTS T** и **FORALL T**.

Заметим, что здесь понятие экземпляра переменной рассматривается в чисто синтаксическом аспекте.

Каждый **экземпляр переменной** кортежа T в ППФ является или **свободным,** или **связанным** в зависимости от выполнения следующих условий:

- в атомах все экземпляры ***свободны***;
- экземпляры переменной T, которые в формуле φ, ***связаны*** в формулах EXISTS T(φ) и FORALL T(φ);
- в логических ППФ экземпляры свободны или связанны в зависимости от того, свободны или связаны они в формулах φ и ψ.

Примеры:

Атомы:

1) **TS.Ном_пост=”S1”**

2) **TS.Ном_пост ≠ TSP.Ном_пост**

Экземпляры TS и TSP в обоих примерах свободны.

Логические формулы:

1) **NOT (TS.Город = “Томск”)**

2) **(TS.Ном_пост = TSP.Ном_пост) AND (TSP.Ном_дет ≠ T.Ном_дет)**

Экземпляры TS и TSP в обоих примерах свободны.

Кванторные формулы:

1) **EXISTS TSP ((TSP.Ном_пост = TS.Ном_пост) AND (TSP.Ном_дет = TP.Ном_дет) AND (TP.Назв_дет ≠ “Смеситель”))**

2) **FORALL TP (TP.цвет = “Белый”)**

В первом примере все три экземпляра TSP связаны, а экземпляры TS и TP – свободны; во втором примере оба экземпляра TP связаны.

Еще раз отметим, что понятие связности необходимо для определения истинности или ложности формулы (соответствующего утверждения): связность принципиальным образом уточняет сформулированное утверждение.

Выражения исчисления кортежей. В общем виде выражение исчисления кортежей имеет вид:

<список_целевых_элементов> [WHERE ППФ],

где:

- **целевой элемент** - имя простой переменной (T) или выражение вида (T.A),
- **WHERE** – условие.

Результатом выполнения выражения является удовлетворение запроса к базе данных.

Примеры. Реализовать в виде выражений следующие запросы:

1) «Определить имена поставщиков, поставляющих деталь «Кран»:

TS.ФИО WHERE EXISTS TSP ((EXISTS TS(TSP.Ном_пост = TS.Ном_пост)) AND (EXISTS TP ((TS.Ном_дет = TP.Ном_дет) AND (TP.Ном_дет = “Кран”)))

2) «Определить имена поставщиков, поставляющих все детали»:

TS.ФИО WHERE FORALL TP (EXISTS TSP)(EXITS TS (TSP.Ном_пост = TS.Ном_пост)) AND (TSP.Ном_дет = TS.Ном_дет)))

Очевидно, что при использовании механизма исчисления пользователь лишь устанавливает определенные характеристики необходимого отношения, оставляя СУБД решать, что необходимо соединять, выбирать, проецировать и т.д. для получения требуемого результата.

Таким образом, можно сказать, что, по крайней мере внешне, формулировка удовлетворения запроса в терминах реляционной алгебры носит предписывающий (**процедурный**) характер, а реляционного исчисления – описательный (**декларативный**). В алгебре описывается **решение** проблемы, запрос, представленный на языке реляционной алгебры, может быть **вычислен** на основе вычисления элементарных алгебраических операций с учетом их старшинства и возможного наличия скобок. В исчислении предикатов описывается, в чем проблема **заключается;** формула только устанавливает условия, которым должны удовлетворять кортежи результирующего отношения.

Эти отличия, однако, существуют только внешне. Доказано, что механизмы реляционной алгебры и реляционного исчисления эквивалентны, то есть для любого допустимого выражения реляционной алгебры можно построить эквивалентную (т.е. производящую такой же результат) формулу Различия связаны с разными **стилями** выражения: алгебра ближе к языкам программирования, а исчисление – естественному языку.

Заметим, что крайне редко алгебра или исчисление принимаются в качестве полной основы какого-либо языка БД. Обычно (как, например, в случае языка SQL) язык основывается на некоторой смеси алгебраических и логических конструкций. Тем не менее, знание алгебраических и логических основ языков баз данных часто бывает полезно на практике.

Подробно реляционная модель данных рассмотрена в [2].

5.1.5. Достоинства и недостатки реляционных моделей

К числу наибольших **достоинств** реляционного подхода можно отнести:

- наличие небольшого набора абстракций, которые позволяют сравнительно просто моделировать большую часть распространенных предметных областей и допускают точные формальные определения, оставаясь интуитивно понятными;
- наличие простого и в то же время мощного математического аппарата, опирающегося главным образом на теорию множеств и математическую логику и обеспечивающего теоретический базис реляционного подхода к организации баз данных;
- возможность ненавигационного манипулирования данными без необходимости знания конкретной физической организации баз данных во внешней памяти.

Среди основных **недостатков** реляционной модели в настоящее время выделяют:

- слабая система типов данных; этим системам присуща некоторая ограниченность (прямое следствие простоты) при использовании в так называемых нетрадиционных областях применения (наиболее распространенными примерами являются системы автоматизации проектирования), в которых требуются предельно сложные структуры данных,
- невозможность адекватного отражения семантики предметной области; другими словами, возможности представления знаний о семантической специфике предметной области в реляционных системах очень ограничены,
- сложности интеграции в новые технологические среды, которые основаны главным образом на объектных моделях.

Современные исследования в области постреляционных систем главным образом посвящены именно устранению этих недостатков (см. раздел 8.).

5.2. Навигационные модели данных

Смысл краткого рассмотрения ранних (дореляционных) СУБД может быть полезен, по крайней мере, по трем причинам: во-первых, эти системы исторически предшествовали реляционным и для правильного понимания причин повсеместного перехода к реляционным системам нужно знать хотя бы что-нибудь об их предшественниках; во-вторых, внутренность реляционных систем во многом основана на использовании методов ранних систем; в-третьих, некоторое знание в области ранних систем будет полезно для понимания путей развития постреляционных СУБД.

5.2.1. Иерархическая модель

Основные понятия. Основным типом логической структуры, поддерживаемой иерархическими СУБД, является **иерархическая** (**древовидная**) структура (или просто **иерархия**).

Иерархическая структура определяется на соответствующих типах записей согласно **схеме базы данных** (**дереву определений, типу дерева**). В этом дереве типы записей являются узлами, а дуги представляют **иерархические связи** («исходный-порожденный», «род-вид», «элемент-класс» и т.п.) между узлами дерева различных уровней. Тип дерева состоит из одного «корневого» типа записи и упорядоченного набора (из нуля или более) типов поддеревьев (каждое из которых является некоторым типом дерева). Тип дерева в целом представляет собой иерархически организованный набор типов записей.

Содержимое иерархической БД представляется упорядоченным набором нескольких **экземпляров** типа дерева.

В общем случае иерархия должна удовлетворять следующим условиям:

1. Одно дерево может иметь только один **корень** (исходный узел).
2. Узел должен иметь один или несколько **атрибутов**, описывающих семантику объекта.

3. У родительского узла может быть несколько потомков.

4. Потомок может быть связан только с одним родительским узлом.

5. Порожденные узлы могут быть добавлены как в горизонтальном, так и в вертикальном направлениях. Ограничения, например, на глубину иерархии, могут накладываться только требованиями конкретной иерархической СУБД.

Если разность уровней двух связанных узлов равна единице, связь является **непосредственной**; кроме того, любые две вершины дерева, принадлежащие одной его ветви, **транзитивно** связаны друг с другом.

Пример. На рис.14. представлена иерархическая база данных:

(а) - дерево определения с пятью узлами (типами записей) A, B, C, D и E, обозначенными прямоугольниками. Символы внутри прямоугольников задают атрибуты объектов,

(б) – загруженная база данных (экземпляры дерева определений R1, R2).

Пусть типы записей базы данных представляют следующие объекты предметной области:

A – «**ДИСЦИПЛИНА**»,

B – «**УЧЕБНЫЙ ПЛАН**»,

C – «**СЕМЕСТР**»,

D – «**ПРЕПОДАВАТЕЛЬ**»,

E – «**СТУДЕНЧЕСКАЯ ГРУППА**».

В такой трактовке экземпляры базы данных могут быть интерпретированы следующим образом:

Экземпляр R1. Дисциплина A1 включена в учебный план; больше о ней ничего не известно.

Экземпляр R2. Дисциплина A2 включена в учебный план. Для нее запланировано проведение занятий в семестрах C1 и C2 преподавателем D1 в группах E1 и E2 и в семестре C3 в группе E3, однако преподаватель еще не определен.

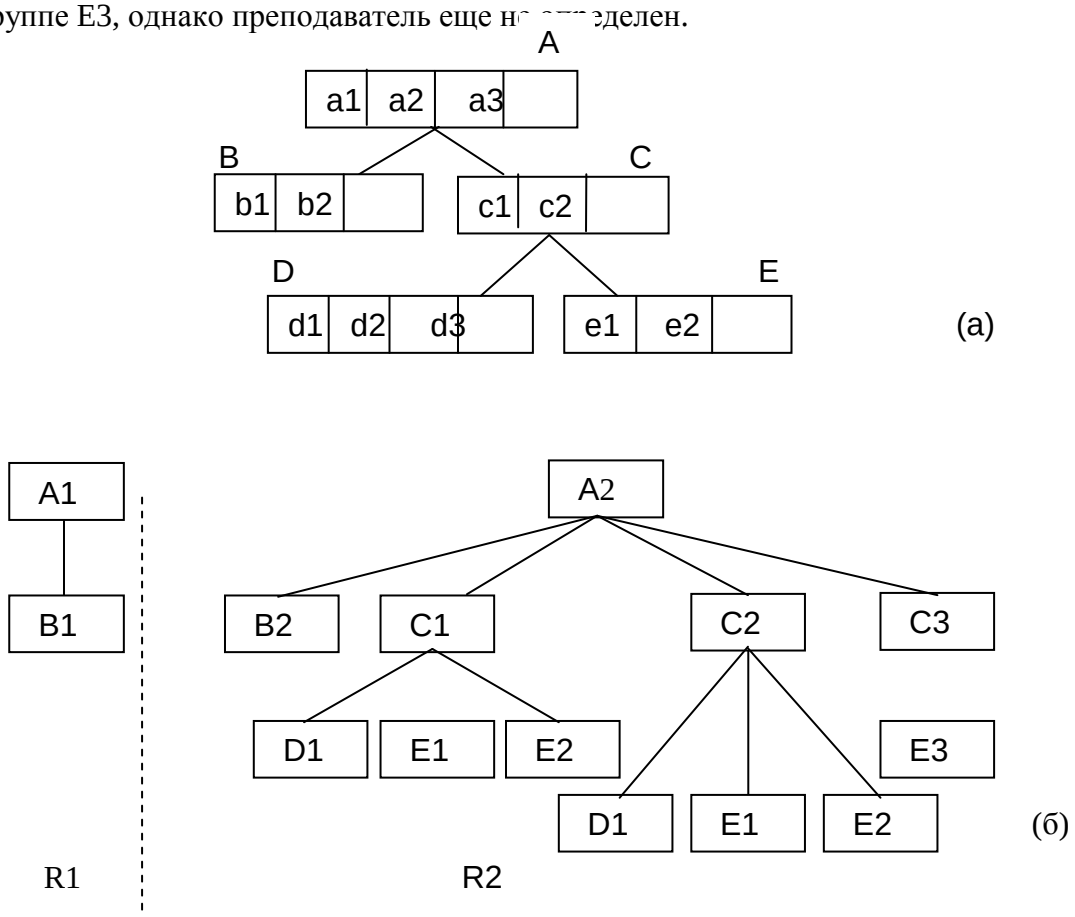


Рис.14. Иерархическая база данных

Принципы организации физического хранения записей. Набор всех экземпляров записей, подчиненных одному экземпляру корневой записи, называется **физической записью**. Порядок хранения экземпляров типов записей в физической записи зависит от особенностей реализации, определяющих способ отображения логических деревьев в физические структуры файлов конкретной СУБД. Наиболее простой способ состоит в линеаризации дерева; при этом экземпляры типов записей хранятся последовательно в порядке обхода дерева сверху вниз слева направо.

Пример. Физическая запись, хранящая экземпляры R1 и R2 при линеаризации, будет выглядеть следующим образом:

A1 B1	A2 B2 C1 D1 E1 E2 C2 D1 E1 E2 C3 E3
Запись 1	Запись 2

Ясно, что при этом в общем случае физические записи будут иметь различную длину, причем длина некоторых из них может оказаться недопустимо большой (как с точки зрения занимаемых ресурсов памяти, так и времени поиска нужной информации). Поэтому в иерархических СУБД широко используется аппарат разделения графа, представляющего дерево определений, на подграфы (поддеревья), хранящиеся (реализуемые) отдельно, но в целом (в совокупности) представляющие единую логическую структуру – дерево определений.

Такое расчленение обычно соответствует делению предметной области на **фрагменты**, каждый из которых описывается своим подграфом дерева определений.

Пример. Дерево определений, представляющее структуру университета (до определенного уровня), может быть представлено следующим графом (рис. 15.):

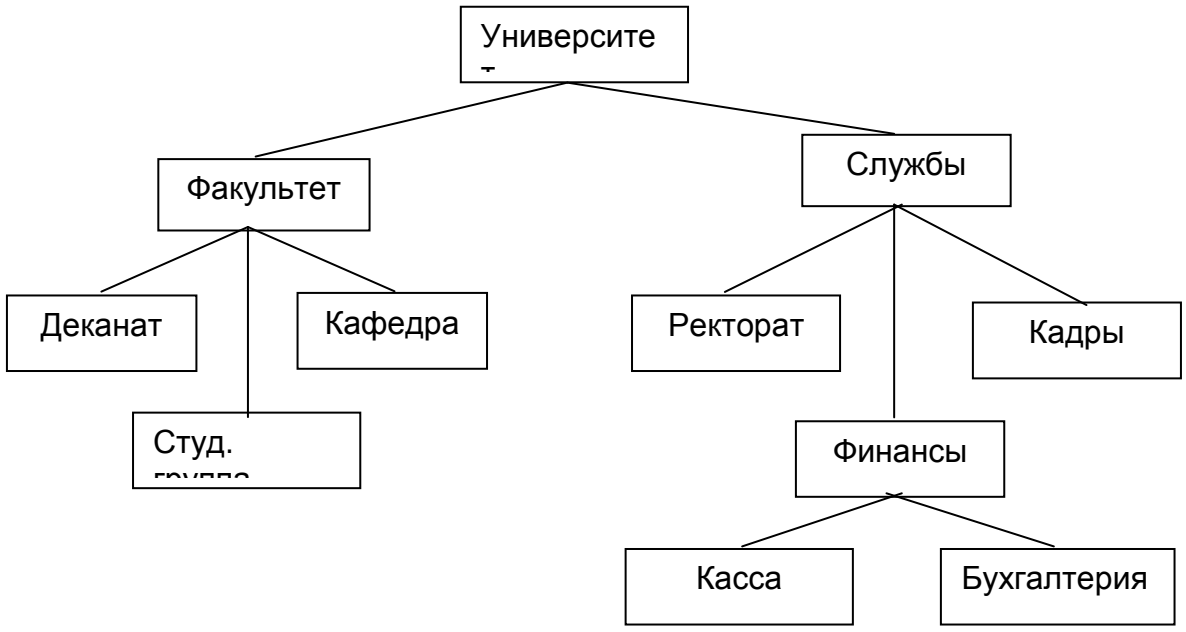


Рис. 15. Упрощенный вид дерева определений ПО «Университет»

В данном дереве можно выделить такие фрагменты, как например, **Деканат**, **Ректорат**, **Кадры**, **Финансы** и т.п. Каждый из фрагментов предметной области представляется отдельной **физической** базой данных (ФБД).

Концептуальная модель. Совокупность взаимосвязанных ФБД образует **концептуальную** иерархическую модель данных. Очевидно, что в простейшем случае дереву определений («бумажной» модели) соответствует одна ФБД. Каждая ФБД на физическом уровне реализуется отдельным набором физических записей. Для каждой ФБД на языке описания данных должно присутствовать как описание ее логической структуры, так и структуры хранения.

Внешняя модель. При работе с иерархической моделью внешняя представляет собой совокупность поддеревьев для физических баз данных, с которыми работает данное приложение. Каждое поддерево внешней модели в обязательном порядке должно содержать корневой тип сегмента соответствующей ФБД концептуальной модели.

Представление внешней модели часто называют **логической базой данных** и определяется она совокупностью блоков связи данного приложения с ФБД, входящими в состав концептуальной схемы иерархической базы данных.

Навигация в иерархических структурах. Язык манипулирования данными (ЯМД) в иерархической модели поддерживает в явном виде навигационные операции, связанные с перемещением указателя, который определяет текущий экземпляр конкретной записи. Поскольку в иерархических СУБД способ реализации операций поиска ориентирован на древовидную структуру, сам поиск должен начинаться с корня и продолжаться в направлении порожденных узлов. Очевидно, что отсутствие в конкретной реализации какого-либо иного способа доступа к требуемому типу записи, кроме последовательного, порождает проблему оптимального физического распределения узлов в дереве для минимизации времени поиска. Как правило, ЯМД иерархических СУБД включает операции прямого доступа к записям, то есть, «горизонтального» просмотра дерева.

Примеры типичных операторов манипулирования иерархическими данными:

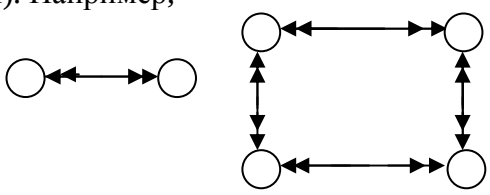
- найти указанное дерево БД (например, предмет “Информатика”);
- перейти от одного дерева к другому;
- перейти от одной записи к другой внутри дерева (например, от одной группы к другой);
- перейти от одной записи к другой в порядке обхода иерархии;
- вставить новую запись в указанную позицию;
- удалить текущую запись.

Ограничения целостности. Автоматически поддерживается целостность ссылок между предками и потомками. Основное правило: никакой потомок не может существовать без своего родителя. Заметим, что аналогичное поддержание целостности по ссылками между записями, не входящими в одну иерархию, не поддерживается.

Общие правила определения целостности БД отсутствуют. В некоторых системах поддерживают ограничения уникальности значений некоторых полей, но в основном все возлагается на прикладную программу.

5.2.2. Сетевые структуры

Основные понятия. Сетевой подход к организации данных является расширением иерархического. В иерархических структурах запись-потомок должна иметь в точности одного предка; в сетевой структуре данных потомок может иметь любое число предков. С точки зрения теории графов сетевой модели соответствует произвольный граф (возможно, имеющий циклы и петли). Например,



Стандарт сетевой модели впервые был определен в 1975г. организацией CODASYL (Conference of Data System Languages), которая определила базовые понятия модели и формальный язык ее описания.

Базовыми объектами модели являются:

- **элемент данных**,

- **агрегат данных**,
- **запись**,
- **набор данных**.

Элемент данных – минимальная информационная единица, доступная с помощью СУБД.

Агрегат данных – соответствует следующему после элемента данных уровню обобщения.

Например, агрегат с именем **Адрес** может быть представлен в виде

Адрес			
Город	Улица	Дом	Квартира

Запись - в общем случае совокупность элементов данных и/или агрегатов, моделирующая некоторый класс объектов реального мира.

Набор – двухуровневый граф, связывающий отношением 1:M два **типа записей**.

Набор фактически отображает и Член набора N язь между двумя типами записей - родительским типом записи (**предком**), называемым **владельцем** набора, и дочерним типом записи (**потомком**) – **членом** набора. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка.

Для любых двух типов записей может быть задано любое количество наборов, которые их связывают. Наличие подобных возможностей позволяет промоделировать отношение M:N между двумя объектами реального мира, что выгодно отличает сетевую модель от иерархической.

Пример. На рис.16. представлены два набора, связывающие сущности «Преподаватель» и «Группа».

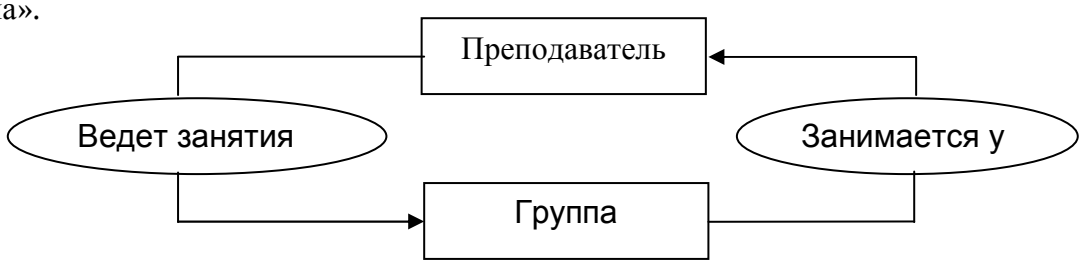


Рис.16. Наборы данных, связывающие сущности «Преподаватель» и «Группа»

Пусть связь между экземплярами типов записей в наборах представляется содержимым таблицы1.

Экземпляров набора «Ведет занятия у» – 3 (по количеству преподавателей), а экземпляров набора «Занимается у» – 4 (по количеству групп).

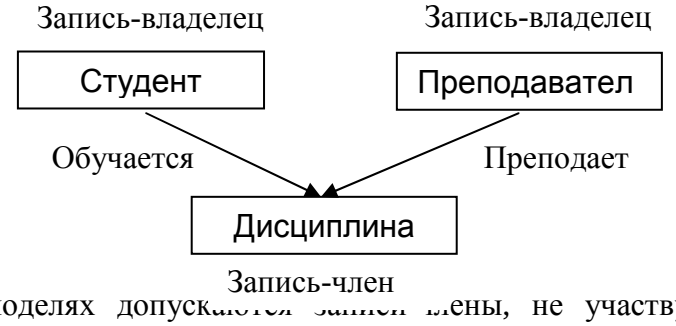
Таблица 1.

Преподаватель	Группа
Иванов	1171
Иванов	1172

Карпова	1172
Карпова	1174
Карпова	723
Смирнов	1171
Смирнов	1174

Учитывая первоначальные рекомендации CODASYL и результаты последующих соглашений, можно сформулировать следующие основные свойства и ограничения, присущие набору:

1. Набор – это поименованная совокупность связанных записей.
2. В каждом экземпляре набора имеется только один экземпляр записи-владельца.
3. Тип набора представляет логическую взаимосвязь 1:M между владельцем и членом
4. Существенное различие между сетевой и иерархической моделями данных состоит в том, что в сетевой модели каждая запись может участвовать в любом количестве наборов:



5. В сетевых моделях допускаются записи-члены, не участвующие в наборах. Это соответствует порожденным узлам дерева, не имеющим исходных.
6. Между двумя типами записей можно определить любое количество наборов, то есть, между ними могут быть определены любые типы отношений.
7. Экземпляр набора существует после заполнения записи-владельца.

На формирование типов связи не накладываются особые ограничения; возможны, например, ситуации:

- Тип записи потомка в одном типе связи может быть типом записи предка в другом типе связи (как в иерархии).
- Данный тип записи может быть типом записи предка в любом числе типов связи.
- Данный тип записи может быть типом записи потомка в любом числе типов связи.
- Предок и потомок могут быть одного типа записи.

Концептуальная модель. Таким образом, сетевая БД состоит из набора записей и набора связей между ними, на концептуальном уровне образующих некоторый граф, а если говорить более точно, из набора экземпляров каждого типа из заданного в схеме БД набора типов записи и набора экземпляров каждого типа из заданного набора типов связи.

Внешняя модель. При сетевой организации данных внешняя модель поддерживается путем описания соответствующей части общего связного графа.

Навигация в сетевой модели. Навигационные операции осуществляют перемещение по БД путем прохождения по связям, которые поддерживаются в схеме БД. В этом случае результатом является новый единичный объект, который получает статус текущего объекта.

Примерный набор операций манипулирования данными может быть таковым:

- найти конкретную запись в наборе однотипных записей;
- перейти от предка к первому потомку по некоторой связи;
- перейти к следующему потомку в некоторой связи;
- перейти от потомка к предку по некоторой связи;
- создать новую запись;

- уничтожить запись;
- модифицировать запись;
- включить в связь;
- исключить из связи;
- переставить в другую связь и т.д.

Искомая взаимосвязь экземпляров типов записей, приведенных в таблице 1 для наборов примера на рис. 16, может быть представлена в виде следующей диаграммы:

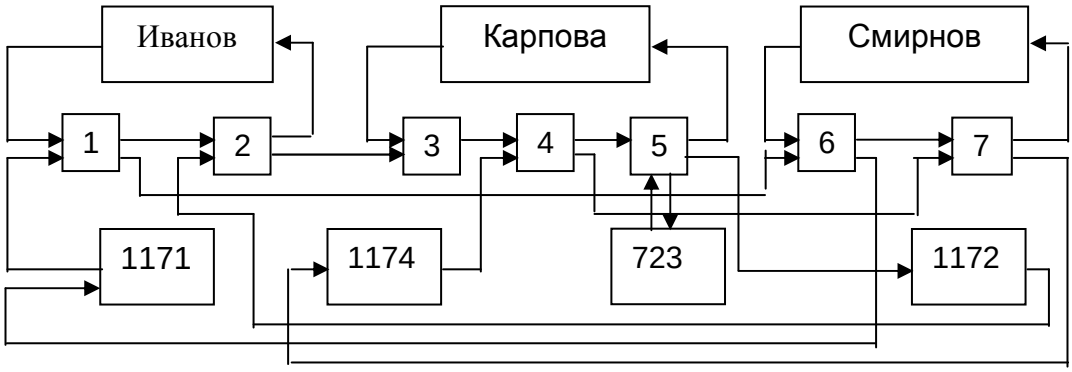


Рис.17. Пример диаграммы связей экземпляров типов записей в сетевой модели

Очевидно, что задачи как физической организации файла, в которую отображается данная структура, так и поиска нужных записей отнюдь не тривиальны.

