МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

И.Л. Чудинов, В.В. Осипова

БАЗЫ ДАННЫХ

*Рекомендовано УМО РАЕ по классическому университетскому
и техническому образованию в качестве учебного пособия
для студентов высших учебных заведений, обучающихся
по направлению подготовки 230700 «Прикладная информатика».
Профиль «Прикладная информатика (в экономике)» (бакалавриат)*

Издательство
Томского политехнического университета
2012

УДК 004.65(075.8)
ББК 32.973.26-018.2я73
Ч-84

Чудинов И.Л.
Ч-84 Базы данных: учебное пособие / И.Л. Чудинов, В.В. Осипова; Томский политехнический университет. – Томск: Изд-во Томского политехнического университета, 2012. – 140 с.

ISBN 978-5-4387-0115-6

Пособие содержит обоснование концепции баз данных, описание роли и функций, обеспечиваемых системой управления базами данных (СУБД). Дается описание особенностей классических структур данных, применяемых для организации БД, среди которых выделена реляционная модель данных, положенная в основу большинства современных СУБД. Особое внимание уделено процессу проектирования БД: рассмотрены этапы проектирования БД, а также используемые при этом методики и модели.

Предназначено для студентов заочного и дистанционного обучения по направлению «Информационные системы в экономике».

УДК 004.65(075.8)
ББК 32.973.26-018.2я73

Рецензенты

Доктор технических наук, профессор ТУСУРа
А.А. Шелупанов

Доктор технических наук, профессор ТУСУРа
Ю.П. Ехлаков

ISBN 978-5-4387-0115-6

СОДЕРЖАНИЕ

1. КОНЦЕПЦИЯ БАЗ ДАННЫХ

1.1. Обоснование концепции баз данных. История вопроса

История возникновения и развития подхода к организации хранения и обработки баз данных (БД) насчитывает порядка 40 лет и органически связана с историей использования компьютеров.

На первом этапе (50-е годы) на компьютерах, их тогда называли электронными вычислительными машинами (**ЭВМ**), выполнялись лишь **вычислительные** задачи, имеющих следующие характеристики:

- объем исходных данных намного меньше, чем объем оперативной памяти компьютера ($V_{\text{исх. данных}} << V_{\text{озу}}$);
- исходные данные – числа;
- время обработки намного больше, чем время ввода исходных данных и вывода результатов обработки ($T_{\text{обр}} >> T_{\text{обмена}}$);
- исходные данные размещаются вместе с программой, описания данных располагаются в программе;
- однократное использование исходных данных;
- форма результата не имеет особого значения.

Типичными примерами решаемых в то время задач является решение систем уровней обыкновенных и дифференциальных уравнений различными численными методами.

Тогда становятся понятными приведенные характеристики таких задач. Если система состоит из нескольких десятков уравнений, то время на ее обработку, даже с помощью компьютера, достаточно велико, в то время как число исходных данных (коэффициентов при неизвестных) будет равно нескольким сотням чисел, а в результате решения получим несколько десятков чисел. Естественно утверждать, что повторно решать систему уравнений с теми же коэффициентами нет смысла, а форма представления результата также не имеет особого значения.

Следует заметить, что компьютеры этого периода (их часто называют первым поколением ЭВМ) имели следующие характеристики:

- техническая элементная база – электронные лампы;
- занимаемая площадь – несколько сотен квадратных метров;
- вес – несколько тонн;
- быстродействие – несколько десятков тысяч операций в секунду;
- объем оперативной памяти в пересчете на современную меру – около 25 Кб;

- внешних запоминающих устройств нет;
- внешние машинно-ориентированные носители информации – перфокарты, перфоленты.

Второй период применения средств вычислительной техники (60-е годы) характеризуется переходом к новому поколению ЭВМ и появлением нового класса задач, которые назовем **информационными**. Они имеют следующие параметры:

- объем исходных данных больше объема оперативной памяти ($V_{\text{исх. данных}} > V_{\text{озу}}$);
- появление нечисловых, символьных данных, совместное хранение с числовыми в виде структур – записей;
- исходные данные хранятся в файлах автономно от программы обработки. Программа содержит описание структуры записи файла;
- время обработки меньше времени ввода-вывода данных ($T_{\text{обр}} < T_{\text{обмена}}$);
- файл используется многократно;
- результат оформляется в виде документа, удобного для анализа пользователем.

Типичными информационными задачами являются задачи в сфере организационного управления. Примером такой задачи может служить расчет конкурса (число заявлений на одно место) в процессе приема заявлений от абитуриентов вуза. Для решения такой задачи достаточно двух файлов F1 – ПЛАН и F2 – ЗАЯВЛЕНИЯ со структурой, приведенной на рис. 1.1.

Необходимость автономного хранения исходных данных (например, файл F2) обуславливается не только значительным объемом, но и необходимостью независимой его корректировки по мере поступления новых заявлений или изменения значения данных (например, факультета, на который подает заявление абитуриент). Для обеспечения такой корректировки обычно использовалась специальная программа, осуществляющая ввод данных с перфокарт или с перфоленты (рис. 1.2). Многократное использование также очевидно, так как для органов управления важно знать конкурс не реже одного раза в день. При этом программа выполняется даже в том случае, если за предшествующий день не было подано ни одного заявления.

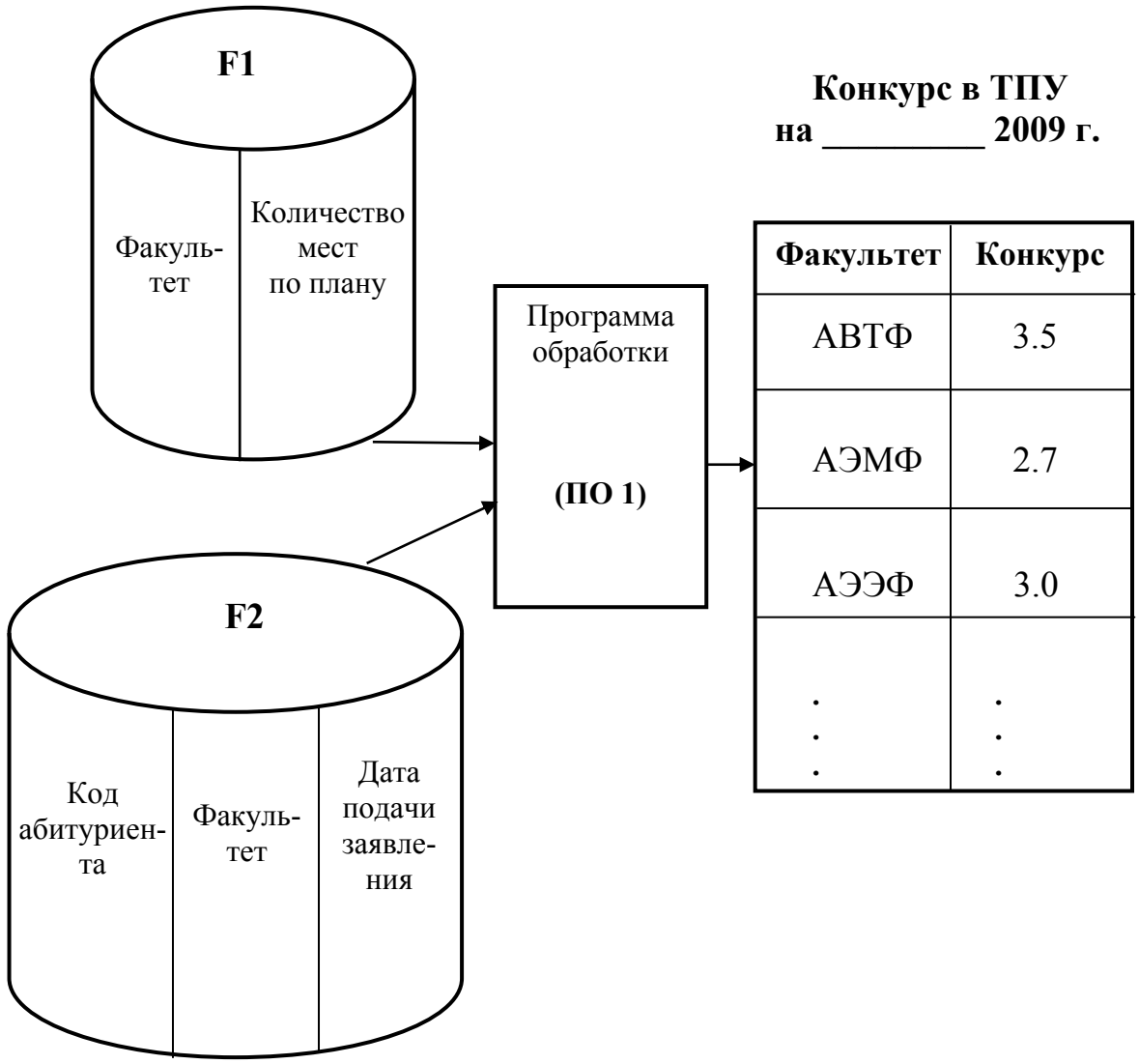


Рис. 1.1. Пример информационной задачи в сфере организационного управления

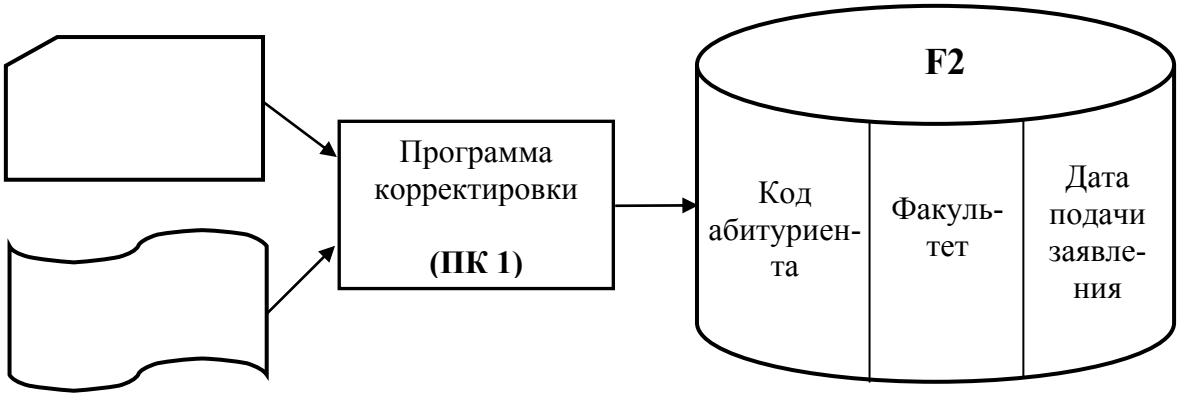


Рис. 1.2. Технология обновления файла «Заявление»

Характерными параметрами ЭВМ второго поколения, существенно влияющими на характеристики информационных задач, являются:

- элементная база – полупроводниковые приборы;
- быстродействие – сотни тысяч операций в секунду;
- объем оперативной памяти – около 250 Кб;
- внешние запоминающие устройства – магнитные ленты, магнитные барабаны и другие внешние устройства, например: алфавитно-цифровое печатающее устройство, устройства передачи информации по каналам связи.

Более совершенные характеристики ЭВМ вступили в противоречие с особенностью информационных задач в том смысле, что при высоком быстродействии вычислений данные информационных задач располагались на «медленных» магнитных запоминающих устройствах. Это противоречие привело к созданию операционных систем, обеспечивающих параллельную работу вычислительной задачи (используется «быстрый» вычислитель) и информационной задачи (используются автономные средства обмена с внешней памятью).

К концу 60-х годов широкое применение информационных задач привело к созданию нового подхода к использованию данных – концепции баз данных.

Дело в том, что в то время преобладал так называемый **позадачный подход** в использовании исходной информации, который характеризуется тем, что для каждой программы обработки используются свои файлы исходных данных.

Так, при необходимости получить документы о распределении абитуриентов по регионам, откуда они прибыли для поступления в вуз, и по видам до вузовского образования имеем две задачи и соответствующие им программы, каждая со своим файлом, что иллюстрируется на рис. 1.3.

Если эти программы создавались в одно и то же время и одним программистом, наверное, использовался бы один файл (например, в F3 добавились бы данные «Окончил учебное заведение»). Однако такие единичные случаи лишь подтверждают принцип позадачного подхода.

При автономном хранении файлов для каждого файла требуется и своя программа корректировки (ПК).

Обобщенную идею позадачного подхода (переходя к абстрактным именам данных) иллюстрируем на рис. 1.4. У каждой программы (программиста) свой файл.

Конструкции **ABC, AD, ABDEF, ACFG** блоков программ обработки имитируют описания обрабатываемых файлов (состав атрибутов записи

файла), которые должны быть приведены в теле программ. Очевидный основной недостаток позадачного подхода – дублирование исходных данных в различных файлах (B, C, D в файлах F1, F2, F3, F4), что приводит к серьезным проблемам при их обновлении (новые заявления абитуриентов должны учитываться во всех файлах), особенно при изменении значений в ранее введенных записях.

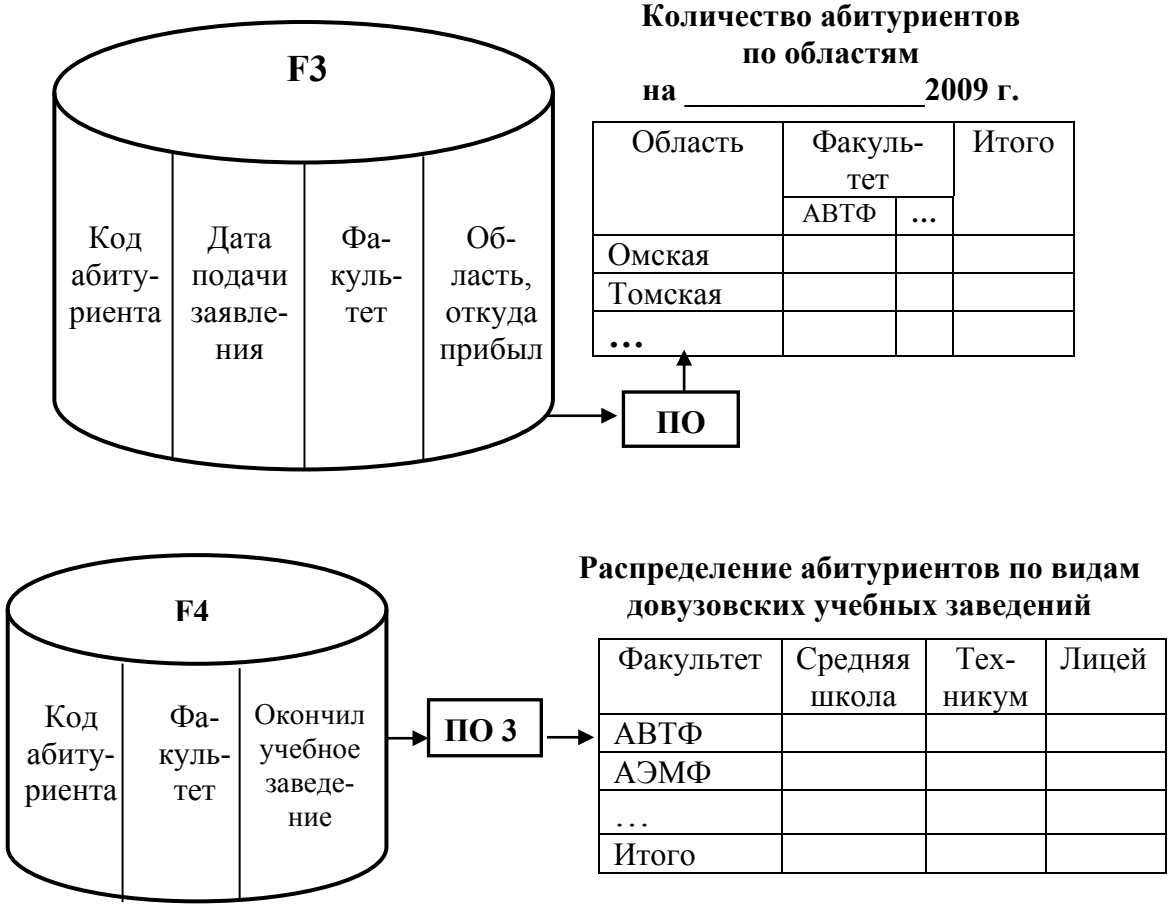


Рис. 1.3. Примеры задач, использующих автономные файлы

Конечно, в реальной ситуации для вновь создаваемых программ старались использовать уже существующие файлы (например, для программы ПО3), однако, если не все требуемые исходные данные имелись в существующих файлах, появилась дополнительная проблема использования нескольких исходных файлов, связанная с необходимостью согласования записей различных файлов. Положение усугублялось, если требовался файл, созданный другим программистом.

Если несколько программ создавались одновременно, то для них мог создаваться единый файл (например, файл F4 для программ ПО4-ПО6), но такой файл мог быть избыточен для каждой программы («лишние» атрибуты выделены).

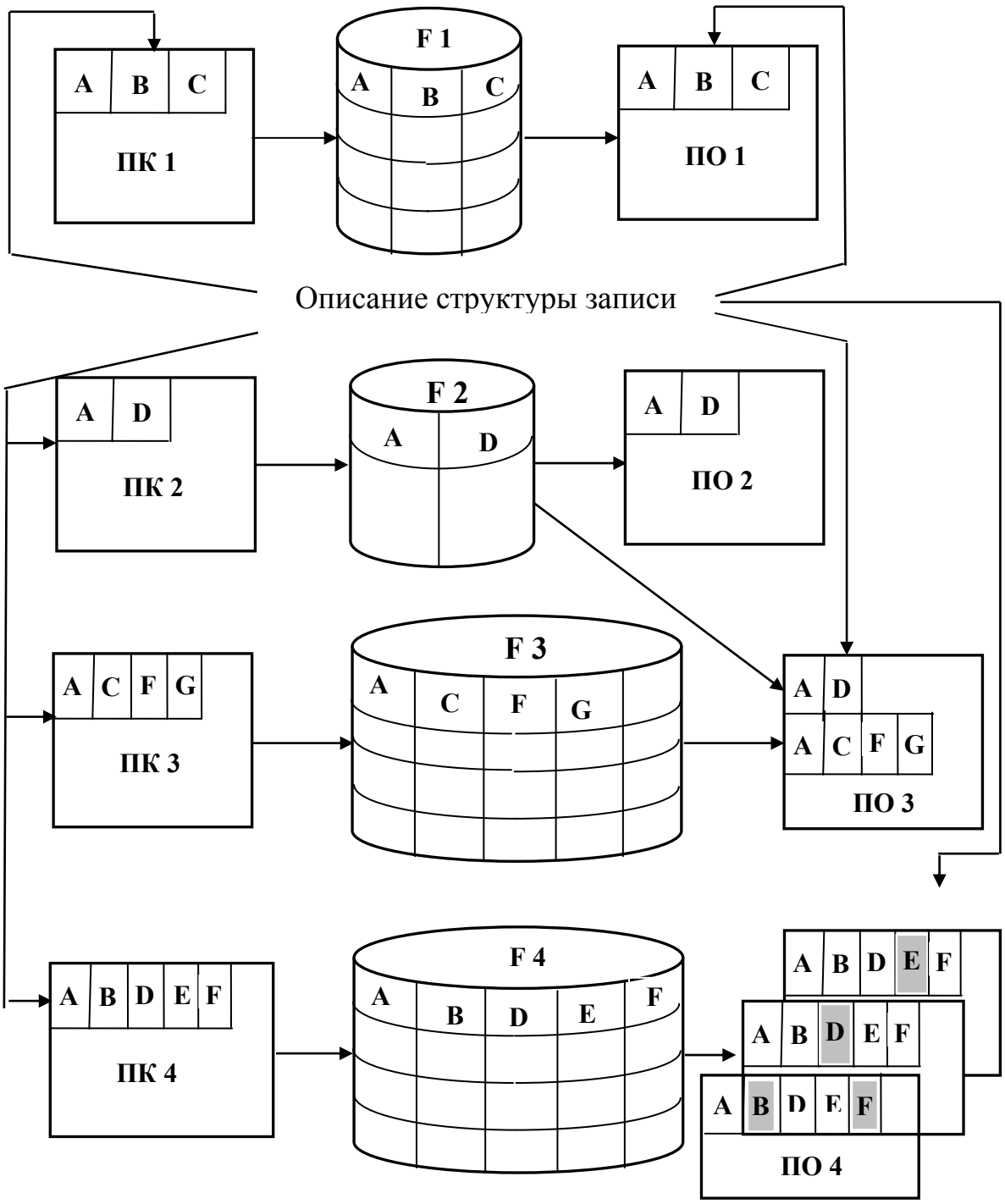


Рис. 1.4. Позадачный подход в обработке файлов

Практика применения такого подхода показала, что при наличии нескольких десятков программ, обрабатывающих файлы, относящиеся к одной предметной области, возникают трудно преодолимые проблемы с обеспечением достоверности исходных данных в приемлемое время (из-за дублирования).

Примерно в середине 60-х годов естественно возникло предложение использовать для описания объектов одного типа единый, пусть большой, файл исходных данных для всех программ обработки (рис. 1.5).

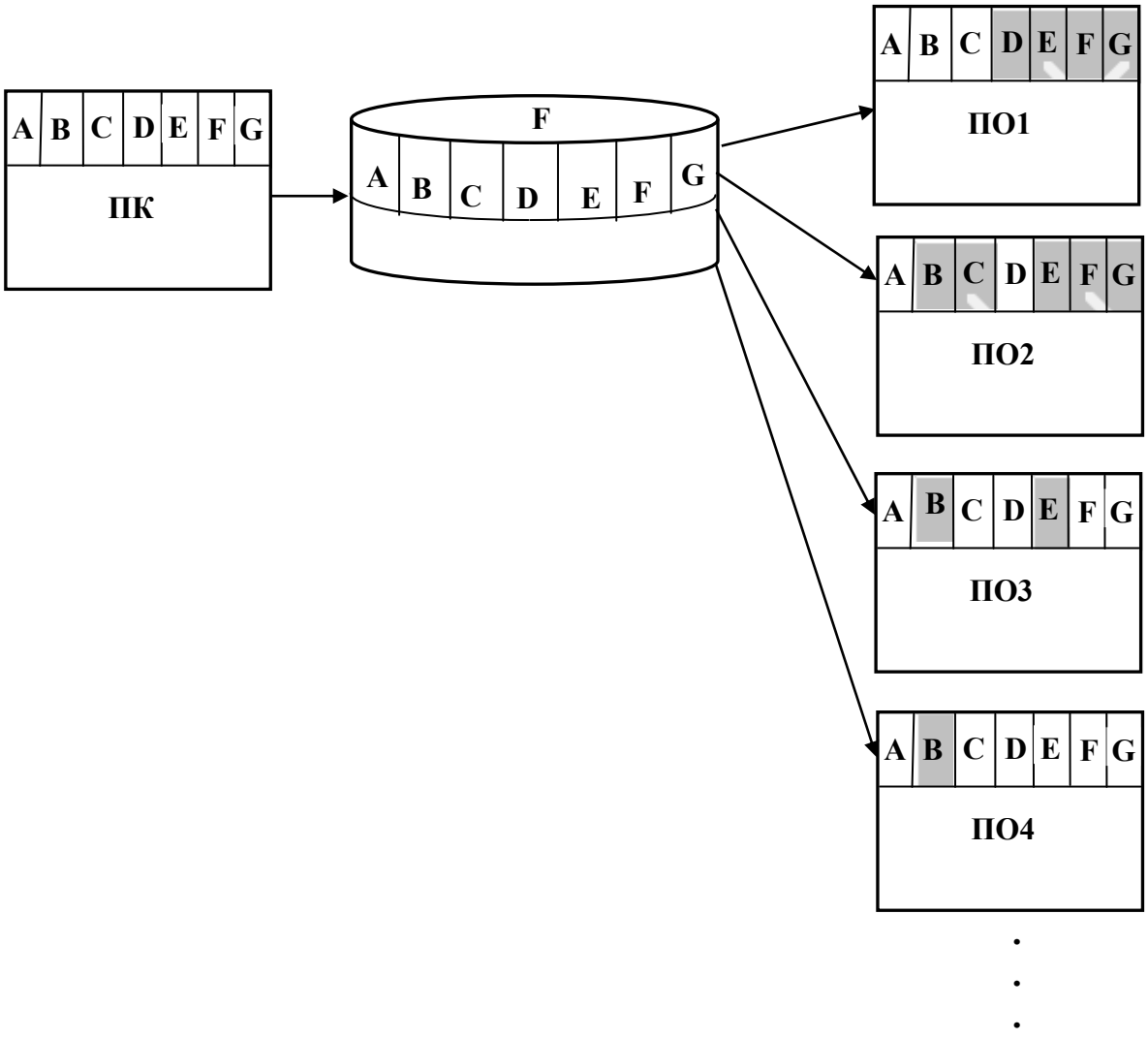


Рис. 1.5. Использование единого для всех программ файла исходных данных

Обеспечивая минимизацию дублирования исходных данных и их однократный ввод, такой подход порождал новые проблемы:

- сложность структуры исходных данных (в то время вся обработка ориентировалась на обработку простых «плоских», линейных файлов);
- избыточность файла для каждой конкретной программы обработки;
- любые изменения в структуре файла исходных данных приводили к необходимости повторной трансляции всех использующих его программ. Даже если эти изменения не затрагивали алгоритма об-

работки, необходимо было менять описание файла в программе, значит вновь осуществлять ее компиляцию и тестирование.

Для разрешения отмеченного противоречия между эффективным, централизованным хранением и актуализацией единого файла с минимальным дублированием информации (безызбыточное хранение) и эффективным использованием каждой программой минимально необходимой исходной информации привело к такой организации системы обработки данных, когда:

- используется единая система взаимосвязанных файлов с минимальной избыточностью, получившая название базы данных, со специальной организацией, эффективной для всех применений в целом;
- для обеспечения эффективного доступа к базе данных со стороны программ (предоставление только необходимой информации, обеспечение независимости от возможных изменений в структуре той части базы данных, которую не обрабатывает программа) используется специальная программа – **система управления базами данных (СУБД)**, по существу берущая на себя функции операционной системы по управлению данными.

Схема на рис. 1.6 иллюстрирует это положение. При этом предполагается, что с точки зрения эффективности всех приложений целесообразно иметь два файла и две программы корректировки.

В этом случае сохраняются достоинства позадачного подхода (когда программа имеет минимально необходимую информацию) и подхода, основанного на едином файле (минимизация дублирования хранимых данных). Появляется возможность изменить распределение данных между файлами хранения, добавить новые, удалить неиспользуемые данные, не требуя каких-то изменений в действующих программах обработки.

В зависимости от способа взаимодействия СУБД и программы обработки либо в программу передается очередная запись требуемой структуры, независимо от того, где физически в БД расположены данные, составляющие требуемую структуру, либо для программы создается временный файл требуемой структуры.

Подводя итог проведенному анализу, можно сформулировать следующие основные принципы (положения), определяющие концепцию данных:

1) автономное безызбыточное хранение данных сложной структуры и значительного объема;
2) комплексное использование хранимой информации;
3) независимость программ обработки от физической структуры исходных данных.

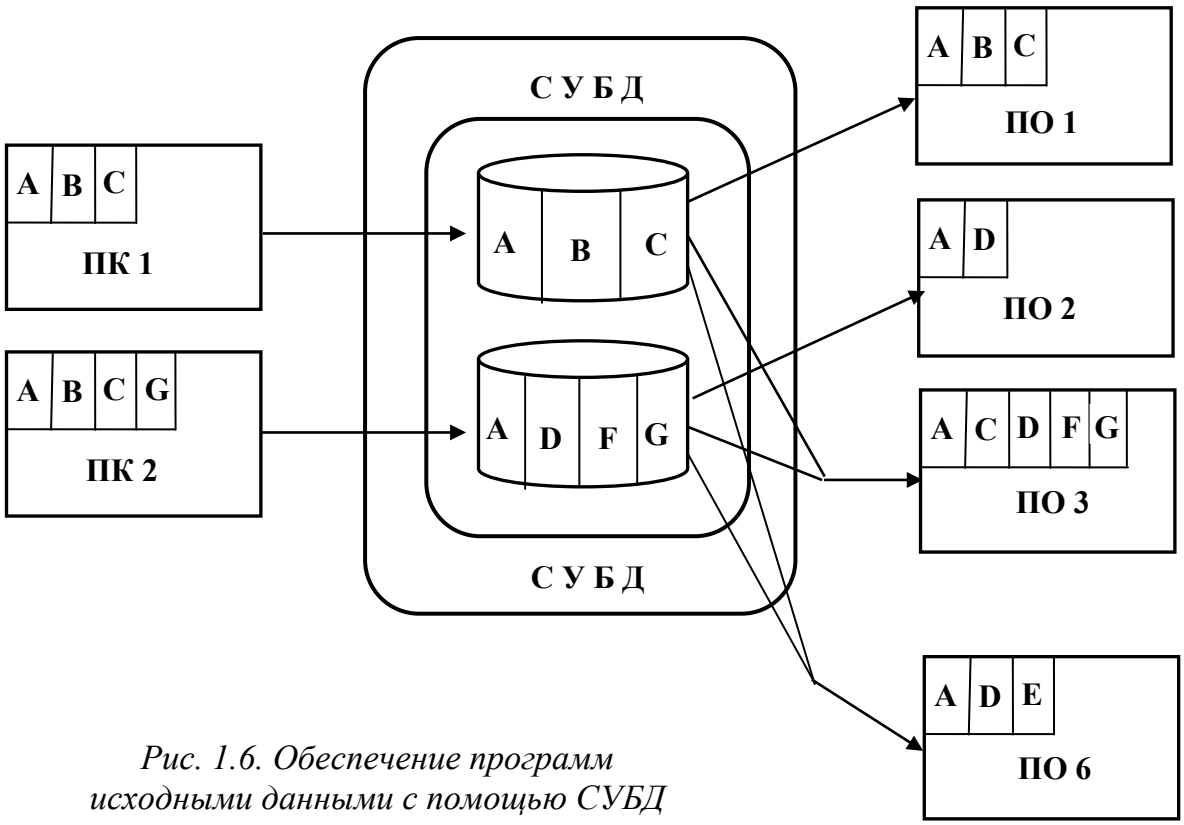


Рис. 1.6. Обеспечение программ исходными данными с помощью СУБД

Дополнительные положения концепции баз данных:

а) БД есть отображение информационной модели предметной области;
б) однократный ввод первичной информации;
в) защита данных (авторизованный доступ, защита от катастрофического разрушения, криптография, ограничения целостности);
г) реорганизация БД (развитие) по мере необходимости с минимальным влиянием на действующие программы.

Эти положения в том или ином виде находят свое отражение в определениях, которые дают базам данных и СУБД признанные авторитеты в этой области.

Одним из наиболее полных определений является следующее, представленное Дж. Мартином [1].

Базу данных можно определить как совокупность взаимосвязанных, хранящихся вместе данных при наличии такой организации и минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений; данные запоминаются и используются так, чтобы они были независимы от программ, использующих эти данные, а программы были бы независимы от способа и структуры хранения данных; для добавления новых или модификации существующих данных, а также для поиска данных в БД применяется общий управляющий способ.

Таким образом, необходимость применения концепции баз данных обусловлена следующими причинами:

- развитием подхода к обработке данных от вычислительных задач к информационным, к объединению последних в комплексы (подсистемы) и с постоянным их развитием, включая расширение состава задач и их ориентирование на широкий круг конечных пользователей;
- противоречием между подзадачным подходом в использовании исходных данных и требованием их эффективной актуализации;
- стремлением отобразить в системе хранимых данных информационную модель определенной предметной области.

Следует заметить, что приведенное выше обоснование отражает естественное развитие подхода к обработке структурированных (фактографических) данных и определяется в основном требованием повышения эффективности процессов создания, функционирования и модернизации программного обеспечения. Принцип обеспечения независимости программ от физической организации данных был в то время определяющим, а простота доступа понималась как возможность простого обращения к БД из программы, написанной на стандартном языке программирования.

Вторым по важности был неявно обозначенный выше принцип информационного моделирования некоторой предметной области в виде БД. Причем СУБД не только устанавливала (или, как еще говорят, «поддерживала») связи между записями различных типов в виде единой сложной структуры, но и обеспечивала выполнение целого ряда процедур на такой структуре, позволяя при этом значительно упростить программу обработки. В этой связи, 70-е годы характеризуются развитием теории баз данных (иерархическая, сетевая и реляционная модели данных), созданием БД сложной структуры. СУБД того времени работали с иерархической (IMS-ОКА, ИНЭС), сетевой (IDS, БАНКОС, СИОД, ADABAS) структуры. Использование информации БД осуществлялись посредством специального обращения к СУБД из программы, написанной на традиционном языке программирования типа Ассемблер, Кобол (это были обычные CALL-обращения к программе СУБД с определенными правилами оформления параметров вызова, т. е. на внутреннем языке манипулирования данными). Реляционная модель в силу простоты используемой структуры данных (проблемы с отображением иерархической и сетевой структуры) считалась не перспективной для практического использования.

Основная особенность применения концепции БД в 80-е годы заключается в переходе на использование персональных компьютеров, что в условиях отсутствия вычислительной сети привело к использованию локальных БД пользователей (так называемых автоматизирован-

ных рабочих мест – АРМов) и практического применения реляционной модели данных. Появилось множество реляционных СУБД (dBASE, PARADOX, CLARION, FOXPro, ACCESS), получивших название «настольных» и имеющих специализированные (ориентированные на работу с БД) языки программирования. В конце периода, с появлением локальных вычислительных сетей практически все эти СУБД способны работать с единой БД, располагаемой на специальном компьютере – сервере, обеспечивающем хранение БД значительного размера. Режим работы с общей БД на локальной сети получил название «технология файл-сервера».

90-е годы ознаменовались появлением корпоративных вычислительных сетей и как следствие усилением принципа комплексного использования информации БД, переходом к обеспечению доступа пользователей и программ к информации БД через запрос на языке SQL. При этом, появляются СУБД (MsSQL, MySQL, DB2, Progress, ORACLE), обеспечивающие работу с БД в режиме «клиент-сервер».

Сравнение технологий «файл-сервер» и «клиент-сервер» (рис. 1.7).

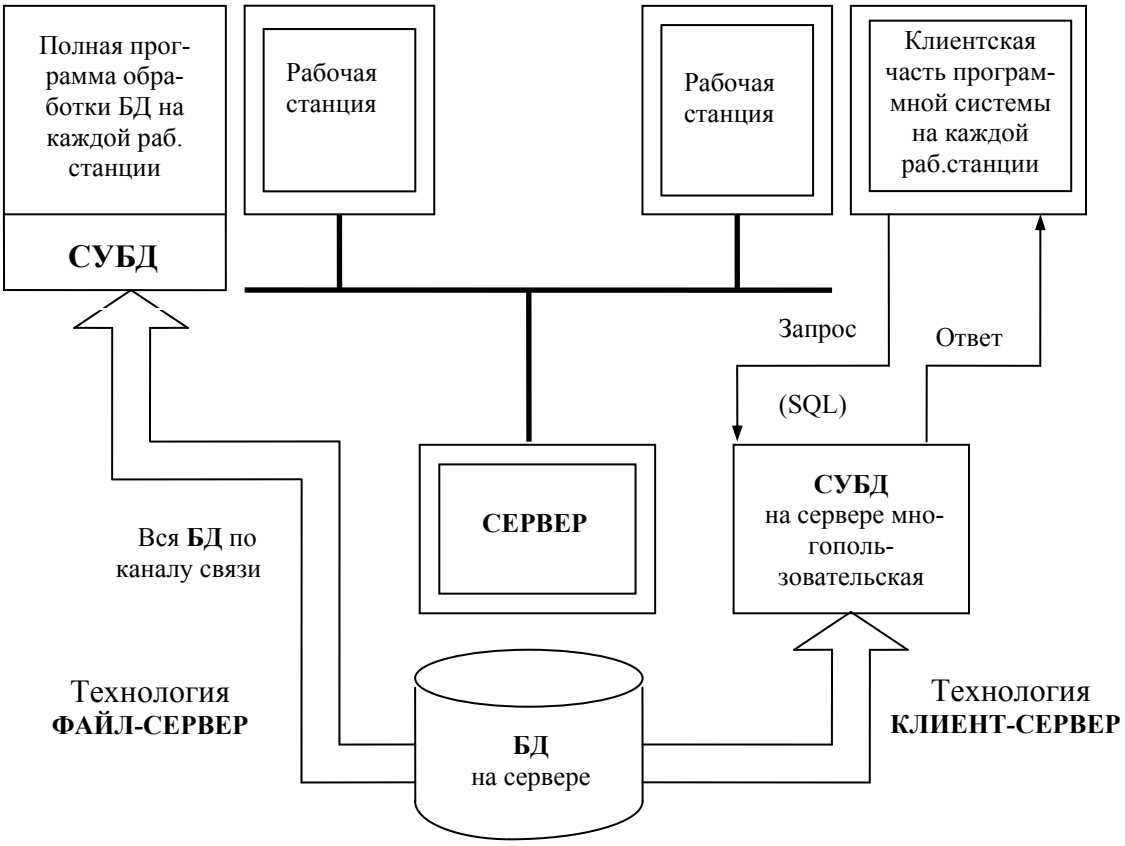


Рис. 1.7. Технологии «файл-сервер» и «клиент-сервер»

В обоих случаях реализуется сетевая технология с выделением в сети сервера – специального компьютера с большой памятью, где размещается БД, но взаимодействие с БД имеет следующие различия.

а) Технология файл-сервер

Для рабочей станции сервер – это нечто вроде специального диска S, дополнительного к линейке A (дискета), C (винчестер на ПК), D (2-й винчестер на ПК или 2-я часть), E – CD диск, S – сервер с БД.

На каждой рабочей станции полное программное обеспечение СУБД и работающая в ее среде программа, обращающаяся к данным БД, т. е. к диску S.

При обращении программы к данным БД сервера вся БД передается по сети на рабочую станцию, где и осуществляется обработка (как правило, отбор необходимой информации из большой БД). В итоге кроме того, что рабочая станция должна быть достаточно мощным компьютером (СУБД там работает), при работе нескольких рабочих станций осуществляется перегрузка сети. Не совершенные в то время средства блокировки при параллельной работе с одной БД нескольких программ, приводили к проблемам одновременной работы с БД даже нескольких пользователей.

б) Технология клиент-сервер

Программа на рабочей станции (клиентская часть) обращается к БД (к SQL-серверу) с запросом, как правило, на языке SQL – это может быть несколько десятков строк.

На сервере размещается не только БД, но и основная часть СУБД (обработчик SQL запроса). Выборка данных и основная обработка осуществляется на мощной машине сервера: программа обращается к БД (к серверу) с незначительным по размеру SQL запросом, а в программу (пользователю) возвращается ответ незначительного объема – один или несколько экранов.

В итоге, сеть не загружена, рабочая станция может быть не такой мощной, проще организовать распределенную обработку и защиту данных, оптимизировать работу БД.

В случае необходимости реализации программ общего пользования, алгоритм которых невозможно реализовать с помощью SQL создается еще один уровень (между SQL сервером и рабочими станциями) – сервер приложений и получаем трех уровневую технологию обработки БД.

Современный этап применения БД связан в основном:

- с повсеместным использованием корпоративных и глобальных вычислительных сетей;
- со значительной «историей» функционирования БД.

В результате наметились следующие направления использования БД:

- параллельная (одновременная) работа удаленных пользователей с одной БД;

- распределение хранения и обработки БД на сети;
- интеграция данных и приложений (программного обеспечения), созданных в разные периоды времени, в различных информационно-программных средах (ОС, СУБД, инструментальные средства);
- новые подходы к использованию исторических данных (Warehouse, OLAP, Data Mining, Management knowledge).

Контрольные вопросы и задания к разделу 1.1

1) Назовите основные характеристики вычислительных и информационных задач.
2) Назовите основные причины, вызвавшие появление концепции баз данных.
3) Охарактеризуйте позадачный подход в использовании исходных данных в системах обработки информации.
4) Назовите основные положения концепции баз данных.
5) Охарактеризуйте основные периоды в использовании концепции БД.
6) Особенности технологий «файл-сервер» и «клиент-сервер».
7) Новые направления в использовании БД.

1.2. Архитектура представления информации в концепции БД

Обеспечение основных принципов концепции баз данных достигается за счет трехуровневого представления информации (рис. 1.8) [2]:

- **концептуальное представление** – логическая структура БД в целом в ограничениях СУБД по структуре данных. Это то, как «видит» БД потенциальный пользователь;
- **физическое представление** – конкретное размещение значений данных в памяти (во внешней и в оперативной), способы и средства представления структурных характеристик (имен, размеров, адресов), установления связей между элементами структуры БД;
- **внешнее представление** – часть структуры БД, используемая в конкретном приложении (запрос, программа получения каких-то документов и т. п.).

Система управления базами данных (СУБД) обеспечивает возможность хранения описания всех этих представлений.

Такое трехуровневое представление данных как раз и обеспечивает соблюдение основных принципов концепции БД. Так, автономное хранение описания физической структуры БД позволяет СУБД настроиться на работу с конкретными данными, если ей переданы только имена данных (из программы или от конечного пользователя в экранном интерфейсе), а следовательно, обеспечивается независимость упомянутых

программ обработки или запроса конечного пользователя от варианта размещения данных в памяти и тем более от возможного расширения состава данных.

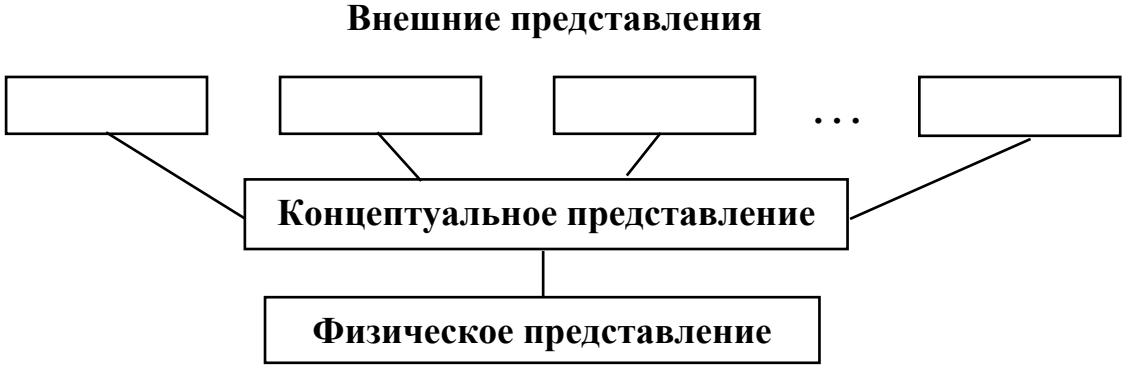


Рис. 1.8. Архитектура представления информации в концепции БД

С другой стороны, соотнесение (и отображение с помощью СУБД) любого внешнего представления с общей концептуальной моделью как раз и является основой обеспечения комплексного использования хранимых данных. Для нового приложения достаточно указать перечень имен, составляющих запись исходных данных и являющихся подмножеством концептуальной модели. СУБД обеспечит выборку необходимой информации из БД.

И, наконец, разделение трех описаний, их автономное хранение и ведение является основой автономного, централизованного хранения информации БД. Следующий пример иллюстрирует понятия концептуального, физического и внешних представлений. Пусть имеем следующее **физическое представление** информации о студентах в трех файлах (рис. 1.9).

Файл 1

Шифр студента	Место рождения	Дата рождения	Пол

Файл 2

Шифр студента	Ф.И.О.	Номер группы	Размер стипендии	Адрес проживания

Файл 3

Шифр студента	**Дисциплина**	Оценка	Количество пропусков	Рейтинг

Рис. 1.9. Физическое представление информации о студентах

Абстрагируясь от варианта размещения информации в физических файлах, в общем виде (концептуально) структуру данных можно представить с помощью двух агрегатов данных (рис. 1.10).

Общие сведения о студенте

Шифр студента	Ф.И.О.	Номер группы	Место рождения	Дата рождения	Пол	Размер стипендии	Адрес проживания

↓

Сведения об учебе

Дисциплина	Оценка	Количество пропусков	Рейтинг

Рис. 1.10. Концептуальное представление информации о студентах

Стрелка, соединяющая описание агрегата данных (записи) типа «Общие сведения о студентах» с описанием записи типа «Сведения об учебе», свидетельствует о том, что каждая запись первого типа связывается с несколькими записями второго типа (равными числу дисциплин, которые изучены студентом).

Внешнее представление. Пусть требуется создать программу, в результате работы которой должен быть получен следующий документ (рис. 1.11).

СВЕДЕНИЯ
о неуспевающих студентах (количество)

	Возраст	
	До 20 лет	Более 20 лет
Всего		
В том числе не получающих стипендию		

Рис. 1.11. Требуемый результат обработки

Тогда следующий состав исходных данных, минимально необходимых для обеспечения требуемого результата, и будет соответствующим внешним представлением (рис. 1.12).

Шифр студента	Дата рождения	Размер стипендии	**Дисциплина**	Оценка

Рис. 1.12. Внешнее представление информации, необходимой для получения сведений о студенте

Другой пример внешнего представления приведен на рис. 1.13. Пусть для проведения экзаменов необходимо получить ведомости (шифр студента совпадает с номером зачетной книжки). Ведомости необходимо выдать по тем дисциплинам, по которым в текущих семестре и учебном году студенты не имеют оценок.

ЭКЗАМЕНАЦИОННАЯ ВЕДОМОСТЬ
Дисциплина _____ Преподаватель_____
Группа _____семестр _____год

Ф.И.О. студента	Номера зачетной книжки	Оценка	Роспись преподавателя

Рис. 1.13. Требуемый результат обработки

Для получения требуемого результата достаточно иметь исходную информацию, структура которой приведена на рис. 1.14.

Шифр студента	Ф.И.О.	Группа	**Дисциплина**	Оценка

Рис. 1.14. Внешнее представление информации, необходимой для получения экзаменационных ведомостей

Контрольное задание к разделу 1.2

1) Дайте определения концептуальному, внешнему и внутреннему представлению.

1.3. Понятия схемы и подсхемы и их использование в СУБД

Описание концептуального и соответствующего ему физического представления (описание структуры БД) хранится автономно, называется **схемой БД** и создается до того, как начнет наполняться БД. Описание подмножества концептуального представления, которое соответствует внешнему представлению для некоторого приложения (описание части структуры БД, доступной программе обработки), называется **подсхемой**. Используя подсхему и схему, СУБД обеспечивает настройку приложения на работу с физической базой данных. Тем самым достигается универсализм СУБД по обеспечению соответствия внешнего представления физическому, а значит, обеспечивается принцип независимости программ обработки от физической структуры БД. С другой стороны, программа обработки может получить только те данные и выполнять только те процедуры (чтения, а возможно обновления данных), которые указаны в ее подсхеме. Тем самым обеспечивается и защита БД от несанкционированного доступа.

Пользователь, проектируя обработку данных для получения требуемого результата, определяет требуемое внешнее представление как подмножество концептуального представления и в принципе может не знать физической организации БД. СУБД, «понимая» соответствие концептуального и физического представления и «зная» внешнее представление, определяет, откуда физически надо выбрать требуемую информацию и в каком виде предоставить ее приложению (программе обработки или непосредственно конечному пользователю). Благодаря такому механизму (схема-подсхема) программа не зависит от адреса файла, а также от возможных изменений в структуре записи таких, как добавление новых данных, изменение характеристик или удаление данных, не используемых в программе. Следует заметить, что в современных СУБД вместо термина *подсхема* используется термин *представление (view)*, что в точности соответствует понятию *внешнее представление* в архитектуре представления информации в концепции БД.

Контрольные вопросы и задания к разделу 1.3

1) Дайте понятие схемы и подсхемы.
2) Сформулируйте назначение схемы и подсхемы в концепции баз данных.
3) Для чего предназначена система управления базами данных?

2. ИНФОРМАЦИОННАЯ СИСТЕМА СФЕРЫ УПРАВЛЕНИЯ – ОСНОВНАЯ ОБЛАСТЬ ИСПОЛЬЗОВАНИЯ БД

Под ИС понимаем совокупность методов и средств, обеспечивающих **представление** некоторой предметной области в виде информационной модели (ИМПО), и **предоставление** пользователям необходимой информации об объектах предметной области.

Основные сферы применения ИС:

- образование (получение новых знаний о предметных областях);
- экономика (использование информации о состоянии объектов и процессов в предметной области с целью управления ее функционированием).

Принципиальное отличие ИС для этих сфер применения заключается в следующем:

- Для целей получения новых знаний используется в основном содержательная, смысловая информация, представленная в виде текстов, рисунков, а в последнее время в виде аудио- и видеообъектов. Типичными объектами предметной области ИС в сфере образования являются книги, статьи, отчеты, пояснительные записки и т. п., чаще всего текстовая информация.
- В сфере экономики используются в основном некоторые фактические данные, отражающие определенные свойства, характеристики, параметры, атрибуты, описывающие состояние объектов предметной области. Типичное внешнее представление – так называемые объектно-характеристические таблицы, в которых строки соответствуют объектам предметной области, а столбцы – характеристикам объектов.

В соответствии с отмеченными характеристиками объектов представления информации ИС классифицируют на *документальные* (ДИС) и *фактографические* (ФИС).

В ФИС всестороннее описание объектов предметной области (ПрО), как раз и отображается в информационную модель, которая хранится в виде единой системы взаимосвязанных файлов, называемой базой данных (БД). Взаимосвязь файлов отражает взаимосвязь объектов разных типов и различных описаний внутри одного типа.

Обобщенная схема ФИС (и место в ней БД) представлена на рис. 2.1.

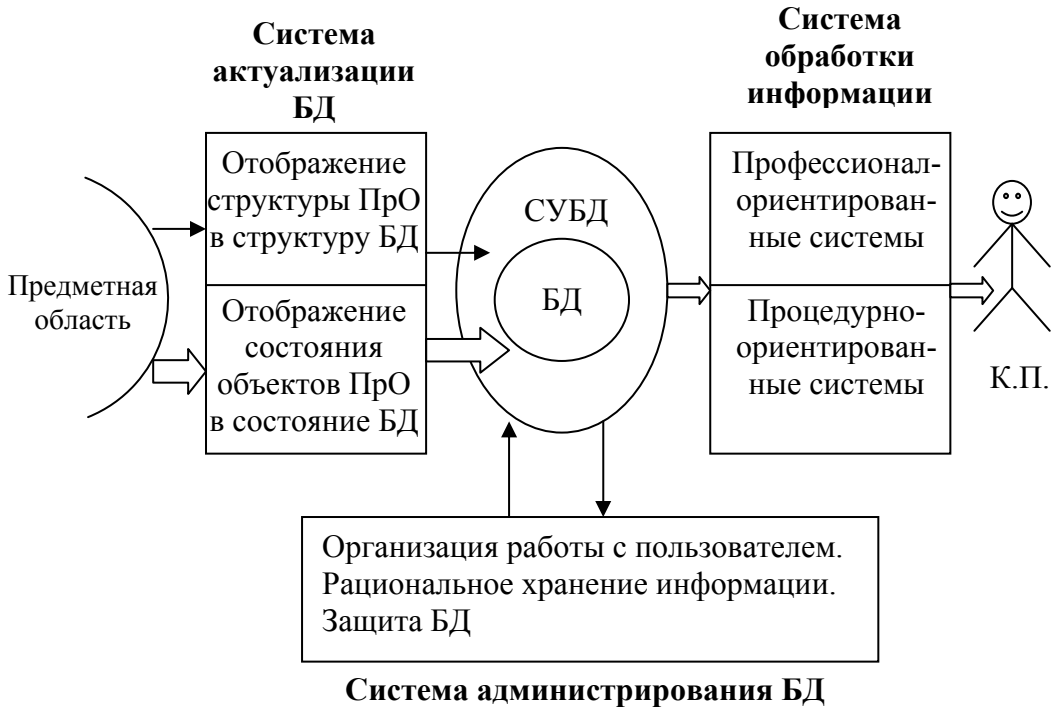


Рис. 2.1. Обобщенная схема ФИС

Рассмотрим каждый элемент схемы подробнее.