Наиболее интересна операция поиска (**FIND**), так как именно она отражает суть навигационных методов, применяемых в сетевой модели. Всего существует 7 типов операций поиска:

- по ключу,
- последовательный просмотр записей данного типа,
- поиск владельца текущего экземпляра набора,
- последовательный просмотр записей-членов текущего экземпляра набора,
- просмотр записей-членов экземпляра набора, специфицированных рядом полей,
- сделать текущей записью процесса текущий экземпляр набора,
- установить текущую запись процесса.

Таким образом, выполнение команд поиска устанавливает **текущий указатель** (указатель на **текущий объект**). Далее найденный объект, если его необходимо модифицировать, должен быть перемещен из БД в рабочую область приложения.

Ограничения целостности. В принципе их поддержание не требуется, но иногда требуют целостности по ссылкам (как в иерархической модели).

5.2.3. Особенности навигационных моделей. Достоинства и недостатки

Особенности:

- Навигационные модели активно использовались в течение многих лет, дольше, чем какая-либо из реляционных СУБД. На самом деле некоторые из ранних систем используются даже в наше время, накоплены громадные базы данных, и одной из актуальных проблем информационных систем является использование их совместно с современными системами.
- Все ранние системы не основывались на каких-либо абстрактных моделях. Понятие модели данных фактически вошло в обиход специалистов в области БД только вместе с реляционным подходом. Абстрактные представления ранних систем появились позже на основе анализа и выявления общих признаков у конкретных систем.

- В ранних системах доступ к БД производился на уровне записей. Пользователи этих систем осуществляли явную навигацию в БД, используя языки программирования, расширенные функциями СУБД. Интерактивный доступ к БД поддерживался только путем создания соответствующих прикладных программ с собственным интерфейсом.
- Навигационная природа ранних систем и доступ к данным на уровне записей заставляли производить всю оптимизацию доступа к БД самого пользователя, без какой-либо поддержки системы.
- После появления реляционных систем большинство ранних систем было оснащено «реляционными» интерфейсами. Однако в большинстве случаев это не сделало их по-настоящему реляционными системами, поскольку оставалась возможность манипулировать данными в естественном для них режиме.

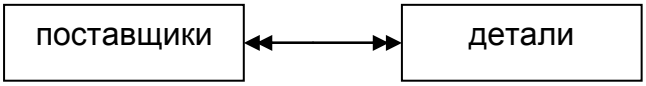
Достоинства:

- Иерархическая модель естественным образом реализует отображение 1:N между исходным и порожденными типами записей. Это отображение полностью функционально, так как дерево не может содержать порожденный узел без исходного.
- Большая выразительность сетевой модели, достигаемая, прежде всего, возможностью установления между сущностями отношений любого типа.
- Развитые средства управления данными во внешней памяти на низком уровне.
- Возможность построения вручную эффективных прикладных систем.
- Возможность экономии памяти за счет разделения подобъектов (в сетевых системах).

Недостатки:

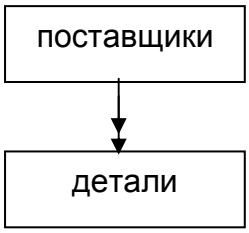
- Для представления отображения типа M:N в иерархических системах необходимо дублирование деревьев.

Пример. Связь

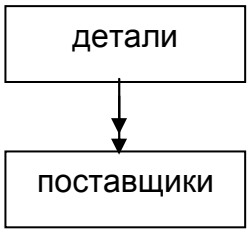


должна быть реализована двумя схемами:

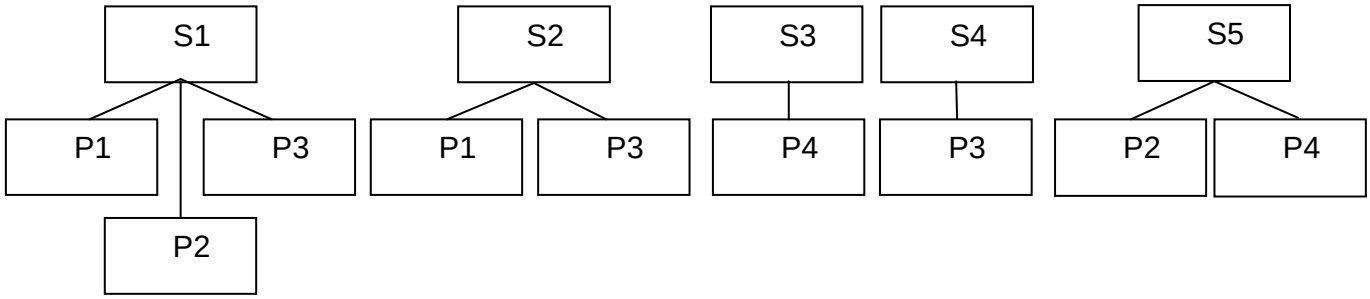
(а)



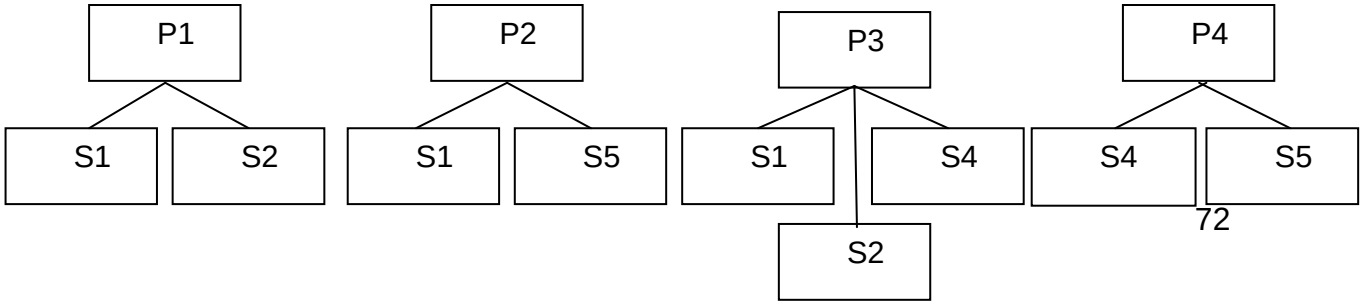
(б)



Пусть загрузочная база данных по схеме (а) включает экземпляры:



Тогда загрузочная база данных по схеме (б) должна включать экземпляры:



Очевидно, что расплатой за необходимость такой реализации отношений M:N выступает высокая степень дублирования данных в базе.

- Моделью слишком сложно пользоваться. Фактически необходимы знания о физической организации.
- Прикладные системы зависят от физической организации. Их логика перегружена деталями организации доступа к БД.
- Разработчик приложения должен детально знать логическую структуру БД, поскольку ему необходимо осуществлять навигацию среди различных экземпляров наборов и типов записей, то есть, программист должен представлять текущее состояние в экземплярах наборов при «продвижении» по БД.
- Проблемой поддержки иерархической модели является невозможность хранения в базе данных порожденных узлов без соответствующих исходных. Следует заметить, что в процессе проектирования такая ситуация встречается достаточно часто (например, когда о родительском узле нет достоверной информации). В этом случае необходимо ввести пустой исходный узел. Наличие пустых узлов создает дополнительные трудности при проектировании некоторых приложений.
- Определенная сложность операций включения информации о новых объектах в иерархическую базу данных и удаления устаревшей (проблема сохранения целостности базы данных: при удалении узла в общем случае удаляется целое поддерево, что может привести к потере информации). В более общей формулировке: возможность потери независимости данных при реорганизации БД (разрушение целостности БД).
- Произвольность структуры сети приводит к принципиальным сложностям в реализации такой модели, поскольку в ней отсутствуют ограничения на способы соединения типов записей в схеме БД.

КОНТРОЛЬНЫЕ ВОПРОСЫ ПО ПЯТОМУ РАЗДЕЛУ

1. Дайте определение базовым понятиям реляционной модели: домен, кортеж, отношение, схема отношения, схема базы данных.
2. Каковы пользовательские представления понятиям схемы отношения и экземпляра отношения.
3. Перечислите свойства отношений.
4. Сформулируйте понятие функциональной зависимости. Определите 1НФ, 2НФ и 3НФ представления реляционной модели.
5. Дайте понятия целостности для сущностей и ссылок. Что такое внешний ключ.
6. В чем отличие в использовании аппарата реляционной алгебры и аппарата реляционного исчисления.
7. Опишите набор традиционных операций над множествами как операций реляционной алгебры.
8. Опишите набор специальных операций реляционной алгебры.
9. Дайте понятие правильно построенной формулы.
10. С чем **связывается** переменная в процессе замыкания. Зачем нужен этот процесс.
11. Перечислите достоинства и недостатки реляционных систем.
12. Сформулируйте основные понятия иерархической модели. Перечислите ее основные.
13. Каково представление концептуального и внешнего уровней иерархической модели.

14. Опишите принципы организации физического хранения записей в иерархической модели.
15. Особенности навигации в иерархических моделях.
16. Сформулируйте основные понятия сетевой модели.
17. Каково представление концептуального и внешнего уровней сетевой модели.
18. Особенности навигации в сетевых моделях.
19. Достоинства и недостатки навигационных моделей.

6. СИСТЕМА УПРАВЛЕНИЯ БАЗОЙ ДАННЫХ

6.1. Назначение и функции СУБД

Как уже было отмечено, традиционных возможностей файловых систем оказывается недостаточно для построения даже простых информационных систем (см. п.1.2.). Было выявлено несколько потребностей, которые не покрываются возможностями систем управления файлами:

- поддержание логически согласованного набора файлов;
- обеспечение языка манипулирования данными; восстановление информации после разного рода сбоев;
- реально параллельная работа нескольких пользователей.

Можно считать, что система управления данными, обладающая перечисленными свойствами, является системой управления базами данных (СУБД).

Таким образом, СУБД играет центральную роль в функционировании автоматизированного банка данных. В общем случае СУБД – это набор программных средств, выполняющих следующие основные функции:

1. Обеспечение пользователей языковыми средствами **определения** и **манипулирования** данными.
2. Обеспечение поддержки **моделей данных** пользователей, то есть средств определения логического представления данных, относящихся к некоторому приложению. Таким образом, здесь под термином **модель данных** понимаются средства определения именно **логического** представления данных.
3. Обеспечение **защиты** данных.
4. Обеспечение **целостности данных.** Вопрос о сохранении целостности данных имеет несколько аспектов, например:
   - проверка **корректности** данных связана с выполнением для новых данных, заносимых в БД, условий, связанных с ограничениями на их значения, например, год рождения служащего не может датироваться 18.. годом,
   - неправильное **управление общими данными при коллективном доступе**.
5. Управление **транзакциями**. Транзакция - это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется, и СУБД фиксирует изменения БД, произведенные искомой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается в состоянии БД.
6. Непосредственное **управление данными во внешней памяти**.
7. Управление **буферами оперативной памяти**.
8. Обеспечение **независимости** данных. СУБД должна поддерживать неизменное представление базы данных на одном уровне архитектуры системы независимо от изменения представлений базы данных на других архитектурных уровнях.
9. Обеспечение **универсальности.** СУБД должна обладать мощными средствами поддержки концептуальной модели данных для отображения пользовательских логических представлений. Другими словами, должна быть способна поддерживать разные модели данных на единой логической и физической основе.
10. Обеспечение **совместимости** и **развития**. СУБД должна сохранять работоспособность при развитии программного и аппаратного обеспечения.
11. Обеспечение **безизбыточности** данных. БД должна представлять собой единую совокупность интегрированных данных с минимальной степенью их дублирования.
12. Поддержка как **централизованных**, так и **распределенных** по информационно-вычислительной сети БД.

Рассмотрим некоторые функции СУБД и соответствующие ее компоненты несколько подробнее.

Языковая компонента. Для работы с базами данных используются специальные языки, в целом называемые **языками баз данных**. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. Подобными средствами являлись **язык определения данных (ЯОД)** и **язык манипулирования данными (ЯМД). ЯОД** служил главным образом для определения логической структуры БД, то есть той структуры БД, какой она представляется пользователям. **ЯМД** содержал набор операторов манипулирования данными, то есть операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные.

Термин **язык данных** обозначает либо оба, либо один из названных языков, а слово данные отличает язык данных от других типов языков (Си, Паскаль, Бейсик и др.). Однако язык данных может быть включен в универсальный язык в виде набора операторов. В этом случае универсальный язык программирования и язык данных называют, соответственно, **включающим языком** и **подъязыком данных**. В свою очередь автономный язык данных называют **языком запросов**. В современных СУБД обычно поддерживается единый интегрированный язык запросов, содержащий все необходимые средства для работы с БД, начиная от ее создания и обеспечивающий базовый пользовательский интерфейс с базами данных. Стандартным языком наиболее распространенных в настоящее время реляционных СУБД является язык SQL (Structured Query Language). Поскольку СУБД полностью обеспечивает пользовательский интерфейс с БД, остановимся подробнее на основных функциях реляционной СУБД, поддерживаемых на «языковом» уровне (т.е. функциях, поддерживаемых при реализации интерфейса SQL).

1. Прежде всего язык SQL сочетает средства ЯОД и ЯМД, то есть позволяет **определять схему** реляционной БД и **манипулировать данными**. При этом именование объектов БД (для реляционной БД - именование таблиц и их столбцов) поддерживается на языковом уровне в том смысле, что компилятор языка SQL производит преобразование имен объектов в их внутренние идентификаторы на основании специально поддерживаемых служебных таблиц-каталогов. Внутренняя часть СУБД (ядро) вообще не работает с именами таблиц и их столбцов.

2. Язык SQL содержит специальные средства определения **ограничений целостности** БД. Ограничения целостности хранятся в специальных таблицах-каталогах, и обеспечение контроля целостности БД производится на языковом уровне, то есть при компиляции операторов модификации БД компилятор SQL на основании имеющихся в БД ограничений целостности генерирует соответствующий программный код.

3. Специальные операторы языка SQL позволяют определять так называемые **представления** БД, фактически являющиеся хранимыми в БД запросами (результатом любого запроса к реляционной БД является таблица) с именованными столбцами. Для пользователя представление является такой же таблицей, как любая базовая таблица, хранимая в БД, но с помощью представлений можно ограничить или наоборот расширить видимость БД для конкретного пользователя. Поддержание представлений производится также на языковом уровне. Очевидно, что представление можно рассматривать как внешнюю схему.

4. **Авторизация доступа** к объектам БД производится на основе специального набора операторов SQL. Идея состоит в том, что для выполнения операторов SQL разного вида пользователь должен обладать различными полномочиями. Пользователь, создавший таблицу БД, обладает полным набором полномочий для работы с этой таблицей. В их число входит полномочие на передачу всех или части полномочий другим пользователям, включая полномочие на передачу полномочий. Полномочия пользователей описываются в специальных таблицах-каталогах, контроль полномочий поддерживается на языковом уровне.

Поддержка целостности. Понятие транзакции необходимо для поддержания логической целостности БД. Если вспомнить пример информационной системы отдела кадров с файлами **СОТРУДНИКИ** и **ОТДЕЛЫ** (см. п.5.2.), то единственным способом не нарушить целостность БД при выполнении операции приема на работу нового сотрудника будет объединение элементарных операций над файлами **СОТРУДНИКИ** и **ОТДЕЛЫ** в одну

транзакцию. Таким образом, поддержание механизма транзакций - обязательное условие даже однопользовательских СУБД. Но понятие транзакции гораздо существеннее во многопользовательских СУБД. При коллективном режиме возможно параллельное использование общих физических данных. Это требует поддержки согласованности одних и тех же данных при работе различных пользователей. Типичные примеры рассогласования при неправильном управлении одновременными модификациями – продажа большего числа товаров, чем имеется на складе, продажа нескольких билетов на одно место. Другими словами, суть сохранения целостности БД – ее адекватность текущему состоянию предметной области. То свойство, что каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения, делает очень удобным использование понятия транзакции как единицы активности пользователя по отношению к БД. При соответствующем управлении параллельно выполняющимися транзакциями со стороны СУБД каждый пользователь может в принципе ощущать себя единственным пользователем СУБД.

Защита данных может рассматриваться с нескольких точек зрения, например:

- защита для обеспечения **секретности** связана с разрешением использования тех или иных данных, хранящихся в системе, только пользователям, имеющим на это право,
- защита от **несанкционированного доступа**, как правило, разрешает неограниченный доступ к данным из БД по чтению (выборке), но запрещает обновление и запись новых данных в “чужие” для данного пользователя разделы БД,
- защита от **разрушения** БД **при сбоях оборудования** предполагает наличие процедур копирования и восстановления (функция **журнализации**).

Журнализация. Одно из основных требований к СУБД - надежное хранение данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя.

Обычно рассматриваются два возможных вида аппаратных сбоев:

- так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания),
- и жесткие сбои, характеризуемые потерей информации на носителях внешней памяти.

Примерами программных сбоев могут быть:

- аварийное завершение работы СУБД (из-за ошибки в программе или некоторого аппаратного сбоя),
- аварийное завершение пользовательской программы, в результате чего некоторая транзакция остается незавершенной.

Первую ситуацию можно рассматривать как особый вид мягкого аппаратного сбоя; при возникновении последней требуется ликвидировать последствия только одной транзакции.

В любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Другими словами, поддержание надежного хранения данных в БД требует **избыточности** хранения данных, причем та их часть, которая используется для восстановления, должна храниться особо надежно. Наиболее распространенный метод поддержания такой избыточной информации - ведение **журнала изменений** БД.

Журнал - это особая часть БД, недоступная пользователям СУБД, в которую поступают записи обо всех изменениях основной части БД. В разных СУБД изменения БД фиксируются на разных уровнях: иногда запись в журнале соответствует некоторой логической операции изменения БД (например, операции удаления строки из таблицы реляционной БД), а порой запись соответствует минимальной внутренней операции модификации страницы внешней памяти. В некоторых системах одновременно используются оба подхода.

Во всех случаях придерживаются стратегии «упреждающей» записи в журнал протокола. Кратко, эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части БД.

Самая простая ситуация восстановления - индивидуальный **откат транзакции**.

Физически журнал обычно представляет собой чисто последовательный файл с записями переменного размера, которые можно просматривать в прямом или обратном порядке. Операции обмена производятся стандартными порциями (страницами) с использованием буфера оперативной памяти (см. пп.7.2.1. и 7.3.). Структура (и тем более, смысл) журнальных записей известна только компонентам СУБД, ответственным за журнализацию и восстановление. Поскольку содержимое журнала является критичным при восстановлении базы данных после сбоев, к ведению файла журнала предъявляются особые требования по части надежности. В частности, обычно стремятся поддерживать две идентичные копии журнала на разных устройствах внешней памяти.

Очевидно, что в многопользовательском режиме работы вопросы журнализации занимают одно из центральных мест при функционировании СУБД.

Управление данными во внешней памяти. Эта функция включает обеспечение необходимых структур внешней памяти (индексов) как для хранения непосредственных данных, входящих в БД, так и для служебных целей, например, для убыстрения доступа к данным (см. пп. 7.2.3. - 7.2.7.).

Управление буферизацией. СУБД обычно работают с БД значительного размера; по крайней мере, этот размер обычно существенно превышает доступный объем оперативной памяти. Понятно, если при обращении к любому элементу данных будет производиться обмен с внешней памятью, то вся система будет работать со скоростью устройства внешней памяти. Единственным же способом реального увеличения этой скорости является **буферизация данных в оперативной памяти**. Поэтому в развитых СУБД поддерживается собственный набор буферов оперативной памяти с собственной дисциплиной замены буферов. При управлении буферами основной памяти приходится разрабатывать и применять согласованные алгоритмы **буферизации**, **журнализации** и **синхронизации**.

Независимость данных. Важнейшими аспектами независимости данных являются:

- **Логическая независимость данных**. Способность механизмов СУБД поддерживать изменения логического представления данных таким образом, чтобы при этом было возможно сохранить неизменными пользовательские представления данных. В терминологии архитектурной модели ANSI/X3/SPARC это означает возможность вариации концептуальной схемы базы данных без необходимости изменений внешних схем.
- **Физическая независимость данных.** Способность механизмов СУБД поддерживать изменения физического представления данных таким образом, чтобы при этом было возможно сохранить неизменным логическое их представление. В терминологии архитектурной модели ANSI/X3/SPARC это означает возможность вариации внутренней схемы базы данных без необходимости изменений концептуальной схемы.
-

6.2. Типовая организация СУБД и упрощенная схема работы

Очевидно, что организация типичной СУБД и состав ее компонентов соответствует рассмотренному набору функций. Логически в современной реляционной СУБД можно выделить:

- внутреннюю часть – **ядро СУБД** (часто его называют DBE - Data Base Engine),
- **компилятор языка БД** (обычно, SQL),
- подсистему **поддержки времени выполнения**,
- **набор утилит**.

В некоторых системах эти части выделяются явно, в других - нет, но логически такое разделение можно провести во всех СУБД.

Ядро СУБД отвечает за управление данными во внешней памяти, управление буферами оперативной памяти, управление транзакциями и журнализацию. Соответственно можно

выделить такие компоненты ядра, как **менеджер данных**, **менеджер буферов**, **менеджер транзакций** и **менеджер журнала**. Функции этих компонентов взаимосвязаны, и для обеспечения корректной работы СУБД все эти компоненты должны взаимодействовать по тщательно продуманным и проверенным протоколам. Ядро СУБД обладает собственным интерфейсом, не доступным пользователям напрямую и используемым в программах, производимых компилятором SQL (или в подсистеме поддержки выполнения таких программ), и утилитах БД. Ядро СУБД является основной резидентной частью СУБД.

Основная функция **компилятора языка** БД - компиляция операторов языка БД в некоторую выполняемую программу. Основной проблемой реляционных СУБД является то, что языки этих систем (а это, как правило, SQL) являются непроцедурными, то есть в операторе такого языка специфицируется некоторое действие над БД, но эта спецификация не процедура, она лишь описывает в некоторой форме условия совершения желаемого действия. Поэтому компилятор должен решить, каким образом выполнять оператор языка, прежде чем произвести программу. Результатом компиляции является **выполняемая программа**, представляемая в некоторых системах в машинных кодах, но более часто в выполняемом внутреннем машинно-независимом коде. В последнем случае реальное выполнение оператора производится с привлечением **подсистемы поддержки времени выполнения**, представляющей собой, по сути дела, **интерпретатор** этого внутреннего языка.

Наконец, в отдельные **утилиты БД** обычно выделяют такие процедуры, которые слишком накладно выполнять с использованием языка БД, например, загрузка и разгрузка БД, сбор статистики, заполнение БД реальными данными, глобальная проверка целостности БД и т.д. Утилиты программируются с использованием интерфейса ядра СУБД, а иногда даже с проникновением внутрь ядра.

Обобщенную структуру СУБД можно представить схемой на рис. 18.

Упрощенно основные этапы реализации СУБД запроса приложения представлены на рис.19.

1. Прикладная программа выдает СУБД запрос на доступ к БД.

2. СУБД анализирует запрос и определяет типы необходимых логических записей, содержащих запрашиваемые данные.

3. СУБД определяет хранимые записи, которые нужно предоставить прикладной программе.

4. СУБД выдает ОС команды поиска и обработки хранимых записей.

5. ОС интерпретирует команды СУБД, взаимодействует с внешним запоминающим устройством, обнаруживая в БД требуемые записи.

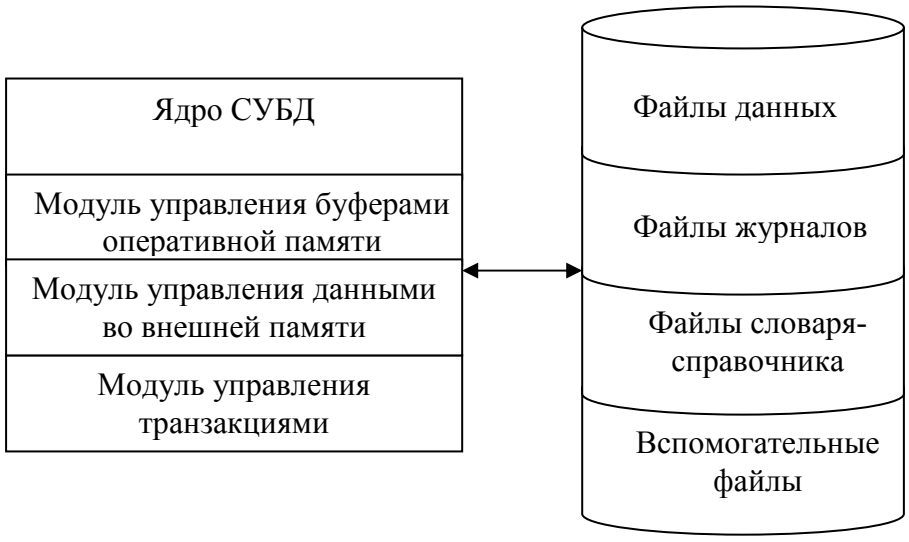


Рис.18. Обобщенная структура СУБД

6. Отобранные записи передаются в системные буферы.