Система актуализации БД включает две подсистемы:

- отображения структуры ПрО в структуру БД;
- отображения состояния объектов ПрО в состояние БД.

Отображение структуры ПрО в структуру БД может трактоваться как проектирование БД, что также является объектом рассмотрения настоящего пособия и будет проведено позднее.

Отображение состояния объектов ПрО в состояние БД осуществляется, как правило, в два этапа:

- фиксации значений параметров объектов предметной области;
- корректировки значений соответствующих данных в БД.

Система обработки информации представляет собой программные средства (приложения) двух основных типов:

- профессионально-ориентированные средства;
- процедурно-ориентированных программные средства.

Профессионально-ориентированные системы обеспечивают прямую компьютерную поддержку существующих ИТ на рабочем месте и максимально учитывают их специфику. Поэтому такие средства часто называют *автоматизированными рабочими местами* (АРМ). Несомненное достоинство таких систем – создание комфортных условий для работников. Основной недостаток – необходимость модификации ПО при

изменении ИТ, связанных с их совершенствованием или директивными решениями. Для обеспечения минимальных затрат на модификацию таких систем их необходимо делать открытыми и использовать средства автоматизированного проектирования. Эти системы наиболее предпочтительны для работников, связанных с обработкой первичной информации, со спецификой в информационной технологии (определенные документы, последовательность действий, определенные участники и т. п.).

Процедурно-ориентированные системы реализуют типовые процедуры поддержки ИТ. Они являются универсальными средствами типа информационно-справочных систем. Нетрудно заметить, что к таким системам относятся средства экранного интерфейса современных СУБД, средства интерпретирующие команды SQL. Несомненным их достоинством является их универсальность, независимость от изменений в конкретных ИТ. Однако реализовать с их помощью специфические технологии на рабочем месте является трудоемкой задачей. Поэтому эти средства целесообразно применять для информационной поддержки руководящих работников, которым необходим справочный режим работы с данными, возникающими на самых разных рабочих местах.

Средства администрирования БД обеспечивают поддержку следующих видов работ администрации БД:

- организация работы с пользователями;
- совершенствование физической организации БД;
- обеспечение защиты хранимой информации.

Представленная схема наглядно свидетельствует о том, что БД является центральным звеном, основой информационной системы, средством информационного моделирования ПрО, для которой создана информационная система.

В течение более чем 40 последних лет ФИС разрабатываются и применяются для различных ПрО, имеющих следующие особенности.

ПрО может быть представлена как множество объектов различной природы: личности, документы, предметы производства, научные исследования, некоторые явления и др. Главное, что объекты в ПрО обладают (могут быть описаны) некоторыми свойствами (параметрами, характеристиками, показателями и т. п.). При этом для разных объектов значение одноименного параметра может быть различным, но выбирается из одного множества возможных значений, называемого словарем (классификатором, доменом). В результате информационного моделирования описание объектов ПрО отображается в компьютерные данные следующим образом:

- каждому параметру объекта ПрО соответствует данное, значению параметра у конкретного объекта – значение данного в записи, соответствующей этому объекту. Идентификатор (имя, название и т. п.) объекта также представляется как данное, но данное особого назначения – оно идентифицирует и запись (входит в идентификатор) и называется ключевым данным записи (по крайней мере, входит в ключ);
- описание множества однотипных объектов с определенной стороны (по некоторой группе параметров) представляется в виде файла, причем одному объекту в файле соответствует одна либо несколько записей. Одному объекту ПрО будет соответствовать несколько записей в файле в том случае, если по упомянутой группе параметров необходимо хранить несколько экземпляров описаний (за разные периоды времени, по технологическим переходам и т. п.);
- полная информация об объектах ПрО (всестороннее описание объектов) хранится в системе взаимосвязанных файлов, называемой БД. Взаимосвязь файлов отражает взаимосвязь объектов разных типов и различных описаний внутри одного типа.

Таким образом, можно сказать, что БД является представлением выбранной части предметного мира, которая моделируется для реализации необходимых бизнес-правил организации.

3. МОДЕЛИ ДАННЫХ

3.1. Понятия модели и структуры данных

Необходимость хранения и эффективного использования информационной модели предметной области явилась одной из основных (наряду с обеспечением независимости программ от структуры хранения данных) причин возникновения концепции БД и использования СУБД.

В период, предшествующий применению СУБД, использовались простые линейные структуры записей файлов, связь между записями различных файлов устанавливалась алгоритмически в программе обработки (описаний связей не было). Имевшаяся в то время возможность описания иерархических структур (например, в языке **PL/1** или **COBOL**) внутри одной записи не привела к широкому применению из-за того, что требовалось указывать и физически резервировать память под максимально возможное число повторений соподчиненных групп (агрегатов) данных.

С возникновением концепции баз данных получила развитие теория структуризации данных, были определены типовые структуры (модели) данных:

- иерархическая;
- сетевая;
- реляционная модель.

В публикациях, связанных с теорией структуризации данных, нет четкого разделения понятий **структура** и **модель**. Если иерархическую и сетевую структуры иногда представляют и как модели данных (без дополнительного пояснения), то всегда говорят о реляционной **модели** данных. Попытаемся выделить это различие в соответствующих определениях.

Под **структурой** данных будем понимать совокупность информационных элементов и связей между ними.

Под **моделью** данных будем понимать соответствующий тип структуры данных и типовые операции по управлению данными.

Следует также заметить, что когда говорят о структуре данных как о модели данных, то имеют в виду **логическую** структуру, под которой понимают представление информационных элементов и связей между ними вне зависимости от способа их размещения в памяти компьютера.

В противовес этому под **физической** структурой понимают представление информационных элементов и связей между ними в памяти компьютера, вплоть до представления символов, а возможно, и указателей связи битовыми кодами.

3.2. Линейная модель данных

Свойства линейной структуры:

- элементами линейной структуры являются простые данные. Простыми считаем данные, разделение которых на составляющие не имеет смысла;
- каждое данное имеет имя (идентификатор) и множество возможных значений, задаваемое словарем, диапазоном или правилом формирования;
- множество данных, составляющих линейную структуру, описывает множество однотипных объектов;
- все экземпляры линейной структуры (записи) однородны в том смысле, что:
  - порядок следования данных во всех экземплярах структуры один и тот же;
  - размер и тип данных одного имени во всех экземплярах структуры одинаковы. Разные данные могут иметь различные размеры и типы;
- линейной структуре в информационных системах соответствует файл однотипных записей.

Отметим особо, что связь между элементами состоит в определенном порядке их следования в экземплярах структуры.

Пример линейной структуры, объектом описания (моделирования) которой является студент, приведен на рис. 3.1.

Схема структуры **СТУДЕНТ**

Код студента	Ф.И.О.	Номер группы	Пол	Дата рождения

Экземпляры структуры **СТУДЕНТ**

⋮				
427101	Гончар Е.	4271	Ж	29.04.78
427102	Драганов М.	4271	М	19.01.79
427103	Зюкин М.	4271	М	26.03.79
477001	Акулинин А.	477	М	29.04.76

Рис. 3.1. Простая линейная структура данных

Представленная структура описывает материальный объект – личность.

Другой пример линейной структуры, объектом описания которой является нечто не материальное что-то вроде семестр студента, приведен на рис. 3.2.

Схема структуры **СЕМЕСТР**

Код студента	Номер семестра	Тип стипендии	Рейтинг за семестр

Экземпляры структуры **СЕМЕСТР**

⋮			
427101	1	стандартная	105
427101	2	стандартная	100
427101	3	не получает	70
⋮			
427102	1	стандартная	100
427102	2	повышенная	130
⋮			
477001	1	стандартная	100
477001	2	стандартная	110
477001	3	стандартная	100
⋮			

Рис. 3.2. Линейная структура, ключ которой состоит из двух данных

Типовые операции по управлению данными линейной структуры:

- вставка – включение новых экземпляров (записей) в структуру;
- удаление – удаление определенных экземпляров из структуры;
- замена – изменение значений некоторых данных в определенных экземплярах структуры;
- выборка – чтение экземпляров для обработки.

При удалении или замене соответствующие записи вначале должны быть найдены.

Не следует путать эти операции с операциями вставки новых атрибутов, удаления и замены существующих атрибутов, являющихся операциями по изменению структуры.

Представленные на рис 3.1 и 3.2 схемы реализованы в нотации Байхмана, что наиболее точно соответствует обычному изображению линейной структуры в виде таблицы, однако вызывает трудности в изображении значительного числа данных-столбцов, особенно при отобра-

жении с помощью компьютера. Поэтому в большинстве СУБД используется другой способ – отображение названий столбцов в виде строк некоторой таблицы (рис. 3.3).

Например, такая структура:

Номер зач. книжки студента	Номер группы	Номер семестра	Рейтинг студента за семестр	Дисциплина	Кол-во пропусков по дисциплине в семестре	Оценка по дисциплине в семестр	Рейтинг по дисциплине в семестре

Представляется в следующем виде:

Номер зачетной книжки студента
Номер учебной группы студента
Номер семестра
Рейтинг студента за семестр
Дисциплина
Количество пропусков по дисциплине в семестре
Оценка по дисциплине в семестре
Рейтинг по дисциплине в семестре

Рис. 3.3. Варианты отображения описания линейной структуры

При таком подходе, кроме простоты компьютерного отображения отдельной линейной структуры появляется возможность использования более точного названия данных.

Следует заметить, что, несмотря на повсеместное использование такой структуры файлов в периоде предшествующим применению БД, понятие линейной, или как ее еще называют плоской, структуры данных появилось в связи с развитием теории баз данных, т. е. с определением иерархической, сетевой и реляционной модели данных. Введение понятия линейной структуры целесообразно еще и потому, что она является элементом более сложных, иерархической и сетевой, структур. Вместе с тем, можно сказать, что реляционная модель является линейной структурой с определенными дополнительными условиями, такими как обязательное определение ключей и удовлетворение нормальным формам.

3.3. Иерархическая структура (модель) данных

Анализируя примеры, приведенные в предыдущем разделе, нетрудно заметить, что каждому экземпляру структуры **СТУДЕНТ** соответствует несколько (по числу семестров, пройденных студентом) экземпляров структуры **СЕМЕСТР**. Это и определяет иерархическую зависимость между этими линейными структурами, причем элемент, каждому экземпляру которого можно поставить в соответствие несколько экземпляров второго, называется **старшим** (**родителем**, **исходным**). В нашем случае – это структура **СТУДЕНТ**. Второй элемент называется **подчиненным** (**потомком**, **порожденным**). В нашем случае – это **СЕМЕСТР**.

Иерархическую связь формально определяют как **1:М** (один ко многим).

На рисунках иерархическую зависимость будем изображать в виде так называемых диаграмм Бахмана, состоящих из обобщенных схем элементов, связанных стрелкой (иногда двойной стрелкой), направленной от старшего к подчиненному. Примером такой диаграммы может быть отображение иерархической связи между элементами **СТУДЕНТ** и **СЕМЕСТР** (рис. 3.4).

Рис. 3.4. Простая иерархическая связь

Когда ведут речь об иерархической структуре, имеют в виду древовидную структуру.

Иерархическая древовидная структура данных – это структура, удовлетворяющая следующим требованиям:

- элементами являются линейные структуры различных типов (различные линейные структуры);
- связанные между собой элементы относятся так, что каждому экземпляру одного из них можно поставить в соответствие несколько экземпляров второго. Первый элемент называется старшим (исход-

ным, «родителем»), второй – подчиненным (порожденным, «ребенком»);

- каждый подчиненный в одной связи может быть старшим в связи с другим элементом;
- один старший может иметь несколько подчиненных различных типов;
- в древовидной структуре любой подчиненный имеет не более одного старшего, корневой элемент не имеет старшего;
- связь между непосредственно связанными элементами формально определяется как «один ко многим» или 1:М.

Элементы иерархической древовидной структуры имеют следующие названия:

- единственный элемент, не имеющий своего старшего, называется корневым;
- элементы, не имеющие подчиненных, называются концевыми или листьями;
- множество элементов, расположенных на одном пути от корневого до концевого элемента называется ветвью;
- максимальное число элементов в ветви (среди всех ветвей) называется рангом иерархической структуры.

Расширим структуру, приведенную на рис. 3.4, дополнительными элементами в виде дерева (рис. 3.5).

Необходимо обратить внимание, что в классической иерархической структуре совсем не обязательно, чтобы ключ подчиненного содержал в себе ключ старшего элемента, как это имеет место в элементе СЕМЕСТР. При работе с экземплярами подчиненных элементов к ним автоматически присоединяются ключи старших элементов. Поэтому во вновь введенных элементах **АТТЕСТАЦИЯ** и **ПРОПУСКИ** нет данных **код студента**, **номер семестра**, а в элементе **БОЛЕЗНИ** нет **кода студента**. Определим пока понятие ключа как одно либо несколько данных, определяющих смысл всех других данных (без него остальные данные становятся бессмысленными). Более детально понятие ключа будет рассмотрено в разделе 3.5. *Реляционная модель данных*, где это понятие является основополагающим.

Отметим также особенность связи между элементами, состоящую в том, что не у каждого экземпляра элемента типа **СТУДЕНТ** есть экземпляры элемента типа **БОЛЕЗНИ**, так как некоторые студенты за все время обучения могли не обращаться к врачебной помощи. Такую связь также относят к типу **1: М**, хотя по существу это связь **1: М** (**0**).

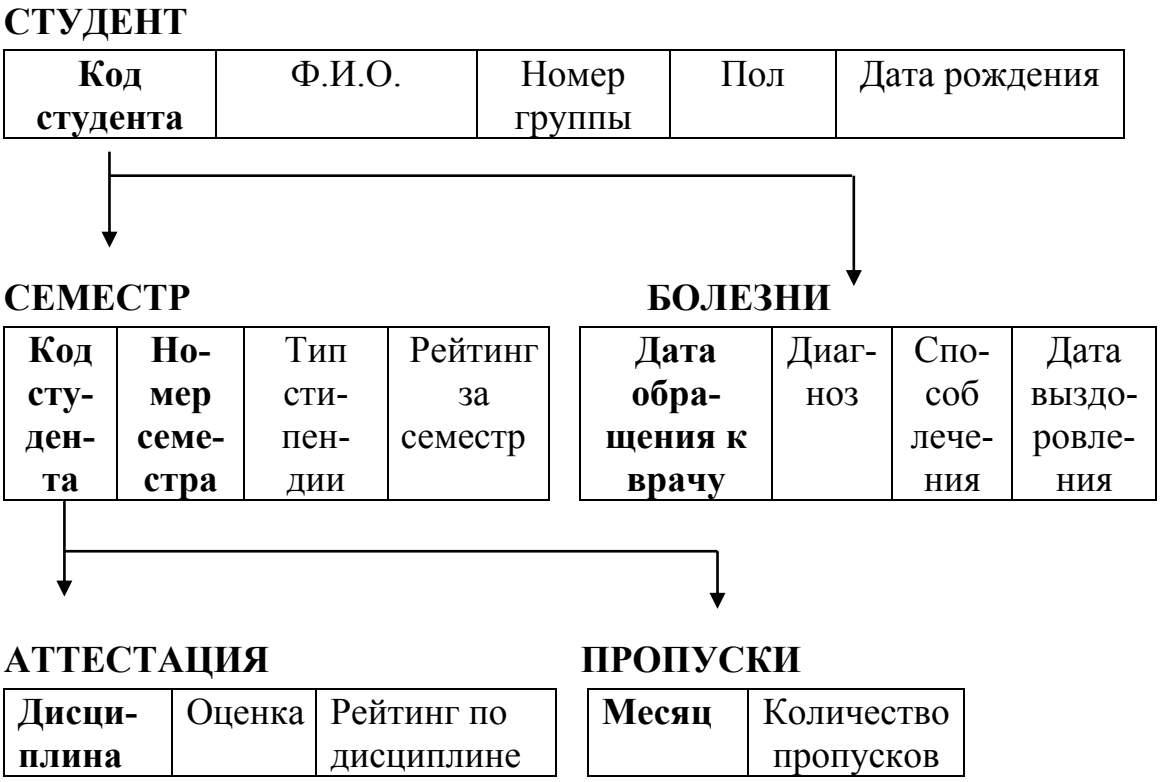


Рис. 3.5. Типичная иерархическая структура

В заключение следует отметить, что по определению иерархической считается связь, когда каждому экземпляру элемента одного типа можно поставить в соответствие несколько экземпляров элементов второго типа, и каждый подчиненный (кроме корневого) должен имеет только один старший, т. е. подчиненный не может существовать без старшего.

В иерархической модели данных определены следующие типовые операции по управлению данными:

- **вставка** аналогична соответствующей операции линейных структур данных с тем лишь отличием, что в случае жесткой иерархии невозможно вставить экземпляр элемента, не имеющего соответствующего старшего;
- **замена** имеет особенности для ключевых данных, так как требуется произвести аналогичные изменения соответствующих данных в ключах подчиненных (если ключ старшего присутствует в ключе подчиненного, как это было в связи **СТУДЕНТ–СЕМЕСТР**) либо перезаписать все подчиненное поддерево на другое место в структуре в соответствии с изменившимся значением ключевого данного старшего элемента. По этой причине иерархические СУБД часто не допускают изменения значения ключевого данного, требуя замены

его на операции вставки всех подчиненных на новое место и последующего удаления поддерева на прежнем месте;

- **удаление** также имеет соответствующую специфику в жесткой иерархии – удаление экземпляра старшего ведет к удалению и всех его подчиненных;
- **выборка** в иерархических СУБД имеет несколько разновидностей, обеспечивающих повышение эффективности программирования обработки связанных линейных структур:
  - выборка следующего **подобного**, т. е. чтение следующего экземпляра того же типа элемента. Это аналогично операции выборки в линейной структуре;
  - выборка **подобного в пределах исходного**. Такая выборка может быть осуществлена только после выборки экземпляра старшего элемента, а читаются последовательно однотипные подчиненные, но только связанные с выбранным «старшим». Например, после выборки записи типа **СТУДЕНТ** могут быть последовательно прочитаны записи типа **БОЛЕЗНИ**, но только для соответствующего студента;
  - выборка **следующего в иерархической последовательности,** т. е. выборка возможно разнотипных элементов в соответствии со структурой по правилу «сверху вниз – слева направо». Так, для структуры, приведенной на рис. 3.4 вслед за чтением записи **СТУДЕНТ** выбирается соответствующая выбранному студенту первая запись типа **СЕМЕСТР**, затем, если есть, соответствующая запись типа **АТТЕСТАЦИЯ** (правило «сверху вниз»). Далее, в связи с отсутствием подчиненных у элемента типа **АТТЕСТАЦИЯ**, следующая, если она есть, запись типа **АТТЕСТАЦИЯ** по другой дисциплине (правило «слева направо»). Когда закончатся записи типа **АТТЕСТАЦИЯ** для того же семестра и студента, читается первая запись типа **ПРОПУСКИ** (правило «слева направо») опять для того же студента и семестра. Коль скоро подчиненных элементов у элемента типа **ПРОПУСКИ** нет (вниз некуда), читается следующая запись типа **ПРОПУСКИ,** но связанная с теми же элементами типа **СТУДЕНТ** и **СЕМЕСТР**. Когда такие записи закончатся (нет ни внизу, ни вправо), читается следующая запись типа **СЕМЕСТР**, но для прежнего значения элемента типа **СТУДЕНТ**. Далее по прежней схеме читаются записи типа **АТТЕСТАЦИЯ** и **ПРОПУСКИ**, а затем следующая запись типа **СЕМЕСТР**. Когда записи типа **СЕМЕСТР** под прежней записью типа **СТУДЕНТ** закончатся, читается запись

типа **БОЛЕЗНИ** (правило «слева направо») для того же студента. И лишь после того, как они закончатся (либо их нет), читается очередная запись типа **СТУДЕНТ** и движение по дереву «сверху вниз, слева направо» повторяется;

– в процессе выполнения операции выборки экземпляров элементов (записей) некорневого уровня к выбираемому экземпляру могут быть присоединены ключи вышестоящих уровней или даже полные связанные экземпляры элементов старших уровней. Может быть заполнено и место расположения в БД любого из этих экземпляров вышестоящих уровней для использования при последующем обращении, например в пределах исходного.

Кроме отмеченного удобства программирования, к достоинствам иерархической структуры данных относятся эффективное использование памяти компьютера и достаточно быстрое выполнение операций над данными. Такая структура удобна для работы с иерархически упорядоченной информацией. Недостатком этой структуры является сложность ее реализации в СУБД.

Вместе с тем иерархия является наиболее удобным средством информационного моделирования самых различных предметных областей. Человек в процессе созидательной деятельности строит материальные объекты по иерархическому принципу (прибор состоит из деталей, здание из этажей и комнат и т. п.), в процессе познавательной деятельности он применяет методы декомпозиции и интеграции, поэтому основой информационных моделей реального мира является иерархия. Любые, более сложные структуры состоят из элементарных (парных) иерархических связей.

3.4. Сетевая структура (модель) данных

Элементами сетевой структуры данных являются линейные структуры. Для сетевой структуры справедливы все положения, характеризующие иерархическую структуру (иерархическая структура является частным случаем сетевой), а кроме того допускается:

- наличие более одной связи между двумя элементами (рис. 3.6, *а*);
- подчиненный может иметь более одного старшего (рис. 3.6, *б*);
- циклические подструктуры (рис. 3.6, *в*, *г*);
- более одной связи между экземплярами одного типа (рис. 3.6, *д*).
- наличие связей типа многие ко многим (M:M) и один к одному (1:1)

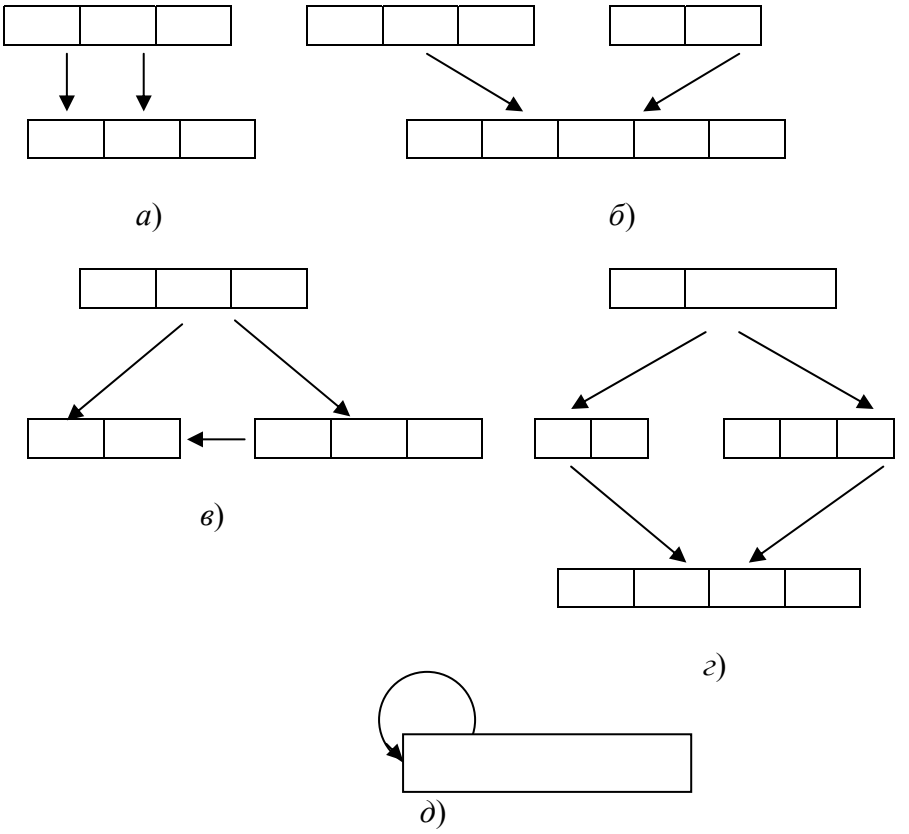


Рис. 3.6. Допустимые виды связей между элементами в сетевой структуре

Пример, иллюстрирующий сетевую структуру данных (рис. 3.7):

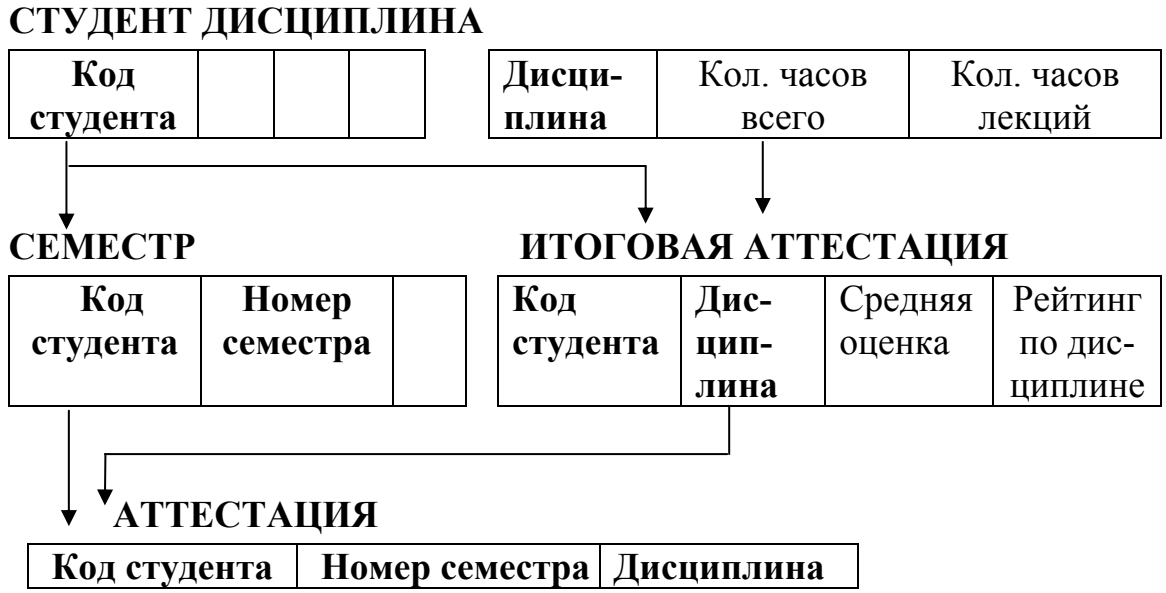


Рис. 3.7. Простая сетевая структура (варианты 3.6 б и г)

Пример на рис. 3.7 хотя и является лишь фрагментом в модели организации учебного процесса в вузе, тем не менее, иллюстрирует наиболее важные особенности сетевой структуры: более одного старшего и циклы. Пример составлен на основе примера на рис. 3.5, поэтому в уже известных элементах опущены имена не ключевых данных. Элемент **ИТОГОВАЯ АТТЕСТАЦИЯ** отличается от элемента **АТТЕСТАЦИЯ** только в том случае, если дисциплина читается несколько семестров, а элемент **ДИСЦИПЛИНА** содержит характеристики дисциплины, единые для всех студентов, ее изучающих.

К сетевым следует отнести также связи многие ко многим (M:M) и связь один к одному (1:1)

Связь M:M – когда каждому элементу одной структуры можно поставить в соответствие несколько элементов другой, но и наоборот, каждому элементу второй структуры можно поставить в соответствие несколько экземпляров первой (рис 3.8 и 3.9).

ДИСЦИПЛИНА

Код студента	**Дисциплина**	Рейтинг студента по дисциплине

СЕМЕСТР

Код студента	**Номер семестра**	Тип стипендии	Рейтинг за семестр

Рис 3.8 Связь типа M:M (вариант 1)

Каждому экземпляру структуры *Дисциплина* (дисциплина, изучаемая **студентом**) можно поставить в соответствие несколько экземпляров структуры *Семестр* (столько, сколько семестров отучился соответствующий студент), и наоборот, каждому экземпляру структуры *Семестр* (любой из семестров, который уже завершил студент) в структуре *Дисциплина* можно поставить несколько экземпляров с каждой из дисциплин, которые уже изучил соответствующий студент.

На одной и той же кафедре обучается множество студентов и работает множество преподавателей, поэтому каждому экземпляру структуры *Студент* можно поставить в соответствие столько экземпляров структуры *Преподаватель*, сколько преподавателей работает на соответствующей кафедре, и наоборот.

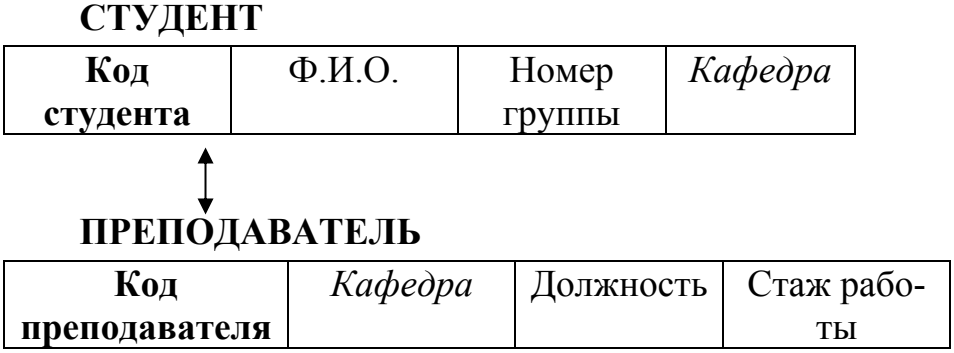


Рис 3.9. Связь типа M:M (вариант 2)

Связь 1:1 – когда каждому элементу одной структуры можно поставить в соответствие только один (или ни одного) экземпляр другой, но и наоборот, каждому элементу второй структуры можно поставить в соответствии только один экземпляров первой (рис. 3.10).

Рис 3.10. Пример связи типа 1:1

Если предположить, что в обеих структурах нет дублирования информации по каждому студенту (противное не имеет смысла), то каждому экземпляру одной структуры можно поставить в соответствии только один экземпляр в другой структуре (для того же студента), и наоборот.

Пример, иллюстрирующий возможность нескольких связей между парой линейных структур (вариант 3.6 *а*), а также типы связей 1:1 и M:M, представлен на рис. 3.11.

Первая связь 1:1 – каждому медицинскому работнику обязательно соответствует один экземпляр в структуре застрахованных личностей, так как каждый медицинский работник должен быть застрахован и застрахован единожды, как любая личность, но не все застрахованные личности являются медицинскими работниками, т. е. связь фактически 1:1 или 1:0.

Вторая связь М:М, по месту работы – на одном месте работы несколько личностей как среди застрахованных, так и среди медицинских работников (свяжутся не все экземпляры застрахованных, а только соответствующих медицинских работников).

Третья связь М:М, по поликлиникам прикрепления застрахованных, которые являются местом работы части медицинских работников (не все работают в поликлиниках), поэтому эта связь типа М:М, но с возможностью отсутствия связи некоторых экземпляров как первой, так и второй структуры.

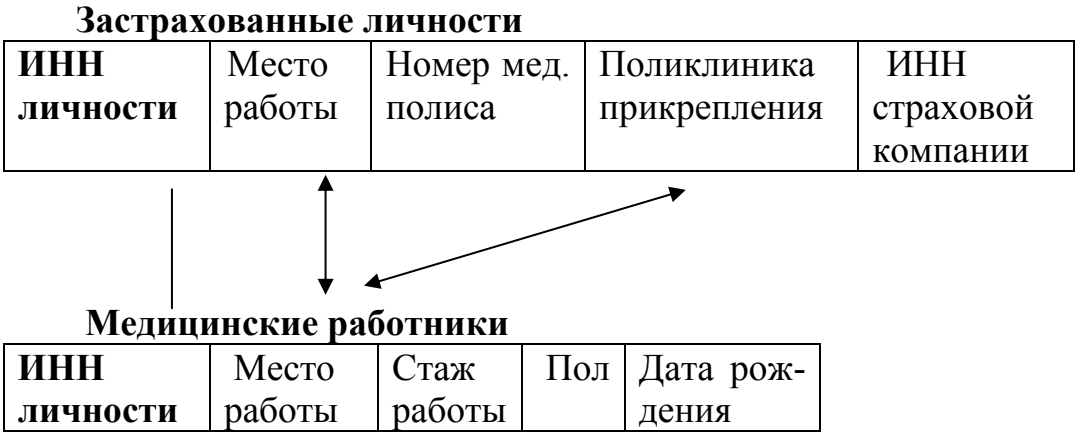


Рис 3.11. Несколько связей между парой линейных структур (связи 1:1, М:М) – вариант 3.6 а)

Связь типа **петля** – например, связь между экземпляром, соответствующим начальнику и экземплярами, соответствующими его подчиненным или работники одного места работы (рис. 3.12).

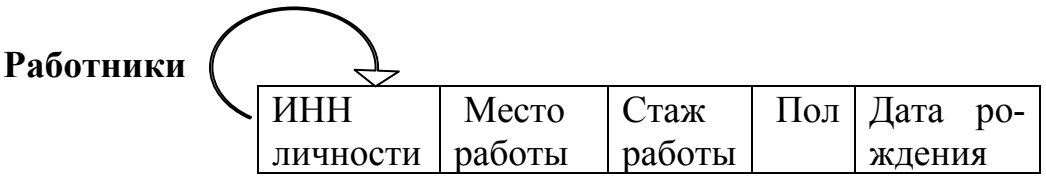


Рис 3.12. Связь типа петля – вариант 3.6. д)

Достоинством сетевой структуры данных является возможность эффективной реализации по показателям затрат памяти и возможности реализации специальных операций, использование которых позволяет существенно упростить программы обработки связанных элементов.

В сравнении с иерархической сетевая структура данных предоставляет большие возможности в смысле допустимости образования произвольных связей. Недостатком этой структуры является высокая слож-

ность для понимания, реализации и выполнения обработки информации в БД. Кроме того, в такой структуре ослаблен контроль целостности связей вследствие допустимости установления произвольных связей между записями.

В процессе рассмотрения особенностей иерархической и сетевой структур данных мы одновременно определили классические типы связей между парой линейных структур – 1:1, 1:M, M:M. В разделе, связанном с проектированием баз данных будет приведен более детальный анализ этих типов связей.

Следует заметить, что в полном объеме сетевая структура не поддерживается ни одной реально действующей СУБД. Она была определена группой CODASYL [3] как универсальная структура, позволяющая реализовать информационную модель практически любой ПрО. Это направление отражало интересы проектировщиков БД и прикладных программистов, с которых во многом снимались проблемы отслеживания связей между записями различных файлов. Однако в 80-ые гг. в связи с широким распространением ПК преобладающее развитие получило второе направление, которое использует реляционную модель данных, описание которой представлено ниже.

Контрольные вопросы и задания к разделам 3.1, 3.2, 3.3, 3.4

1. Что такое структура данных?
2. Что такое логическая и физическая структуры данных?
3. Назовите основные требования, которым должна удовлетворять:
   – линейная структура данных;
   – иерархическая структура данных;
   – сетевая структура данных.
4. Назовите и поясните существо типовых связей между парой линейных структур.

3.5. Реляционная модель данных

3.5.1. История применения реляционной модели данных

Появление концепции баз данных сопровождалось развитием теории структуризации данных, в которой наметились два направления. Первое, реализуемое группой CODASYL, было ориентировано на построение сложных информационных моделей различных предметных областей (сетевые модели) и их поддержку средствами СУБД. Это направление отражало интересы системных аналитиков, проектировщиков баз данных и прикладных программистов, с которых во многом снима-

лись проблемы отслеживания связей между записями различных файлов.

Второе направление, предложенное Коддом [4], с одной стороны, подкреплено реляционной алгеброй (строгой математикой), а значит, можно было ожидать создания корректно работающих СУБД. С другой стороны, оно было ориентировано на конечного пользователя, так как использовало простую интерпретацию основных понятий и столь же простое отображение информационной модели в виде традиционно используемых пользователем таблиц.

Первое направление преобладало в 70-е годы, видимо, потому что конечные пользователи чисто технически не имели возможности по непосредственному доступу к базам данных из-за отсутствия вычислительных сетей.

В 80-е годы, в связи с широким распространением персональных компьютеров, применением их непосредственно конечными пользователями (неспециалистами в области программирования), преобладающее развитие (точнее широкое практическое применение) получило второе направление.

3.5.2. Основные понятия реляционной модели данных

В основе реляционной модели данных лежит понятие **отношения**. Неформализованное отношение представляется в виде двумерной таблицы, на которую накладываются определенные ограничения. Столбец таблицы соответствует понятию **атрибута** отношения, строка – понятию **кортежа** отношения. Множество возможных значений, которые могут появляться в столбце таблицы, – понятию **домена**, на котором определен соответствующий атрибут.

Аналогично можно установить соответствие и с понятиями, используемыми при определении файлов линейной структуры: отношение–файл; атрибут–данное; кортеж–запись файла; домен–множество возможных значений данного.

Формализованное определение основных понятий реляционной модели данных базируется на теории множеств.

Отношение, с одной стороны, представляется (по «горизонтали») как множество атрибутов, а с другой (по «вертикали») – как множество кортежей. Каждому элементу множества атрибутов ставится в соответствие множество возможных значений – домен. Тогда можно осуществить следующую формальную запись основных понятий реляционной модели данных.

Пусть дана совокупность множеств $D_1, D_2, ..., D_n$, каждое из которых (D_i) представляет собой множество возможных значений *i*-го атрибута (A_i) отношения *R*. Множества *D* могут пересекаться и даже совпадать, т. е. различные атрибуты могут быть определены на пересекающихся или даже на одном и том же домене. Например, атрибуты **дата приема на работу** и **дата увольнения**.

Схемой отношения *R* называется конечное множество имен атрибутов $\{A_1, A_2, ..., A_n\}$, причем каждому атрибуту ставится в соответствие домен $A_i \to D_i$. Схема отношения *R* записывается в виде:

$$R(A_1, A_2, ..., A_n) \text{ или } R(D_1, D_2, ..., D_n).$$

Пусть множество *D* есть декартово произведение доменов схемы

$$D = \{D_1 * D_2 *, ..., *D_n\}.$$

Тогда отношение *R* со схемой $R(D_1, D_2, ..., D_n)$ есть подмножество $D, R \subset D$.

По существу, отношение соответствует рассмотренной нами линейной структуре с соблюдением всех присущих ей требований (один и тот же порядок следования атрибутов, один и тот же размер и тип значений одного атрибута во всех кортежах отношений). Дополнительные, присущие реляционной модели требования – обязательное наличие в отношении ключа и удовлетворение нормальным формам. Так, приведенные на рис. 3.1 и 3.2 линейные структуры **СТУДЕНТ** и **СЕМЕСТР** в реляционной модели данных будут отношениями, а данные – *№ зачетной книжки, Ф.И.О., Пол, Дата рождения* и *Номер семестра, Оценка, Рейтинг по дисциплине* – атрибутами отношений с ключами, отмеченными жирным шрифтом на рис. 3.13.

Схема отношения **СТУДЕНТ**

№ зачетной книжки	Ф.И.О.	Номер группы	Пол	Дата рождения

Схема отношения **СЕМЕСТР**

№ зачетной книжки	**Номер семестра**	Тип стипендии	Рейтинг за семестр

Рис. 3.13. Схемы простых отношений

В реляционной модели обязательно устанавливается и имеет определяющее значение понятие **функциональной зависимости** одного атрибута *X* от другого *Y* в отношении *R*, означающее, что если известно значение первого атрибута (*детерминанта*) *X,* то значение второго атрибута (*зависимой части*) *Y* определено и имеет единственное значение.

Формальное определение функциональной зависимости: $X \rightarrow Y$. Например, если в отношении **Студент** известен *№ зачетной книжки*, то для любого кортежа отношения *Студент* однозначно определены значения его атрибутов *ФИО*, *Номер группы*, *Пол*, *Дата рождения*, т. е. *№ зачетной книжки* $\rightarrow$ {*ФИО*, *Номер группы*, *Пол*, *Дата рождения*}, аналогично для отношения *Семестр: № зачетной книжки, Номер семестра* $\rightarrow$ {*Тип стипендии*, *Рейтинг за семестр*}.

3.5.3. Ключ отношения

В реляционной модели понятие **функциональной зависимости** прежде всего связано с понятием ключа. В отношении должны быть такие атрибуты, называемые ключом, что все остальные (называемые неключевыми) функционально зависят от ключа. Другими словами, если известны значения атрибутов ключа, то не ключевые атрибуты отношения имеют вполне определенное, единственное значение.

Другое определение ключа – один либо несколько атрибутов, значения которых однозначно определяют (идентифицируют) каждый кортеж отношения.

Требование обязательности наличия ключа в отношении приводит к важному свойству – **в отношении не может быть двух одинаковых кортежей.**

Учитывая, что все данные также будут являться ключом, речь идет о минимальном числе данных, сохраняющем свойство ключа. Такой ключ называют также **первичным.**

Рассмотрим понятие ключа более пристально для отношений, приведенных ранее на рис. 3.13.

Драганов М. получает дополнительное образование и имеет, в связи с этим, две зачетной книжки. Есть два различных студента с одинаковой фамилией Гончар А. (например, Анна и Алексей), поэтому Ф.И.О. не является ключом. Ключом является атрибут *№ зачетной книжки*, так как для конкретного значения этого атрибута значения остальных атрибутов имеют единственное значение (рис. 3.14).

Здесь некоторое сомнение вызывает тот факт, что в кортежах, относящихся к Драганову М. мы имеем дублирование (неоднозначность), однако, по прежнему, для любого значения атрибута *№ зачетной книжки* остальные атрибуты имеют единственное значение (функционально зависят от него), а в отношении нет двух одинаковых кортежей-строк, т. е. атрибут *№ зачетной книжки* является ключом. Здесь тем не менее надо отметить, что существует некоторая ненормальность, связанная с отмеченным дублированием, которая может (и должна) быть исключена при проектировании БД как системы взаимосвязанных отношений.

Схема отношения **СТУДЕНТ**

№ зач. книжки	Ф.И.О.	Номер группы	Пол	Дата рождения

Кортежи отношения **СТУДЕНТ**

427101	Гончар А.	4271	Ж	29.04.78
427102	Драганов М.	4271	М	19.01.79
427103	Зюкин М.	4271	М	26.03.79
⋮	⋮	⋮	⋮	⋮
477003	Акулинин А.	4770	М	29.04.76
549015	Драганов М.	5490	М	19.01.79

Рис. 3.14. Ключ состоит из одного атрибута

Рассмотрим вариант, когда вместо *№ зачетной книжки* возьмем атрибут *ИНН студента* (личности) (рис. 3.15). Тогда для одного значения *ИНН студента* может быть два различных кортежа для студентов обучающихся параллельно на нескольких специальностях (студент Драганов М.).

ИНН студента	Ф.И.О.	**Номер группы**	Пол	Дата рождения
703702022416	Гончар Е.	4271	Ж	29.04.78
701802022306	Драганов М.	**4271**	М	19.01.79
703702022237	Зюкин М.	4271	М	26.03.79
…	…	…	…	…
703702132509	Акулинин А.	477	М	29.04.76
701802022306	Драганов М.	**5490**	М	19.01.79

Рис. 3.15. Изменение ключевого атрибута

В этом случае ключ отношения будут составлять два атрибута *ИНН студента* и *Номер группы* потому, что только комбинация значений этих атрибутов уникальна в отношении (нет двух одинаковых кортежей).

Другой, более простой пример отношения, ключ которого состоит из двух атрибутов *№ зачетной книжки* и *Номер семестра*, приведен на рис. 3.16.

Схема отношения **СЕМЕСТР**

№ зачетной книжки	**Номер семестра**	Тип стипендии	Рейтинг за семестр

Кортежи отношения **СЕМЕСТР**

427101	1	стандартная	105
427101	2	стандартная	100
427101	3	не получает	70
⋮	⋮	⋮	⋮
427102	1	стандартная	100
427102	2	повышенная	130
⋮	⋮	⋮	⋮
477001	1	стандартная	100
477001	2	стандартная	110
477001	3	стандартная	100
⋮	⋮	⋮	⋮

Рис. 3.16. Отношение, ключ которого состоит из двух атрибутов

Следует отметить, что значения атрибутов ключа определяют смысл остальных (неключевых) атрибутов. Так, если в последнем примере убрать первый столбец, то информация в таблице-отношении потеряет смысл.

3.5.4. Нормализация отношений

Общие положения

Кроме отмеченных выше ограничений на обязательное наличие ключа, в реляционной модели данных вводится требование нормализации отношений. Теория реляционной модели данных различает пять нормальных форм, однако для практических целей достаточно соблюдать первые три формы.

Нормализация заключается в исключении частных функциональных зависимостей путем разбиения не нормализованных отношений на более простые, в которых все не ключевые атрибуты функционально зависят от полного ключа.

Требование нормализации отношений направлено на обеспечение такой их структуры, которая исключает некорректное обновление значений некоторых атрибутов и ошибки в выполнении определенных операций выборки.

Теория реляционной модели данных рассматривает пять уровней нормализации отношений – пять нормальных форм, но наибольшую практическую значимость имеют первые три нормальных формы. Аномалии более высоких форм не оказывают существенного влияния на результаты обработки отношений и встречаются крайне редко.

Первая нормальная форма (1НФ)

Отношение удовлетворяет первой нормальной форме, если все его атрибуты атомарны (неделимы), т. е. среди атрибутов нет составных или с множественными значениями.

Например, приведенное на рис. 3.17 отношение СТУДЕНТ не удовлетворяет первой нормальной форме, так как имеет атрибут место рождения, состоящий из атрибутов республика, область, город (село), а также атрибут иностранный язык (которым владеет студент) с множественными значениями для некоторых студентов.

СТУДЕНТ

Код студента	ФИО	Место рождения			Иностранный язык
		Республика	Область	Город (село)	

Рис. 3.17. Отношение не удовлетворяющее 1НФ по наличию составного атрибута

Когда говорится о невозможности иметь составной атрибут (например, место рождения) имеется в виду, что невозможно одновременно иметь (обращаться к ним) атрибуты республика, область, город и те же самые значения именовать как место рождения. Необходимо принять либо первый (детальные атрибуты), либо второй вариант. Если все же необходимо кроме места рождения иметь возможность обращаться и к атрибуту город, то необходимо ввести дополнительный атрибут город, где родился.

Более значительная специфика связана с атрибутом с множественными значениями.

Соблюдая требования одного размера атрибута во всех кортежах, мы должны бы были представить исходное отношение либо в виде рис. 3.18, либо в виде рис. 3.19.

В этом случае размер атрибута *иностранный язык*, рассчитанный под максимально возможное число языков, будет неоправданно большим, к тому же процедура поиска кортежей с заданным значением языка будет отличаться от аналогичной процедуры для атрибутов с атомарными значениями.

Код студента	Ф.И.О.	№ группы	Иностранный язык
427101	Гончар А.	4271	немецкий, английский
427102	Драганов М	4271	немецкий, шведский, польский
427103	Зюкин М.	4171	латынь
...	…	…	…
477001	Акулин А.	477	не владеет

Рис. 3.18. Размещение множества значений в одном атрибуте

Код студента	Ф.И.О.	Номер группы	Иностранный язык
427101	Гончар А.	4271	немецкий
427101	Гончар А.	4271	английский
427102	Драганов М.	4271	немецкий
427102	Драганов М.	4271	шведский
427102	Драганов М.	4271	польский
427103	Зюкин М.	4271	латынь
…	…	…	…
477001	Акулинин А.	477	не владеет

Рис. 3.19. Организация хранения атрибутов с множественными значениями в виде типичной для реляционной модели однородной структуры

Такое представление снимает предыдущие проблемы, однако порождает новые, связанные с дублированием значений первых атрибутов для студентов, владеющих несколькими языками. Кроме излишнего расхода памяти здесь возникают проблема обработки, например подсчета числа студентов, родившихся в некоторый день или в диапазоне дат. Кроме того, что более опасно, возможна аномалия обновления. Например, решено исключить значение *латынь* из учитываемых языков и при удалении соответствующего кортежа в приведенном примере приведет к потере всей информации о студенте Зюкине М.

Реляционная модель требует нормализации (приведения к 1НФ) путем разбиения исходного отношения на два следующим образом: из исходного отношения СТУДЕНТ исключается атрибут с множественным значением и получается новое отношение СТУДЕНТ1, а исключенный атрибут вместе с ключом исходного отношения образуют новое отношение СТУДЕНТ2 (рис. 3.20), причем оба атрибута являются ключевыми.

СТУДЕНТ1

Код студента	Ф.И.О.	Дата рождения

СТУДЕНТ2

Код студента	**Иностранный язык**

Рис. 3.20. Результат нормализации первичного отношения СТУДЕНТ

Необходимо обратить внимание на то, что если бы в исходном отношении был еще хотя бы один атрибут, зависимый от атрибутов *Иностранный язык* и *Код студента*, например, атрибут *Степень владения языком*, то исходное отношение не удовлетворяло бы 2НФ (рис. 3.21).

Код студента	Ф.И.О.	Номер группы	**Иностранный язык**	Степень владения языком

Рис. 3.21. Отношение, не удовлетворяющее второй нормальной форме

Вторая нормальная форма (2НФ)

Отношение удовлетворяет второй нормальной форме, если оно удовлетворяет 1НФ и не содержит атрибутов, зависящих от части ключа.

На рис. 3.22 приведено отношение СЕМЕСТР, не удовлетворяющее 2НФ по следующей причине. Ключ отношения составляют атрибуты код студента и номер семестра, так как комбинация значений именно этих атрибутов уникальна для любого кортежа отношения. Вместе с тем, если атрибуты тип стипендии в семестре и рейтинг в семестре, зависят от полного ключа, то Ф.И.О. и дата рождения являются характеристиками студента вне зависимости от семестра, т. е. зависят только от части ключа – от атрибута код студента.

При использовании такого ненормализованного отношения возникают проблемы подсчета числа объектов, отмеченные при рассмотрении примера нормализации по 1НФ. Возможна и отличная от предыдущего случая аномалия обновления. Например, если студентка с № зачетной книжки 427101 Гончар А. (Анна) меняет фамилию, то возникает опасность, что обновится значение только в первом кортеже или обновится фамилия и Гончара А. (Алексей, Андрей) в других записях.

Приведение отношения к 2НФ (нормализация по 2НФ) заключается в разбиении исходного отношения на два, одно из которых включает атрибуты ключа исходного отношения и атрибуты, зависящие от полного ключа, а второе – атрибуты зависящего от части ключа вместе с ат-

рибутами этой части. Результат нормализации исходного отношения приведен на рис. 3.23.

Схема отношения **СЕМЕСТР**

Код студента	Ф.И.О.	Дата рождения	**Номер семестра**	Тип стипендии	Рейтинг за семестр

Кортежи отношения **СЕМЕСТР**

427101	Гончар А.	29.04.78	1	стандартная	105
427101	Гончар А.	29.04.78	2	стандартная	100
427101	Гончар А.	29.04.78	3	не получает	70
⋮	⋮	⋮	⋮	⋮	⋮
427102	Драганов М.	19.01.79	1	стандартная	100
427102	Драганов М.	19.01.79	2	повышенная	130
⋮	⋮	⋮	⋮	⋮	⋮
477001	Акулинин А.	29.04.76	1	стандартная	100
477002	Гончар А.	20.01.79	2	стандартная	110
477002	Гончар А.	20.01.79	3	стандартная	100
⋮	⋮	⋮	⋮	⋮	⋮

Рис. 3.22. Отношение, не удовлетворяющее второй нормальной форме

СТУДЕНТ

Код студента	Ф.И.О.	Дата рождения

СЕМЕСТР

Код студента	**Номер семестра**	Тип стипендии в семестре	Рейтинг студента в семестре

Рис. 3.23. Результат нормализации отношения приведенного на рис. 3.19

Предлагаем читателю самостоятельно нормализовать отношение, приведенное на рис. 3.21.

Третья нормальная форма (3НФ)

Отношение удовлетворяет третьей нормальной форме, если оно удовлетворяет 2НФ, и среди его неключевых атрибутов нет зависящих от другого неключевого атрибута (нет атрибутов, транзитивно зависящих от ключа). На рис. 3.24 приведено отношение, не удовлетворяющее 3НФ.