7. СУБД выделяет **логические записи**, которые необходимо передать приложению.

8. СУБД передает выделенные данные из системных буферов в рабочую область прикладной программы, предварительно преобразовав их в формат, указанный в запросе.

9. СУБД передает прикладной программе информацию о результатах обработки запроса.

10. При успешном завершении СУБД обработки запроса приложение использует полученные данные для дальнейшей работы, в противном случае запрос должен быть повторен.

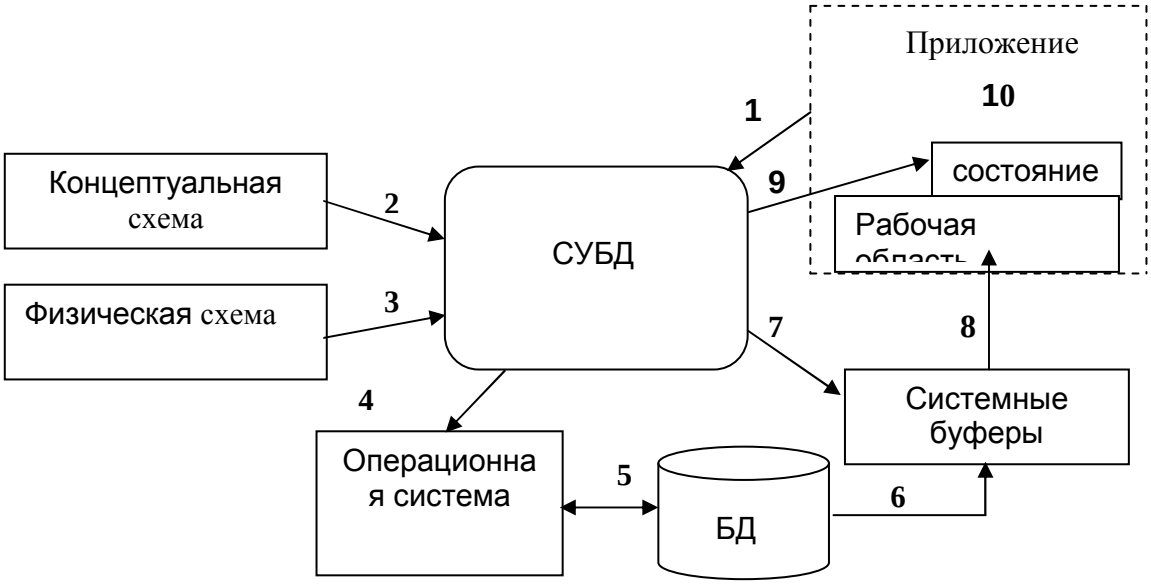


Рис.19. Схема реализации запроса к СУБД

Замечание. Понятия концептуальной и внешней схем для навигационных моделей достаточно очевидны. Для реляционной модели внешнюю схему можно ассоциировать со схемой отношения, являющегося результатом выполнения запроса. А вот понятие концептуальной схемы как результата реализации соответствующей модели инструментарием СУБД заменяется понятием даталогической (логической) схемы базы данных как взаимосвязанного набора схем отношений, формирующих базу.

Замечание. Для более эффективной работы с внешней памятью СУБД может взять управление на себя, минуя систему управления файлами ОС. Более подробно этот материал рассмотрен в п. 7.3..

Взаимодействие компонентов информационной системы между собой, предметной областью и ее моделью может быть представлена схемой на рис.20.

КОНТРОЛЬНЫЕ ВОПРОСЫ ПО ШЕСТОМУ РАЗДЕЛУ

1. Перечислите основные функции СУБД.
2. Опишите языковую компоненту СУБД.
3. Определите понятие транзакции. Назначение и суть механизма журнализации.
4. Какие основные аспекты сохранения целостности учитываются при функционировании СУБД.
5. Какие основные аспекты защиты данных должны учитываться при функционировании СУБД.
6. Сформулируйте понятия логической и физической независимости данных.
7. Приведите обобщенную схему СУБД.
8. Приведите упрощенную схему функционирования СУБД.

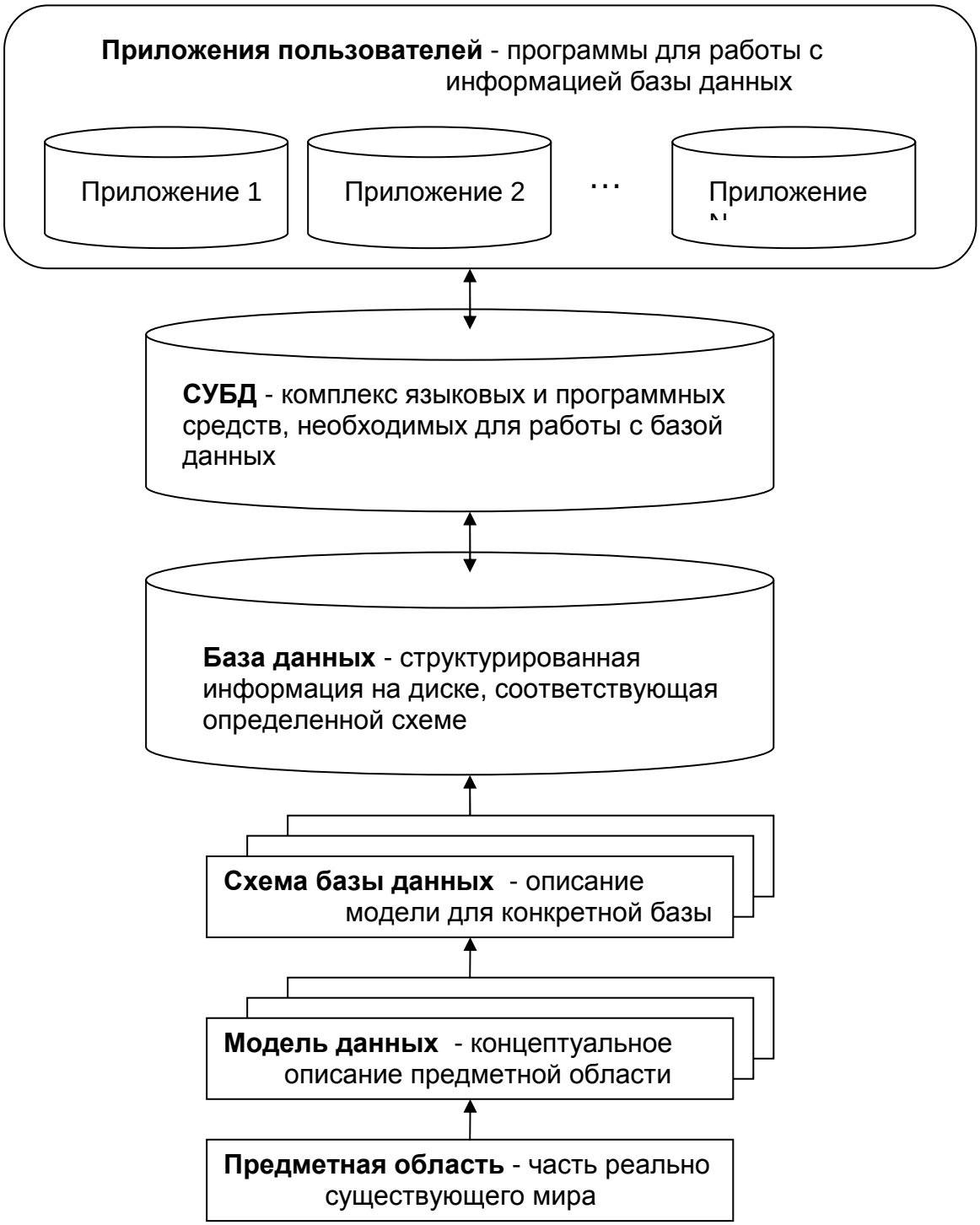


Рис.20 Взаимодействие компонентов информационной системы

7. ОСНОВЫ ФИЗИЧЕСКОГО ПРОЕКТИРОВАНИЯ

7.1. Файловые и страничные системы хранения информации

Физическая организация данных оказывает основное влияние на эксплуатационные характеристики проектируемой базы данных, так как именно на этом уровне осуществляется ее привязка к физической памяти.

Знание того, как большинство СУБД физически хранят данные во внешней памяти, представление о параметрах этого хранения и соответствующих методах доступа может очень помочь при проектировании баз данных, обладающих заданной производительностью.

Возможность работы со множеством различных логических представлений на основе единого физического представления составляет существо концепции БД. Согласно этой концепции, отделение логического представления от физического является главной гарантией независимости данных.

Поскольку исторически первыми системами хранения и доступа были файловые структуры и системы управления файлами (СУФ), входящие в состав ОС, при работе СУБД резонно было использовать средства СУФ для работы с файлами, в которые отображались хранимые данные одной или нескольких баз данных.

Однако универсальные механизмы управления файловыми структурами, разработанные для решения традиционных задач по работе с файлами, становятся крайне неэффективными с ростом объемов хранимых данных, то есть в среде специфических задач хранения и управления данными СУБД. В связи с этим произошел постепенный переход от использования базовых файловых структур к непосредственному управлению размещением хранимых данных во внешней памяти самой СУБД (управление страничными системами).

Таким образом, можно выделить два механизма, используемых СУБД при управлении хранимыми данными:

- управление **файловыми системами**,
- **бесфайловое** управление внешней памятью.

Следует отметить, что при этом механизмы, применяемые в файловых системах, во многом перешли и в страничные системы хранения информации.

Итак, пространство для базы данных на внешних устройствах распределяется большими фрагментами:

1. **Файлами**. В этом случае доступ к диску осуществляется операционной системой, что дает определенные преимущества, например, работа с файлами средствами ОС.

2. **Разделами диска (страницами).** В этом случае с внешним устройством работает сама СУБД. При этом достигается более высокая производительность. Диски особенно эффективны для ускорения операций записи данных.

Определим несколько основных понятий, связанных с проблемами проектирования физического уровня базы данных.

Хранимая запись. Под хранимой записью будем понимать запись какого-либо типа, хранимую во внешней памяти. Очевидно, по аналогии с логическими записями, можно говорить о **типе** хранимой записи и **экземпляре** хранимой записи некоторого типа.

Метод доступа определим как совокупность технических и программных средств, обеспечивающих возможность хранения и выборки данных (представленных хранимыми записями), расположенных во внешней памяти.

В формировании метода доступа важны два компонента:

- структура памяти,
- механизм поиска.

По **структуре** различают два типа внешней памяти:

- память с последовательным доступом (например, магнитные ленты),
- память с прямым доступом (например, дисковая память).

Механизм поиска задается алгоритмом, определяющим специфический путь доступа, который возможен в рамках конкретной структуры памяти, и количество шагов вдоль этого пути для нахождения искомых данных (хранимых записей).

Очевидно, что выбор метода доступа регулируется соображениями эффективности работы с БД (по использованию ресурсов - памяти и времени). Как правило, современные СУБД используют несколько методов доступа к хранимым записям.

Таким образом, в общем случае процесс **проектирования физической БД** (**физическое проектирование**) – это процесс создания по заданной логической структуре ее эффективной физической реализации, исходя из информационных требований конечных пользователей. Выделим в процессе физического проектирования следующие задачи:

- определение методов доступа,
- распределение и управление внешней памятью.

7.2. Файловые структуры. Классификация методов доступа

Прежде всего, дадим несколько определений.

Файл. Именованная структура данных, представляющая множество аналогично построенных (имеющих одинаковый формат) экземпляров хранимых записей одного типа.

Физическая база данных (БД на физическом уровне). Совокупность совместно хранимых взаимосвязанных данных, представляющая один или несколько типов хранимых данных, упакованных в один или несколько файлов.

Блок записей. Группа последовательно хранимых записей, которая используется при операциях обмена как **единое целое**. Обмен с внешней памятью блоками записей позволяет сократить время доступа, однако требует большего размера буфера в оперативной памяти. Вопрос о выборе размера блока будет рассмотрен ниже.

В каждой СУБД по-разному организовано хранение и доступ к данным; однако, существуют некоторые файловые структуры, использующиеся для хранения информации во внешней памяти, которые имеют общепринятые способы организации и широко применяются во всех СУБД. На рис.21. приведена возможная классификация подобных структур.

7.2.1. Способы последовательной организации

Физически последовательная организация

На рис.22. представлена простейшая для анализа организация, при которой записи размещаются в смежных областях (блоках):

Предполагается, что записи имеют постоянный размер и могут быть как неупорядоченными, так и упорядоченными по возрастанию или убыванию значений первичного ключа.

Последовательный метод доступа, заключающийся в последовательном переборе записей (с начала или конца файла) используется, в подавляющем большинстве случаев, для реализации операторов поиска всех или большинства записей. Однако, возможность упорядочивания записей позволяет в рамках физически-последовательной организации осуществлять достаточно эффективно и поиск уникальных записей, используя, например, алгоритм двоичного поиска.

Основной **недостаток** физически-последовательной организации связан со сложностью расширения файла.

Связная последовательная организация

Позволяет размещать записи в несмежных участках памяти (рис.23.).

Таким образом, участки файла образуют список. Такая организация существенно облегчает процесс модификации файла (рис.24.).

На рисунке пунктирной линией обозначена связь, которая разрывается при добавлении новой записи с ключем 04.

Список может быть двусвязным (рис.25.). При такой организации появляется возможность эффективной реализации поиска предшествующей записи.

Главным **недостатком** связанной последовательной организации является большая сложность при необходимости упорядочивания записей.

7.2.2. Прямые методы доступа. Хеширование

Файлы с прямым методом доступа обеспечивают механизм поиска записей по первичному ключу, отличный от последовательного просмотра. Прямой доступ к записи реализуется с помощью какого-либо метода отображения значения ключа в адрес. Наиболее распространенными методами отображения являются **хеширование** и **индексирование**.

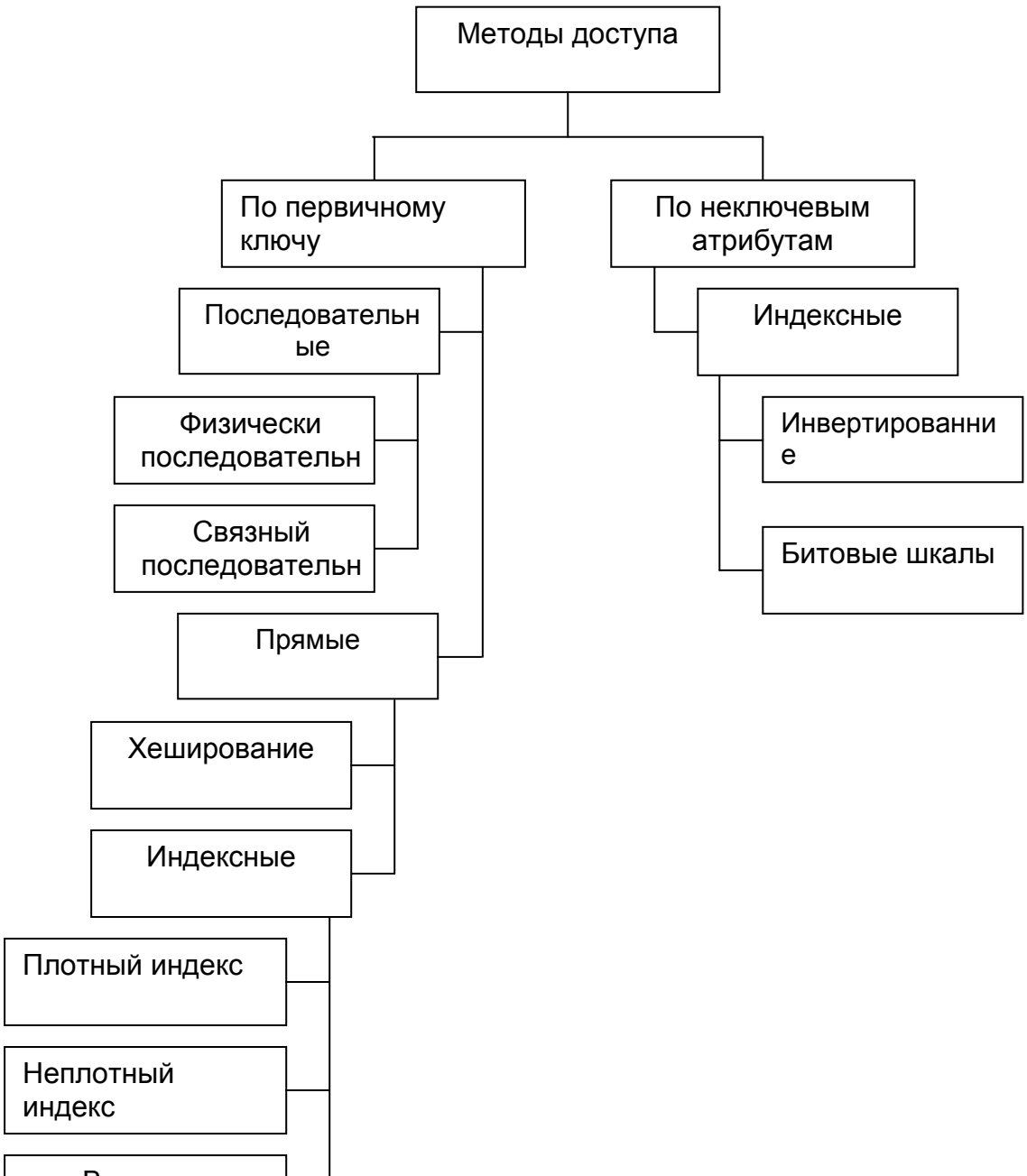


Рис.21. Классификация методов доступа

01, **02**, **03** – значения первичного ключа.

Рис.22. Пример физически последовательной организации

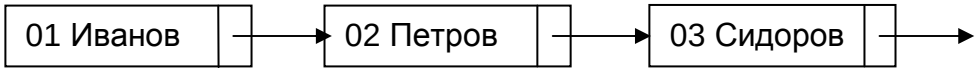


Рис.23. Пример связной последовательной организации

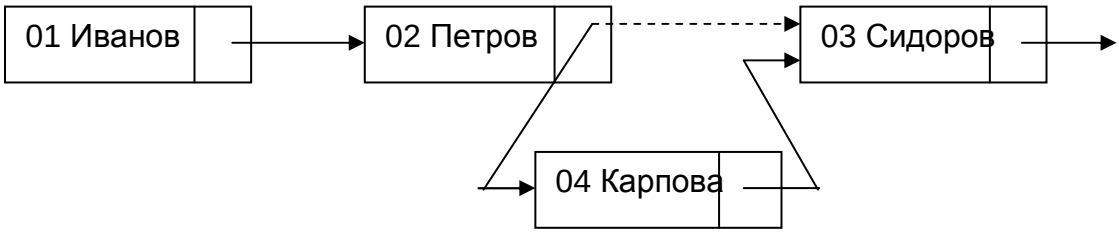


Рис.24. Механизм добавления записи в связную последовательную структуру

Рис.25. Двусвязная последовательная организация

Общей идеей методов **хэширования** является применение к значению ключа некоторой **функции свертки** (**Хэш-функции**), вырабатывающей значение меньшего размера. Свертка значения ключа затем используется для доступа к записи. В самом простом случае свертка ключа используется как адрес в таблице, содержащей ключи и записи.

В реальности записи файла разделяются между **участками**, каждый из которых содержит один или несколько блоков памяти. В этом случае хеширование обеспечивает прямую адресацию записи путем преобразования значения первичного ключа в абсолютный или относительный адрес **участка**.

Пусть v есть значение ключа записи и h – Хеш-функция. Тогда h (v) - адрес участка, в котором должна находиться искомая запись (в том случае, если она присутствует вообще). Общая схема организации хешированного файла представлена на рис.26.

Проблема синонимов

Очевидно, что при реализации Хеш-функции отношения 1:1 между значениями ключей и номерами участков размер справочника участков становиться неприемлемо большим, а величина самих участков неприемлемо малой, что приводит к нерациональному расходу памяти.

Реальным выходом из этой ситуации является принятие соглашения, при котором в общем случае Хеш-функции осуществляет отображение типа 1:M; однако в этом случае фиксируется эффект возникновения **синонимов**, когда записи с различными значениями ключей направляются для хранения в один участок, что приводит, в конечном счете, к различной степени загруженности участков.

И, если при использовании связанной последовательной организации блоков внутри участков (именно такая организация представлена на рис.26.) наличие синонимов приводит, в основном, только к различию во времени поиска в пределах отдельных участков, то при использовании физически последовательной организации могут возникнуть дополнительные проблемы, связанные с необходимостью введения области переполнения (рис.27.).

Очевидно, что возникновение слишком большого количества цепочек переполнения ведет к потере главное преимущества хэширования - доступа к записи практически всегда за одно обращение. Переход на использование новой хэш-функции (со значением свертки большего размера) требует перестройки всех участков основного файла, что в случае баз данных являются абсолютно неприемлемым. Поэтому обычно вводят промежуточные таблицы-справочники, содержащие значения ключей и адреса записей, а сами записи хранятся отдельно. Тогда при переполнении справочника требуется только его переделка, что вызывает меньше накладных расходов.

Замечание. Конечно, структура самой области переполнения может быть связанной последовательной или физически последовательной.

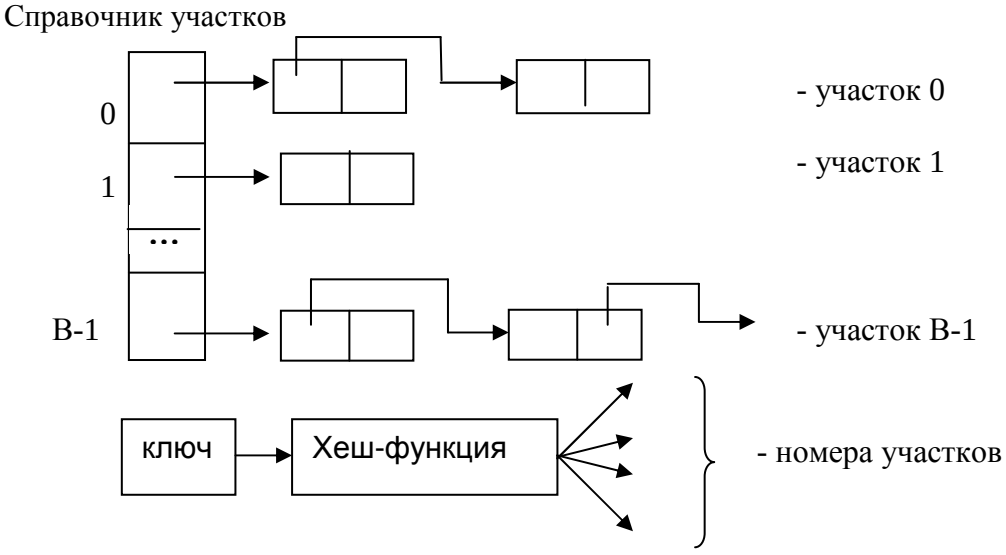


Рис.26. Общая схема организации процесса хеширования

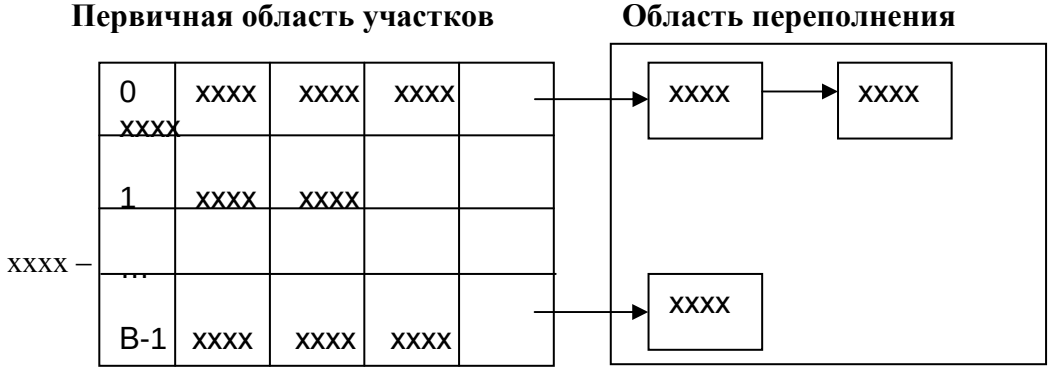


Рис.27. Схема использования физически последовательных участков при хешировании

Итак, ключевым моментом выбора Хеш-функции является требование равномерного распределения синонимов по участкам.

Пример. Пусть исходные записи имеют следующие значения ключей:

36	26	12	5	95

Вариант 1. Значением Хеш-функции является выражение: <значение_ключа> (mod10) +1
Распределение синонимов по участкам примет вид:

№ уч-ка	Записи
1	
2	
3	12
4	
5	
6	5 95
7	36 26
8	
9	
10	

Вариант 2. Значение Хеш-функции - это выражение: {сумма цифр ключа} (mod10) +1
Распределение синонимов по участкам примет вид:

№ уч_ка	Записи
1	
2	
3	
4	12
5	95
6	5
7	
8	
9	26
10	36

7.2.3. Прямые методы доступа. Классификация методов индексирования

Индекс – зто вспомогательная структура данных в, файловых системах, среде хранения базы данных, средах хранилищ данных других типов, служащая для:

- повышения производительности выполнения операций поиска данных, удовлетворяющих заданному критерию,
- обеспечения обработки данных в соответствии с порядком значений ключа и/или вычисления различных статистических характеристик, зависящих только от значений ключей.