Код студента	Ф.И.О. студента	Дата рождения	*Адрес общежития*	Ф.И.О. коменданта общежития
427101	Гончар Е.	29.04.78	Лыткина, 2	Королева И.В.
427102	Драганов М.	19.01.79	Лыткина, 4	Никитина Г.Г.
427103	Зюкин М.	26.03.79	Лыткина, 2	Королева И.В.
477001	Акулинин А.	29.04.76	Лыткина, 4	Никитина Г.Г.
...	…	…	…	…

Рис. 3.24. Отношение, не удовлетворяющее третьей нормальной форме

В этом примере ключ состоит из одного атрибута *Код студента*, атрибут *Ф.И.О. коменданта общежития* однозначно связан (с не ключевым атрибутом *Адрес общежития*, где проживает студент, т. е. функционально зависит от него (точно так же, как и от ключа). Естественно, одно и то же значение атрибута *Ф.И.О. коменданта общежития* будет повторяться во всех кортежах, относящихся к студентам, проживающим в одном и том же общежитии. При использовании такого ненормализованного отношения также возникают проблемы аномального обновления (изменения фамилии коменданта) и выборки кортежей.

Приведение отношения к 3НФ (нормализация по 3НФ) заключается в разбиении исходного отношения на два (рис. 3.25), одно из которых есть исходное отношение без атрибутов, зависящих от не ключевого атрибута. Второе отношение состоит из атрибута, от которого в исходном отношении зависели исключенные атрибуты (оно станет ключом в новом отношении) плюс атрибуты, исключенные из исходного отношения.

СТУДЕНТ

Код студента	Ф.И.О. студента	Дата рождения	Адрес общежития

ОБЩЕЖИТЕ

Адрес общежития	Ф.И.О. коменданта общежития

Рис. 3.25. Результат нормализации отношения рис. 3.24.

Нормальная форма Бойса-Кодда (НФБК)

В рассмотренных ранее НФ предполагалось, что отношения имеют только один семантический ключ, но это не всегда верно. Рассмотрим это на следующем примере.

В результате нормализации по 3НФ (нормальная форма) имеем следующее отношение *Экзаменационная ведомость* (рис. 3.26).

Номер зачетки	**Номер семестра**	**Дисциплина**	Оценка	Преподаватель
8Г6101	1	Матанализ	5	Иванов И.И.
8Г6102	1	Матанализ	4	Волков В.В.
8Г6101	1	Программирование	5	Петров П.П.
8Г6101	2	Матанализ	4	Иванов И.И.
8Г6201	1	БД и БЗ	5	Сидоров О.П.
8Г6105	1	БД и БЗ	5	Сидоров О.П.
8Г6102	2	Программирование	4	Яковлев Д.Ю.

Рис. 3.26. Отношение, удовлетворяющее 3НФ и не нормализованное по НФБК

В этом отношении однозначно можно сказать, какой преподаватель у какого студента принял экзамен по дисциплине в указанном семестре и какую оценку при этом студент получил. Однако необходимо отметить, что:

1. Один преподаватель может принимать экзамен только по одной дисциплине.
2. По одной и той же дисциплине могут принимать экзамены различные преподаватели.

В результате можно заметить, что существует ФЗ Преподаватель→ Дисциплина. Если удалить какую-нибудь запись из представленного отношения, например последнюю (где студент с зач. книжкой № 8Г6102 во 2-м семестре сдал экзамен по Программированию на 4 преподавателю Яковлеву Д.Ю.), то будет потеряна информация, что преподаватель Яковлев Д.Ю. принимает экзамен по дисциплине Программирование (это так называемая аномалия удаления). В этом случае есть еще один потенциальный ключ (Номер зачетки, Номер семестра, Преподаватель).

Таким образом, отношение, нормализованное по 3НФ, может оказаться нежелательным в структуре БД, так как 3НФ и как предшествующие НФ не учитывают того обстоятельства, что в отношении и может быть несколько потенциальных ключей, которые при этом могут иметь пересечения по составу атрибутов. Этот недостаток устранен в

усиленной формулировке требования 3НФ, данной Р. Бойсом (один из разработчиков SQL) и Э. Коддом (создатель РМД), соответствующая НФ отношения называется нормальной формой Бойса-Кодда (НФБК).

Определение НФБК: отношение находится в НФБК, если каждый детерминант отношения является потенциальным ключом, т. е. при нормализации по НФБК отсутствуют ФЗ атрибутов составного ключа от неключевых атрибутов.

В результате нормализации по НФБК исходное отношение декомпозируется на два отношения (теорема Хеза): в первом отношении в состав ключа включается детерминант неучтенной выявленной ФЗ (в нашем случае Преподаватель), а зависимая часть этой ФЗ (в нашем случае Дисциплина) из ключа исходного отношения исключается и вместе с детерминантом составляет второе отношение, в котором ключом и становится детерминант ФЗ. Представленное выше отношение в результате нормализации по НФБК декомпозируется на следующие отношения, представленные на рис. 3.27.

Номер зачетки	**Номер семестра**	**Преподаватель**	Оценка
8Г6101	1	Иванов И.И.	5
8Г6102	1	Волков В.В.	4
8Г6101	1	Петров П.П.	5
8Г6101	2	Иванов И.И.	4
8Г6201	1	Сидоров О.П.	5
8Г6105	1	Сидоров О.П.	5
8Г6102	2	Яковлев Д.Ю.	4

Преподаватель	Дисциплина
Иванов И.И.	Матанализ
Волков В.В.	Матанализ
Петров П.П.	Программирование
Сидоров О.П.	БД и БЗ
Яковлев Д.Ю.	Программирование

Рис. 3.27. Отношение, нормализованное по НФБК

НФБК является более сильным и независимым от 3НФ, так как не требует удовлетворения нормализации по другим НФ (при этом должны быть выявлены все ФЗ отношения, детерминанты которых входят в состав ключа), а также не используется понятие транзитивной зависимости.

Четвертая нормальная форма (4НФ)

Предположим, необходимо хранить информацию о приемной компании вуза, а точнее, какой абитуриент на какой факультет подал заявление и какие предметы ему при этом необходимо сдавать (рис. 3.28).

При этом необходимо учитывать, что:

- один абитуриент имеет право подавать заявление на несколько факультетов одновременно;
- каждый факультет имеет свой список сдаваемых предметов;
- один и тот же предмет может сдаваться на нескольких факультетах;
- абитуриент обязан сдавать все предметы, указанные для факультета, на который он поступает, несмотря на то, что он, может быть, уже сдавал такие же предметы на другом факультете.

Код абитуриента	Факультет	Предмет
1001	АВТФ	Математика
1001	АВТФ	Физика
1001	ИЭФ	Математика
1001	ИЭФ	Обществознание
1002	АВТФ	Математика
1002	АВТФ	Физика
1003	ИГНД	Математика
1003	ИГНД	Русский язык

Рис. 3.28. Отношение, удовлетворяющее по НФБК и не нормализованное по 4НФ

Все атрибуты этого отношения взаимно независимы и в совокупности составляют единственный первичный ключ. Отношение находится в НФБК.

Однако при использовании такой структуры может возникнуть две нежелательные ситуации. Во-первых, при попытке добавить в это отношение новую запись, например (1004, ИЭФ, Математика), мы обязаны добавить также и записи (1004, ИЭФ, Обществознание), так как все абитуриенты ИЭФ обязаны иметь один и тот же список сдаваемых предметов. Во-вторых, при попытке удалить запись (1004, ИЭФ, Математика), мы обязаны удалить также и кортеж (1004, ИЭФ, Обществознание) по той же самой причине. Кроме того, если мы удалим кортеж (1003, ИГНД, Математика), а вместе с ним и кортеж (1003, ИГНД, Русский язык), то будет потеряна информация о предметах, которые должны сдаваться на ИГНД. Таким образом, вставка и удаление кортежей не может быть выполнена независимо от других кортежей отношения.

Декомпозиция этого отношения для устранения указанных ситуаций не может быть выполнена на основе ФЗ, так как это отношение не содержит никаких ФЗ. Это отношение является полностью ключевым, но можно понять, что какая-то взаимосвязь между атрибутами имеется. Эта взаимосвязь описывается понятием **многозначной зависимости**.

Для каждого факультета (для каждого значения из X) каждый поступающий на него абитуриент (значение из Y) сдает один и тот же список предметов (набор значений из Z), и для каждого факультета (для каждого значения из X) каждый сдаваемый на факультете экзамен (значение из Z) сдается одним и тем же списком абитуриентов (набор значений из Y). В отношении «Абитуриент–Факультет–Предмет» имеется многозначная зависимость Факультет→→Абитуриент|Предмет ($X \to\to Y|Z$). Эта многозначная зависимость является нетривиальной, так как не существует ФЗ $X \to Y$ и $X \to Z$. Именно наличие этой зависимости не позволяет независимо вставлять и удалять кортежи: кортежи обязаны вставляться и удаляться одновременно целыми наборами.

Для разрешения этой аномалии, избавления многозначной зависимости, исходное отношение необходимо декомпозировать следующим образом (теорема Фейджина): если имеется набор непересекающихся атрибутов (X, Y, Z) отношения R и существует многозначная зависимость $X \to\to Y|Z$, то декомпозиция отношения R выполняется на проекции $R_1 = R[X, Y]$ и $R_2 = R[X, Z]$.

Определение 4НФ: отношение находится в 4НФ тогда и только тогда, когда отношение находится в НФБК и не содержит нетривиальных многозначных зависимостей.

Приведенное отношение «Абитуриент–Факультет–Предмет» находится в НФБК, но не в 4 НФ. Для того чтобы отношение было нормализовано по 4НФ, необходимо убрать из него многозначную зависимость, для чего декомпозируем исходное отношение на следующие отношения, представленные на рис. 3.29.

В полученных отношениях устранены аномалии вставки и удаления, характерные для отношения «Абитуриенты–Факультеты–Предметы». Заметим, что полученные отношения остались полностью ключевыми, и в них по-прежнему нет ФЗ.

Отношения с нетривиальными многозначными зависимостями возникают, как правило, в результате естественного соединения двух отношений по общему атрибуту. Фактически это приводит к попытке хранить в одном отношении информацию о двух независимых, разных сущностях. В качестве еще одного примера можно привести ситуацию, когда в одном отношении храниться информация о том, что студент изучает несколько дисциплин и посещает несколько спортивных секций, что приводит к возникновению нетривиальной многозначной зависимости Студент→→Дисциплина|СпортСекции.

Факультет	Код абитуриента
АВТФ	1001
ИЭФ	1001
АВТФ	1002
ИГНД	1003

Факультет	Предмет
АВТФ	Математика
АВТФ	Физика
ИЭФ	Математика
ИЭФ	Обществознание
ИГНД	Математика
ИГНД	Русский язык

Рис. 3.29. Отношение, нормализованное по 4НФ

Пятая нормальная форма (5НФ)

До сих пор предполагалось, что единственной необходимой или допустимой операцией в процессе нормализации является замена переменной отношения по правилам декомпозиции без потерь точно двумя ее проекциями. Однако существуют отношения, для которых нельзя выполнить декомпозицию без потерь на две проекции, но которые можно подвергнуть декомпозиции без потерь на три или большее количество проекций.

Рассмотрим следующий пример (рис. 3.30).

Преподаватель	Дисциплина	Группа
Иванов И.И.	Программирование	8551
Петров П.П.	Матанализ	8551
Петров П.П.	Матанализ	8552
Сидоров О.П.	ИС	8552
Сидоров О.П.	БД	8МЯ61
Яковлев Д.Ю.	Матанализ	8МЯ61

Рис. 3.30. Отношение, не нормализованное по 5НФ

В этом отношении один преподаватель может читать лекции по нескольким дисциплинам, при этом по каждой дисциплине он может читать лекции для разных групп. Одну дисциплину могут читать несколько преподавателей, но у одной группы по определенной дисциплине читает лекции только один преподаватель.

Представленное отношение находится в 4НФ. Однако при увольнении преподавателя необходимо удалить все записи, в которых указано, по каким дисциплинам и у каких групп он читал лекции.

Если отношение «*X-Y-Z*» спроецировать на составные парные атрибуты {*X-Y*}, {*Y-Z*}, {*X-Z*}, то соединение этих проекций дает исходное отношение. Это означает, что в рассматриваемом отношении существовала **зависимость соединения**. В результате отношение будет находиться в 5НФ.

Определение 5НФ: отношения находятся в 5НФ, которую иногда иначе называют проекционно-соединительной нормальной формой (ПСНФ), тогда и только тогда, когда каждая нетривиальная зависимость соединения в отношении определяется его потенциальным ключом.

Применим нормализацию по 5НФ для рассматриваемого примера. Разбиваем отношение «Преподаватель–Дисциплина–Группа» на три проекции: {Преподаватель–Дисциплина} (ПД), {Дисциплина–Группа} (ДГ), {Преподаватель–Группа} (ПГ) (рис. 3.31):

ПД

Преподаватель	**Дисциплина**
Иванов И.И.	Программирование
Петров П.П.	Матанализ
Сидоров О.П.	ИС
Сидоров О.П.	БД
Яковлев Д.Ю.	Матанализ

ДГ

Дисциплина	**Группа**
Программирование	8551
Матанализ	8551
Матанализ	8552
ИС	8552
БД	8МЯ61
Матанализ	8МЯ61

ПГ

Преподаватель	**Группа**
Иванов И.И.	8551
Петров П.П.	8551
Петров П.П.	8552
Сидоров О.П.	8552
Сидоров О.П.	8МЯ61
Яковлев Д.Ю.	8МЯ61

Рис. 3.31. Проекции исходного отношения для нормализации по 5НФ

Получим попарные соединения полученных трех приведенных выше проекций, которые будут иметь вид (рис. 3.32):

ПД-ПГ

Преподаватель	Дисциплина	Группа
Иванов И.И.	Программирование	8551
Петров П.П.	Матанализ	8551
Петров П.П.	Матанализ	8552
Сидоров О.П.	ИС	8552
Сидоров О.П.	ИС	8МЯ61
Сидоров О.П.	БД	8МЯ61
Сидоров О.П.	БД	8552
Яковлев Д.Ю.	Матанализ	8МЯ61

ПД-ДГ

Преподаватель	Дисциплина	Группа
Иванов И.И.	Программирование	8551
Петров П.П.	Матанализ	8551
Петров П.П.	Матанализ	8552
Петров П.П.	Матанализ	8МЯ61
Сидоров О.П.	ИС	8552
Сидоров О.П.	БД	8МЯ61
Яковлев Д.Ю.	Матанализ	8551
Яковлев Д.Ю.	Матанализ	8552
Яковлев Д.Ю.	Матанализ	8МЯ61

ДГ-ПГ

Преподаватель	Дисциплина	Группа
Иванов И.И.	Программирование	8551
Петров П.П.	Программирование	8551
Иванов И.И.	Матанализ	8551
Петров П.П.	Матанализ	8551
Петров П.П.	Матанализ	8552
Сидоров О.П.	Матанализ	8552
Петров П.П.	ИС	8552
Сидоров О.П.	ИС	8552
Сидоров О.П.	БД	8МЯ61
Яковлев Д.Ю.	БД	8МЯ61
Сидоров О.П.	Матанализ	8МЯ61
Яковлев Д.Ю.	Матанализ	8МЯ61

Рис. 3.32. Отношения, полученные в результате соединения проекций исходного отношения

Далее в результате пересечения полученных проекций получаем исходное отношение, представленное на рис. 3.30.

5НФ является последней из известных НФ. Условия ее получения довольно нетривиальны, и поэтому она почти не используется на практике. Кроме того, она имеет определенные недостатки: в полученных отношениях на основе проекций исходного отношения содержатся кортежи с ложной информацией (которой нет в исходном отношении), так как были эти отношения были получены в результате соединения проекций. Поэтому для правильной интерпретации необходимо рассматривать совместно все проекции основного отношения. На практике проектирования БД ограничиваются нормализацией по 3НФ или НФБК.

3.5.5. Операции реляционной алгебры

Общие положения

Оригинальность подхода Кодда состояла в том, что кроме традиционных для СУБД того времени операций вставки, замены, удаления и простой выборки, он предложил применять к отношениям стройную систему операций реляционной алгебры.

Это имеет важное практическое значение потому, что, имея в основе строгий формальный аппарат, можно создать более надежные системы управления базами данных.

Кроме того, весьма важно то, что **операциям реляционной алгебры соответствуют простые типовые задачи по обработке данных** и при наличии СУБД, реализующих эти операции, программистам **облегчается задача проектирования приложений**, а при создании соответствующего интерфейса даже **конечные пользователи могут непосредственно обращаться к реляционным базам данных** с простыми информационными запросами.

Восемь основных операций над отношениями реализуются в реляционной модели данных:

- пять традиционных операций над множествами – **объединение, пересечение, разность, декартово произведение, деление;**
- три специальные реляционные операции – **проекции, выбора (селекции) и соединения.**

Учитывая, что операции **объединения, пересечения** и **разности** выполняются над парой отношений, которые должны удовлетворять определенным требованиям, определим следующее понятие.

Два отношения являются **совместимыми по объединению**, если ***имеют одинаковое число атрибутов и i-й атрибут одного отноше-***

ния должен быть определен на том же домене (значения должны быть из того же домена), что и i-й атрибут второго отношения.

Объединением ***двух совместимых по объединению отношений R1 и R2 является отношение R3, содержащее множество всех кортежей, принадлежащих или R1, или R2, или обоим вместе.***

Другой вариант завершения этого определения: ***«…, принадлежащих R1 и тех кортежей R2, которые не принадлежат R1»***.

Последнее определение подчеркивает, что **в результирующем отношении не должно быть совпадающих кортежей (дублей)**. Учитывая, что в отношении по определению не может быть одинаковых кортежей, далее мы не будем подчеркивать это требование к результирующему отношению.

Естественно, что **R3** также совместимо по объединению с **R1** и **R2**.

Ниже приведем пример (рис. 3.33).

ПРЕПОДАВАТЕЛЬ

Личный номер преподавателя	Ф.И.О. преподавателя	Дата рождения	Пол	Адрес

НАУЧНЫЙ РАБОТНИК

Личный номер работника	Ф.И.О. научного работника	Дата рождения	Пол	Адрес

СОТРУДНИК

Личный номер сотрудника	Ф.И.О. сотрудника	Дата рождения	Пол	Адрес

Рис. 3.33. Отношения, совместимые по объединению

Этот пример, как и все последующие, подтверждает приведенный ранее тезис о том, что **с помощью одной реляционной операции решается простейшая, но типичная информационная задача.**

Типичные практические задачи, решаемые через операцию объединения – слияние файлов однотипных записей с исключением дублирующих записей. **Например, слияние файлов приемных комиссий факультетов в единый файл вуза.**

Пересечением двух совместимых по объединению отношений R1 и R2 является отношение R3, содержащее кортежи, принадлежащие как R1, так и R2.

Рассмотрим пример (рис. 3.34). Пусть имеем отношения **СТУДЕНТ**, содержащее сведения о всех студентах института, и **ЖИЛЕЦ**, содержащее сведения о всех проживающих в общежитии.

СТУДЕНТ

Ф.И.О. студента	**Серия и номер паспорта**	Пол	Дата рождения	Семейное положение

ЖИЛЕЦ

Ф.И.О. жильца	**Серия и номер паспорта**	Пол	Дата рождения	Семейное положение

Рис. 3.34. Отношения для пересечения

В результате выполнения операции пересечения получаем отношение **СТУДЕНТ0**, содержащее сведения о студентах, проживающих в общежитии (рис. 3.35).

СТУДЕНТ0

Ф.И.О. студента	**Серия и номер паспорта**	Пол	Дата рождения	Семейное положение

Рис. 3.35. Отношение, полученное в результате пересечения исходных отношений

Разностью ***двух совместимых по объединению отношений R1 и R2 является отношение R3, кортежи которого принадлежат R1, но не принадлежат R2 (т. е. кортежи из R1, которых нет в R2).***

На примере, приведенном ранее на рис. 3.34, в результате вычитания отношения **СТУДЕНТ** из отношения **ЖИЛЕЦ** получаем отношение **НЕСТУДЕНТ**, содержащее сведения о всех проживающих в общежитии, но не являющихся студентами (рис. 3.36).

А в результате вычитания отношения **ЖИЛЕЦ** из отношения **СТУДЕНТ**, получаем отношение **СТУДЕНТ**, содержащее сведения о студентах, не проживающих в общежитии.

НЕСТУДЕНТ

Ф.И.О. студента	**Серия и номер паспорта**	Пол	Дата рождения	Семейное положение

Рис. 3.36. Отношение, полученное в результате разности исходных отношений

Здесь также с помощью одной операции выполняются важные для управленцев информационные запросы.

Декартовым произведением *отношения А со схемой* $(A_1, A_2, ..., A_n)$ ***и отношения В со схемой*** $(B_1, B_2, ..., B_m)$ ***является отношение С, со схемой*** $(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m$ ***равной объединению схем отношений А и В, кортежи которого получены путем конкатенации (присоединения) каждого кортежа отношения В с каждым кортежем отношения А.***

ПРИМЕР. Пусть имеем отношение АВТ, **содержащее список участников команды факультета** Автоматики и вычислительной техники **по шахматам и отношение** РАЦ, **содержащее аналогичный список команды** Российско-американского центра**. Тогда декартовым произведением** АВТ × РАЦ **будет отношение** ВСТРЕЧИ, **содержащее список пар участников, которые должны играть друг с другом (рис. 3.37).**

АВТ

Ф.И.О. игрока АВТ	Разряд	Дата рождения

РАЦ

Ф.И.О. игрока РАЦ	Разряд	Дата рождения

ВСТРЕЧИ

Ф.И.О. игрока АВТ	Разряд игрока АВТ	Дата рождения игрока АВТ	Ф.И.О. игрока РАЦ	Разряд игрока РАЦ	Дата рождения игрока РАЦ

Рис. 3.37. Пример декартово произведения

Операция* деления *одного отношения (делимого) на другое отношение (делитель) может быть выполнена, если все множество атрибутов делителя является подмножеством атрибутов делимого.

Результирующее отношение содержит только те атрибуты делимого, которых нет в делителе. Пример приведен на рис. 3.38.

В него включаются только те кортежи, декартово произведение которых с делителем содержится в делимом (является подмножеством делимого).

Проекция – это операция построения нового отношения путем выбора одних и исключения других атрибутов из исходного отношения.

ДЕЛИМОЕ

Ф.И.О.	**Иностр. язык**	**Степень владения языком**

ДЕЛИТЕЛЬ

Иностр. язык	**Степень владения языком**

ДЕЛИМОЕ

Ф.И.О.	**Иностр. язык**	**Степень владения языком**
Иванов	английский	свободно
Сидоров	английский	свободно
Сидоров	японский	разговорный
Сидоров	турецкий	со словарем
Кузнецова	английский	со словарем
Кузнецова	турецкий	свободно
Иванова	немецкий	разговорный
Иванова	турецкий	свободно
Иванова	английский	со словарем

ДЕЛИТЕЛЬ

Иностр. язык	**Степень владения языком**
английский	со словарем
турецкий	свободно

ЧАСТНОЕ (результат)

Ф.И.О.
Кузнецова
Иванова

Рис. 3.38. Пример деления отношения

Второй вариант определения: ***«… нового отношения, множество атрибутов которого является подмножеством атрибутов исходного отношения»***.

Здесь вновь уместно напомнить о необходимости исключения дублирующих кортежей, так как если из исходного отношения не выбран хотя бы один атрибут ключа, в результирующем отношении, как правило, появятся дублирующие друг друга кортежи.

На примере отношения СТУДЕНТ выполняем операцию проекции по атрибуту **номер группы**, получаем отношение **группа**, представляющую собой список всех групп, в которых учатся студенты (рис. 3.39).

СТУДЕНТ

Код студента	Ф.И.О. студента	Номер группы	Дата Рождения

ГРУППА

Номер группы

Рис 3.39. Пример проекции отношения

На практике чаще всего проекция выполняется с целью сокращения размерности (числа атрибутов) для отображения информации конечному пользователю. При этом ключевые атрибуты, как правило, включаются в результирующее отношение и сокращения числа кортежей не происходит.

Выбор (селекция) – операция получения нового отношения с той же схемой, что и исходное отношение, но кортежи которого являются подмножеством кортежей исходного. В него включаются только те кортежи исходного отношения, значения определенных атрибутов которых удовлетворяют заданным ограничениям.

Ограничения могут быть заданы в виде логического выражения, элементами которого являются простейшие условия типа **«Имя атрибута – знак сравнения – значение атрибута»** (в общем случае «Выражение – знак сравнения – выражение»), которые могут соединяться булевыми операторами **И (AND), ИЛИ (OR),** иметь знак отрицания (**NOT**), а также использовать **круглые скобки** для изменения старшинства выполнения булевых операций по правилам математической логики.

Знаки сравнения **>, <, =, ≠, >=, <=.**

Например, для отношения СТУДЕНТ (рис. 3.40) требуется отобрать кортежи, относящиеся к первому году учебы, причем для студентов, не получающих стипендию, но имеющих достаточно высокий рейтинг – от 800 до 900 баллов.

Условие отбора **(номер_семестра = 1 *ИЛИ* номер_семестра = 2) *И* тип стипендии в семестре = не получает *И* (рейтинг за семестр ≥ 800 *И* рейтинг за семестр ≤ 900).**

СТУДЕНТ

Код студента	Ф.И.О. студента	Номер семестра	Тип стипендии в семестре	Рейтинг за семестр

Рис. 3.40. Отношение для селекции

Операции проекции и выбора являются основными в информационно-справочных системах и чаще всего используются совместно. Так в приведенном примере не имело смысла включать в результирующее отношение атрибут **тип стипендии в семестре**, так как во всех кортежах он будет равен значению **не получает**.

Соединением *отношения A по атрибуту X с отношением B по атрибуту Y называется множество всех кортежей, являющихся конкатенацией таких кортежей a* $\in$ *A и кортежей b* $\in$ *B, для которых выполняется условие X * Y (под * понимается одна из операций сравнения* $=, \neq, <, >, \geq, \leq$*). X и Y должны быть определены на одном и том же домене.*

Чаще всего условие содержит знак **=**, а сама операция служит для соединения двух таблиц в одну с целью упрощения последующей обработки.

Например, если для расчета суммарного рейтинга студентов (рейтинг группы равен сумме рейтингов студентов за семестры) необходима информация из двух отношений:

СТУДЕНТ

Код студента	Ф.И.О.	Номер группы	Дата рождения

СЕМЕСТР

Код студента	**Номер семестра**	Тип стипендии	Рейтинг за семестр

в результате операции *соединения* получим отношение (рис. 3.41).

Код студ-та	Ф.И.О. студ-та	Номер группы	Дата рождения	***Код студ та***	Номер семестра	Тип стипендии	Рейтинг за семестр

Рис. 3.41. Пример соединения отношений

Алгоритм обработки такого единого отношения проще, чем при работе с двумя отношениями за счет исключения необходимости согласования кортежей из разных отношений. Но необходимо помнить о дублировании значений некоторых атрибутов.

Чаще всего операция *соединения* используется вместе с операциями *проекции* и *селекции* (или предшествует им). В приведенном примере нет смысла иметь два атрибута **Код студента.**

Тот факт, что при соединении часто нарушается требование нормальности по **2НФ**, не существенен, так как соединение реализуется временно, на период обработки связываемых отношений (**денормализация**).

3.5.6. Языки обработки реляционных БД

В качестве языка обработки реляционных БД рассмотрим язык структурированных запросов **SQL**.

История SQL

В начале 1970-х годов после появления в июне 1970 г. работы Э. Кодда «Реляционная модель данных для больших совместно используемых банков данных» корпорация IBM была разработана экспериментальная реляционная СУБД IBM System R, для которой был создан специальный язык SEQUEL, позволявший относительно просто управлять данными в этой СУБД [5]. Целью разработки было создание простого непроцедурного языка, которым мог воспользоваться любой пользователь, даже не имеющий навыков программирования. Собственно разработкой языка запросов занимались Дональд Чэмбэрлин (Donald D. Chamberlin) и Рэй Бойс (Ray Boyce). Пэт Селинджер (Pat Selinger) занималась разработкой стоимостного оптимизатора (cost-based optimizer), Рэймонд Лори (Raymond Lorie) занимался компилятором запросов.

Первоначальное название языка было SEQUEL – акроним «Structured English Query Language» – «английский язык структурированных запросов» и произносится как «сиквэл». Позже по юридическим соображениям[1] язык SEQUEL был переименован в SQL – аббревиатура «Structured Query Language» – «язык структурированных запросов», и официальное произношение названия стало побуквенным «эс-кью-эль» (однако многие англоговорящие разработчики БД до сих пор используют первое произношение названия языка) [6]. В 1979 г. корпорация Relational Software (впоследствии ставшей Oracle) представила коммерческую реализацию SQL для СУБД Oracle V2 для машин VAX. Корпорация Oracle отобрала у IBM права на SQL и сейчас является собственником всех выпускаемых на рынок SQL продуктов.

---

[1] «SEQUEL» был торговой маркой британской авиастроительной группой компаний «Hawker Siddeley».

Стандарты SQL

После появления этого языка стал появляться ряд конкурирующих программ SQL на рынке, необходимо было иметь какой-то стандарт языка, к которому они должны быть приведены и следовать. В 1986 г. первый официальный стандарт языка SQL-86 был принят ANSI (American National Standards Institute – Американский национальный институт стандартов), который был одобрен ISO (International Organization for Standardization – Международной организацией по стандартизации) в 1987 г. и несколько уточнен в 1989 г. Дальнейшее развитие языка поставщиками СУБД потребовало принятия в 1992 г. нового расширенного стандарта SQL-92 (SQL2), был определен базовый уровень (Entry level) соответствия реализации этому стандарту. Следующим стандартом стал SQL:1999 (SQL3), в котором добавлена поддержка регулярных выражений, рекурсивных запросов, поддержка триггеров, базовые процедурные расширения, нескалярные типы данных и некоторые объектно-ориентированные возможности. В настоящее время действует принятый в 2003 г. стандарт SQL:2003, в котором введены расширения для работы с XML-данными, оконные функции (применяемые для работы с OLAP-БД), генераторы последовательностей и основанные на них типы данных. Позднее были внесены некоторые модификации в этот стандарт: функциональность работы с XML-данными значительно расширена, появилась возможность совместно использовать в запросах SQL и XQuery – в результате появился стандарт SQL:2006. В стандарте SQL:2008 улучшены возможности оконных функций, устранены некоторые неоднозначности стандарта SQL:2003[7].

В виду того, что стандарт SQL определяется ANSI, который устанавливает в ходе стандартизации определенные ограничения в язык, некоторые коммерческие программные реализации БД расширяют SQL без уведомления ANSI, добавляя другие особенности в язык, которые, как они считают, будут весьма полезны. Поэтому на текущий момент все усилия по проверке СУБД на соответствие стандарту ложатся на ее производителя. Стандарт SQL, определенный ANSI, – это вид минимального стандарта и можно реализовывать больше, чем он позволяет, хотя и необходимо выполнять его указания при выполнении задач, которые он определяет.

Преимущества использования SQL

✓ Независимость от конкретной СУБД.

Несмотря на наличие диалектов и различий в синтаксисе программ, содержащих SQL-запросы, они могут быть достаточно легко перенесе-

ны из одной СУБД в другую. Существуют системы, разработчики которых изначально ориентировались на применение по меньшей мере нескольких СУБД.

✓ Наличие стандартов.

Наличие стандартов и набора тестов для выявления совместимости и соответствия конкретной реализации SQL общепринятому стандарту только способствует «стабилизации» языка.

✓ Декларативность.

С помощью SQL программист описывает только то, какие данные нужно обработать. То, каким образом это сделать, решает СУБД непосредственно при обработке SQL-запроса. Однако программист при этом должен представлять, как СУБД будет разбирать текст его запроса. Чем сложнее сконструирован запрос, тем больше он допускает вариантов написания, различных по скорости выполнения, но одинаковых по итоговому набору данных.

Недостатки использования SQL

✓ Несоответствие реляционной модели данных.

Создатели реляционной модели данных Э. Кодд, К. Дейт и их сторонники указывают на то, что SQL не является истинно реляционным языком. В частности, они указывают на следующие проблемы SQL [8]:

- повторяющиеся строки;
- неопределенные значения (null);
- явное указание порядка столбцов слева направо;
- столбцы без имени и дублирующиеся имена столбцов;
- отсутствие поддержки свойства «=»;
- использование указателей;
- высокая избыточность.

✓ Сложность.

Хотя SQL и задумывался как средство работы конечного пользователя, в конце концов он стал настолько сложным, что превратился в инструмент программиста.

✓ Отступления от стандартов

Несмотря на наличие международного стандарта ANSI SQL-92, многие компании, занимающиеся разработкой СУБД (например, Oracle, Sybase, Microsoft, MySQL AB), вносят изменения в SQL, применяемый в разрабатываемой СУБД, тем самым отступая от стандарта. Таким образом, появляются специфичные для каждой конкретной СУБД диалекты языка.

Структура языка

Основное и единственное назначение SQL – использование языка в написании запросов. Запрос SQL – команда на языке SQL, которую выполняет СУБД для получения необходимой информации из БД.

В зависимости от цели использования существуют пять типов SQL-запросов [9]:

1. **DDL** (Data Definition Language – язык определения данных) позволяет:
   - создавать (CREATE), изменять (ALTER), удалять (DROP) объекты БД (таблицы (TABLE), представления (VIEW), индексы (INDEX), последовательности (SEQUENCE), триггеры (TRIGGER) и др.);
   - назначать (GRANT) и лишать (REVOKE) привилегий и ролей (ROLE) пользователей;
   - анализировать (ANALYZE) информацию таблицы, индекса, кластера;
   - проводить аудит (AUDIT) – процесс регистрации действий, производимых с объектами БД;
   - добавлять комментарии (COMMENT) к объектам БД.
2. **DML** (Data Manipulation Language – язык манипулирования данных) позволяет получить доступ и манипулировать данными в существующих объектах БД (таблицах, представлениях):
   - выбирать (SELECT);
   - вставлять (INSERT);
   - обновлять (UPDATE);
   - удалять (DELETE);
   - выполнять слияние (MERGE).
3. **Transaction Control** (управление транзакциями) позволяет управлять изменениями, сделанными в результате DML-запросов. Пользователь группирует DML-запросы (один или несколько) в логическую единицу, называемую *транзакцией*, которая является атомарной, т. е. выполняется либо целиком, либо не выполняется вовсе. Для того чтобы зафиксировать текущее состояние системы, используется точка сохранения, с целью возврата в любой следующий момент к зафиксированному с помощью нее (точки сохранения) состоянию. Благодаря использованию этого блока можно выполнять следующие действия:
   - фиксировать (COMMIT) транзакции;
   - откатывать (ROLLBACK) транзакции;
   - устанавливать точки сохранения (SAVEPOINT).

4. **Session Control** (управление сессиями) позволяет динамически управлять свойствами текущей сессии пользователя:
   - устанавливать роли (SET ROLE) пользователю, делать их доступными или недоступными;
   - изменять настройки сессии (ALTER SESSION), например языковые, временные и др.;
5. **System Control** (управление системой) позволяет динамически управлять свойствами экземпляра БД (ALTER SYSTEM), например изменять количество разделяемых серверов, убивать сессию и др.

Необходимо отметить, что SQL широко используется как встроенный язык, т. е. SQL-запросы (DDL, DML, Transaction Control) включаются в программы, написанные на процедурных языках (например, Pro*C/C++, Pro*COBOL и др.); прекомпилятор Oracle интерпретирует встроенные SQL-запросы и переводит их в выражения, которые понятны компилятору процедурного языка.

Синтаксис записи формата запросов SQL

1. Слова, написанные прописными латинскими буквами, являются зарезервированными словами **SQL**.
2. Слова, написанные строчными буквами и заключенные в кавычки, именуют конструкцию, которую необходимо раскрыть дополнительно.
3. Слова, написанные строчными буквами и незаключенные в кавычки, именуют элементарное (не требующее дополнительного описания) понятие.
4. То же, что и п. 3., но ограниченное символами / является комментарием.
5. Фрагменты, заключенные в фигурные скобки и разделенные символом **|**, являются альтернативными **{ | }**. При записи команды для конкретного применения необходимо выбрать одни из них.
6. Фрагмент, заключенный в квадратные скобки **[ ]**, возможно не будет использоваться при записи команды для конкретного применения.
7. Круглые скобки **( )** используются:
   - в выражениях для изменения порядка выполнения операций;
   - для объединения некоторых фрагментов в единое целое.
8. Многоточие, стоящее перед закрывающейся квадратной или круглой скобкой, означает, что заключенный в эти скобки фрагмент может быть повторен много раз.
9. Символ **:=** служит для соединения левой раскрываемой части с раскрывающей ее правой частью и читается как «по определению есть».

Ниже рассмотрим синтаксис и примеры использования SQL-запросов типа DML, к числу которых относится запрос на выборку (SELECT).

Синтаксис запроса на выборку записей из таблицы БД имеет следующий вид:

SELECT {* | **[{DISTINCT** | **ALL}]** «выражение»[, «выражение»…]}

[**INTO** список переменных включающего языка]

FROM «ссылка на таблицу» [,«ссылка на таблицу» …]

[**WHERE** [«условие соединения» [**AND** «условие соединения»…]]

«условие фильтра» [{**AND** | **OR**} «условие фильтра»…]]]/возможно применение круглых скобок для изменения порядка выполнения операции **AND** и **OR**

[**GROUP BY** «столбец группировки» [,«столбец группировки»…]]

[**HAVING** «условие фильтра»]

[**ORDER BY** «столбец упорядочения» [{**ASC** | **DESC**}] [,«столбец упорядочения» [{**ASC** | **DESC**}] …]]

ALL выбор всех, в т. ч. дублируемых записей (по умолчанию)

DISTINCT – исключаются дубли записей

«выражение» := «первичный» | «первичный» «оператор» | «выражение»

«первичный» := «имя столбца» | «литерал» | «функция агрегирования» | «встроенная константа» | «нестандартная функция»

«имя столбца» := [«ссылка на таблицу».]«идентификатор»

«Оператор» := + | – | / | *

«Литерал» := «строка» | математическое выражение

«Строка» := строка любых символов, заключенная в кавычки

«Функция агрегирования» := **AVG** («имя столбца») | **COUNT** (*) | **COUNT** ([{**DISTINCT** | **ALL**}] «имя столбца») | **MIN** («имя столбца») | **MAX** («имя столбца») | **SUM** («имя столбца»)

COUNT (*) /подсчет числа записей в группе/

COUNT ("имя столбца") /подсчет числа записей с **NOT NULL** значениями столбца.

«Ссылка на таблицу»: = «имя таблицы» [«алиас»] / алиас – синоним имени таблицы. Необходим при связывании таблицы самой с собой. Действует только на время выполнения команды

«Имя таблицы»: = «идентификатор»

«идентификатор»:= буква [{буква | цифра | символ подчеркивания}…]

«Условие соединения»: = «имя столбца 1» «оператор сравнения» «имя столбца 2»

«Имя столбца 1» и «имя столбца 2»: = имена столбцов из связываемых таблиц, причем ссылка на таблицу, которой принадлежит столбец, обязательна

«Оператор сравнения»: = = | < | > | <= | >= | <>

«Условие фильтра» : = {«Выражение» «оператор сравнения» «выражение» |«Выражение» **[NOT]** **BETWEEN** «выражение» **AND** «Выражение» | «Выражение» [**NOT**] **IN** {«список значений» |«подзапрос»} | «Имя столбца» [**NOT**] **LIKE** «образец» | «имя столбца» **IS[NOT] NULL** | Выражение» «оператор сравнения» {**ALL** | **ANY** | **SOME**} «подзапрос» [**NOT] EXISTS** «подзапрос»}

«Подзапрос»: = **SELECT…**

В условии фильтра могут использоваться **( )** для изменения порядка выполнения логических операторов, соединяющих элементарные условия фильтра.

«Столбец упорядочения»: = «имя столбца» из выражений фразы **SELECT**

«Столбец группировки»: = «имя столбца» из выражений фразы **SELECT** /группировка применяется для функций агрегирования/

Синтаксис запроса на создание таблицы БД:

CREATE TABLE «имя таблицы» («имя столбца» «тип данных» [«размер»] [«ограничение на столбец»…] [«значение по умолчанию»] [,«имя столбца» «тип данных» [«размер»] [«ограничение на столбец»…] [«значение по умолчанию»]…] «ограничение на таблицу» [, «ограничение на таблицу»…])

“ограничение на столбец” : = **NOT NULL** | **UNIQUE** | **PRIMARY KEY** | **CHECK** (условие) |

REFERENCES имя таблицы [(имя столбца)]

/столбцы, являющиеся внешним ключом/

PRIMERY KEY и **UNIQUE** одинаковое действие на уникальность значения, а

PRIMERY KEY указывает на первичный ключ

«Ограничение на таблицу»: = **UNIQUE** (список столбцов) |

PRIMARY KEY (список столбцов) |

/Не в стандарте / **FOREIGN KEY** (список столбцов) |

REFERENCES имя таблицы [(список столбцов)]

CHECK (условие)

«Условие» = «условие фильтра» /в случае ограничения на таблицу по нескольким столбцам/

«Значение по умолчанию» : = **DEFAULT VALUE** = «выражение»

Примеры запросов на выборку из следующих таблиц

Все примеры запросов SQL рассматриваются на примере структуры БД, состоящей из следующих таблиц (рис. 3.42).

Во всех приведенных примерах запросов SQL используются следующие правила:

- Слова, написанные прописными латинскими буквами, являются зарезервированными словами **SQL**.
- Слова, написанные строчными курсивными буквами, именуют конструкцию, которую необходимо раскрыть дополнительно.
- Запрос заканчивается знаком «;».

РАСПИСАНИЕ

номер группы	**тип недели**	**день недели**	**номер занятия**	корпус	аудитория	Дисциплина	вид занятий	ФИО преподавателя

СТУДЕНТ

номер зачетки	номер группы	ФИО	размер стипендии	суммарный рейтинг

ДЕНЬ НЕДЕЛИ

день недели	номер дня недели

НЕДЕЛЯ

дата пн	**дата сб**	номер недели	тип недели

АУДИТОРИЯ

корпус	**аудитория**	число мест	состояние доски

Рис. 3.42. Исходные таблицы для примеров запросов SQL

❖ В самом простом виде запрос представляет собой выбор (**SELECT)** всех строк по указанным столбцам из (**FROM)** таблицы и выглядит так:

SELECT *имена_столбцов*
FROM *таблица;*

Если необходимо вывести все столбцы таблицы, тогда используется знак * вместо перечисления всех имен столбцов.

SELECT *
FROM *таблица;*
Например, вывести список дней недели и их номера:

SELECT день_недели, номер_дня_недели
FROM день_недели;

Для вывода уникальных записей используется DISTINCT после SELECT и указывается имя столбца, по которому устанавливается уникальность записей.

Например, вывести список групп студентов, так как в таблице о студентах хранятся дублирующие значения групп (в одной группе учатся много студентов), то необходимо использовать DISTINCT:

SELECT DISTINCT группа
FROM студент;

❖ Можно задать ограничение на выбор определенных значений, указав после фразы WHERE условие на те столбцы, значения которых должны ему удовлетворять.

Условие может включать операторы сравнения (=, >, <, <>), логические выражения (true, false) и др. Условие должно возвращать логический результат (быть истинным или ложным). Условие может содержать несколько подусловий, которые соединяются между собой логическими операциями AND (и) и OR (или), и порядок выполнения подусловий можно изменять при помощи круглых скобок. Причем сначала выполняются операторы сравнения, затем AND и OR.

SELECT *имена_столбцов*
FROM *название_таблицы*
WHERE *условие;*

Например, вывести информацию об аудиториях 19-го корпуса, в которых число мест больше 30:

SELECT аудитория, число_мест, состояние_доски

FROM аудитория
WHERE корпус = ’19’ AND число_мест > 30;

Кроме того, в SQL существуют собственные операторы сравнения, к числу которых относятся:

- **BETWEEN … AND …** проверяет вхождение в диапазон значений, указанных между словом AND;
- **IN (list)** – проверяет вхождение в указанный список значений;
- **LIKE** – проверяет на соответствие с заданной маской;
- **IS NULL** – проверяет, является ли неизвестным значением.

Например, вывести информацию о студентах, учащиеся в группах, номер которых начинается на ‘7’:

SELECT *
FROM студент
WHERE номер_группы like ‘7%’;

В маске знак _ означает любой символ; знак % означает любую последовательность символов.

Для каждого типа данных существуют набор функций для манипулирования значениями, к которому (типу) они относятся. Подробнее см. документацию.

Символьные функции

Для преобразования регистра символов

- **LOWER (все символы строчные);**
- **UPPER (все символы прописные);**
- **INITCAP** (первая буква прописная, остальные строчные).

Для манипулирования символами

- **CONCAT** (конкатенация – соединение двух строк в одну);
- **SUBSTR** (получение подстроки из строки);
- **LENGTH** (длина строки);
- **INSTR** (поиск подстроки в строке);
- **TRIM** (удаление символов с начала / конца строки);
- **LPAD/RPAD** (добавление символов с начала / конца строки до определенной длины);
- **REPLACE** (замена символов в строке).

Например, вывести все группы, в которых учатся Ивановы:

SELECT номер_группы, ФИО

FROM студент
WHERE UPPER(SUBSTR(ФИО,1,6)) like ‘ИВАНОВ’;
Функции для дат:

- **MONTH_BETWEEN** (получение числа месяцев между двумя датами);
- **ADD_MONTH** (добавление календарных месяцев к дате);
- **NEXT_DAY** (ближайшая дата, когда наступит заданный день недели);
- **LAST_DAY** (последняя дата текущего месяца);
- **SYSDATE** (текущая дата).

Например, вывести ближайшую дату от текущей даты, когда наступит пятница:

```
SELECT NEXT_DAY(SYSDATE, ‘ПЯТНИЦА’)
FROM DUAL;
```

В этом запросе используется системная таблица Oracle DUAL, которая состоит из одного столбца DUMMY (фиктивный) и одной строки со значением ‘X’; эта таблица используется для временного хранения констант, подсчета выражений, т. е. для возвращения однократного результата.

К числовым функциям относятся все математические операции над числами. Выполняя операции над значениями таблицы, получаем определенные результаты, которые необходимо вывести во фразе SELECT. Таким образом, результаты вычислений в виде выражений представляют собой вторичные данные.

Общие функции – это функции, которые одинаково работают для нескольких типов данных, к их числу относятся:

- **TRUNC** (усечение значения до заданной точности) – для дат и чисел;
- **ROUND** (округление значения до заданной точности) – для дат и чисел.

Например, вывести число мест, округленных до десятков, в аудиториях 19-го корпуса:

```
SELECT аудитория, ROUND (число_мест, -1)
FROM аудитория
WHERE корпус = ‘19’;
```

В функциях ROUND, TRUNC 2-м аргументом выступает точность, которая представляет собой положительное число для округления (отсечения) десятичных знаков числа, и отрицательное число – для целой части числа, при этом значение точности (1, 2 и т. д.) зависит от отдаленности округления (отсечения) от точки (начала целой части), т. е. 1 – для десятых/-ков, 2 – для сотых/-ен и т. д.

❖ Существует возможность отсортировать результаты запроса, указав после фразы **ORDER BY**, по каким столбцам (название столбца или его порядковый номер во фразе SELECT) в какой последовательности упорядочивать и способ упорядочивания: по возрастанию (**ASC** – по умолчанию) или убыванию (**DESC**).

SELECT *имена_столбцов*
FROM *таблица*
ORDER BY *имя(номер)_столбца способ_упорядочивания;*

Например, вывести информацию о студентах группы 8521 упорядоченно по ФИО в алфавитном порядке (т. е. по возрастанию). И в случае наличия студентов с одинаковым ФИО сначала вывести студента с большим рейтингом (т. е. по убыванию).

SELECT номер_зачетки, ФИО, размер_стипендии, суммарный_рейтинг
FROM студент
WHERE номер_группы = ‘8521’
ORDER BY ФИО, суммарный_рейтинг DESC;

❖ Зачастую бывает необходимость представить значения в сгруппированном виде и применить групповые функции для каждой группы значений. Для этого используется фраза **GROUP BY**, в которой указаны названия столбцов, по которым группируются значения. В результате группировки все значения таблицы разбиваются по группам, которые указаны во фразе GROUP BY. К каждой полученной группе значений можно применить групповые функции: **COUNT** (подсчет количества значений), **SUM** (сумма значений), **AVG** (среднее значение), **MAX** (максимальное значение), **MIN** (минимальное значение) (SUM, AVG применяются только к числовым значениям). При этом групповые функции можно использовать и без группировки, тогда все записи таблицы будут представлять одну группу. При использовании группировки во фразе SELECT перечисляются только те столбцы, по которым происходит группировка, или групповые функции к сгруппированным значениям,

возвращаемое количество строк равно количеству полученных групп. Для исключения групп применяется фраза HAVING, в которой указывается условие для групповой функции. При этом для ограничения состава групп и других значений используется WHERE.

SELECT *имена_группирующих_столбцов, групповые_функции(имя_столбца)*
FROM *таблица*
GROUP BY *имена_группирующих_столбцов*
HAVING *условие;*

Например, вывести информацию о группах: номер группы (должен начинаться на ‘8’ и упорядочен по возрастанию), количество в них студентов (должно быть больше 10 человек) и сумму размера стипендии по каждой группе.

SELECT номер_группы, count(номер_зачетки), sum (размер_стипендии)
FROM студент
WHERE номер_группы LIKE ‘8%’
GROUP BY номер_группы
HAVING count(номер_зачетки) > 10
ORDER BY *имя(номер)_столбца способ_упорядочивания*;

Порядок выполнения такого запроса следующий:

1) СУБД просматривает все записи из указанной таблицы во фразе FROM (всех студентов);
2) сравнивает значение указанного в условии столбца каждой записи с условием, указанном во фразе WHERE и выбирает только те записи, которые удовлетворяют этому условию (только тех студентов, которые учатся в группах, начинающихся на ‘8’);
3) полученные записи группирует по столбцам, указанным во фразе GROUP BY (разбивает полученные записи о студентах на группы, начинающихся на ‘8’);
4) из группированных записей выбирает только те, которые удовлетворяют условию во фразе HAVING (из полученных групп, начинающихся на ‘8’, отбираются только те, у которых количество студентов больше 10 человек);
5) для каждой полученной группы выполняются групповые функции для сгруппированных записей, указанные во фразе SELECT (по каждой учебной группе кроме подсчета количества студентов вычисляется сумма размеров стипендий);

6) в последнюю очередь выполняется сортировка по тем столбцам или групповым функциям, которые указаны в ORDER BY (они должны обязательно входить в состав SELECT).

До сих пор рассматривались запросы, в которых происходит выборка только из одной таблицы. Далее рассмотрим примеры запросов для выбора данных из нескольких таблиц. В случае наличия одинаковых столбцов в 2-х таблицах для их различения и улучшения производительности работы с БД в запросах перед именем столбца указываются через точку имена таблиц (для одной таблицы СУБД автоматически расставляет имя таблицы для всех ее столбцов). Иногда вместо имени таблицы используют алиас – это псевдоним, который назначается пользователем в самом запросе:

- таблицам – для краткости (чтобы каждый раз не писать полное имя) или для смысла (понятно для запоминания);
- столбцам, функциям или выражениям во фразе SELECT для отображения их названия в выводе результатов запроса.

Если происходит соединение нескольких таблиц и не указано условие, по которому их соединять, то образуется декартово произведение, т. е. все строки одной таблицы соединяются со всеми строками второй таблицы. Во избежание этого используется условие WHERE, причем необходимо руководствоваться следующим правилом: для соединения n таблиц требуется, по крайней мере, (n–1) условий соединения.