Следует заметить, что повышение производительности системы хранения данных при использовании техники индексирования достигается за счет выделения дополнительных ресурсов памяти для хранения индексов, а также вычислительных ресурсов для их модификации при выполнении операций вставки, удаления и обновления данных.

Используются различные подходы к организации индексов, ориентированные на поддержку различных операций доступа к данным в среде хранения. Классификацию индексов можно провести по разным основаниям. При этом различают:

1. по **уровням**

- **одноуровневые** индексы, в записях (**статьях**) которых непосредственно используются адреса индексируемых объектов и пространств памяти базы данных,
- **многоуровневые** индексы, организованные в виде **дерева страниц памяти**; в статьях страниц-листьев этого дерева содержатся указатели самих индексируемых объектов данных, а в страницах (блоках) более высоких уровней – указатели номеров страниц более низкого уровня, где нужно продолжать поиск.

2. по **значению ключей**

- **первичный** индекс файла по значению **первичного ключа**; каждой статье такого индекса соответствует единственная (уникальная) запись в файле,
- **вторичный** индекс файла по значению вторичного ключа; поскольку значения вторичного ключа не являются уникальными, каждой статье вторичного индекса может соответствовать, вообще говоря, несколько записей файла.

3. по **способу организации**

- **плотный** (или **полный**) индекс, включающий явным образом статьи для всех актуальных (имеющихся на данный момент) значений ключа индексирования,
- **неплотный** (или **неполный**, **разреженный**) индекс, содержащий статьи не для всех актуальных значений ключа индексирования, а только для их диапазонов; тем самым индекс указывает не точный адрес данных в среде хранения, а некоторую достаточно ограниченную область (например, страницу), где они содержатся,
- индекс со структурой **B-дерева**; многоуровневый индекс, в котором страницы верхних уровней являются разреженными индексами страниц следующего нижнего уровня, а страницы самого нижнего уровня (листья дерева) образуют, по существу, плотный индекс для индексируемого множества записей,
- индекс на **битовых шкалах** (**битовый** индекс).

Одна организация индекса отличается от другой, главным образом, в способе поиска ключа с заданным значением. В следующих пунктах этого раздела рассматриваются основные методы доступа с использованием **первичного** индекса.

7.2.4. Доступ с полным (плотным) индексом

Метод доступа с **полным** (**плотным**) индексом (или **индексно-произвольный** метод) представляет собой такую организацию файла, при которой для каждого экземпляра записи в файле предусмотрен соответствующий элемент индекса (рис. 28.). Этот элемент включает значение ключа записи и указатель на блок, содержащий искомую запись. Обычно для ускорения поиска в индексе его элементы упорядочиваются.

Достоинством данного метода доступа является произвольное расположение записей данных в основном файле, что обеспечивает их физическую независимость при хранении.

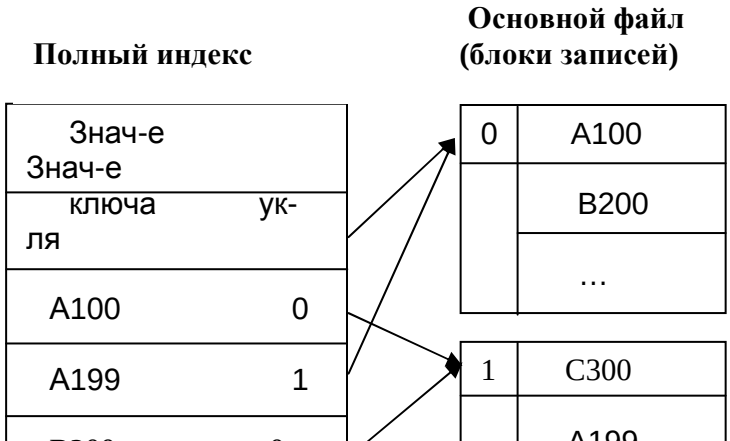


Рис.28. Схема организации метода доступа с плотным индексом

Основной **недостаток** проявляется в тех случаях, когда:

1. Выдается оператор выборки всех или большинства записей, и при этом требуется упорядочивание полученных данных.

2. Сложность процесса обновления основного файла, особенно при добавлении в него новых записей (требуется перестройка индекса).

7.2.5. Доступ с неплотным индексом

Доступ с неплотным индексом (индексно-последовательный метод доступа) строится на основе физически упорядоченного по возрастанию значения ключей последовательного файла и совокупности пронумерованных индексных элементов (индексе), каждый из которых содержит ключ подобно записям основного файла; элементы в индексе упорядочиваются по возрастанию значений ключей. Значение ключа в индексном элементе представляет наибольший (или наименьший) из значений ключей записей, входящих в блок основного файла с номером, совпадающим с номером индексного элемента.

Структура индексно-последовательного файла представлена на рис.29.

Алгоритм поиска при данной организации файла очевиден и включает два этапа:

1. Поиск в индексе элемента, указывающего на блок, в котором должна находиться искомая запись, используя максимальное (или минимальное) значение ключей записей, размещенных в блоках основного файла.

2. Последовательный просмотр записей найденного блока.

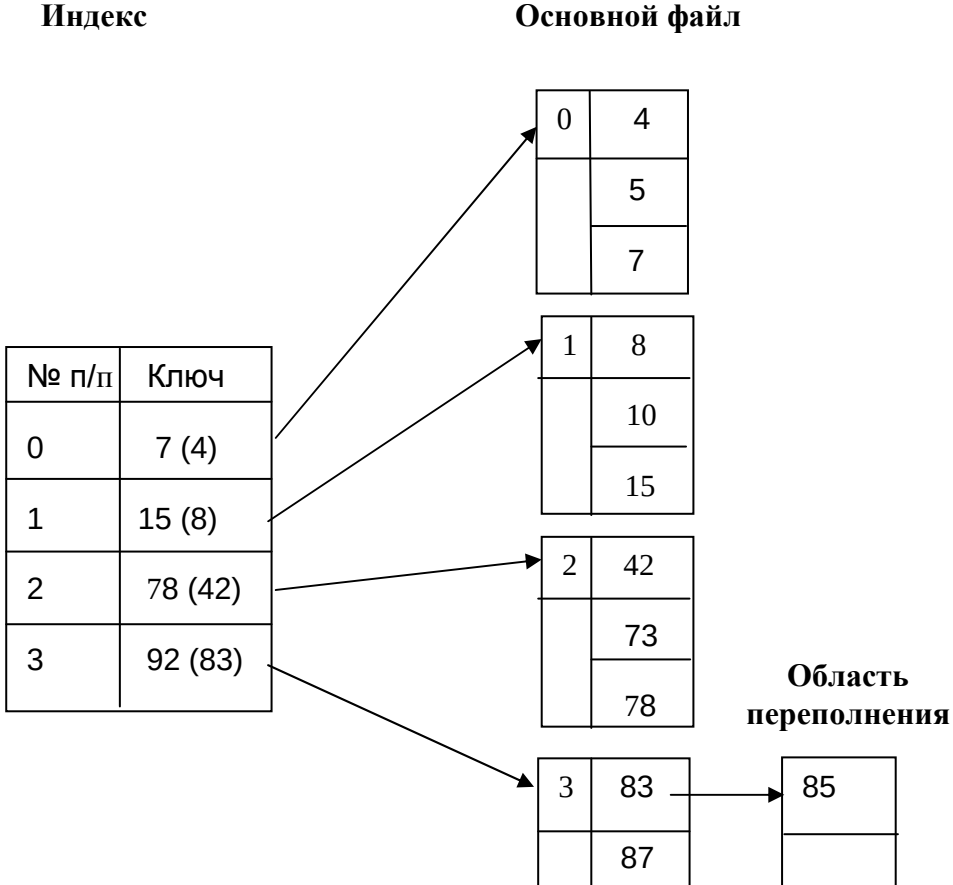


Рис.29. Схема организации индексно-последовательного файла

Таким образом, к записям индексно-последовательного файла с помощью индекса осуществляется **прямой доступ** к блоку (странице), включающему требуемую запись, и **последовательный** доступ в соответствии с упорядоченностью записей по этому ключу индексирования.

Использование индексно-последовательной организации наиболее эффективно, когда модификация исходного файла не предполагает его расширения. В противном случае, как праавило, необходимо введение области переполнения, существование которой принципиально ломает простоту алгоритм поиска, присущую индексно-последоватльному методу доступа

Сравнение метода полного индекса с индексно-последовательной организацией

1. В методе полного индекса не предусмотрена обработка переполнения; вместо этого всякий раз при включении новой записи в основной файл выполняется переупорядочивание индекса.
2. При отсутствии переполнения поиск всех записей в обоих методах имеют одинаковую производительность.
3. В обоих методах достаточно эффективно выполняется операция поиска записей с уникальными ключами.
4. Вследствие физически последовательного размещения записей операции типа ПОЛУЧИТЬ СЛЕДУЮЩУЮ и ПОЛУЧИТЬ ПРЕДЫДУЩУЮ выполняются гораздо эффективнее в методе неплотного индекса.
5. Добавление, а так же изменение значений первичных ключей в основном файле в обоих методах трудоемко, поскольку, как правило, влечет обновление индекса.

7.2.6. Организация индексов в виде B-деревьев

Построение B-деревьев связано с простой идеей создания индекса над уже построенным индексом. Например, при построении индексно-последовательного файла сама индексная область может быть рассмотрена как основной файл, над которым можно построить неплотный индекс, и так далее, пока индексную область не будет представлять только один блок.

В общем случае, может быть построено некоторое дерево, каждый родительский блок которого связан с одинаковым количеством подчиненных блоков, количество которых равно количеству индексных записей, размещенных в родительском блоке. Число шагов при этом для поиска любой записи основного файла одинаково и равно количеству уровней в дереве.

Такие деревья называют **сбалансированными** (путь от корня до любого листа одинаков) Таким образом, термин **B**-дерево происходит от английского **balance** (баланс). Пример организации B-дерева представлен на рис.30.

Использование техники B-деревьев в настоящее время, скорее всего, является наиболее популярным подходом к организации индексов в базах данных. С точки зрения **внешнего логического** представления, B-дерево - это сбалансированное, сильно ветвистое дерево во

внешней памяти. Ветвистость дерева - это свойство каждого узла дерева ссылаться на большое число узлов-потомков. С точки зрения **физической организации**, B-дерево представляется как мультисписочная структура страниц внешней памяти, то есть каждому узлу дерева соответствует блок внешней памяти (**страница**, см. п.7.3.2.). Внутренние и листовые страницы обычно имеют разную структуру.

Поиск в B-дереве - это прохождение от корня к листу в соответствии с заданным значением ключа.

B-деревья универсальны и обеспечивают хорошую скорость доступа как при просмотрах по диапазонам, так и при выборке единичной записи по значению ключа, однако характеризуются относительно большим объемом памяти для хранения и затратами на поддержание в актуальном состоянии, включающими обычно балансировку дерева.

7.2.7. Инвертированный файл (доступ по неключевым атрибутам)

Рассмотренные выше методы доступа осуществляют поиск записей по значению первичного ключа. Однако выполнение операции типа ПОЛУЧИТЬ НЕКОТОРЫЕ предполагает указание критерия отбора экземпляров записей по значениям одного или нескольких неключевых атрибутов (часто называемых **вторичным**и ключами). Заметим, что соответствующий запрос на языке SQL включает предложение WHERE.

Основным методом организации поиска по значениям неключевых атрибутов является **инвертирование** файла.

Инвертированный файл – это файл, для которого поддерживается **плотный вторичный** индекс по значениям некоторого поля содержащихся в нем записей. Инвертированный файл состоит из многоуровневого индекса и набора **списков указателей доступа**, обеспечивающих доступ к записям данных.

Список указателей доступа представляет собой физически последовательный список указателей на записи, содержащие идентичные значения соответствующего атрибута.

В наиболее распространенном варианте используется двухуровневый индекс. Схема организации инвертированного файла представлена на рис.31.

Индекс 2-го ур-ня **Индекс 1-го ур-ня** **Основной файл**

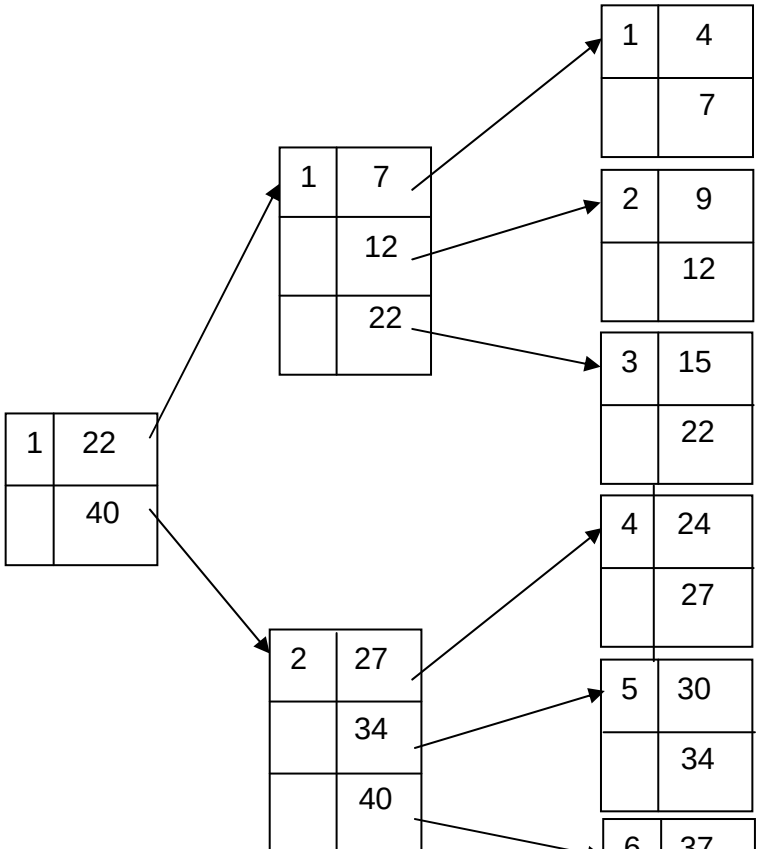


Рис.30. Схема организации B-дерева

Частично инвертированный файл инвертируется по выборочному количеству неключевых атрибутов.

Полностью инвертированный файл инвертируется по каждому неключевому атрибуту, то есть это файл, для которого поддерживается плотный вторичный индекс по значениям каждого поля содержащихся в нем записей, не являющегося первичным ключем.

Выше рассмотрено построение индексов из одиночных атрибутов (типов элементов данных). Очевидно, что можно использовать **составные индексы**, элементы которых соответствуют значениям **сцепленных** элементов различных типов.

7.2.8. Использование битовых шкал

Этот вид индекса определяется на множестве записей, в котором **каждому свойству записей** ставится в соответствие **битовая строка** (статья индекса). Количество битов в строке равно общему количеству записей файла, и соответствующий данной записи бит принимает единичное значение тогда и только тогда, когда запись обладает рассматриваемым свойством. Очевидно, что при этом записи файла СУБД пронумеровываются, так что соответствие между битами строки и записями устанавливается по этим внутрисистемным номерам. Свойство может иметь различный смысл, например, может означать, что некоторый атрибут записи имеет данное значение.

Пример 1. Пусть основной файл содержит следующие записи:

01	S01	Иванов	Томск
02	S02	Петров	Томск
03	S03	Сидоров	Кемерово
04	S04	Кузнецов	Барнаул
05	S05	Петров	Омск

Значения элементов данных в полях записей отображают значения:
Поле1 – внутрисистемных номеров записей,
Поле2 – атрибута - первичного ключа,
Поле3 – атрибута – **Фамилия_поставщика**,
Поле4 - **Город**, где живет поставщик.
Сформируем битовые строки, соответствующие следующим свойствам:

Свойство	**Статья (битовая строка)**
Фамилия_поставщика = “Петров”	0 1 0 0 1
Город = ”Томск”	1 1 0 0 0

Такая организация индекса имеет два важных достоинства:

1. Если индексирование осуществляется по некоторому ключу и мощность множества значений этого ключа невелика, для хранения битового индекса требуется сравнительно небольшой объем памяти, а в некоторых случаях он может полностью помещаться в оперативную память. Это обстоятельство обеспечивает дополнительное существенное повышение производительности операций доступа к данным за счет уменьшения количества операций ввода-вывода, связанных с обращением к индексу.

2. Битовое представление индекса позволяет с высокой эффективностью выполнять операции выборки (селекции) записей по сложным логическим критериям, содержащим логические связки AND, OR и NOT, с помощью поразрядных логических операций над битовыми строками.

Возможны различные другие схемы организации битовых индексов. Например, можно статьи индекса ставить в соответствие не свойствам записей, а самим записям. При этом статья индекса будет содержать уникальный идентификатор записи (не обязательно соответствующий первичному ключу) и битовую строку, позиции которой будут соответствовать значениям различных свойств.

Пример 2. По этой схеме сформируем битовые строки для записей файла из Примера 1, соответствующие следующим свойствам:

Свойство 1. **Фамилия_поставщика = “Петров”**

Свойство 2. **Город = “Томск”**

Состав индекса:

Номер статьи	Значение уникального идентификатора	Битовая строка
01	S01	0 1
02	S02	1 1
03	S03	0 0
04	S04	0 0
05	S05	1 0

Очевидно, что такая организация битовых шкал освобождает СУБД от необходимости перенумеровывать записи файла.

Заметим, что формирование битовой шкалы по второй схеме является в некотором роде транспонированием битовой шкалы, составленной по первой схеме.

Решения, принимаемые на этапах физического проектирования и настройки, чаще всего представляют собой компромисс между достижением требуемых характеристик, которые часто противоречат друг другу. За выигрыш в скорости обработки запросов, которую дает индекс, приходится платить дополнительными ресурсами памяти на его размещение и процессорным временем для его поддержки в актуальном состоянии.

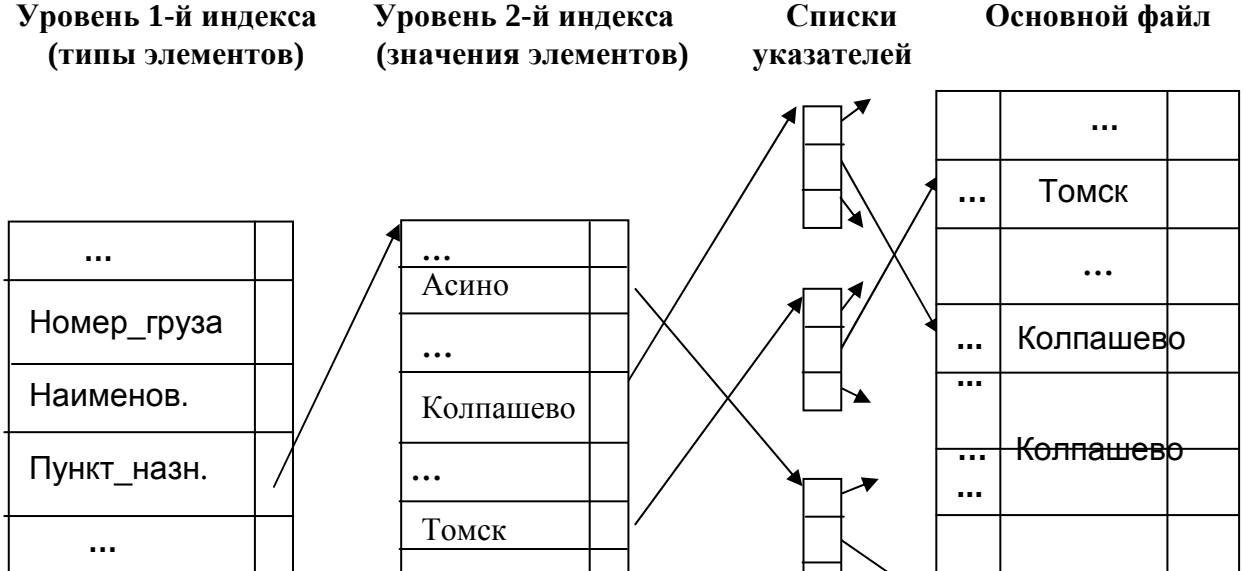


Рис.31. Схема механизма инвертирования

Замечание. В правильно спроектированной базе данных каждая таблица содержит первичный ключ, что означает автоматическое наличие индекса, построенного самой СУБД.

Данный обзор показывает, что современные СУБД предоставляют достаточно широкий набор различных методов доступа, которые чаще всего являются теми или иными видами индексирования — способа отображения ключа индексирования в адрес хранимой записи (включая и хэширование). Наиболее часто используемыми являются индексные структуры на основе B-дерева (B–tree); на основе хэш-функции или хеширование (hashing); на базе битовых шкал или индексов (bitmap). Индекс может служить различным целям: для ускорения доступа к записям одной таблицы и для ускорения операций соединения, тогда он называется индексом соединения. Если в качестве ключа индексирования используется некоторая функция атрибутов таблицы, такой индекс называют «основанным на функции» (function-based).

7.2.9. Достоинства и недостатки основных методов доступа

B-деревья универсальны и обеспечивают хорошую скорость доступа как при просмотрах по диапазонам, так и при выборке единичной записи по значению ключа, однако характеризуются относительно большим объемом памяти для хранения и затратами на поддержание в актуальном состоянии, включающими обычно балансировку дерева.

Такой индекс имеет один существенный недостаток — он может быть использован только в запросах по ведущим столбцам. Это означает, что если используется составной индекс, то поиск по всем атрибутам, входящим в индекс, начиная со второго, будет медленным. Допустим, определен индекс index1(id1, id2); в этом случае поиск значений, удовлетворяющих условию id2=1, будет медленным (не исключено, что оптимизатор вообще не будет использовать этот индекс для обработки данного условия и будет принято решение о полном сканировании данных), а поиск значений, удовлетворяющих условию id1=1 and id2=1, будет быстрым. Несмотря на этот недостаток, индексы B-деревьев наиболее распространены.

Хэш-индекс имеет небольшие накладные расходы на хранение, однако требует, чтобы распределение значений ключа индексирования было относительно постоянно, в противном случае потребуется частая переделка индекса на основе новой хэш-функции. Индексы на основе хэш-функций хорошо подходят для различного рода справочных таблиц.

Битовые индексы также очень компактны и полезны для столбцов с большим процентом повторения значений ключа. Обычно используют следующее правило: если количество

повторяющихся значений столбца более 99% от общего количества строк таблицы, тогда целесообразно рассмотреть использование битового индекса.

Битовые индексы обладают очень важным свойством: если производится запрос, включающий сложное условие выборки, которое составлено из предикатов OR, AND, NOT и «=«, то оптимизатор может использовать имеющиеся по конкретным столбцам битовые индексы, объединяя их. B-деревья этого делать не позволяют (для этого потребовалось бы построить составной индекс по этим столбцам, специально для ускорения данного запроса).

Битовые карты полезны в хранилищах, где преобладают длинные транзакции и данные читаются чаще чем записываются, однако они неэффективны в приложениях с короткими транзакциями,

Данные особенности следует учитывать при определении индексов в схеме базы данных, а именно:

- индексировать нужно атрибуты, по которым наиболее часто осуществляется поиск или соединение. Наличие индекса замедляет операции модификации, но ускоряет поиск;
- наличие индекса обязательно, если для атрибута или набора атрибутов указано ограничение unique. Такие индексы СУБД создает автоматически, если в описании таблицы указаны ограничения unique;
- индекс может быть использован для выборки данных в заданном порядке. В этом случае не вызывается процесс сортировки ответа, а используется уже готовый индекс;
- атрибуты, входящие во внешний ключ, также следует индексировать, если СУБД не делает эту операцию автоматически при декларации внешнего ключа;
- в некоторых СУБД поддерживаются bitmap-индексы, которые очень эффективны при поиске на равенство, но для поиска на этот тип индексов не годится;
- в некоторых СУБД поддерживаются хеш-индексы, например для кластеров. Такие индексы эффективно используются при поиске на равенство.

Для того чтобы выбрать тот или иной тип индекса, требуется внимательно изучить руководство администратора СУБД. Оптимизатор SQL использует различные типы доступа к данным при обработке запросов, и индексирование существенно влияет на выбор оптимизатора.

7.3. Бесфайловая организация внешней памяти

При файловой организации обмен данными между оперативной и внешней памятью осуществляется под управлением операционной системы (ОС). Основу этого обмена составляют блоки записей, входящие в состав файлов, и буфера оперативной памяти, выделяемые ОС. Ясно, что всю работу по управлению буферами берет на себя ОС.

Причины передачи управления пространством внешней памяти от ОС к СУБД были названы в начале главы; еще раз отметим, что при бесфайловой организации внешней памяти операционная среда не получает непосредственного доступа к этому пространству.

Замечание 1. Физическая организация современных баз данных является наиболее закрытой и является коммерческой тайной для большинства коммерческих СУБД. И здесь не существует никаких стандартов. Однако в своей основе СУБД ориентированы на реализацию реляционных моделей данных, поэтому можно сформулировать какие-то общие подходы к управлению внешней памяти.

Замечание 2. Следует отметить, что некоторые механизмы, описываемые ниже, используются и при файловом управлении.

7.3.1. Особенности реляционных СУБД

Реляционные СУБД обладают рядом особенностей, влияющих на организацию внешней памяти. К наиболее важным особенностям можно отнести следующие.