В общем виде запрос из нескольких таблиц выглядит следующим образом:

SELECT *таблица1.столбец1, таблица2.столбец1, таблица2.столбец2, таблица3.столбец1*
FROM *таблица1, таблица2, таблица 3*
WHERE *таблица1.столбец1 = таблица2.столбец1*
AND *таблица2.столбец2 = таблица3.столбец1;*

Например, вывести расписание группы 8521 в порядке следования дней недели (для краткости таблиц используются алиасы).

SELECT р.тип_недели, р.день_недели, р.номер_занятия, р.корпус, р.аудитория, р.дисциплина, р.вид_занятий, р.ФИО_преподавателя
FROM расписание р, день_недели дн
WHERE р.день_недели = дн.день_недели AND р.номер_группы = ‘8521’
ORDER BY р.тип_недели, дн.номер_дня_недели, р.номер_занятия;

Соединение между таблицами в запросах может быть 2-х типов:

- **эквисоединение**, когда соединение происходит по первичному и внешнему ключам таблиц. Приведенный выше пример относится к запросу такого типа.
- **неэквисоединение**, когда соединение между таблицами происходит по условию ограничения значений таблиц. Этот тип соединения используется редко, так как значения должны быть определены на одном домене и обязательно удовлетворять указанному условию.

В случае эквисоединения, когда в связующем столбце одной таблицы есть значение, которое отсутствует в связующем столбце другой таблицы, такое соединение является внешним (в условии в скобках ставится + у того столбца, у которого отсутствует значение). Например, вывести расписание всех групп, даже тех, у которых не проводятся занятия (т. е. нет для них расписания).

```
SELECT p.*
FROM расписание p, студент c
WHERE p. номер_группы (+) = c. номер_группы
ORDER BY p.номер_группы, p.тип_недели, дн.номер_дня_недели, p.номер_занятия;
```

Существуют ситуации, когда необходимо выполнить соединение таблицы с собой. Допустим, есть следующая структура таблицы, содержащая информацию о сотрудниках и их руководителей (кому они подчиняются), которые также являются сотрудниками, т. е. связь-петля таблицы с собой.

Рис. 3.43. Связь типа петля вариант 3.6. д)

Например, если необходимо вывести список ФИО сотрудников и ФИО их руководителей, то запрос будет выглядеть так:

```
SELECT с.ФИО сотрудник, р.ФИО руководитель
FROM сотрудник с, сотрудник р
WHERE с.ид_руководителя = р.ид_сотрудника
ORDER BY 1;
```

В таких запросах важную роль играют алиасы таблиц и столбцов для их семантического различения.

Выше в запросах на соединение нескольких таблиц использовался традиционный синтаксис по стандарту SQL-86, когда во фразе FROM через запятую перечисляются таблицы, а во фразе WHERE – условие, по которому их надо соединять. Стандарт ANSI SQL-92 предлагает другой вариант синтаксиса запросов на соединение таблиц: во фразе FROM указывается первая таблица, а далее во фразе JOIN указывается другая таблица и во фразе ON условие, по которому они соединяются. Для внешнего соединения вместо (+) используются слова LEFT/RIGHT для указания стороны, у которой отсутствует значение, или OUTER для обоих случаев [10]:

SELECT *таблица1.столбец1, таблица2.столбец1, таблица2.столбец2*
FROM *таблица1*
[LEFT/RIGHT/OUTER] JOIN *таблица2* ON *таблица1.столбец1 = таблица2.столбец1*
[LEFT/RIGHT/OUTER] JOIN *таблица3* ON *таблица2.столбец2 = таблица3.столбец1*
WHERE *условие;*

Несмотря на то, что смысл самого запроса не поменялся, а это всего лишь другая запись, синтаксис, запроса, она имеет некоторые преимущества от традиционного:

1. Вся информация о соединении всех таблиц располагается в одном месте. Чтобы отделить условия соединения от ограничений результатов запроса, больше не нужно формировать сложную фразу WHERE.
2. Невозможно будет не описать условия соединения, так как СУБД потребует, чтобы использовался оператор ON или другой оператор для явного описания условия соединения. Если требуется полное декартово произведение, то его необходимо задать явно.

Например, вывести расписание занятий, проходящих в 19-м корпусе в 507 аудитории в порядке следования дней недели (не выводить дни недели, когда нет занятий).

```
SELECT р.тип_недели, р.день_недели, р.номер_занятия, р_номер_группы, р.дисциплина, р.вид_занятий, р.ФИО_преподавателя
FROM расписание р
JOIN день_недели дн ON р.день_недели = дн.день_недели
WHERE р.корпус = ‘19’ AND р.аудитория = ‘507’
ORDER BY р.тип_недели, дн.номер_дня_недели, р.номер_занятия;
```

На практике достаточно часто бывает потребность в подзапросах, когда необходимо сначала получить промежуточный результат (подсчитать или вывести список), а затем использовать его для основного запроса. Синтаксис подзапроса следующий:

```
SELECT имена_столбцов
FROM таблица
WHERE выражение оператор
                                (SELECT имена_столбцов
                                FROM таблица
                                WHERE условие)
```

Например, вывести ФИО всех студентов, которые учатся в одной группе с Ивановым Иван Ивановичем.

```
SELECT ФИО
FROM студент
WHERE номер_группы = (SELECT номер_группы
                                FROM студент
                                WHERE ФИО = ‘Иванов Иван Иванович’);
```

Подзапрос – внутренний запрос, который заключен в скобки. Главный запрос – внешний относительно подзапроса.

При использовании подзапросов существуют следующие правила:

1. Подзапрос выполняется один раз до главного запроса. Результат подзапроса используется главным запросом.
2. Таблицы в главном запросе и в подзапросе могут как совпадать, так и различаться.
3. Подзапрос должен находиться справа от оператора сравнения.

Однострочные подзапросы – возвращают как результат одну строку, используют однострочные операторы сравнения: =, <, >, <>. Приведенный выше пример относится к использованию однострочного подзапроса, так как предполагается, что студент может учиться только в 1-й группе.

Еще один пример однострочного подзапроса такой: вывести всех студентов, которые получают максимальную стипендию.

```
SELECT *
FROM студент
WHERE размер_стипендии =
                              (SELECT max(размер_стипендии)
                              FROM студент);
```

Многострочные подзапросы – возвращают как результат более одной строки, используют многострочные операторы сравнения:
IN – равно любому значению списка;
ANY – сравнение значения с любым значением, возвращаемым подзапросом;
ALL – сравнение значения с каждым значением, возвращаемым подзапросом.

Пример использования многострочного подзапроса следующий: вывести информацию о всех студентов, у кого стипендия выше, чем вся стипендия у студентов группы 8521:

```
SELECT *
FROM студент
WHERE размер_стипендии > ALL
                              (SELECT размер_стипендии
                              FROM студент
                              WHERE номер_группы = '8521');
```

Кроме того, подзапросы можно использовать во фразе FROM, когда результат подзапроса играет роль таблицы – источника данных для главного запроса, или во фразе SELECT.

Операции реляционной алгебры в SQL

Далее рассмотрим, как в SQL реализуются операции реляционной алгебры.

Несколько примеров самой простой операции **Выбор** (селекция) ранее уже были приведены, когда при выборке значений используется условие WHERE для ограничения выбираемых значений.

Операция **Проекция** также была показана на примере (получение списка групп, в которых учатся студенты) и реализуется, когда во фразе SELECT выбирается необходимый столбец, по которому устанавливается уникальность записей с помощью слова DISTINCT.

Упоминалось и про операцию **Декартово произведение**, когда используется соединение нескольких таблиц без указания условия соединения. Пример реализации декартова произведения может быть следующим: студенты участвуют в чемпионате по шахматам, тогда результирующая таблица встреч игроков друг с другом представляет собой декартово произведение таблицы СТУДЕНТ с самой собой:

```
SELECT c1.ФИО игрок1, c1.разряд разряд_игрока1, c2.ФИО игрок2, c2.разряд разряд_игрока2
FROM студент c1, студент c2;
```

Операция **Соединение** представляет собой частный случай декартово произведения, для которого указывается условие соединения таблиц друг с другом. Достаточно примеров (расписание для группы и в аудитории) было приведено для этой операции и с использованием традиционного синтаксиса, когда условие соединения таблиц указывается во фразе WHERE, и по стандарту ANSI с помощью фразы JOIN…ON.

Операция деление одной (делимое) таблицы на вторую (делитель) возможна, когда все записи второй таблицы содержатся в первой, а результатом этой операции являются таблица с записями, декартово отношение которых с делителем содержится в делимом. Например, вывести только то расписание, занятия по которому проходят и в 19-ом корпусе в 507 аудитории, и в КЦ в 313 аудитории. Тогда делимое – это таблица РАСПИСАНИЕ со всеми записями в ней (занятия во всех аудиториях всех корпусов), делитель представляет собой таблицу со следующей структурой и содержанием (запрашиваемое условие):

ДЕЛИТЕЛЬ

корпус	**аудитория**
19	507
КЦ	313

В результате деления получаем необходимое расписание групп по запрашиваемому условию, SQL-запрос содержит вложенные подзапросы и выглядит следующим образом:

```
SELECT *
FROM расписание p
WHERE NOT EXISTS
                (SELECT *
                FROM делитель д
                WHERE NOT EXISTS
                                (SELECT *
                                FROM расписание p2
                                WHERE p2.номер_группы = p.номер_группы
                                AND p2. тип_недели = p.тип_недели
                                AND p2.день_недели = p.день_недели
                                AND p2.номер_занятия = p.номер_занятия
                                AND p2.корпус = д.корпус
                                AND p2.аудитория = д.аудитория));
```

Для операций объединение, пересечения и разность в SQL существуют собственные операторы UNION, INTERSECT, MINUS соответственно. Так как эти операции выполняются над 2-мя таблицами, они (последние) должны иметь одинаковый состав атрибутов, определенных на одинаковых доменах. Рассмотрим использование этих операторов SQL на примерах.

Например, для операции объединения имеем следующие таблицы о преподавателях и сотрудниках вуза (рис. 3.44):

ПРЕПОДАВАТЕЛЬ

ид преподавателя	ФИО	дата рождения	пол	адрес

СОТРУДНИК

ид сотрудника	ФИО	дата рождения	пол	адрес

Рис. 3.44. Отношения для объединения

В результате объединения этих таблиц получим таблицу, содержащую сведения о преподавателях и сотрудниках:

```
SELECT *
FROM преподаватель
UNION
SELECT *
FROM сотрудник;
```

Например, для операции пересечения имеем следующие таблицы о студентах и о жильцах, проживающих в общежитии:

СТУДЕНТ

ФИО студента	**Серия и номер паспорта**	Пол	Дата рождения

ЖИЛЕЦ

ФИО жильца	**Серия и номер паспорта**	Пол	Дата рождения

Рис. 3.45. Отношения для пересечения

В результате пересечения этих таблиц, получим таблицу, содержащую сведения о студентах, проживающих в общежитии:

```
SELECT *
FROM студент
INTERSECT
SELECT *
FROM жилец;
```

Для операции разность в зависимости от порядка использования таблиц меняется результат выборки. Для примеров используем те же самые таблицы, что и для операции пересечение.

В результате вычитания таблицы студент из таблицы жилец, получаем жильцов, не являющихся студентами:

```
SELECT *
FROM жилец
MINUS
SELECT *
FROM студент;
```

В результате вычитания таблицы жилец из таблицы студент, получаем студентов, не проживающих в общежитии:

```
SELECT *
FROM студент
MINUS
SELECT *
FROM жилец;
```

Далее рассмотрим пример создания структуры таблиц с помощью DDL-операции (CREATE) SQL. Например, имеем следующую структуру, представленную на рис. 3.46.

Ограничения в структуре таблицы, которым должна удовлетворять каждая запись таблицы, могут быть представлены как на уровне определенного столбца, так и на уровне всей таблицы: уникальность, ограничения целостности или условия для значений столбцов).

Для представленной выше структуры обозначим некоторые ограничения. В таблице студент ограничения будут следующими:

- номер зачетки уникален у каждого студента;
- рассматриваем студентов, зачисливших не моложе 14 лет и позднее 1950 г.;
- значения для пола студентов и типа финансирования вставляются из соответствующих таблиц-классификаторов;
- проживание в общежитии хранится как факт (да или нет), так как булевского типа в СУБД Oracle нет, поэтому этот факт хранится в виде 1 (да) или (нет).

Рис. 3.46. Отношения для создания таблиц БД

Запрос на создание такой структуры таблиц выглядит следующим образом:

```
CREATE TABLE СТУДЕНТ (
 номер_зачетки                VARCHAR2(10) NOT NULL PRIMARY
                              KEY,
 ФИО                          VARCHAR2(100) NOT NULL,
 дата_рождения                DATE NOT NULL,
 пол                          NUMBER(1) NOT NULL REFERENCES
                              пол (ид),
 номер_группы                 NUMBER(4) NOT NULL REFERENCES
                              группа (ид),
 дата_зачисления              DATE NOT NULL CHECK (да-
                              та_зачисления>TO_DATE('01.01.1950','DD
                              .MM.YYYY')),
 тип_финансирования           NUMBER(1) NOT NULL REFERENCES
                              тип_финансирования (ид),
 проживание_в_общежитии       NUMBER(1) NOT NULL CHECK (прожи-
                              вание_в_общежитии IN (0,1)),
CHECK (дата_зачисления>(дата_рождения+14*365)));

CREATE TABLE ИТОГОВАЯ_АТТЕСТАЦИЯ (
 номер_зачетки          VARCHAR2(10) NOT NULL REFERENCES сту-
                        дент (номер_зачетки),
 дисциплина             NUMBER(4) NOT NULL REFERENCES дисцип-
                        лина (ид),
 средняя_оценка         NUMBER(1),
 PRIMARY KEY (номер_зачетки, дисциплина));

CREATE TABLE АТТЕСТАЦИЯ_ПО_СЕМЕСТРАМ (
 номер_зачетки          VARCHAR2(10) NOT NULL,
 дисциплина             NUMBER(4) NOT NULL,
 семестр                NUMBER(2) NOT NULL CHECK (семестр < 13),
 оценка                 NUMBER(1) NOT NULL CHECK (оценка
                        BETWEEN 2 AND 5),
 PRIMARY KEY (номер_зачетки, дисциплина, семестр),
 FOREIGN KEY (номер_зачетки, дисциплина)
     REFERENCES итоговая_аттестация (номер_зачетки, дисциплина));
```

Для того чтобы задать условие для столбцов нескольких таблиц, необходимо это условие прописывать в отдельной процедуре, которая будет находиться за пределами формирования описанной выше структуры этих таблиц и вызываться при обработке данных таблиц. Для приведенного выше примера такое условие в процедуре обработки будет следующим в зависимости:

- если средняя_оценка по дисциплине проставляется вручную, условие будет:

средняя_оценка > MIN(аттестация_по_семестрам.оценка) AND < MAX(аттестация_по_семестрам.оценка));

- если средняя_оценка по дисциплине автоматически вычисляется как среднее арифметическое, условие будет:

средняя_оценка = ROUND(AVG(аттестация_по_семестрам.оценка)),1);

4. ПРОЦЕСС ПРОЕКТИРОВАНИЯ БД

Эффективность современных информационных систем (ИС) в сфере управления во многом определяется результатом проектирования базы данных, являющейся информационной моделью предметной области (ПрО). Традиционно процесс проектирования баз данных включает три **этапа**:

1. **Концептуальное проектирование**, в результате которого собранные требования к данным ПрО анализируются и оформляются в виде концептуальной информационной модели предметной области (КИМПО).
2. **Выбор СУБД**, в среде которой физически будет реализована база данных.
3. **Проектирование физической структуры БД**, при котором концептуальная схема БД преобразуется во внутреннюю схему БД, решаются вопросы физического размещения и организации эффективного доступа к БД.

Последние два этапа проектирования, как правило, не представляют сложностей для их выполнения. Так, выбор СУБД зависит от известных факторов, выполнение которых позволит удовлетворить основным требованиям создаваемого проекта ИС. После того, как был принято решение об использовании выбранной СУБД, этап физического проектирования БД благодаря существованию современных CASE-средств полностью автоматизирован и не требует непосредственного участия человека. В отличие от последних этапов концептуальное проектирование традиционно представляет собой сложный процесс, плохо поддающийся формализации, отсутствуют готовые рецепты в «приготовлении» КИМПО для сложной ПрО, к числу которых относятся и вузы. Поэтому этот процесс рассматривается как своего рода искусство [11].

Любая методика проектирования схемы БД определяется:

- процедурой – последовательностью действий формирования модели ПрО;
- моделью данных (понятиями и нотацией) – язык, на котором будет описана полученная модель ПрО.

Далее рассмотрим каждый этап процесса проектирования БД в соответствии с этим определением.

4.1. Построение концептуальной информационной модели ПрО

4.1.1. Основные подходы для концептуального моделирования

На сегодняшний день существует ряд методик построения концептуальной схемы данных ПрО. В зависимости от основного используемого метода (анализа или синтеза), к которому они сводятся, можно выделить следующие подходы к концептуальному проектированию: декомпозиционный и интеграционный. Оба эти подхода позволяют построить концептуальную схему данных в виде объектов ПрО и связей между ними. Однако использование только одного из подходов не дает гарантии выявления полного перечня типов объектов ПрО. Далее опишем каждый подход и его применение для концептуального проектирования.

4.1.1.1. Декомпозиционный подход

Декомпозиционный подход базируется на системном анализе ПрО и предполагает последовательное, многоуровневое разбиение моделируемой системы на подсистемы до тех пор, пока не станет очевидным информационное поле составных частей. Это позволяет учесть не только существующие, но и будущие потребности. Главным недостатком такого подхода является сложность его реализации (необходимо активное участие руководителей различных уровней). Успех и значение декомпозиционного метода состоит не только и не столько в том, что сложное целое расчленяется на все менее сложные и в конечном счете простые части, а в том, что, будучи соединены надлежащим образом, эти части снова образуют единое целое. В результате последовательная декомпозиция системы на подсистемы приводит к формированию иерархической древовидной структуры [12].

Для проведения декомпозиции необходимо сформировать совокупность оснований декомпозиции – стандартных моделей, учитывающих специфику ПрО, – и определить порядок их использования. Высокий уровень абстрактности этих моделей позволяет использовать их для любых типов систем, причем для описания различных аспектов систем. Однако, чтобы применить формальную модель к рассматриваемой системе, необходимо придать ей конкретное содержание, т. е. решить, какие аспекты реальной системы включать как элементы модели избранного типа, а какие нет, считая их несущественными.

Для выявления основных типов объектов ПрО предлагается использовать методику декомпозиции целей, описанную в [13] для применения в системах организационного управления (система управления,

объектом которой являются коллективы людей). В соответствии с методикой после формулирования глобальной цели системы определяются основания декомпозиции глобальной цели на подцели, которые, как правило, являются неэлементарными, и последующие этапы связаны с их дальнейшей декомпозицией с использованием выявленных оснований в качестве соответствующих классификаторов. Для каждой итерации процедуры декомпозиции необходимо определять основные разделы требований и уточнение полученных в результате предыдущих этапов требований с учетом интерпретации оснований декомпозиции. Основная проблема использования выбранной методики состоит в нетривиальности интерпретации типовых оснований декомпозиции к рассматриваемой ПрО.

Системный анализ позволяет проводить анализ проблем управления большими и сложными системами. Система представляет собой конечное множество связей и отношений, выделенных из среды по признаку их причастности к конечному продукту деятельности системы. Система с очень большим числом подсистем и элементов разнообразной природы называется большой, а система с непредсказуемым поведением – сложной. К числу больших и сложных систем относятся большинство оргсистем. Системный подход позволяет упорядочить исходную информацию о сложной системе, понизить уровень ее сложности, неопределенности, осуществить решение задач проектирования и управления сложными по отношению к интеллектуальным возможностям человека объектами.

Важнейшим понятием системного подхода является цель – долгосрочный желаемый результат, недостижимый за рассматриваемый промежуток времени, но доступный в будущем. Сформулировать цель значит определить тот конечный результат, ради которого создается и действует система. Для формулирования глобальной цели системы, представления ее как системы подцелей разного уровня, имеющих более конкретный характер, необходимо осуществить системную декомпозицию глобальной цели, результатом которой является построение дерева целей и функций.

Методика построения дерева целей и функций [14] предназначена для анализа оргсистем и основана на следующих положениях:

- использование принципа декомпозиции;
- использование достаточно общих оснований декомпозиции;
- использование некоторых дополнительных принципов и ограничений.

Иерархическая многоуровневая структура целей и функций является прямым результатом использования принципа декомпозиции в про-

цессе структуризации целей системы. Разбиение цели *i*-го уровня на подцели (*i*+1)-го уровня осуществляется путем использования стандартных оснований декомпозиции – стандартных моделей системного анализа, описывающих инвариантные характеристики сложных систем, и определения порядка их использования. Высокий уровень абстрактности этих моделей позволяет использовать их для любых типов систем, причем для описания различных аспектов систем. Однако, чтобы применить формальную модель к рассматриваемой системе, необходимо придать ей конкретное содержание, т. е. решить, какие аспекты реальной системы включать как элементы модели избранного типа, а какие нет, считая их несущественными.

К числу дополнительных принципов и ограничений для построения ДЦФ относятся:

- принцип полноты – достижение совокупности возникающих при декомпозиции целей должно быть достаточным условием для достижения глобальной цели;
- принцип суперпозиции целей – цели одного уровня дерева должны быть относительно независимыми, для того чтобы цель любого уровня являлась аддитивной суммой входящих в нее подцелей;
- принцип конечности декомпозиции – алгоритм декомпозиции должен завершать свою работу за конечное число шагов, а результатом его работы должно быть конечное дерево.

Принцип полноты учитывается, если в ДЦФ все цели являются существенными, т. е. без достижения существенной цели невозможно достижение цели более высокого уровня ДЦФ, тем самым из ДЦФ исключаются несущественные цели. Принцип конечности выполняется, когда при достижении элементарной цели процесс декомпозиции прекращается, т. е. на основе имеющейся информации и опыта элементарная цель может считаться заведомо достижимой.

Далее необходимо выяснить, какие стандартные модели как основания декомпозиции предлагает использовать методика построения ДЦФ. Для декомпозиции глобальной цели оргсистемы при соблюдении описанных выше основных положений и принципов используются базовые модели системного анализа, к числу которых относятся также модель черного ящика, модель состава системы и модель структуры (рис. 4.1). Эти виды моделей широко используются для формирования моделей организаций. Например, модель черного ящика используется для описания взаимодействия организации с окружающей средой. Модель состава используются для отображения состава функций организации, целей, задач, персонала и т. д. Модель структуры используется для отображения структуры подчиненности в организации, коммуникаци-

онных взаимодействий и т. д. Указанные виды моделей систем используются чаще всего в статическом варианте, но они также могут использоваться и в динамическом варианте [12].

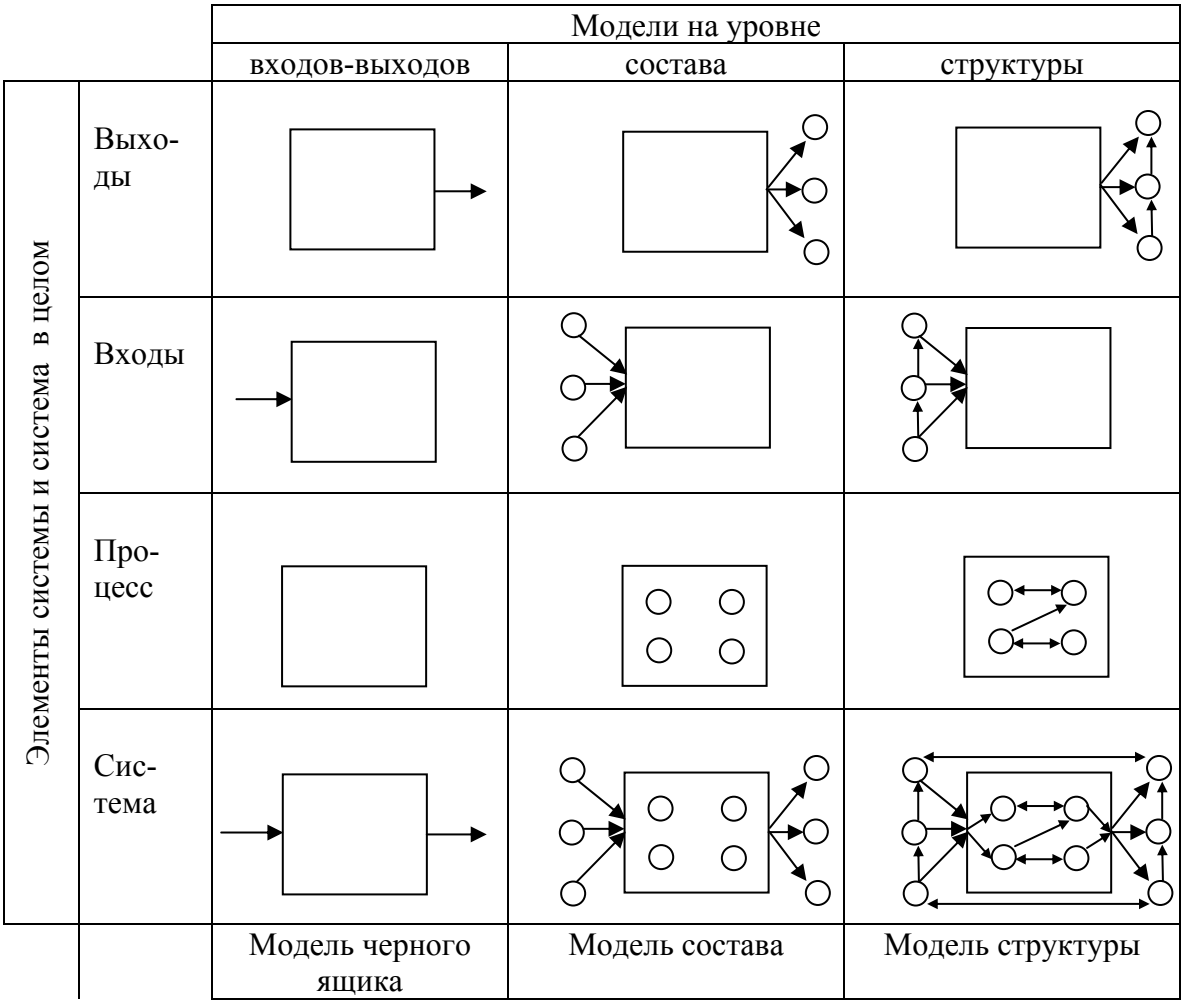


Рис. 4.1. Базовые модели системного анализа

Поскольку оргсистемы, как правило, являются большими и сложными, их выходы – ***конечные продукты*** деятельности – отличаются значительным разнообразием и составляют следующий классификатор:

- материальная продукция;
- энергия;
- информация;
- кадры;
- финансы.