Наличие двух уровней системы:

уровня **непосредственного управления данными** во внешней памяти (а также обычно управления буферами оперативной памяти, управления транзакциями и журнализацией изменений БД),

языкового уровня (например уровня, реализующего язык SQL).

При такой организации подсистема нижнего уровня должна поддерживать во внешней памяти набор базовых структур, конкретная интерпретация которых входит в число функций подсистемы верхнего уровня.

Поддержка **отношений-каталогов** (справочников). Информация, связанная с именованием объектов базы данных и их конкретными свойствами (например, структура ключа индекса), поддерживается подсистемой языкового уровня. С точки зрения структур внешней памяти, отношение-каталог ничем не отличается от обычного отношения базы данных.

Регулярность структур данных. Поскольку основным объектом реляционной модели данных является плоская (в 1НФ) таблица, главный набор объектов внешней памяти может иметь очень простую регулярную структуру. При этом необходимо обеспечить возможность эффективного выполнения операторов языкового уровня как над одним отношением (простые операции селекции и проекции), так и над несколькими отношениями (наиболее распространена и трудоемка операция соединения нескольких отношений). Для этого во внешней памяти должны поддерживаться дополнительные индексы.

Для выполнения требования надежного хранения баз данных необходимо поддерживать **избыточность хранения** данных, что обычно реализуется в виде журнала изменений базы данных.

Соответственно, возникают следующие разновидности объектов во внешней памяти базы данных:

строки отношений - основная часть базы данных, большей частью непосредственно видимая пользователям;

управляющие структуры - индексы, создаваемые по инициативе пользователя (администратора) или верхнего уровня системы из соображений повышения эффективности выполнения запросов и обычно автоматически поддерживаемые нижним уровнем системы;

журнальная информация, поддерживаемая для удовлетворения потребности в надежном хранении данных;

служебная информация, поддерживаемая для удовлетворения внутренних потребностей нижнего уровня системы; набор структур служебной информации зависит от общей организации системы, но обычно требуется поддержание следующих служебных данных:

- внутренние каталоги (справочники), описывающие физические свойства объектов базы данных, например число атрибутов отношения, их размер и, возможно, типы данных;
- описание индексов, определенных для данного отношения;
- описатели свободной и занятой памяти в страницах внешней памяти, распределенных для хранения отношений; такая информация требуется для нахождения свободного места при занесении кортежей.
-

7.3.2. Базовые структуры памяти

7.3.2.1. Структура и типы страниц

Основной единицей хранения и манипулирования данными при бесфайловой организации является **страница памяти** (или **блок данных**) - часть пространства памяти среды хранения базы данных, организованного таким образом, что оно состоит из последовательности таких частей (страниц), имеющих одинаковую длину.

Страница является единицей обмена с внешней памятью. Размер страницы фиксирован для базы данных и устанавливается при ее (базы) создании. Страницы памяти имеют **уникальные идентификаторы**, в качестве которых обычно используются их последовательные номера. Содержимое страницы памяти может быть прочитано в буфер обмена или записано во внешнюю память из буфера за одно обращение к устройству внешней памяти. В некоторых системах страницы памяти могут иметь внутреннюю организацию, например, могут обладать **индексом**, обеспечивающим прямой доступ к содержащимся на странице хранимым записям. Страницы с простейшей организацией, предусматривающей последовательное размещение в них записей, в некоторых методах доступа называются **блоками записей**.

Выделяют четыре типа страниц:

- страницы данных,
- страницы индексов,
- страницы blob-объектов,
- битовые страницы.

Страница данных. Основная единица осуществления операций обмена. Структура страницы данных представлена на рис. 32.

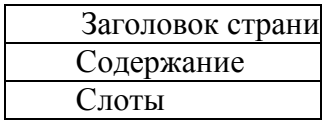


Рис. 32. Структура страницы данных

Заголовок страницы включает внутрисистемную информацию, используемую СУБД в механизме управления страницами.

Данные на странице представляются в виде **строк**. Каждая строка соответствует некоторому кортежу отображаемого отношения.

Слоты характеризуют размещение строк данных на странице. В базе данных каждый кортеж имеет уникальный внутрисистемный идентификатор, включающий номер страницы и номер строки на странице, в которую отображается данный кортеж. Содержимое слота и составляет идентификатор соответствующей ему (по номеру на странице) строки. При упорядочивании кортежей отношения по значению какого-либо атрибута физического перемещения строк на соответствующих страницах не происходит. Вместо этого производится перестройка содержимого слотов.

Страница индексов. Страницы индексов предназначены для хранения индексных структур, используемых СУБД в реализации методов доступа, и организованы в виде B-деревьев.

Страница blob. Страницы blob (**B**inary **L**arge **Ob**ject) предназначены для хранения слабоструктурированной информации, содержащей тексты большого объема, графическую информацию, двоичные коды. Эти данные рассматриваются как потоки байтов произвольного размера, а в страницах данных формируются ссылки на эти страницы. Данные таких типов в ранних СУБД относились к типу MEMO.

Битовая страница. Битовые страницы содержат описатели других типов страниц. Описатель страницы включает две составляющих – **тип** страницы и ее **состояние** (**свободна**/**занята**).

7.3.2.2. Табличные пространства

Общим для СУБД является понятие **пространства** (для некоторых СУБД **табличное пространство**). В табличных пространствах размещены различные логические структуры данных, такие как таблицы и индексы, временные таблицы и словарь данных. Группировка хранимых данных по пространствам производится по ряду признаков: частота изменения данных, характер работы с данными (преимущественно чтение или запись и т.п.), скорость роста объема данных, важность и т.п. Таким образом, например, только читаемые таблицы помещаются в одно пространство, для которого установлены одни параметры хранения, таблицы транзакций размещаются в пространстве с другими параметрами и т.д. (рис. 33).

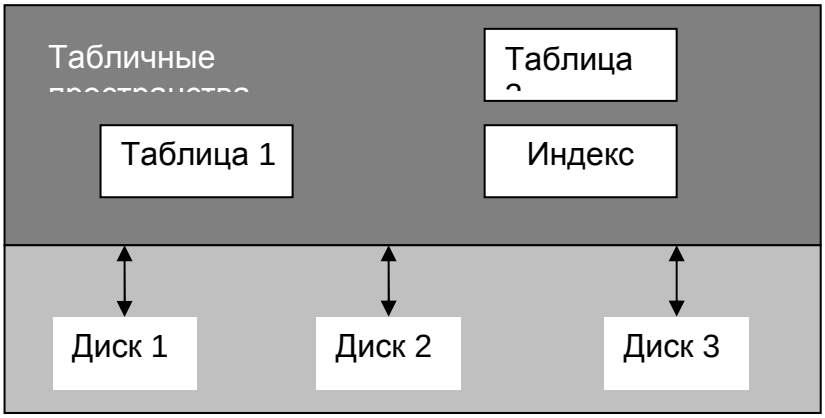


Рис.33. Физическое размещение данных по устройствам

Одна логическая единица данных (таблица или индекс) размещается точно в одном пространстве, которое может быть отображено на несколько физических устройств или файлов. При этом физически разнесены (располагаться на разных дисках) могут не только логические единицы данных (таблицы отдельно от индексов), но и данные одной логической структуры (таблица на нескольких дисках). Такой способ хранения называется **горизонтальной фрагментацией** (или **секционированием**): таблица делится на фрагменты по строкам. Фрагментация — один из способов повышения производительности.

Могут применяться различные схемы записи данных во фрагментированные таблицы. Одна из них — круговая, когда некоторая часть вставляемых в таблицу строк записывается в первый фрагмент, другая часть — в следующий и так далее по кругу. В данном случае за счет распараллеливания может быть увеличена производительность операций модификации данных и запросов.

Существует и другая схема, включающая логическое разделение строк таблицы по ключу (**кластеризация**). Данная схема позволяет избежать перерасхода процессорного времени и уменьшить общий объем операций ввода/вывода. Ее суть в том, что при создании таблицы все пространство значений ключа таблицы разбивается на несколько интервалов, а строкам с ключами, принадлежащими разным интервалам, назначаются различные месторасположения.

Впоследствии, при обработке запроса, данная информация учитывается оптимизатором. Если производится поиск по ключу, то оптимизатор может удалять из рассмотрения фрагменты таблицы, не удовлетворяющие условию выборки.

Пусть, например, для таблицы **Person** создаются два раздела **part1** и **part2**, каждый из которых размещен в своем табличном пространстве (**tblspace1** и **tblspace2**). Записи со значением поля **Num** от 1 до 499 будут располагаться в первом разделе, а записи с номерами от 500 до 1000 — во втором (рис. 34.).

Тогда при запросе:

```
SELECT FIO FROM person
 WHERE Num BETWEEN 10 AND 40
```

оптимизатор будет производить поиск только в разделе part1, что может дать ощутимый выигрыш в производительности в таблице с десятками тысяч строк.

Подобные механизмы фрагментации данных поддерживают практически все современные СУБД, что часто используется при создании систем высокой производительности.

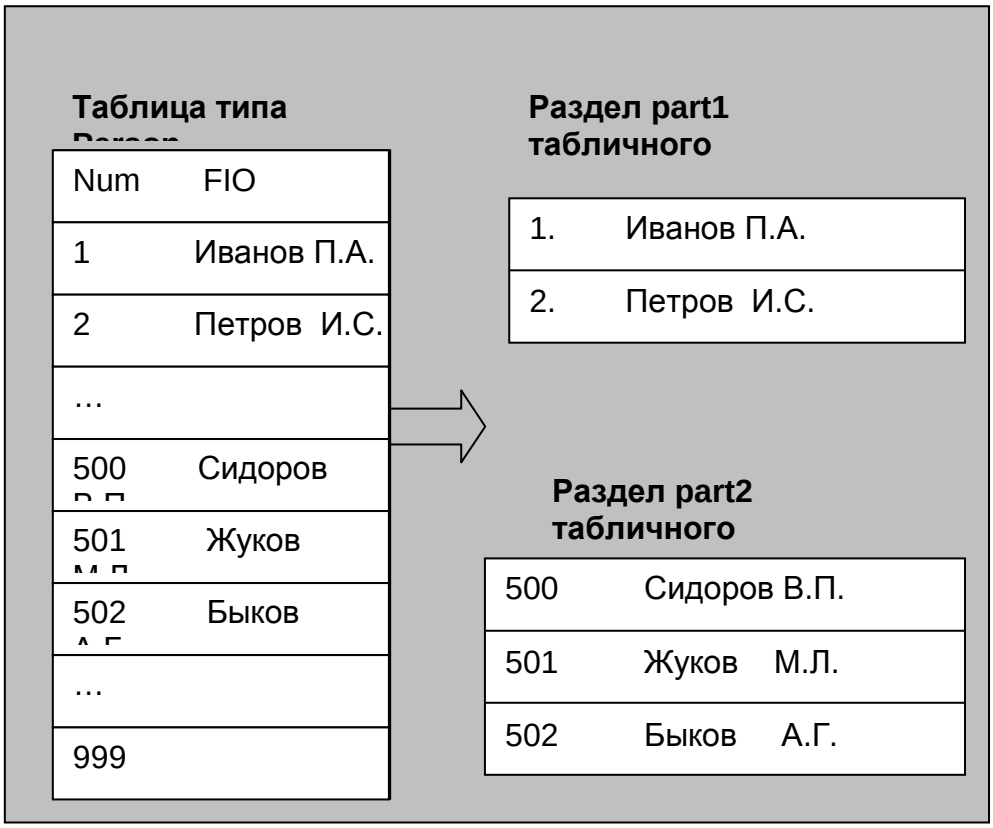


Рис. 34. Пример кластеризации записей

7.3.2.3. Понятие экстента и буферизация

Управление пространствами внешней памяти осуществляется с помощью **экстентов**, то есть непрерывных последовательностей страниц. Информация о наличии экстентов для объекта схемы данных находится в специальных управляющих структурах. При таком подходе модель объекта во внешней памяти представляется списком экстентов.

На рис. 35. иллюстрируется взаимосвязь блоков (страниц), экстентов и разделов (файлов) баз данных.

Рис.35. Блоки, экстенты и файлы базы данных

Операции манипулирования данными выполняются механизмами СУБД непосредственно над содержимым **буферов**. Система управления буферами реализует:

- создание и поддержку совокупности буферных областей в оперативной памяти,
- некоторый механизм перенесения содержимого буферов во внешнюю память, тем самым обеспечивая сохранение произведенных изменений в хранимой базе данных.

Следует отметить несколько важных проблем, возникающих в процессе физического проектирования.

7.3.3. Проблемы и параметры управления внешней памятью

Кроме рассмотренных выше проблем выбора методов доступа к записям, на этапе физического проектирования возникают и проблемы, связанные с управлением внешней памятью.

Выбор размера блока данных (страницы)

Размер блока оказывает большое влияние на производительность базы данных — при больших размерах скорость операций чтения/записи растет (особенно это характерно для полных просмотров таблиц и операций интенсивной загрузки данных), однако возрастают накладные расходы на хранение (база увеличивается) и снижается эффективность индексных просмотров. Меньший размер блока позволяет более экономно расходовать память, но вместе с тем относительно дорог. Длинные блоки (16, 32 или 64 Кбайт) лучше использовать для больших объектов данных: полнотекстовые фрагменты, мультимедиа-объекты, длинные строки и т.п. Короткие блоки (2 или 4 Кбайт) лучше подходят для значений числовых типов, недлинных строк, значений даты и времени. Следует также учитывать размер блока операционной системы (ОС), он должен быть кратен размеру блока базы данных. Малый размер блока лучше подходит для систем оперативной обработки транзакций, потому что, если сервер блокирует данные на уровне блоков, то это позволяет большему числу пользователей работать, не мешая друг другу. В системах поддержки принятия решений, для которых более критичным является не общая пропускная способность (количество транзакций в единицу времени), а среднее время отклика, больший блок предпочтительнее.

Очень важно сразу правильно выбрать размер блока: в работающей базе изменить его практически невозможно (для этого часто проводят ряд испытаний базы данных - прототипа).

Выбор размера экстента

Информация о наличии экстентов для объекта схемы данных находится в специальных управляющих структурах, реализация которых зависит от СУБД. На управление экстентами (выделение пространства, освобождение, слияние) тратятся определенные ресурсы, поэтому для достижения эффективности нужно правильно определять их экстентов.

Фрагментация

Распространенной проблемой для администраторов при управлении физическим хранением является фрагментация различных структур внешней памяти, которая чаще обусловлена операциями удаления. В отличие от преднамеренной фрагментации, такая «фрагментация» обычно ухудшает производительность. Встречается фрагментация на уровне блоков, когда в блоке остается свободное пространство после удаления строк из таблицы, на уровне экстентов, когда заполненные экстенты чередуются с незаполненными, появившимися после операций удаления таблиц, и т.п.

Фрагментацией могут быть обусловлены следующие важные для эксплуатации базы данных проблемы.

1. Выделение свободного пространства. При вставке строк в таблицу СУБД выдает сообщение о нехватке свободного пространства, хотя на первый взгляд пространства достаточно. Пространство выделяется экстентами и, если в базе данных нет непрерывного свободного блока нужного размера, то выдается это сообщение.

2. Расход свободного пространства и общее увеличение времени обработки данных. В случае сильной фрагментации обе величины могут вырасти вдвое.

Фрагментация неизбежна и поэтому является нормальным явлением, и не всегда ухудшает характеристики базы данных. Индексы также подвержены проблемам фрагментации.

Параметры проектирования физического уровня

Таким образом, организация хранения объектов данных является одной из самых сложных задач проектирования схемы базы данных, для решения которой привлекаются администраторы баз данных. И, хотя универсальных решений, которые подошли бы для любой СУБД, не существует, так как каждый производитель СУБД создает свой способ хранения и доступа к данных, хранение данных во внешней памяти в известных СУБД организовано очень похожим образом.

Вследствие этого, в литературе определяется совокупность следующих основных параметров, оказывающих влияние на проектирование схемы базы данных, общая для большинства реляционных СУБД:

- размер табличных пространств для хранения таблиц;
- размер табличных пространств для хранения индексов;
- размер табличных пространств для хранения BLOB;
- кластеры и их параметры;
- размер словаря данных, включая код всех хранимых процедур, функций, триггеров, пакетов;
- управляющие файлы;
- файлы журнала;
- интенсивность потока запросов, модифицирующих данные и индексы;
- файлы временных табличных пространств (для хранения временных таблиц, которые строятся, например, при выполнении group by, а также других временных объектов);
- интенсивность потока запросов, инициирующих создание временных таблиц;
- потоки транзакций read-write, read-only, объем модифицируемых и считываемых ими данных, характеристики параллельной работы транзакций (какие и сколько их);
- количество приложений, работающих параллельно с базой данных;
- количество соединений с базой данных для каждого приложения;
- входные и выходные данные, генерируемые пользовательскими программами.

В процессе проектирования все данные должны быть отражены в **журнале проектирования.** Однако многие тонкости в вопросах размещения объектов данных и

конфигурации сервера баз данных на этапе проектирования не могут быть учтены, так как требуют полномасштабного тестирования.

КОНТРОЛЬНЫЕ ВОПРОСЫ ПО СЕДЬМОМУ РАЗДЕЛУ

1. В чем различие в принципах управления хранилищем данных с использованием файловых и страничных (бесфайловых) систем.
2. Сформулируйте основные понятия физического уровня: хранимая запись, формат хранимой записи, метод доступа, механизм поиска.
3. Сформулируйте основные задачи этапа физического проектирования.
4. Приведите общую классификацию методов доступа.
5. Опишите способы последовательной организации.
6. Опишите метод доступа – хеширование. В чем состоит проблема синонимов.
7. Опишите метод доступа с полным индексом и индексно-последовательный метод доступа. Сравните эти методы. В чем достоинства и недостатки каждого из них.
8. В чем суть инвертирования.
9. Что такое B-дерево.
10. Опишите механизмы использования битовых шкал.
11. В чем суть бесфайловой организации внешней памяти. Опишите общую структуру страницы.

8. ОСОБЕННОСТИ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ СУБД

8.1. Основные понятия объектно-ориентированного подхода

В середине 70-х годов в некоторых языках программирования была реализована концепция **абстрактного типа данных**, то есть программистам была предоставлена возможность определять собственные (пользовательские) типы данных в дополнение к предопределенным (стандартным) типам, определенным в языке. Соответствующее описание типа включает его **идентификацию** и определение **структуры** (примером такой возможности является конструкция **struct** языка C++).

Со временем понятие **типа данных** в языках программирования было расширено, в него стали вкладывать не только структурные свойства, но и элементы **поведения**. Такое расширение понятия типа послужила в дальнейшем основой для формирования концепции **объекта**, на которой базируются современные **объектные модели**.

Таким образом, объект - это сущность (предмет или явление), обладающая состоянием и поведением. Состояние объекта определяется совокупностью его атрибутов, которые могут принимать значения определенных типов. Поведение, в свою очередь, определяется совокупностью операций (функций, или в терминологии объектно-ориентированного программирования **методов**), специфицированных для этого объекта, которые могут быть выполнены самим объектом или над состоянием объекта. Для объектов поддерживается их индивидуальность, которая не изменяется при изменении состояния объекта, то есть изменения значения его свойств.

Объекты могут иметь динамическую или статическую природу: динамические объекты используются в одном цикле воспроизводства, например заказы на продукцию, счета на оплату, платежи; статические объекты используются во многих циклах воспроизводства, например, оборудование, персонал, запасы материалов.

Объединение в одном описании структуры и поведения для типа объектов определяет второе базовое понятие объектно-ориентированного подхода - понятие **класса,** обладающего рядом важных свойств (принципов объектно-ориентированного программирования)**.**

Инкапсуляция. Это свойство определяет два момента при создании класса.

1. **Сокрытие декларативной информации** означает закрытие прямого доступа ко всем или к части атрибутов класса (доступ к таким атрибутам осуществляется только путем обращения к соответствующим методам класса). Заметим, что закрытыми, что реже, могут быть и некоторые методы класса.

2. **Сокрытие процедурной информации**. Принципиально важно, что различаются ***интерфейсы*** объектов и их ***реализации***:

- интерфейс определяет свойства объекта**,** ***видимые*** **пользователю,** - его свойства и сигнатуры (возможный набор) операций.
- реализация определяет ***внутренние*** свойства объекта, которые **инкапсулируются** интерфейсом и остаются скрытыми от пользователя.

Таким образом, класс представляет собой своеобразную **капсулу**, в которой от других программных объектов, как правило, частично скрыта декларативная информация и всегда и полностью процедурная.

Наследование для классов объектов означает возможность построения новых классов на основе существующих, причем осуществляется наследование не только атрибутов объектов от вышестоящего (родительского) класса объектов к нижестоящему классу (потомку), но и методов (**иерархия обобщения**); как и в реальности, класс-потомок в общем случае может

быть богаче родительского класса: кроме родительских свойств и методов, в порождаемый класс могут быть добавлены новые атрибуты и методы.

Полиморфизм (множественность форм**)** обычно проявляется в двух формах:

1. переопределения (**перегрузки)** некоторых методов (функций и операций); для этого требуется описать несколько одноименных методов, отличающихся типом возвращаемого значения, списками параметров, телом кода;

2. переопределения кода метода родительского класса в классах-потомках (такая функция называется **виртуальной**).

Заметим, что кроме предусмотренной встроенной иерархической связи “тип-подтип”, обеспечивающей наследование свойств типа объектами подтипа, объектные модели поддерживают типы связей, определенные пользователями.

Современные объектные модели наряду с атомарными типами объектов поддерживают сложные типы-контейнеры и типы-коллекции.

8.2. Предпосылки появления объектно-ориентированных СУБД

В середине 70-х годов XX века начали рождаться новые идеи, которые привели впоследствии к формированию объектного подхода в технологиях баз данных.

1. Объектный подход в программировании в значительной мере побудил интерес к воплощению объектной парадигмы в технологиях баз данных, поскольку возникла необходимость в обеспечении эффективной среды хранения объектных данных для поддержки многочисленных прикладных программных систем, реализованных средствами объектных языков. Эту функцию было естественно возложить на СУБД, основанные на объектных моделях данных, и обладающие интерфейсами прикладного программирования (API) для объектных языков.

Объектно-ориентированная СУБД (ООСУБД) — средство, которое обеспечивает запись объектов в базу данных **«как есть».** Данное обстоятельство стало решающим аргументом в пользу разработки технологии ООСУБД как средства для переноса семантики объектов и процессов реального мира в сферу информационных систем, то есть переноса объектов реального мира в виртуальный мир.

2. Исследования по созданию ООСУБД связаны со сложившимся пониманием неэффективности использования реляционных систем в целом ряде приложений. Многие из таких приложений нуждаются в более богатых по сравнению с предоставляемыми реляционными СУБД средствами моделирования предметной области (с более развитыми системами типов данных, возможностями структурирования данных, обеспечивающих во многих случаях более естественное отображение семантики предметной области в базах данных).

Дело в том, что одним из основных положений реляционной модели данных является требование нормализации отношений: поля кортежей могут содержать лишь атомарные значения, то есть, по существу, требование примитивности структур данных, лежащих в основе реляционной модели. Для традиционных приложений реляционных СУБД (РСУБД) - банковских систем, систем резервирования (билетов, посадочных мест и т.д.) - это вовсе не ограничение, а даже преимущество, позволяющее проектировать экономные по памяти БД с предельно простой и понятной структурой. Запросы с соединениями в таких системах сравнительно редки, для динамической поддержки целостности используются соответствующие средства SQL. Заметим, что плоские нормализованные отношения универсальны и теоретически достаточны для представления данных любой предметной области.

Однако такой подход является весьма неестественным при управлении наборами данных или данными, которые необходимо описать с большей степенью детализации. Допустим, в столбце **Адрес** таблицы **СЛУЖАЩИЙ** может быть указан полный домашний адрес, причем различные приложения могут обрабатывать различные компоненты адресов. Однако РСУБД будет трактовать данные столбца **Адрес** как единое целое (строку символов), не замечая компонентные элементы данных; поэтому либо каждому приложению придется распаковывать содержимое поля, либо, что более соответствует духу реляционной модели, для каждой компоненты адреса будет выделен отдельный столбец.

Еще большие проблемы возникают при использовании РСУБД в прикладных системах, основанных **на знаниях** – системах автоматизированного проектирования (САПР), системах искусственного интеллекта и т.п., которые обычно оперируют с объектами сложной структуры. Для формирования (воссоздания) цельного объекта из плоских таблиц РСУБД приходится выполнять запросы, почти всегда требующие соединения отношений. Поэтому попытки использования РСУБД в соответствии с требованиями таких нетрадиционных приложений приводят к появлению в базе данных сотен и тысяч таблиц, над которыми постоянно выполняются дорогостоящие операции соединения, необходимые для воссоздания сложных структур данных, присущих предметной области.

Другими словами, в руки проектировщиков даются настолько же мощные и гибкие средства структуризации данных, как те, которые были присущи иерархическим и сетевым системам баз данных.

3. Проблема, название которой можно перевести как **импеданс** (или расхождение) моделей. Она выражается в том, что модель данных, используемая в приложении, отличается от модели данных базы данных. Это различие, в свою очередь, приводит к двум проблемам в приложениях, в результате которых и возникает расхождение:

- Программист, разрабатывающий приложение, должен оперировать двумя различными языками программирования, с различным синтаксисом, семантикой и типами систем, а именно, прикладным языком программирования (например, Си++) и языком манипулирования базами данных (то есть, SQL). Логика приложения реализуется средствами прикладного языка, в то время как SQL используется для создания и манипулирования данными в базе.
- При извлечении данных из реляционной базы, они должны быть переведены из представления, в котором они там хранились, в представление, соответствующее представлению данных в памяти, характерному для данного приложения. Аналогично, все обновления данных должны быть переданы базе данных при помощи другого предложения SQL, то есть требуется еще одно преобразование из представления, используемого в приложении, в представление базы данных. Весь обмен данными между базой данных и приложением приводит к ненужной дополнительной обработке, которой можно было бы избежать, будь модели данных приложения и базы данных одинаковыми.

8.3. Объектная модель данных. ООСУБД

В связи с объектной парадигмой был предложен новый подход, при котором модель данных рассматривается как **система типов**. Такой подход обеспечивает естественные возможности интеграции баз данных и языков программирования.

Ориентируясь на современное понимание типа данных как носителя свойств, определяющих и состояние экземпляров типа, и их поведение, можно дать следующее определение понятия модели данных (ср. с определением этого понятия, приведенным в п. 2.4.):

Модель данных – это система типов данных, типов связей между ними и допустимых видов ограничений целостности, которые могут быть для них определены**.**

Стандарт на ООСУБД выработан консорциумом Object Database Management Group (ODMG), состоящим в основном их производителей таких СУБД.

В соответствии со стандартом ODMG, объектная модель данных характеризуется следующими свойствами.

- Базовыми примитивами являются **объекты** и **литералы.** Каждый объект имеет уникальный идентификатор, литерал не имеет идентификатора**.**
- Объекты и литералы различаются по **типу.** Все элементы одного типа имеют одинаковый диапазон изменения состояния (множество свойств) и одинаковое поведение (множество определенных операций). Объект, на который можно установить ссылку, называется экземпляром; он хранит определенный набор данных.
- Состояние объекта определяется набором значений множества свойств. Этими свойствами могут быть атрибуты объекта или связи между объектом и одним или несколькими другими объектами.
- Поведение объекта определяется набором операций, которые могут быть выполнены над объектом или самим объектом. Операции могут иметь список входных и выходных параметров строго определенного типа. Каждая операция может также возвращать типизированный результат.
- База данных хранит объекты, позволяя совместно использовать их различным пользователям и приложениям. База данных основана на **схеме данных,** определяемой языком определения данных, и содержит экземпляры типов**,** определенных схемой.

Каждый тип имеет внешнюю спецификацию и одну или несколько реализаций. Спецификация определяет внешние характеристики типа: пользователю для работы с объектом предоставляется набор операций и набор атрибутов объекта, при помощи которых можно работать с реальными экземплярами. Реализация определяет внутреннее содержание объектов, например операции.

Тип также является объектом. ООСУБД обслуживает множество баз данных, каждая из которых содержит определенное множество типов. В базах данных могут содержаться объекты соответствующего типа из этого множества. Тип имеет набор свойств, а объект характеризуется состоянием в зависимости от значения каждого свойства. Операции, определяющие поведение типа, едины для всех объектов одного типа. Свойство едино для всего типа, а все объекты типа также имеют одинаковый набор свойств. Значение свойства относится к конкретному объекту.

Таким образом, одним из принципиальных отличий объектных баз данных от реляционных является возможность создания и использования новых типов данных.

Основные элементы ООСУБД представлены на рис.36., а применение базовых понятий объектной модели в ООСУБД на рис.37.

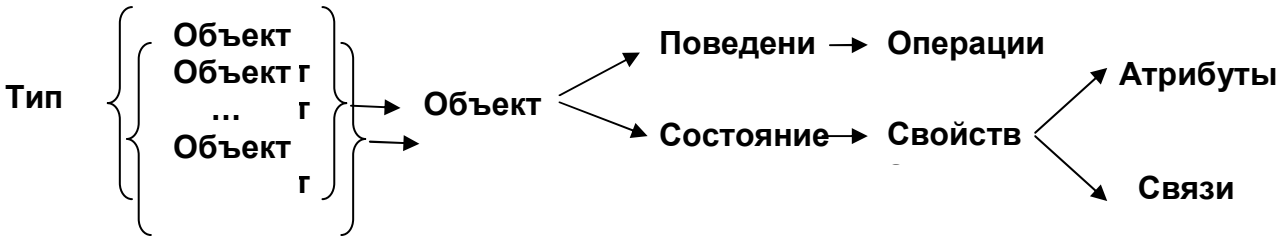


Рис.36. Основные элементы ООСУБД

Замечание 1. Очевидно родство понятия **класса** объектно-ориентированной парадигмы программирования и понятия **типа** объектно-ориентированной технологии баз данных.

Замечание 2. Напомним, что термин объект в РСУБД **обозначает** либо тип объектов, либо экземпляр типа объектов. В ООСУБД и тип, и экземпляр типа **являются** объектами.

Стандартный механизм объектно-ориентированного программирования — наследование, реализуется в ООСУБД путем поддержки иерархии супертипов (родительских типов) и подтипов (типов-потомков). Как правило, в ООСУБД для объектов, которые предполагается хранить в базе (постоянные объекты), требуется, чтобы их предком был конкретный базовый тип, определяющий все основные операции взаимодействия с сервером баз данных. Поэтому для создания своего типа необходимо унаследовать свойства любого имеющегося типа, наиболее подходящего по своему поведению и состоянию к типу, который требуется получить, расширить недостающие операции и атрибуты и переопределить, по необходимости, уже имеющиеся.

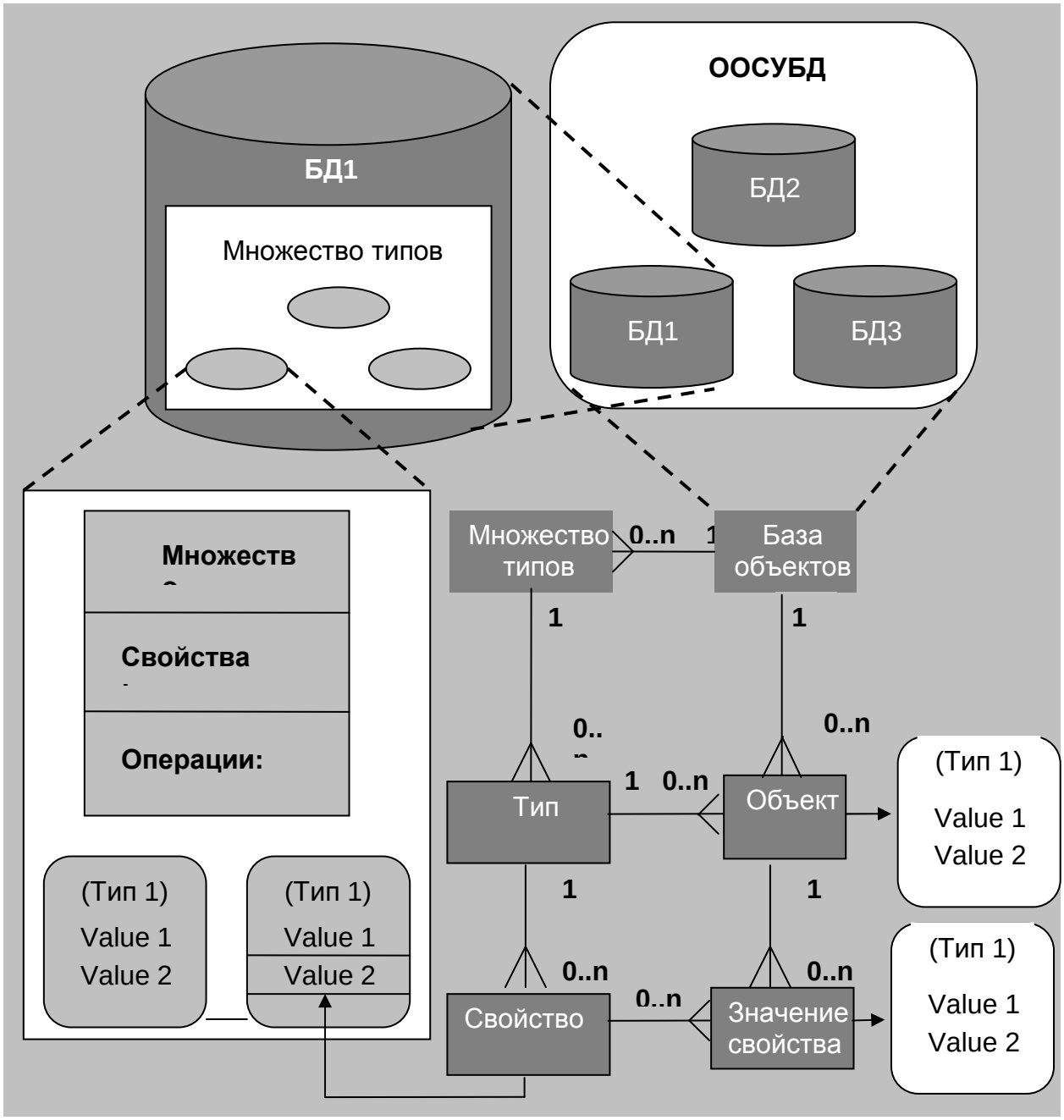


Рис.37. Применение базовых понятий объектной модели в ООСУБД

Пример на рис.38. иллюстрирует возможность наследования типа «Человек», который может быть «Мужчиной», «Женщиной», «Взрослым» или «Ребенком». Соответственно, возможны попарные пересечения этих типов. Каждый из этих типов может иметь свой набор свойств и операций. Любой объект типа Мужчина, Женщина, Взрослый или Ребенок является объектом типа Человек. Аналогично объект подтипа МужчинаВзрослый, полученного наследованием типов Мужчина и Взрослый, является человеком, мужского пола, взрослым и, соответственно, может пользоваться свойствами и операциями всех своих супертипов.

Идея, положенная в основу ООСУБД, состоит в том, что лучший способ обеспечить возможность долговременного хранения объектов — это реализовать долговременное хранение объектов, используемых в объектно-ориентированных языках программирования, таких как Си++. Поэтому ООСУБД часто называют системами долговременного хранения на базе языков программирования. Однако ООСУБД — это нечто больше, чем язык программирования с добавленными возможностями долговременного хранения. Исторически сложилось так, что объектно-ориентированные СУБД создавались в расчете на рынок приложений, где крайне важны такие функции, как быстрый доступ с возможностями навигации, поддержка версий и обработка транзакций, предполагающих выполнение последовательности операций в базе данных. Таким образом, объектно-ориентированные СУБД поддерживают приложения, обладающие такими возможностями, как поддержка долговременного хранения объектов, созданных в разных языках программирования, распределение данных, улучшенные модели обработки транзакций, поддержка версий, динамическая генерация новых типов.

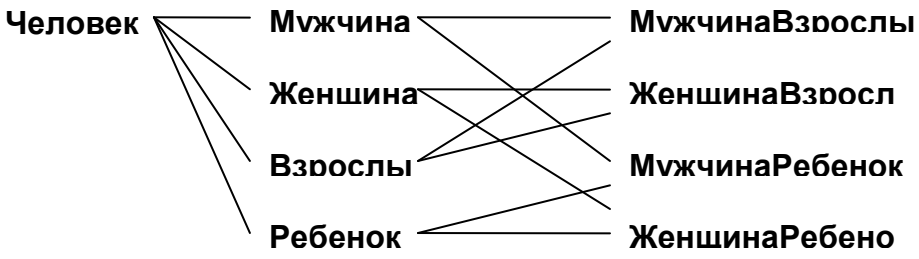


Рис.38. Пример наследования типов

8.4 Объектно-реляционные СУБД

В определенный момент возникло ощущение, что объектно-ориентированные СУБД вытеснят реляционные и в силу согласованности с объектно-ориентированной парадигмой программирования, и в силу возможности навигации по указателям (ссылкам), что позволяет на новом уровне реализовать лучшие изобразительные средства иерархических и сетевых моделей.

Однако, несмотря на определенные достоинства, из-за ряда причин ООСУБД так и не смогли оказать значительного влияния на положение дел в области технологии баз данных:

1. Отсутствие в ООСУБД базовых средства, к которым пользователи систем баз данных привыкли.

Чтобы ООСУБД смогли оказать значительное влияние на рынок баз данных, требовалось учесть привычку разработчиков к использованию средств РСУБД и огромный объем

прикладного программного обеспечения, использующего реляционные базы данных. Для этого необходимо было:

- превратить ООСУБД в системы баз данных, обладающих достаточной совместимостью с РСУБД, чтобы обеспечить сосуществование имеющихся и создающихся новых продуктов,
- унифицировать модели данных РСУБД и ООСУБД,
- унифицировать архитектуры РСУБД и ООСУБД**.**

2. Чтобы стать полноценными СУБД, ООСУБД должны поддерживать весь спектр соответствующих средств:

- стандартная алгебра, средства обеспечения и оптимизации запросов,
- управление транзакциями,
- поддержка ограничений целостности,
- организация безопасности,
- и т.д.

3. Наличие в ООСУБД основных базовых средства, которые пользователи хотели бы видеть;

- широкие возможности настройки производительности, присущие РСУБД,
- поддержка сложных объектов,
- поддержка динамических изменений определений классов,
- полная интеграция с объектно-ориентированными системами программирования.

Значительные сложности учета перечисленных факторов привел к тому, что рынок ООСУБД в настоящее время, по существу, исчез**.**

Основные трудности объектно-ориентированного моделирования данных проистекают из того, что не существует развитого математического аппарата, на который могла бы опираться общая объектно-ориентированная модель данных.

Замечание. Некоторые авторы утверждают, что общая объектно-ориентированная модель данных в классическом смысле и не может быть определена по причине непригодности классического понятия модели данных к парадигме объектной ориентированности.

В начале 90-х годов была предложена идея **объектно-реляционных систем** (ОРСУБД),

поддерживающих **объектно-реляционную модель данных** - гибридную модель данных, сочетающая возможности реляционных моделей с объектными свойствами данных (например, продукты типа Oracle включили в себя расширения и возможности для реализации определенных средств объектно-ориентированных СУБД). Такие модели стали использоваться как **паллиатив (**полумера), обеспечивающий преодоление ограниченных возможностей реляционной модели**.**

Суть объектно-реляционных СУБД (ОРСУБД) в предположении, что наилучший способ удовлетворить требования объектно-ориентированных приложений — это расширить реляционную модель. Реляционная модель убедительно доказала свою состоятельность, а SQL, по существу, стал мировым стандартом. Поэтому, объектно-реляционные СУБД добавляют поддержку объектно-ориентированных моделей данных за счет расширения реляционной модели данных и языка запросов, сохраняя в относительной неприкосновенности технологии реляционной СУБД.

Объектно-реляционная модель включает в себя принцип вложения классов, уникальные идентификаторы объектов, методы, приписываемые к классу, наследование атрибутов и методов в соответствии с иерархией классов, а также связи обобщения/специализации между классами, расположенными на разных уровнях иерархии классов. Модифицируется и язык SQL путем добавления объектных расширений.

В настоящее время производители ОРСУБД поддерживают стабильное хранение объектов для объектно-ориентированных языков программирования, включая C++ и Java. В сущности, они обеспечивают *обертку* (**wrapper**) между объектно-ориентированной программой и

ОРСУБД, сглаживающую различия между моделью объектно-ориентированного языка и моделью ОРСУБД.

Подход ОРСУБД — это «восходящий» подход, ориентированный на данные (или базу данных). Он лучше других позволяет расширить область практического применения существующих, унаследованных данных, хранящихся в реляционных базах данных. ОРСУБД дают возможность устранить расхождение моделей и компенсировать снижение производительности, возникающие при обращении к реляционным данным из объектно-ориентированного языка программирования. Однако модель данных, которая используется приложением, может оказаться близкой, но не идентичной модели данных, применяемой в СУБД для хранения данных приложения. ОРСУБД решают эту задачу за счет обеспечения возможности организации сложных запросов и разносторонней поддержки исполняемых компонентов приложения на сервере базы данных. Кроме того, они обладают самой высокой надежностью, лучшими характеристиками при поддержке параллельности и восстановления после сбоя среди систем всех трех классов. Приложения, которым требуется хорошая поддержка обработки запросов, хороший уровень защиты, целостности, поддержки параллельности и надежности, а также высокая скорость обработки транзакций — лучшие кандидаты на использование ОРСУБД.

Подход ООСУБД — это «нисходящий» подход, ориентированный на приложение (или язык программирования). Это самое приемлемое решение для хранения объектов приложений. Данный подход обеспечивает бесконфликтное долговременное хранение с точки зрения языка программирования. ООСУБД позволяет устранить расхождение моделей, предлагая обширную поддержку возможностей моделирования данных одного или нескольких объектно-ориентированных языков программирования. Таким образом, в ООСУБД модель данных, которая используется приложением, идентична модели, применяемой в СУБД для хранения данных приложения. ООСУБД не предлагают столь же качественные возможности обслуживания запросов, как ОРСУБД. Подход ООСУБД в первую очередь рассчитан на приложения, которым необходима высокая скорость при навигации, которые не поддерживают сложные запросы, и готовы в определенной мере пожертвовать целостностью и безопасностью в пользу высокой производительности. Другими словами, почти все современные ООСУБД — не столько системы баз данных, сколько системы стабильного хранения данных для некоторого объектно-ориентированного языка программирования.

Таким образом, в принципах построения современных ООСУБД можно выделить два подхода:

1. **Революционный** - полный отказ от использования идей реляционных моделей данных.
2. **Эволюционный** – совмещение реляционной платформы с объектно-ориентированной парадигмой**.**

8.4.1. Объектно-реляционное отображение

При проектировании информационных систем с использованием ОРСУБД возникает задача реализации объектно-ориентированного отображения. Существует два способа решения этой проблемы, часто называемых «**сверху вниз**» и «**снизу вверх**».

Основная идея метода «сверху вниз» состоит в том, чтобы начать с объектной модели, после чего от нее производится реляционная схема хранения данных, то есть, ставится задача сохранения объектов в реляционных таблицах.

Метод «сверху вниз» позволяет сосредоточиться на логике приложения, например, для описания общей объектной модели. При этом особое внимание уделяется тому, что производная реляционная схема хранения данных проектируется так, чтобы обеспечить производительность и целостность данных. Возможность использовать простое отображение

«один-к-одному» между классами и таблицами существует далеко не всегда. Очень часто схему объектных данных нужно адаптировать под специфику реляционного мира. Например, никогда нельзя забывать о нормализации. Однако слишком высокая нормализация приводит к низкой производительности, сильно фрагментированным схемам данных и чрезмерной сложности. По этой причине излишне высокой нормализации (выше ЗНФ) все разработчики стараются избегать.

Метод «снизу вверх» начинает с реляционной схемы хранения данных, от которой производит объектную модель, что позволяет в дальнейшем осуществлять доступ к реляционным таблицам с помощью объектов. Этот метод используется достаточно часто в тех случаях, когда реляционная схема данных уже существует, а вносить изменения в нее крайне нежелательно. В подобных случаях проводится анализ реляционной схемы, после чего предпринимается попытка произвести от этой схемы объектную модель, которая бы отражала, с одной стороны, логику приложения, а с другой — собственно реляционную схему. Достижение подобных результатов может оказаться достаточно сложной задачей в случае нормализованной реляционной схемы. В подобных ситуациях возникает необходимость отобразить несколько таблиц на один класс, благодаря чему один экземпляр объекта представляет несколько строк с данными. Очень часто этот метод называется упаковкой, поскольку один класс содержит несколько таблиц.

Отображение классов на таблицы. Рассмотрим способы отображения классов на таблицы подробнее.

Основная идея объектно-ориентированного отображения достаточно проста:

- классы отображаются на таблицы,
- создаваемые объекты данного класса представляются в виде строк, а атрибуты объектов — в виде столбцов реляционной таблицы.

Основная схема объектно-ориентированного отображения приведена на рис.39.

Наследование

Реляционная модель не предполагает наследования. Таким образом, отображение иерархии наследования объектной модели на реляционную схему является достаточно непростым заданием. Существует несколько способов отображения наследования на уровне объектной модели на реляционную модель, рассмотренных ниже, каждый из которых обладает определенными преимуществами и недостатками.

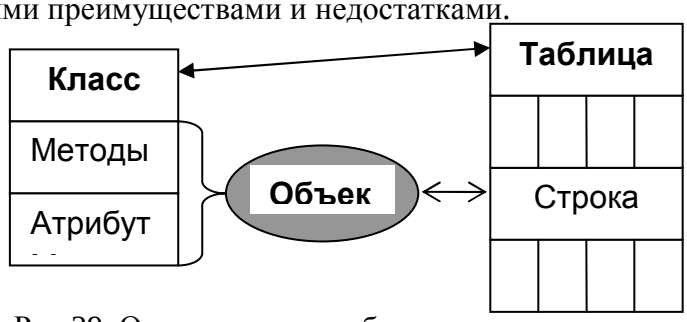


Рис.39. Основная схема объектно-ориентированного отображения

Пример. Пусть имеется следующая иерархия классов, представленная на рис.40.

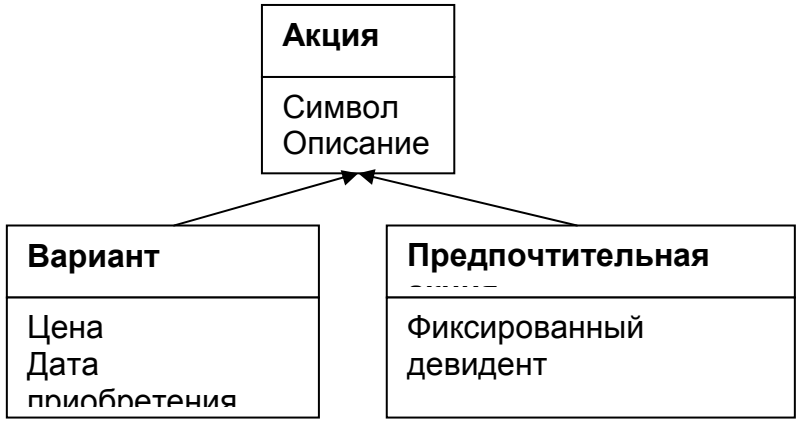


Рис.40. Пример иерархии наследования

Горизонтальное разделение

Горизонтальное разделение поддерживает множественное наследование. Именам столбцов могут предшествовать имена классов, для которых первоначально были определены атрибуты. Горизонтальное разделение обеспечивает отображение, поддерживающее простую идею *отобрази-каждый-класс-на-таблицу*, а также обеспечивает приемлемую производительность операций манипулирования строго типизированными данными. Пример, приведенный на рис.41., иллюстрирует, как выглядит горизонтальное разделение для базового класса **Акция** с двумя потомками **Вариант** и **Предпочтительная акция.** Запрос на получение сведений обо всех объектах предпочтительных акций требует одного простого запроса, который вернет целую коллекцию данных, в частности SQL-вызов

SELECT * FROM Предпочтительная акция

Однако горизонтальное разделение обладает и целым рядом недостатков. При внесении изменений в один из абстрактных базовых или конкретных классов, находящихся в основании графа наследования, например, добавления нового атрибута, претерпят изменения все конкретные классы или таблицы, наследующие от этого класса, С этой точки зрения каждый запрос требует **осуществления запросов к нескольким таблицам,** результаты которых агрегируются. Например, для того чтобы извлечь все объекты **Акция,** нам необходимо провести три отдельных SQL-запроса SELECT; первый для извлечения всех экземпляров **Акция,** второй для извлечения всех экземпляров **Вариант** и третий — для извлечения всех экземпляров **Предпочтительная акция.** Все полученные результаты затем объединяются для получения унифицированной коллекции, содержащей сведения обо всех акциях. Конечно, повысить эффективность подобных запросов можно, используя оптимизацию, основанную на специфике реляционных способов отображения, которые бы расширяли возможности схемы горизонтального разделения,.

Вертикальное разделение

Каждый производный класс отображается на новую таблицу, содержащую атрибуты, которые определены только для этого класса. Помимо этого, каждая таблица содержит внешний ключ, указывающий на таблицу базового класса. Множественное наследование поддерживается путем включения нескольких внешних ключей в таблицу производного класса. Расширения и изменения очень легко добавить по мере необходимости в одну и ту же таблицу. Однако любой запрос на извлечение экземпляров производного класса или таблицы требует как минимум одного соединения, а в случаях глубокой иерархии наследования — даже нескольких соединений. Все это сказывается на производительности всех операций, манипулирующих данными (вставка, запрос, обновление и удаление). Пример метода вертикального разделения приведен на рис.42.

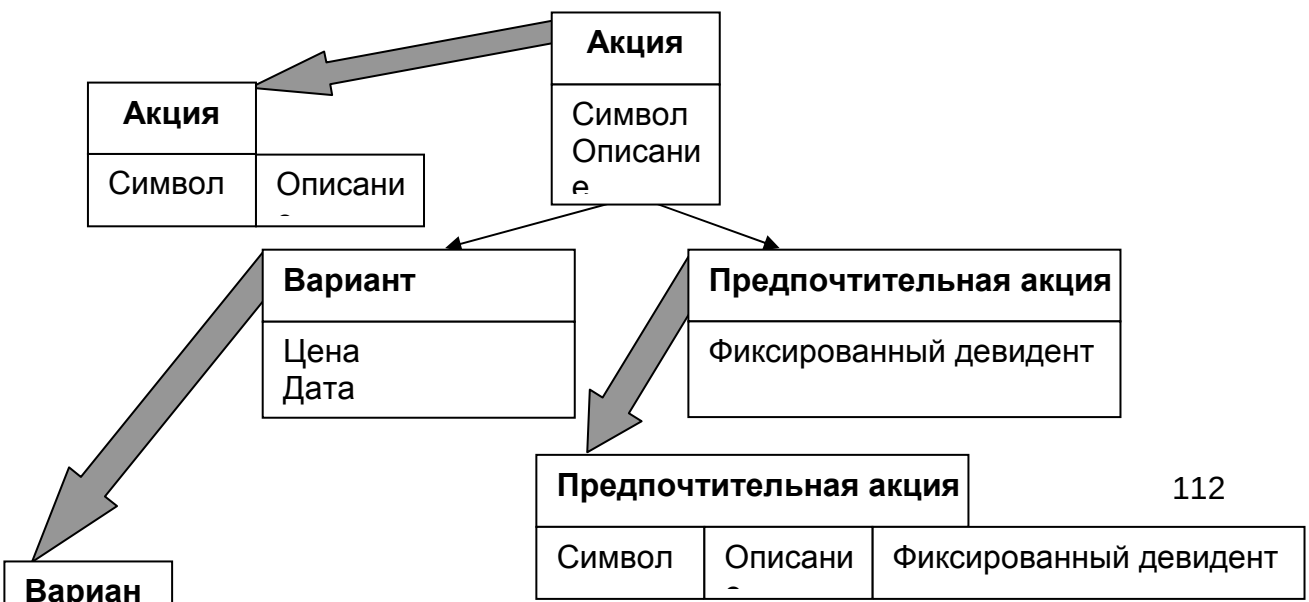


Рис.41. Горизонтальное разделение

Унификация

Все абстрактные и конкретные классы в рамках одной ветви иерархии отображаются на одну таблицу. Для идентификации типа каждой строки используется дополнительный столбец, называемый *дискриминатором типа*. Во многих случаях в этих целях может использоваться имя реального класса. Такой способ отображения не совсем соответствует правилам нормализации. Следовательно, таблицы могут быть очень слабо заполненными, особенно « том случае, если один дочерний класс характеризуется большим количеством атрибутов. Такой вариант отображения оказывается наиболее выгодным при использовании абстрактных родительских классов; это означает, что не существует реальных экземпляров абстрактных классов, несмотря на то, что все определенные для них атрибуты содержатся во всех на следующих классах. Пример метода унификации приведен на рис.43.

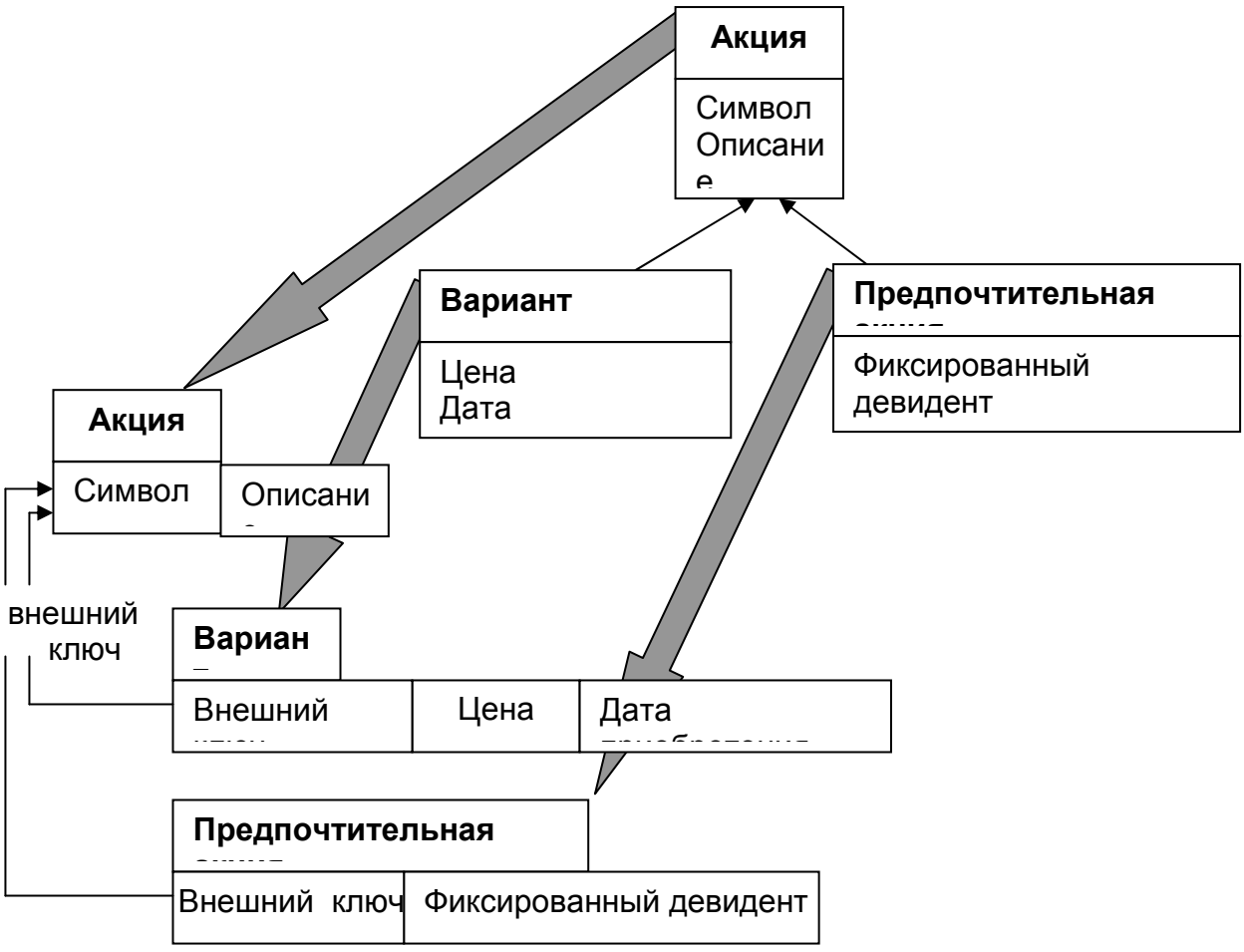


Рис. 42. Вертикальное разделение

8.5. Управление ресурсами. Сервер объектов и сервер страниц

В клиент-серверной архитектуре СУБД необходимо обеспечить эффективное использование этих ресурсов как на клиенте, так и на сервере. Клиент реляционной СУБД обращается к данным на сервере с помощью механизма, известного как доставка запроса (см. рис.44.).

Клиент посылает SQL-запрос на сервер. При получении запроса сервер его оптимизирует, выбирает подходящий план доступа, используя все имеющиеся индексы, и выполняет запрос. Результат запроса — это набор реляционных кортежей, которые возвращаются клиенту. Кортежи, возвращаемые сервером клиентам — это таблицы в первой нормальной форме, за исключением крупных объектов (например, blob).

Появление относительно недорогих рабочих станций с мощными процессорами и памятью большой емкости позволило изменить используемую в компьютерных системах модель, ориентированную на сервер, на модель, предусматривающую более сбалансированное распределение рабочей нагрузки между клиентом и сервером. Эта эволюция затронула и ООСУБД и позволила осуществлять иное распределение ресурсов между клиентом и сервером, чем в реляционных СУБД. Традиционная архитектура ООСУБД предусматривает перенос большей части работы с сервера на клиента. Во многих ООСУБД вся работа, за исключением наиболее существенных операций, связанных с управлением памятью, параллелизмом и восстановлением, переносится на клиента. ООСУБД, таким образом, осуществляет передачу ***данных***, а не передачу ***запросов***, как это происходит в реляционных СУБД. Данные, пересылаемые между клиентом и сервером, могут быть либо *объектами* (то есть сервер предоставляет объекты клиенту), либо *страницами* (когда сервер предоставляет клиенту страницы, содержащие объекты, не зная какие именно объекты размещены на странице).

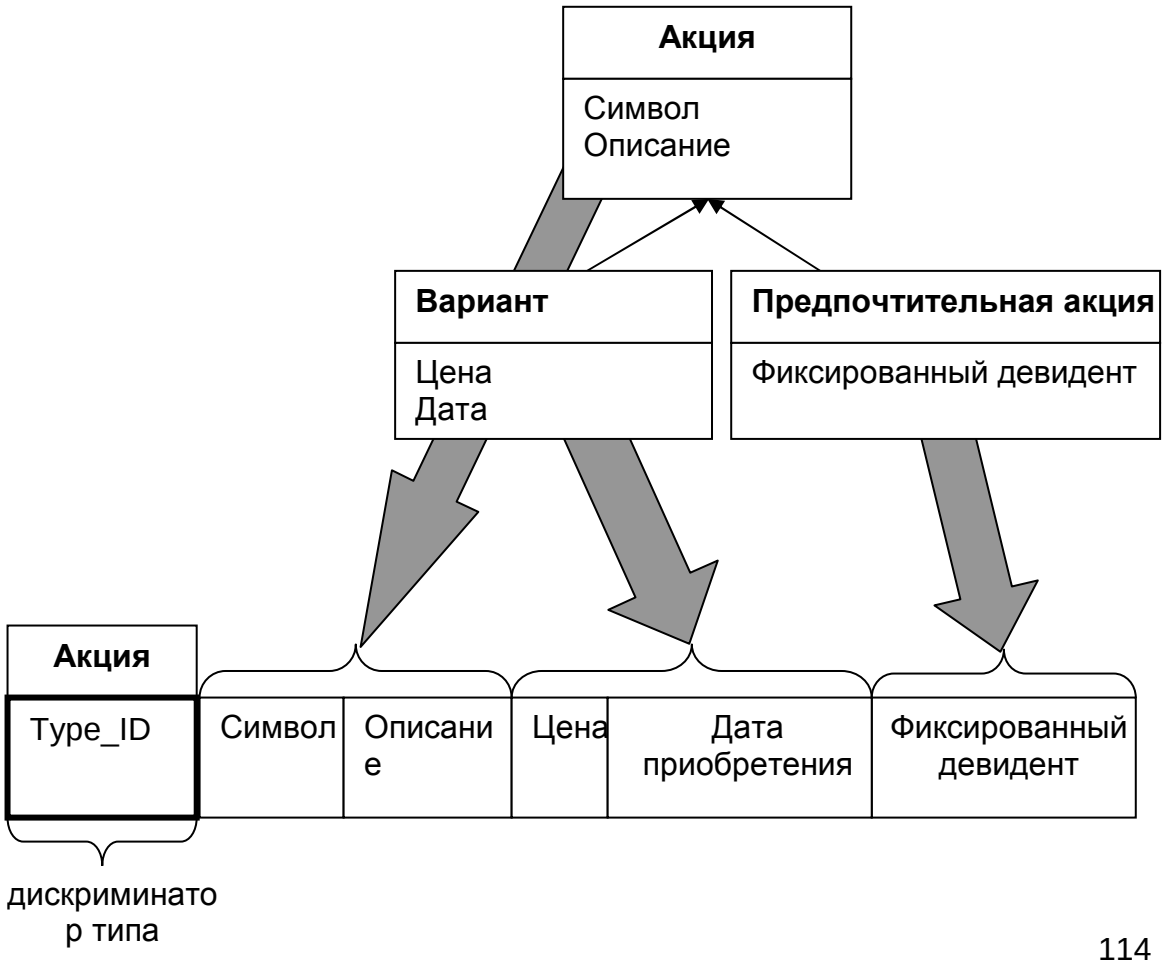


Рис.43. Унификация

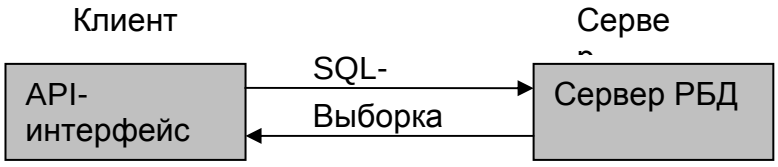


Рис.44. Взаимодействие клиента и сервера

Архитектура **сервера объектов** представлена на рис.45. Сервер объектов может получать запросы или на один объект (используя, к примеру, идентификатор объекта), или на набор объектов. Как показано на рис.45., связь между клиентом и сервером базируется на объектах, и кэш объектов, расположенный в оперативной памяти, поддерживается как на клиенте, так и на сервере (**Диспетчер буфера объектов**). Сами же объекты хранятся на диске, физически объединенные в страницы, но управление ими полностью осуществляется сервером, при этом клиенту практически ничего не известно об этом представлении.

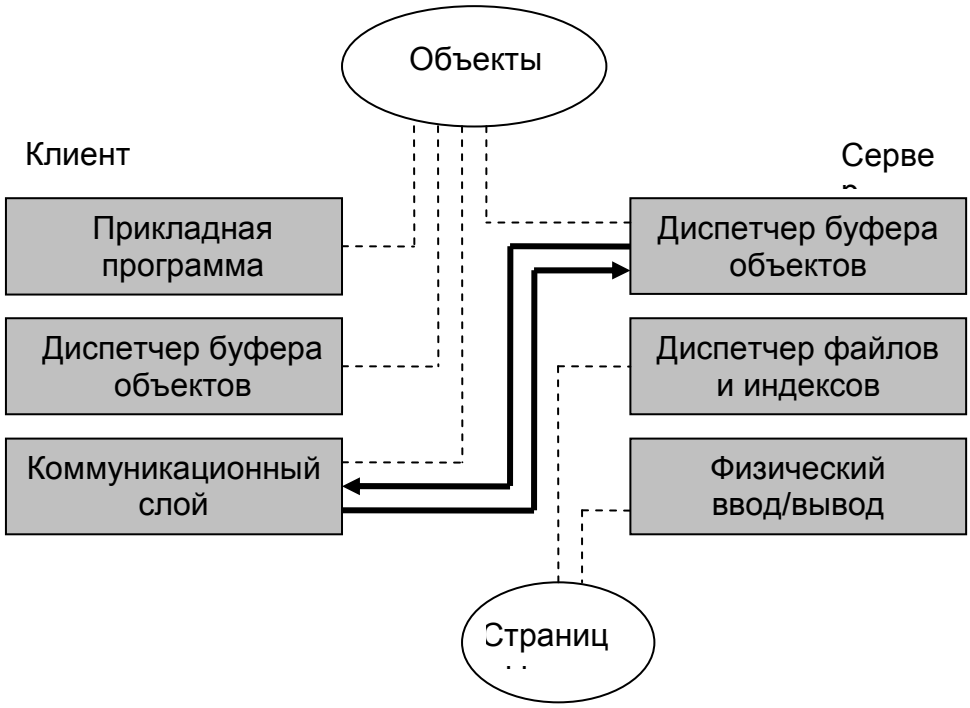


Рис.45. Механизм взаимодействия клиента и сервера объектов

Преимущество архитектуры сервера объектов состоит в том, что запросы, и, что более важно, методы могут исполняться как на клиенте, так и на сервере. Это значит, что взаимодействие между сервером и клиентом можно минимизировать, используя вспомогательные структуры данных, такие как индексы, с тем, чтобы проанализировать запросы на сервере. Управление блокировкой и ограничениями значительно упрощается, поскольку критически важные задачи могут выполняться на сервере. Архитектура сервера объектов позволяет реализовать предоставление полномочий и защиту того же уровня, что и в реляционных СУБД. Фактически, реляционную СУБД можно рассматривать как сервер объектов, который обслуживает примитивные объекты (кортежи в первой нормальной форме).

С другой стороны, серверы объектов имеют множество недостатков, что затрудняет их реализацию. К примеру, обработка запросов в случае обновления данных существенно усложняется, поскольку:

1. или измененные на клиенте данные должны передаваться на сервер перед исполнением там любого запроса,
2. или запросы должны параллельно исполняться на клиенте и на сервере, а полученные результаты — объединяться.

Первая стратегия обеспечивает слишком низкую производительность, и в случае обновления данных кэширование становится практически бесполезным.

Вторая стратегия порождает серьезные проблемы, связанные с обработкой распределенных запросов. Когда объект одновременно существует и на сервере, и на клиенте, возникает практический вопрос, связанный с объектно-ориентированными языками программирования, для работы которых требуется среда реального времени. Исполнение методов на сервере означает, что программа, реализующая этот метод, сама должна храниться в базе данных, а СУБД должна ”понимать” требования среды исполнения всех языков программирования, объекты которых размещаются в базе данных. Наконец, поскольку объекты хранятся на страницах жесткого диска сервера, сервер объектов должен упаковывать и распаковывать эти объекты для обмена ими с клиентами. Это увеличивает рабочую нагрузку сервера и может привести к появлению узких мест.

Производительность сервера объектов может измениться и за счет необходимости передавать полные объекты через API-интерфейс, что может значительно снизить скорость работы в случае больших объектов.

Альтернативой архитектуре сервера объектов является архитектура **сервера страниц** (рис.46.), где передача между клиентом и сервером осуществляется постранично. Сервер в архитектуре сервера страниц выполняет минимальные функции, такие как восстановление и хранение страниц. Объекты существуют только на клиенте.

Сервер страниц упрощает реализацию СУБД за счет максимального сокращения функций, выполняемых сервером, а высокая производительность может быть достигнута за счет удобной компоновки объектов на страницах. Сервер страниц минимизирует нагрузку сервера, и такая системная архитектура позволяет серверу обслуживать большее число клиентов.

Сервер страниц имеет ряд недостатков, проявляющихся, в частности в том, что клиенты значительно увеличиваются в размере, поскольку они должны обладать всей функциональностью базы данных. Размер сообщений при обращении к небольшим объектам может оказаться больше, чем это необходимо (так как страница является минимальной единицей обмена). Для вычисления результата запроса с сервера клиенту может пересылаться больше данных, чем требуется, потому что запросы нельзя передавать на сервер для удаленного исполнения. При использовании сервера страниц намного сложнее становится обеспечение поддержки блокировки на уровне объекта.

Сравнение сервера страниц и сервера объектов, показывает, что сервер в архитектуре сервера страниц менее функционален, чем в архитектуре сервера объектов, однако реализовать его намного проще.

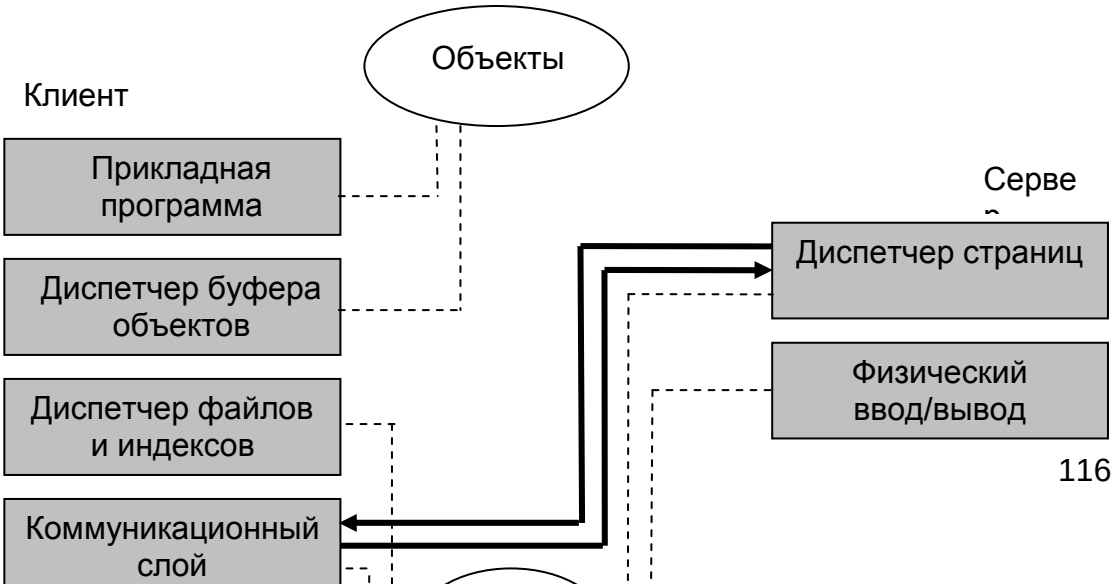


Рис.46. Механизм взаимодействия клиента и сервера страниц

КОНТРОЛЬНЫЕ ВОПРОСЫ ПО ВОСЬМОМУ РАЗДЕЛУ

1. Перечислите побудительные мотивы к началу исследований по созданию ООСУБД.
2. Каковы особенности объектной модели данных.
3. Каковы достоинства и недостатки ООСУБД.
4. В чем основные отличия ООСУБД от ОРСУБД.
5. Приведите способы преобразования иерархии типов в реляционную модель данных.
6. Опишите способы взаимодействия сервера и клиента, используемые в ООСУБД.

9. ВОПРОСЫ РАСПРЕДЕЛЕННЫХ БАЗ ДАННЫХ

9.1. Централизованные и децентрализованные СУБД

СУБД и централизация обработки информации позволили устранить такие недостатки традиционных файловых систем, как несвязность, несогласованность и избыточность данных. По мере роста баз данных и, особенно, при их использовании в территориально разделенных организациях появляются другие проблемы. Так, для централизованной СУБД, находящейся в узле телекоммуникационной сети, с помощью которой различные подразделения организации получают доступ к данным, с ростом объема информации и количества транзакций возникают следующие трудности:

- большой поток обмена данными,
- низкая надежность,
- низкая общая производительность,

Хотя для централизованной БД легче обеспечить **целостность** и **непротиворечивость** информации при обновлении, перечисленные проблемы создают определенные трудности.

В качестве возможного решения этих проблем предлагается **децентрализация** данных. При децентрализации достигается:

- повышение степени распараллеливания обработки вследствие распределения нагрузки,
- повышение эффективности использования данных на местах,

На рис.47. приведен пример логической схемы распределенной БД.

Рассмотрим приведенную систему подробнее.

Предположим, что существует система обработки банковских операций на основе сети. Клиент в городе А работает с системой с помощью **локальных** запросов (**географическая** локальность). Если клиент переезжает в город Б и хочет в банке этого города получить свой вклад, для проверки его счета в банке города А из банка города Б будет выдан **удаленный** запрос. Кроме того, большинство запросов головного учреждения являются удаленными (**глобальная** удаленность). На их основе строится протокол работы всей системы.

Между банками одного района могут также осуществляться обмены информацией. На основании локальных запросов, например, кредитные операции с фермерами этого района (функциональная локальность).

Заметим, что на рис.47. показана логическая схема **пользователя.** На этом уровне распределения БД представляет собой логическую совокупность локальных и удаленных (распределенных) данных.

Таким образом, можно сказать, что распределенная БД - это набор реляционных отношений, хранящихся в разных узлах информационной сети, логически связанных таким образом, чтобы составлять единую совокупность данных.

9.2. Стратегии хранения данных. Достоинства и недостатки

- **Централизация.** Единая копия БД располагается в одном узле.

Преимущества:
- простота поддержки целостности,
- простота обеспечения защиты,
- простота реализации,
- минимальная степень дублирования данных.

Недостатки:
- ограниченный объем внешней памяти,
- ограничения на надежность, определяемые надежностью работы узла,
- возможное значительное (а часто и недопустимое) время отклика на запрос при большом сетевом трафике (ограниченный доступ).

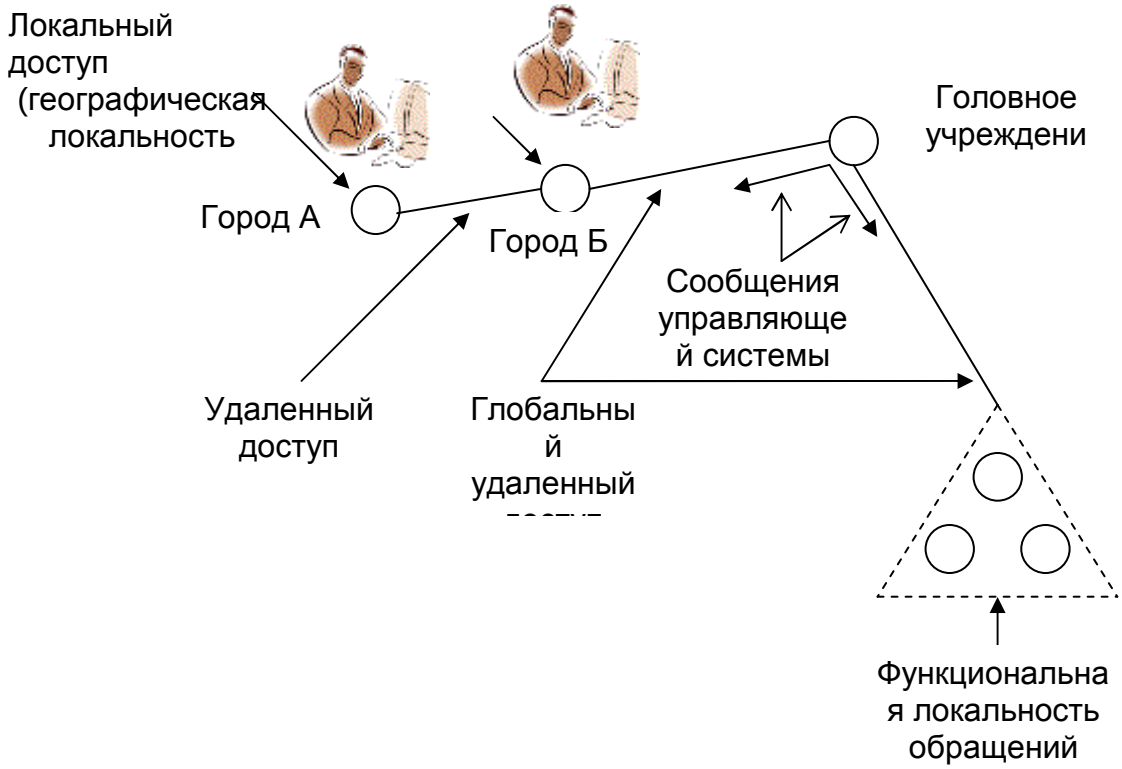


Рис.47. Методы доступа в распределенных базах данных

- **Распределение**. Различают два способа распределения данных с целью их приближения к месту использования и сокращения тем самым сетевого трафика - **репликация** и **фрагментация**.

Репликация (тиражирование, дублирование). Технология, которая предусматривает поддержку полных копий базы данных или некоторых ее фрагментов в нескольких узлах сети.

Полная репликация. Все данные дублируются в каждом узле.

Частичная репликация. Некоторые, часто используемые на удаленных узлах, данные могут дублироваться в соответствующих узлах.

Преимущества:
- максимальный уровень надежности,
- минимизация времени отклика при выборке данных (повышение степени доступности).

Недостатки:
- необходимость синхронизации при модификации БД.

Фрагментация (расщепление, разделение). Технология, предусматривающая разбиение полной базы данных каким-либо методом на непересекающиеся

составные части (фрагменты), хранимые в разных узлах сети. В свою очередь расщепление может быть **вертикальным -** выделение подмножества **полей** (**атрибутов**), и **горизонтальным** - выделение подмножества **записей** (**кортежей**).

Преимущества**:**

- максимальная внешняя память (суммарная для всех узлов сети),
- минимальное время отклика на локальный запрос,
- большая степень распараллеливания,
- высокая степень доступности и надежности.

Недостатки:

- высокая стоимость связи в запросах, касающихся многих «чужих» локальных БД.

Замечание. Ключевым фактором, влияющим на доступность и надежность БД, является **локализация ссылок**, то есть расположение запрашиваемых данных, исходя из удовлетворения запросов пользователей. Если БД распределена по сети так, что данные, расположенные в узле, запрашиваются почти исключительно пользователями этого узла, говорят, что существует **высокая** степень локализации ссылок. Очевидно, что надежность и доступность БД снижаются при понижении степени локализации ссылок.

Возможным подходом к распределению данных может быть и применение смешанной стратегии - сочетанию репликации и фрагментации.

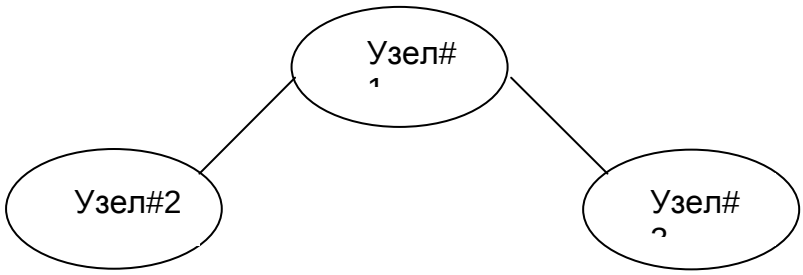
Пример стратегий распределения

Пусть схема распределенной по трем узлам БД состоит из трех отношений:

СЛУЖАЩИЕ(**слж**#, **фио**, **отд**#, **зарпл**, **рук**)

ОТДЕЛ(**подраздел**#, **отд**#)

РАЗМЕЩЕНИЕ(**подраздел**#, **город**)



В **Узел#1** находится головное учреждение и хранятся полные копии всех отношений (**СЛУЖАЩИЕ, ОТДЕЛ, РАЗМЕЩЕНИЕ).**

В **Узел#2** обрабатываются данные о зарплатах и налогах. Для этого нужны только атрибуты **слж**#, **фио**, **зарпл** из отношения **СЛУЖАЩИЕ.**

В **Узел#3** хранятся полные данные о служащих, зарабатывающих меньше 50000 руб. в год; об остальных служащих хранятся только сведения о номере отдела, в котором служащий работает, и руководителе отдела.

Построим в **Узел#2** отношение **СЛЖ1**(**слж**#, **фио**, **зарпл**) как проекцию:

СЛЖ1 = СЛУЖАЩИЕ[**слж**#, **фио**, **зарпл**]

Построим в **Узел#3** отношения

СЛЖ2(слж#, **фио**,**отд#,зарпл,рук)** и **СЛЖ3(слж**#, **фио**,**отд#,рук),**

используя операции селекции (выборки) и проекции:

СЛЖ2 = СЛУЖАЩИЕ WHERE зарпл < 50000

СЛЖ3 = СЛУЖАЩИЕ WHERE зарпл ≥ 50000[слж#, **фио**, **отд**#, **рук]**

Таким образом, в **Узел#1** хранятся полные кортежи, в **Узел#2** - вертикальный фрагмент (расщепление по вертикали), в **Узел#3 -** горизонтальный и смешанный фрагменты отношения **СЛУЖАЩИЕ.**

Замечание. Правильное восстановление полных кортежей отношения **СЛУЖАЩИЕ** по кортежам отношений **СЛЖ1, СЛЖ2** и **СЛЖ3** осуществляется с использованием значений ключевого атрибута **слж**#.

9.3. Проблемы распределенных баз данных

- **Логическая прозрачность.** Необходимость наличия единой концептуальной схемы распределенной по сети БД, другими словами, необходимость наличия общей модели данных распределенной системы. Принцип логической прозрачности позволяет пользователю формировать запросы ко всем данным распределенной БД так, как если бы он работал с централизованной базой.
- **Прозрачность размещения**. Выполнение данного принципа позволяет пользователю при формировании запроса не указывать местоположение узлов, откуда необходимо получить данные для удовлетворения запроса. Очевидно, что реализация прозрачности размещения требует наличия схемы, определяющей местонахождение данных в сети.
- **Прозрачность преобразования**. Распределенные БД могут быть однородными или неоднородными в смысле использования аппаратных и программных (СУБД) средств. Проблема неоднородности аппаратной, но однородности программной решается сравнительно просто. Если же в узлах сети используются разные СУБД, необходимы средства преобразования структур данных и языков.
- **Управление словарями**. Для обеспечения всех видов прозрачности в распределенной БД нужны программы, управляющие многочисленными справочниками и словарями.
- **Координация процессов.** Методы выполнения запросов в распределенных БД отличаются от аналогичных методов централизованных СУБД, поскольку отдельные части запроса нужно выполнять на месте расположения соответствующих данных и передавать частичные результаты на другие узлы, при этом должна быть обеспечена координация всех процессов.
- **Непротиворечивость данных**. Необходим сложный механизм управления одновременной обработкой, который, в частности, должен обеспечивать синхронизацию при обновлениях информации, что гарантирует непротиворечивость данных.
- **Развитая методика репликации** и **расщепления** данных.

Выполнение запросов в распределенных базах данных. Поскольку данные распределены, а также могут быть расщеплены, запросы, составленные пользователем, который рассматривает данные целиком, должны быть приведены к виду, учитывающему разделение и расщепление данных. Таким образом, требуется декомпозиция запроса, так как программа, реализующая запрос, не может быть выполнена полностью в одном узле. Кроме того, поскольку для завершения выполнения программы необходимо перемещать целые отношения (файлы) или результаты промежуточных вычислений, необходимо оптимизировать:

- саму программу,
- методику выполнения программы (запроса) с учетом необходимых перемещений данных.

9.4. Одновременная работа

В связи с управлением одновременной работой СУБД по выполнению запросов от разных пользователей, рассматриваются понятия **транзакции** и **расписания**.

Транзакция - это разовый прогон программы, реализующей запрос (другими словами, такая единица работы), при котором БД остается в состоянии целостности до и после выполнения транзакции.

Замечание. В процессе выполнения транзакции целостность БД может нарушаться.

Расписанием совокупности транзакций будем называть порядок, в котором выполняются элементарные шаги этих транзакций.

Очевидно, что расписание при одновременной работе представляет порядок выполнения транзакций во времени.

Примеры расписаний. Введем следующие обозначения:

T - транзакция**,**

R-A - элементарный шаг транзакции, представляющий чтение значения поля A,

W-A - элементарный шаг транзакции, представляющий запись значения в поле A.

Расписание 1.

T1 : R-A
T1 : W-A
T2 : R-A
T2 : W-A

Нарушения целостности (рассогласования) БД возникнуть не может, так как транзакции T1 и T2 выполняются последовательно.

Расписание 2.

T1 : R-A
T2 : R-A
T1 : W-A
T2 : W-A

Модификация транзакции T1 затирается транзакцией T2, следовательно, она теряется.

Расписание 3.

T1 : W-A
T2 : W-A
T1 : отменить

Потеря модификации транзакции T2, поскольку после ее окончания следует сигнал отмены модификации.

Расписание 4.

T1 : W-A
T2 : R-A
T1 : прервать

Ситуация известна под названием «неправильное считывание», поскольку выбранное транзакцией T2 значение впоследствии из БД удаляется.

Расписание 5

T2: R-A
T1: W-A

T2 : R-A

Две выборки дадут различные значения.

Пример рассогласования БД.

Пусть транзакции T1 и T2 имеют вид:

R-A

A=A+1

W-A

Шаги выполнения транзакций приведены в следующей таблице

Такты	1	2	3	4	5	6
Значение A в БД	5	5	5	5	6	6
T1	R-A		A=A+1			W-A
T2		R-A		A=A+1	W-A	
Значение A в раб. пространстве T1	5	5	6	6	6	6
Значение A в раб. пространстве T2		5	5	6	6	

Очевидно, что целостность БД нарушена, поскольку нельзя было допустить чтение и модификацию поля A транзакцией T2 до того, пока транзакция T1 свое выполнение не закончит, то есть до записи модифицированного транзакцией T1 значения в поле A.

Согласованные состояния БД обеспечивают **последовательные** или **приводимые к последовательным** расписания.

Расписание называется **последовательным**, если все транзакции выполняются строго друг за другом.

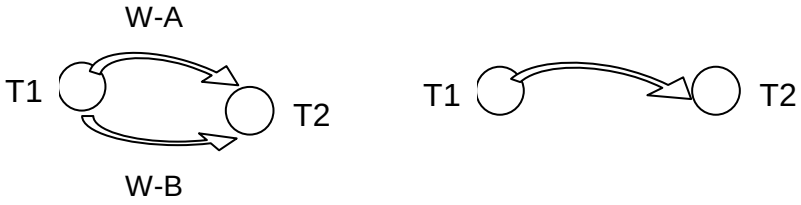
Расписание называется **приводимым к последовательному**, если результат применения этого расписания к БД эквивалентен результату последовательного расписания.

Для выявления расписаний, приводимых к последовательным и благодаря этому сохраняющих согласованное состояние БД, можно использовать метод, основанный на построении **графа зависимостей.**

В графе зависимостей транзакциям соответствуют узлы, а направленная дуга графа, связывающая узлы Ti и Tj , обозначает, что вход транзакции Tj зависит от выхода транзакции Ti , то есть Tj использует значение поля, определенное Ti. Если граф зависимостей содержит циклы, то это означает, что соответствующее расписание **не** приводится к последовательному.

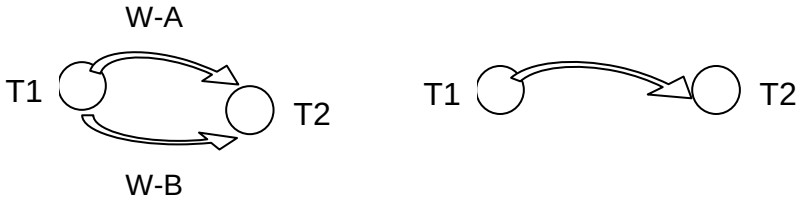
Пример 1. **Расписание последовательное**

T1 : W-A
T1 : W-B
T2 : W-A
T2 : W-B

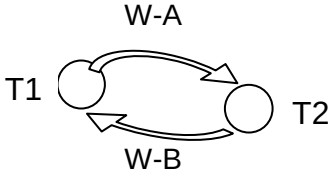


Пример 2. **Расписание, приводимое к последовательному**

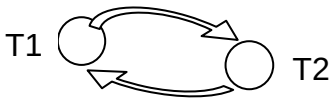
T1 : W-A
T2 : W-A
T1 : W-B
T2 : W-B



Пример 3. Расписание, не приводимое к последовательному



T1 : W-A
T2 : W-A
T2 : W-B
T1 : W-B



9.5. Управление блокированием

В распределенных базах данных в общем случае одновременно выполняются несколько транзакций; при этом могут использоваться одни и те же данные, возможно, продублированные в разных узлах. При чтении для получения нужной информации достаточно **одной** копии, если СУБД обеспечивает непротиворечивость всех копий. При обновлениях (записи в БД) для сохранения непротиворечивости **все** копии должны быть своевременно модифицированы.

Очевидно, что обновления нужно производить в соответствии с последовательными протоколами. Если учитывать одновременное выполнение транзакций над многочисленными копиями, последовательность обновлений требует развитых средств **синхронизации**, которая прежде всего предполагает монопольное владение поля транзакцией до полного окончания использования этого поля.

Монопольное использование значения поля в процессе работы СУБД можно обеспечить с помощью блокировки (захвата). При этом предполагается, что любая транзакция представима в виде последовательности

ЗАХВАТ - ДЕЙСТВИЕ - ОСВОБОЖДЕНИЕ

Управление блокированием - это функциональный элемент СУБД, который назначает и регистрирует блокирование, а также играет роль арбитра между несколькими запросами на блокировку одного элемента. Очевидно, что блокировка действует как примитивная синхронизация выполнения транзакций. То есть, если транзакция пытается блокировать (ЗАХВАТ) что-то уже заблокированное, она должна ждать, пока блокировка не будет снята (ОСВОБОЖДЕИЕ) транзакцией, установившей блокировку.

Итак, всякая транзакция в конце концов должна разблокировать все, что она ранее заблокировала.

Таким образом, в распределенных БД необходимо обеспечить передачу следующих сообщений о блокировках:

1) послать запросы на блокировку во все узлы,

2) получить подтверждение о реализации блокирования (при поступлении запроса на блокировку прежде всего проверяется, не заблокирован ли элемент данных другой транзакцией; если элемент не заблокирован, он блокируется),

3) после завершения операции (чтения или обновления) требуется посылка запросов на снятие блокировки во все узлы, где хранится информация.

Еще раз отметим, что при работе с блокировками транзакции чтения достаточно блокировать одну копию данных, находящуюся в соответствующем узле сети, а транзакция обновления должна блокировать все копии данных, находящихся в разных узлах сети. При этом, вообще говоря, транзакция может прочитать элемент данных из любой копии, содержащей этот элемент, даже если она блокирована по чтению другой транзакцией. Однако обновить элемент транзакция может только в том случае, если блокированы по записи все копии, содержащие этот элемент данных, именно этой транзакцией. Блокировка по чтению (возможность чтения) будет обеспечена до тех пор, пока другая транзакция не установит блокировку по записи. Другими словами, для одного и того же элемента данных не могут одновременно существовать блокировки по чтению и записи. Таким образом, блокировка по чтению

необходима, в общем случае, только для того, чтобы до завершения чтения никакая транзакция не могла модифицировать этот элемент данных.

9.6. Методы синхронизации распределенных обновлений

Метод 1. Использование **графа предшествования.** В графе предшествования узлам соответствуют транзакции, а ребрам - сведения о блокировках и освобождениях данных. Направленное ребро выходит из узла, который снимает блокировку элемента данных (Освободить), и ведет к узлу, который эти данные блокирует (Захватить). Наличие циклов в таком графе означает существование тупиковых ситуаций, когда ни одна транзакция не закончена, и все они находятся в состоянии ожидания разблокирования элементов данных, необходимых для продолжения их выполнения.

Пример. Пусть имеется следующая последовательность запросов и действий транзакций, представленная на рис.49. (A, B, C - элементы данных):

Таким образом, тупик (цикл) возникает, если для двух различных элементов данных необходимо исполнение двух транзакций в разном порядке.

Очевидно, что порядок, в котором различные транзакции блокируют элемент данных, должен соответствовать приводимому к последовательному расписанию.

В схеме с основным узлом установка и снятие блокировок для всех транзакций происходят для основного узла. Когда поступает запрос на блокировку, проверяется, не заблокирован ли элемент данных, находящийся в основном узле, другой транзакцией. Если элемент не заблокирован, он блокируется.

При синхронизации распределенных обновлений заблокировать можно не один основной узел, а несколько узлов. В этом случае в каждом узле находится свой локальный граф предшествования, в котором связь с остальной частью сети представлена специальной вершиной.

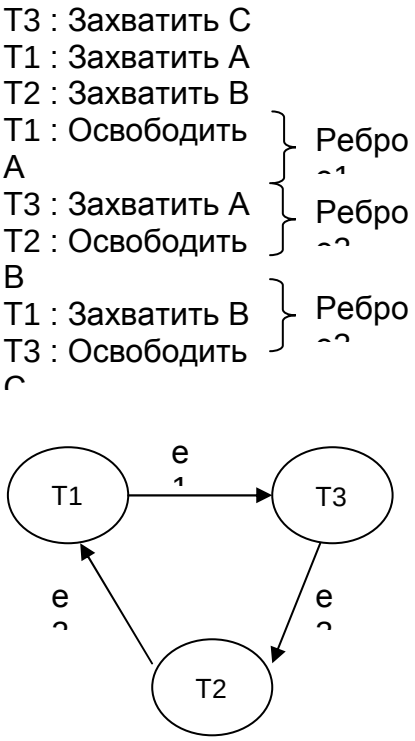


Рис.49. Расписание и соответствующий граф предшествования

В этой схеме возможна глобальная тупиковая ситуация, хотя ни один из локальных графов не содержит циклов. Для обнаружения подобной ситуации узлы связывают свои локальные графы попарно через специальные внешние вершины до тех пор, пока не будет обнаружена тупиковая ситуация.

При синхронизации распределенных обновлений на основе блокировок в центральном узле может находиться общий граф предшествования. В этом случае все узлы посылают запросы на установку и снятие блокировок в центральный узел.

Метод 2. Временные метки. При использовании временных меток исключается возможность возникновения тупиковых ситуаций.

Каждой транзакции при входе в сеть присваивается уникальная метка, например, показание часов глобальной сети, дополненное идентификатором сети. Каждый элемент в БД имеет временные метки последних выполненных над ним транзакций чтения (метка чтения) и обновления (метка записи).

Если транзакция T1 запрашивает операцию, которая вступает в конфликт с операцией, уже выполняемой в соответствии с более поздней по времени транзакцией T2, T1 запускается вновь.

Конфликт между T1 и T2 возникает:

Случай 1. Если операция транзакции T1 есть операция чтения, но объект уже записан транзакцией T2.

Анализ ситуации:

1) объект прочитан транзакцией T2, то есть своего состояния не изменил, и конфликт не возникает;

2) конфликт может возникнуть, поскольку объект записан транзакцией T2, но транзакция T1 не знает, что объект изменен и может быть настроена на обработку объекта, находящегося в состоянии до его изменения транзакцией T2.

Случай 2. Если операция транзакции T1 есть операция записи, но объект уже был записан или прочитан транзакцией T2.

Анализ ситуации:

Конфликт может возникнуть в любом случае, поскольку транзакция Т2 не знает, что состояние объекта изменено транзакцией Т1.

Если транзакция перезапускается, ей присваивается новая временная метка.

Замечание. Хотя тупиковые ситуации исключаются, могут возникать избыточные отказы и перезапуски (рестарты), например, последовательность действий R_{T2} W_{T1} R_{T2} может быть абсолютно корректной, но приводит к рестарту транзакции Т1 (случай 2).

Метод 3. **Системы с голосованием.** Каждому узлу дана возможность «голосовать» в пользу обновления. Известны два подхода.

Подход 1. Запрос на обновление выполняется, если за проведение обновления проголосовали **все** узлы.

Подход 2. Запрос на обновление выполняется, если за проведение обновления проголосовало **большинство** узлов.

Замечание. Считается, что ни один узел не может одновременно голосовать за выполнение двух конфликтных транзакций.

При наличии запроса на обновление узел может выдать один из следующих сигналов:

1) проведение обновления,
2) отклонение обновления,
3) возможна тупиковая ситуация,
4) узел воздерживается от голосования по данному запросу.

9.7. Завершение транзакции. Журнал транзакций

Возможны два варианта завершения транзакции.

Вариант 1. Все операции транзакции выполнены успешно и в процессе выполнения не произошло никаких сбоев программного или аппаратного обеспечения. В этом случае транзакция **фиксируется**, то есть, производится запись во внешнюю память изменений в БД, которые были сделаны в процессе выполнения транзакции.

Замечание. До тех пор, пока транзакция не зафиксирована, допустимо аннулирование этих изменений путем восстановления БД в то состояние, в котором она была на момент начала выполнения транзакции.

Вариант 2. Если в процессе выполнения транзакции возникла ситуация, которая делает невозможным ее нормальное завершение, БД должна быть возвращена в исходное состояние, то есть, должен быть произведен **откат.** Таким образом, **откат** транзакции - это совокупность действий, обеспечивающих аннулирование всех изменений данных в БД, которые были сделаны операторами SQL в теле незавершенной транзакции.

Согласно стандартным соглашениям, транзакция завершается одним из 4-х возможны путей:

1) выполнение оператора COMMIT - фиксация успешного завершения транзакции;
2) успешное завершение программы, в которой было инициировано выполнение транзакции;
3) выполнение оператора ROLLBACK - реализация отката;
4) аварийное завершение программы, в которой было инициировано выполнение транзакции.

Реализация в СУБД принципа сохранения промежуточных состояний, подтверждения или отката транзакции обеспечивается специальным механизмом,

для поддержки которого создается некоторая системная структура, называемая **журналом транзакций**. Однако, назначение журнала транзакций гораздо шире. Он предназначен для обеспечения надежного хранения данных в БД.

Возможны следующие ситуации, при которых требуется произвести восстановление состояние БД.

1) Индивидуальный откат транзакции:

a) стандартная ситуация отката и явное аварийное завершение транзакции по оператору ROLLBACK;

b) аварийное завершение работы прикладной программы, которое логически эквивалентно выполнению оператора ROLLBACK, но физически имеет иной механизм выполнения;

c) принудительный откат транзакции в случае взаимной блокировки при параллельном выполнении транзакций; в подобном случае для выхода из тупика данная транзакция может быть выбрана в качестве «жертвы».

2) Восстановление после внезапной потери содержимого оперативной памяти (мягкий сбой):

a) при аварийном выключении питания;

b) неустранимый сбой процессора и, таким образом, потеря данных, находящихся в оперативной памяти.

3) Восстановление после поломки основного внешнего носителя БД (жесткий сбой).

9.8. Свойства транзакций

Существуют различные модели транзакций, которые могут быть классифицированы на основании различных свойств, включающих:

- структуру транзакции,
- параллельность внутри транзакции,
- продолжительность и т.п.

Стандартно выделяют 4 классических свойства транзакций:

Атомарность (**A**tomicity) - транзакция должна быть выполнена в целом или не выполнена вовсе.

Согласованность (**C**onsistency) - гарантирует, что по мере выполнения транзакции данные переходят из одного согласованного состояния в другое, то есть транзакция не разрушает взаимной согласованности данных.

Изолированность (**I**solation) - конкурирующие за доступ к БД транзакции физически обрабатываются последовательно, изолированно друг от друга, но для пользователей это выглядит так, как будто они выполняются параллельно.

Долговечность (**D**urability) - если транзакция завершена успешно, то те изменения в данных, которые были ею произведены, не могут быть потеряны ни при каких обстоятельствах (даже в случае последующих ошибок).

Традиционные (плоские) транзакции, обладающие всеми стандартными свойствами, иногда называют ACID-транзакциями.

КОНТРОЛЬНЫЕ ВОПРОСЫ ПО ДЕВЯТОМУ РАЗДЕЛУ

1. Перечислите модели доступа и основные стратегии распределения данных, их достоинства и недостатки.
2. Перечислите проблемы распределенных баз данных.

3. Перечислите свойства транзакций. Варианты завершения транзакции.

4. Сформулируйте понятие расписания. Приведите примеры рассогласования.

5. Управление блокированием. Перечислите основные методы синхронизации распределенных обновлений.

6. Перечислите свойства транзакций

ЛИТЕРАТУРА

Основная:

1. Дейт К. Введение в системы баз данных. 7-е издание: Пер. с англ. – М.: Вильямс, 2001.

2. Тарасенко Ф.П. Прикладной системный анализ. (Наука и искусство решения проблем): Учебник. – Томск: Изд-во Том. ун-та, 2004.

3. Тиори Т., Фрай Дж. Проектирование структур баз данных. В 2-х кн.: Пер с анл. – М.: Мир, 1985.

Дополнительная:

1. Гарсиа-Молина Г., Ульман Д., Уидом Д. Системы баз данных. Полный курс: Пер. с англ. – М.: Вильямс, 2003.

2. Когаловский М.Р. Энциклопедия технологий баз данных. – М.: Финансы и статистика, 2002..

3. Кренке Д. Теория и практика построения баз данных: Пер. с англ. – СПб.: Питер, 2003.

4. Цикритзис Д., Лоховски Ф. Модели данных: Пер. с англ. - М.: Финансы и статистика, 1985.

5. Ульман Дж., Видом Дж. Введение в системы баз данных: Пер. с англ. – М.: ЛОРИ, 2000.