Информационные и ресурсные входы любой оргсистемы определяются ее ***взаимодействиями с системами целеполагания*** (рис. 4.2), к числу которых относятся:

- вышестоящие системы – формирующие глобальную цель и основные требования к конечному продукту деятельности системы;
- нижестоящие (подведомственные) системы, возможности которых определяют объем и качество конечного продукта, при этом их объективные потребности должны своевременно удовлетворяться исследуемой системой;
- актуальная среда – объединяющая в себе системы, которые потребляют конечные продукты деятельности исследуемой системы, и следовательно, предъявляют основные требования как к количеству, так и к качеству конечных продуктов.

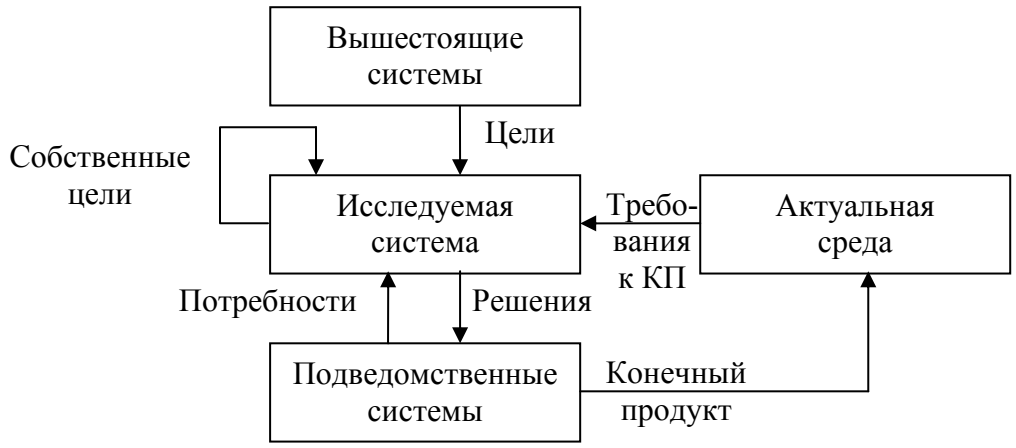


Рис. 4.2. Модель взаимодействия системы с системами окружающей среды

Цели, требования и ограничения перечисленных типов систем, а также собственные цели исследуемой системы и образуют основные классы входных воздействий на исследуемую систему.

В совокупности целеполагающие системы как входы и классификатор конечных продуктов как выходы составляют модель черного ящика исследуемой системы.

Далее рассмотрим «внутреннее содержание» (***структуру***) исследуемой системы. Основными ***элементами*** любой ***оргсистемы***, как известно, являются субъект деятельности, объект деятельности и средства деятельности (рис. 4.3). Субъект деятельности (кадры – К) – человек, который осуществляется производство конечных продуктов. Объект деятельности (предметы труда – ПД) – собственно конечные продукты, их составляющие либо промежуточные объекты, являющиеся объектами преобразования. Средства деятельности (СД) – средства, с помощью которых обеспечивается выполнение этапов жизненного цикла конечных продуктов.

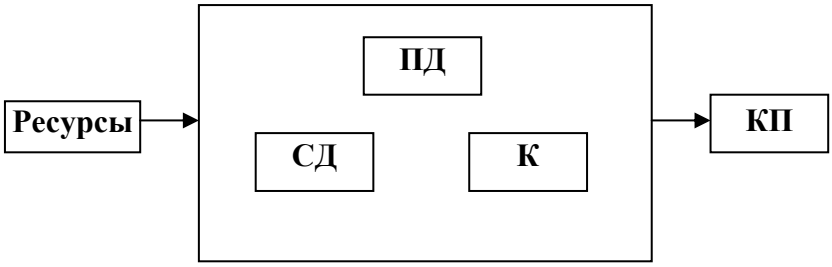


Рис. 4.3. Модель состава оргсистемы

Кроме того, необходимо выявить взаимодействие элементов структуры оргсистемы по достижению намеченных целей, сделать описание того, как достичь конечных результатов. Так как действия по производству конечных продуктов осуществляется в процессе деятельности, то модель структуры представляет собой модель процесса и опирается на ***модель жизненного цикла*** деятельности по производству конечных продуктов, представленную на рис. 4.4. Для получения любого КП системы, необходимо осуществить его жизненный цикл, состоящий из следующих этапов: выявление потребности в КП, обеспечение производства КП и его потребления.

Рис. 4.4. Модель жизненного цикла конечного продукта

Перечисленных стандартных моделей и соответствующих им оснований декомпозиции достаточно для построения ДЦФ оргсистемы. Детальность такого дерева определяется полнотой соответствующих классификаторов. Представленная в дереве совокупность целей дает при этом полное описание того, что должно быть достигнуто в результате деятельности системы. Эта структура инвариантна оргсистеме и дает наглядное представление об алгоритме построения ДЦФ для конкретной системы.

Ниже рассмотрим применение описанной методики построения ДЦФ для исследуемой системы вуз.

1. Построение ДЦФ начинается с формулирования глобальной цели исследуемой системы. Глобальная цель описывает конечный продукт деятельности системы в наиболее общей, качественной форме, удобной для последующей декомпозиции. Для любого вуза глобальной целью, по-видимому, является обеспечение подготовки

специалистов с высшим образованием, научных кадров в различных областях знаний, проведение фундаментальных и прикладных научных исследований, а также распространения знаний среди населения (повышения квалификации, образовательные курсы).

2. Далее производим построения ДЦФ, производя интерпретацию описанных в методике стандартных моделей и соответствующих им оснований декомпозиции.

2.1. Принимая во внимание представленный в методике классификатор конечных продуктов, выделяем конечные продукты деятельности вуза как системы:

- специалисты различной квалификации;
- ученые;
- фундаментальные и прикладные знания;
- изобретения (результаты интеллектуальной деятельности).

Перечисленный состав конечных продуктов выделен по признакам существенности для вуза в настоящее время. Естественно, что каждый из перечисленных классов может быть подвергнут дальнейшей декомпозиции. Так специалисты могут разделены по уровням квалификации (бакалавры, магистры), по профилям обучения (гуманитарные, технические), по специальностям и т. д. В такой детализации, однако, нет необходимости при построении ДЦФ для выявления основных типов объектов ПрО ввиду отсутствия существенных различий в технологии подготовки специалистов.

2.2. К системам целеполагания, которые взаимодействуют с вузом как с системой, относятся:

- органы управления ВПО (Федеральное агентство по образования, Министерство образования и науки) – в качестве вышестоящей системы, определяющей основные требования к конечным продуктам вуза;
- предприятия (как «заказчики» конечных продуктов) и собственно население (потребности которого удовлетворяются вузом) – в качестве подведомственных систем;
- население (как потребители конечных продуктов) – в качестве актуальной среды.

Если на этих первых этапах предлагаемые методикой основания соответствовали требованиям рассматриваемой ПрО, то на последних этапах необходимо применить усилия для интерпретации типовых оснований декомпозиции в условиях исследуемой системы.

3. Определяем основные разделы требований выявленных систем, оказывающих влияние на вуз, к его конечным продуктам.

3.1. Уточнение требований целесообразно осуществить в соответствии с этапами жизненного цикла производства КП: выявление потребности в КП – производство КП – потребление (использование) КП.

3.2. Дополнительное уточнение требований, сформулированных на предыдущем этапе предлагается провести с учетом элементов, участвующих в производстве конечного продукта и отношением между ними: средства труда – предметы труда – кадры.

Далее для рассматриваемой ПрО в соответствии с методикой приводим интерпретацию требований, выявленных для вышестоящей системы, к конечному продукту – специалистам, выпускаемых вузом.

- Выявление потребности – обеспечение возможности получения высшего образования.
  - Предмет труда:
    - абитуриент, желающий получить высшее образование;
    - документ, подтверждающий полученные им необходимые знания (аттестат, диплом).
  - Средства труда:
    - абитуриент (с помощью своих знаний);
    - денежные средства (для платного обучения).
  - Кадры:
    - приемная комиссия.
- Производство КП – подготовка дипломированного специалиста (обучение населения для получения высшего образования).
  - Предмет труда:
    - студент, получающий высшее образование.
  - Средства труда:
    - преподаватель (с помощью своих знаний);
    - оборудование;
    - учебная литература;
  - Кадры:
    - преподаватель.
- Потребление КП – выпуск специалиста.
  - Предмет труда:
    - Выпускник-специалист, получивший высшее образование;
    - диплом.
  - Средства труда:
    - студент (с помощью своих знаний).
  - Кадры:
    - государственная аттестационная комиссия.

В результате использования методики построения ДЦФ, руководствуясь принципами элементарности и существенности целей и функций, уже при декомпозиции на 3-м уровне выявляются следующие типы объектов ПрО:

- абитуриент;
- студент;
- выпускник;
- преподаватель;
- документ;
- оборудование;
- литература.

Каждый объект имеет собственные характеристики-свойства, которые в дальнейшем будут определены.

Несмотря на принцип полноты, который утверждается декомпозиционным подходом, этот процесс не может гарантировать абсолютную полноту анализа, т. к. необходимо придерживаться принципа существенности: в модель включаются только компоненты, существенные по отношению к целям анализа. К тому же с течением времени необходимо развитие информационной модели, что является нетривиальным для декомпозиционного подхода. Далее для дополнения типов объектов вуза рассмотрим применение интеграционного подхода.

4.1.1.2. Интеграционный подход

Интеграционный подход основан на анализе существующих потребностей пользователей в информации. Считается, что потребности эти отражены в текущих документах и дополнительно могут быть выявлены в результате специального опроса пользователей. Основным недостатком такого подхода является необходимость постоянного развития и модернизации схемы, связанной с естественным расширением информационных потребностей пользователей с течением времени, т. е. отсутствие необходимости пользователей в той или иной информации не отражается в концептуальной схеме.

Существует методика Case*Method, изложенная в [11], основанная на интеграционном подходе, она представлена совокупностью правил и указаний по представлению информационных запросов, циркулирующих в организации. Методика содержит набор типовых вопросов, которые предлагается задавать экспертам ПрО в ходе интервьюирования:

- какие в организации имеются объекты, заслуживающие внимания (сущности);
- какими эти объекты обладают качествами и свойствами (атрибутами);

- какие между этими сущностями и атрибутами просматриваются взаимосвязи.

Однако на практике такой подход требует взаимодействия с пользователем и грамотного формирования таких информационных запросов для полного выявления сущностей, атрибутов и связей между сущностями, а также бизнес-правил ПрО и контроля качества создаваемых схем.

Более распространенной процедурой проектирования является методика моделирования IDEF1X, в которой используются различные уровни моделирования для представления данных ПрО. Анализируя потребности пользователей в информации, которые отражены в существующих документах и дополнительно могут быть выявлены в результате специального опроса пользователей, на начальном этапе определяются сущности и связи между ними, отражающие основные бизнес-правила ПрО – формируется модель сущность-связь. На следующем уровне представляется более подробное представление данных: выполняется уточнение выделенных объектов и выделений декларативных ограничений целостности – модель данных, основанной на ключах. На завершающем этапе выполняется детальное представление структуры данных: полное описание всех сущностей, их атрибутов и связей между ними – полная атрибутивная модель. Таким образом, процесс концептуального проектирования в IDEF1X фактически представляет собой продвижение по уровням моделирования от диаграмм уровня сущности до полных атрибутивных диаграмм.

Далее рассмотрим использование интеграционного подхода для процесса обучения студентов в вузе. С помощью декомпозиционного подхода выявили следующие объекты: студент, получающий высшее образование, преподаватель, который дает ему знания с помощью своих знаний и/или оборудования и учебной литературы. На начальном этапе интеграционного подхода можно сказать, что студент связан с преподавателем через объект Занятие, которое последний проводит для первого; однако занятие проводится не для конкретного студента, а для Группы, в которой он учится (рис. 4.5).

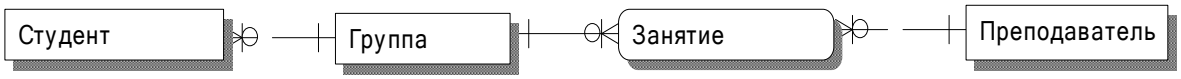


Рис. 4.5. Фрагмент схемы с выявленными сущностями ПрО

В результате дальнейшего выполнения уточнения структуры данных на последующих этапах получим представление, приведенное на рис. 4.6.

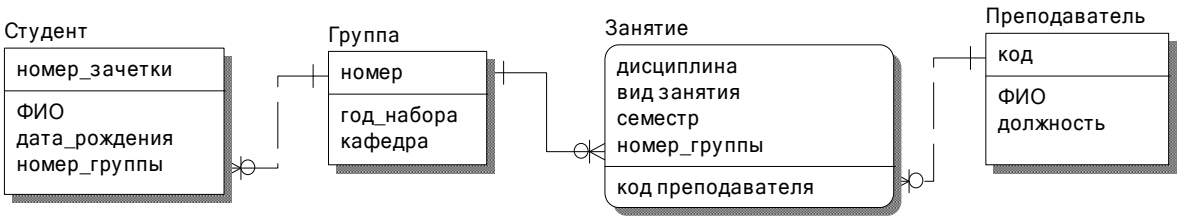


Рис. 4.6. Фрагмент схемы с выявленными сущностями ПрО, атрибутами и связями

Таким образом, применение комбинированного подхода (совместно декомпозиционного и интеграционного подходов) к концептуальному проектированию позволяет построить наиболее полную концептуальную схему данных для сложной ПрО, к которым относится и вуз.

Развитие интеграционного подхода

Разновидностью описанного классического интеграционного подхода является интеграционная методика, предложенная Чудиновым И.Л., Исаевым И.В. [15]. В основу этой методики положен интеграционный подход, но предлагается развитие теории моделирования за счет объединения основных идей модели данных «сущность–связь» и фундаментальных идей реляционной теории, а также переосмысления традиционно принятого порядка концептуального проектирования. Результатом использования предлагаемого подхода является построенная концептуальная информационная модель ПрО (КИМПО) – это модель данных, предназначенная для моделирования информационных представлений о ПрО с целью дальнейшей их реализации в реляционной БД. По существу, КИМПО представляет собой гибридную модель данных, основанную на идеях реляционной модели и модели «сущность–связь». Наиболее близкой к КИМПО является модель данных IDEF1X.

Интеграционную методику проектирования КИМПО можно представить в виде следующих этапов:

- анализ информационных потребностей пользователей и представление их в виде множества исходных отношений;
- уточнение множества исходных отношений, представление их в виде нормализованных, простых структур данных;
- связывание простых структур, являющихся представлениями информационных потребностей пользователей с базовой (текущей) КИМПО.

Основные принципы предлагаемого подхода:

1. Последовательный характер проектирования, заключающийся в постепенном формировании схемы данных за счет включения опи-

саний новых информационных потребностей и модификации существующих описаний.

2. Фрагментированное взаимодействие участников процесса проектирования (источников информационных потребностей и аналитиков) в рамках двухуровневой организации процесса проектирования:
   - первичная формализация информационных потребностей для отдельных задач;
   - формализованная интеграция описаний новых потребностей с базовой схемой данных.
3. Описание информационных потребностей в виде структур, близких к естественным формам представления информации (документ, запрос, файл).
4. Декларативное описание доменов и отношений между ними как часть процесса фиксации базовых знаний о ПрО.
5. Использование информации о доменах для определения наличия и типа отношений между элементами схемы данных.

Ниже рассмотрим каждый этап методики проектирования КИМПО.

1. **Анализ информационных потребностей пользователей**

На этом этапе осуществляется основной семантический анализ ПрО (через информационные потребности пользователей), результатом которого является первоначальная формализация ее информационного описания в виде множества исходных отношений.

Анализ информационных потребностей пользователей заключается в следующем:

- идентификация информационных потребностей пользователей;
- представление выявленных форм отображения информационных потребностей пользователей в виде исходных отношений.

1.1. **Идентификация информационных потребностей пользователей** предполагает:

- выявление конкретных форм отображения информационных потребностей пользователей;
- определение характеристик выявленных форм отображения информационных потребностей пользователей, необходимых для последующего анализа.

При выявлении конкретных форм отображения информационных потребностей необходимо учитывать, что даже при широком внедрении компьютеров во все сферы деятельности человека, документы и ныне являются основной формой представления информации о состоянии объектов разной природы.

Однако документы отражают сложившиеся информационные требования, в то время как новые и потенциальные потребности можно, видимо, выявить только в процессе интервьюирования конечных пользователей с формализацией таких интервью в виде предполагаемых информационных запросов.

И, наконец, сегодня трудно найти предметную область, где бы не эксплуатировались компьютерные системы обработки информации. Обрабатываемые файлы также можно считать формой отображения информационных потребностей пользователей, причем в большинстве случаев это интеграция потребностей многих пользователей.

Таким образом, типовыми формами отображения информационных потребностей пользователей являются:

- документы;
- информационные запросы пользователей;
- файлы существующей системы обработки информации.

1.1.1. **Выявление информационных документов** осуществляется в процессе специального обследования подразделений организации, деятельность которой и является объектом информационного моделирования.

Перечень документов можно выявить в результате:

- опроса работников подразделения;
- анализа архива документов;
- анализа должностных инструкций работников;
- во многих организациях утверждены и поддерживаются в актуальном состоянии альбомы документов, которые применяются в организации.

Весьма важным для последующего анализа и формализации информационных потребностей пользователей является степень структуризации документов.

По уровню структурированности информации документы можно классифицировать на:

- неструктурированные;
- слабоструктурированные;
- хорошо структурированные.

К неструктурированным относятся документы с преимущественно текстовой формой представления информации: письма, отчеты, пояснительные записки, справки, статьи, методические материалы и другие текстовые документы.

К слабоструктурированным документам относятся документы, смысловое содержание которых представлено в текстовом виде, но оформлено

по определенным стандартам. Это такие документы, как: приказы, контракты, акты, протоколы, представления и другие документы.

Значение **Виртуальный** устанавливается для документов, пока еще не существующих в исследуемой ПрО. Прежде всего, это информация, необходимая для расчета вторичных атрибутов, но не входящая ни в один из реальных документов. Виртуальные документы будут определены на более поздних этапах анализа ПрО.

Значение **Запрос** устанавливается для документов, в виде которых оформляется ответ на запрос.

1.1.2. **Выявление информационных запросов** осуществляется в процессе интервьюирования работников подразделений, в особенности тех, кто заполняет документы (источники) и тех, кто использует содержащуюся в них информацию (потребителей). При этом вначале необходимо предложить пользователю представить свои запросы в виде документа – ответа на запрос, а если запрос связан с обобщением некоторой информации (своды, max, min, среднее значение, группирование объектов и т. п.), то с приведением документов, содержащих исходную информацию. Если пользователь затрудняется оформить запрос в виде документа, проектировщик должен сделать это сам и представить результат пользователю для согласования. Необходимо определить тип запроса в следующей классификации:

- **регламентный** – запрос, по фиксированной форме выполняемый по определенному регламенту (регулярно или по запросу пользователя). Его еще можно назвать каталогизированным запросом;
- **нерегламентный** – запрос, для которого невозможно определить ни время (частоту) появления, ни конкретный состав атрибутов. Можно лишь определить множество атрибутов, из которых будут формироваться конкретные запросы. Например, запросы по кадровой информации работников подразделения.

1.1.3. **Выявление информационных файлов** лучше всего осуществлять вместе со службой эксплуатации систем обработки информации. При этом, прежде всего, необходимо выбрать те файлы, которые содержат исходную (первичную) информацию. Для них необходимо зафиксировать и способ (технологию) отображения значения данных в файле. Это может быть:

- документ, информация из которого переносится в файл;
- ввод значений данных в процессе анализа объекта (опрос, наблюдение и т. п.);
- импортирование из файла внешних предметных областей.

Далее необходимо выявить те файлы, информация которых отображается в выходные документы или экспортируется во внешние предметные области. И, наконец, необходимо выявить файлы с промежуточной информацией и провести опрос потенциальных пользователей с целью выявления запросов по этой информации.

1.2. **Представление выявленных форм отображения информационных потребностей пользователей в виде исходных отношений** предусматривает выполнение следующих действий:

- выявление атрибутов и порядка их следования в отношении;
- выявление множественности значения атрибута;
- определение домена атрибута;
- выявление вторичных атрибутов;
- оформление результатов идентификации атрибутов в виде строки таблицы описания исходных отношений.

1.2.1. **Рекомендации по выявлению атрибутов**

Для выявления атрибутов исходного отношения, полученного на основе документов, предлагаются следующие рекомендации:

- каждое заполняемое поле анкетного документа представляется отдельным атрибутом в том же порядке следования;
- каждый столбец (графа) табличного документа представляется отдельным атрибутом в том же порядке следования;
- анкетная часть заголовка табличного документа представляется таким же образом, как документ типа анкета, и располагается в начале отношения;
- в названии документа может быть употреблено понятие, которое можно, при определенных условиях, вынести в начало анкетной части документа в виде заполняемого поля, а следовательно, представить в виде атрибута;
- в названии таблицы многотабличного документа может быть употреблено понятие, которое можно при определенных условиях внести в таблицу в качестве нулевого столбца, а следовательно, представить в виде атрибута;
- если в подвале табличного документа есть анкетная часть, то она представляется в виде атрибутов таким же образом, как и документ типа анкета с расположением атрибутов в отношении после атрибутов, соответствующих анкетной части документа (перед атрибутами, соответствующими первой таблице).

Некоторым понятиям из названия документа (таблицы) может соответствовать атрибут исходного отношения если:

- это понятие является одним из возможных значений некоторого классификатора;
- в предметной области возможно заполнения такого же документа для альтернативного понятия.

Многотабличный документ можно представить в виде нескольких (по числу таблиц) отношений. При этом анкетная часть (заголовка, подвала и возможно из названия документа) вначале присоединяется к первой таблице. Затем определяются ключевые атрибуты, входящие в анкетную часть документа. Они присоединяется к каждой таблице многотабличного документа.

При установлении содержательного имени рекомендуется использовать в качестве базы:

- имя столбца в табличном документе;
- понятие, соответствующее имени заполняемого поля в документе типа анкета или в анкетной части табличного документа;
- имя типа объекта для описания которого используется атрибут;
- значимые слова из имен ключевых атрибутов.

При анализе содержательных имен атрибутов одной таблицы следует обратить внимание на следующие особенности и зафиксировать их при обнаружении:

- имеется связь между атрибутами типа «всего – в том числе…» или «всего – из них»;
- в содержательных именах разных атрибутов можно выделить понятия, которые могут образовать классификатор.

Для выявления атрибутов исходного отношения, получаемого на основе запросов, предлагаются следующие рекомендации:

- в результате опроса пользователей им предлагается представить каждый запрос в виде табличного или анкетного документа;
- проводить дальнейший анализ запроса как документа.

Для выявления атрибутов исходного отношения, формируемого на основе существующих файлов, предлагаются следующие рекомендации:

- в качестве атрибутов выбираются поля файлов;
- определение содержательного имени атрибута необходимо осуществить с администратором данных или постановщиком задачи по актуализации файла.

1.2.2. **Определение вторичных атрибутов**

Вторичность атрибутов выявляется на основе анализа правила формирования значения:

- *первичным* назовем атрибут, значение которого для объекта фиксируется, например Ф.И.О., должность, адрес и т. п.
- *вторичным* назовем атрибут, значение которого определяется на основе значений других атрибутов, например объем продаж, количество сотрудников и т. п.

Вторичный атрибут чаще всего имеет числовой тип и определяется на основе расчета, что может служить формальным признаком выбора атрибута для анализа на его вторичность.

Типовыми для организационных систем являются следующие процедуры расчета вторичных атрибутов:

- суммирование значений атрибута в группе, определяемой другим атрибутом (своды);
- подсчет числа записей (объектов), удовлетворяющих определенным условиям (в группе);
- максимальное, минимальное и среднее значение атрибута в группе;
- отклонение от максимального, минимального, среднего;
- доля, процент, пропорциональное разбиение, расчет по норме.

Для вторичного атрибута необходимо:

- выявить фактический документ или файл, с использованием значений атрибутов которого осуществляется определение его значения;
- сформировать виртуальный документ, с использованием атрибутов которого осуществляется определение его значений (если фактического документа или файла нет);

Как правило, множество вторичных атрибутов одного документа рассчитывается на основе атрибутов, которые также можно представить в виде одного документа, поэтому вторичные атрибуты одной таблицы следует анализировать в комплексе.

При определении концептуальной модели не принимается решение о целесообразности хранения атрибутов – это дело проектирования физической БД и ИС. Поэтому на данном этапе важно отразить потребность во вторичных атрибутах и выявить первичные атрибуты, необходимые для их получения. Если выполняется комплексный проект вплоть до создания ИС, то для вторичных атрибутов целесообразно зафиксировать и алгоритм расчета, включив его в ранее упомянутый раздел правила формирования значений атрибутов.

1.2.3. В результате **анализа доменов атрибутов** должен быть получен один из следующих результатов:

- диапазон возможных значений;
- правила формирования исходных значений;
- словарь возможных значений.

Если для атрибута невозможно определить ни диапазон возможных значений, ни словарь возможных значений, то проставляется значение правило N, организуется специальный раздел правила формирования атрибутов, и туда вносится описание правила под соответствующим номером N.

Определяя домен атрибутов в виде словаря возможных значений, необходимо учитывать:

- могут ли быть множественными значения атрибута у некоторых объектов в анализируемом документе;
- могут ли в ПрО встретиться объекты, для которых в домене нет необходимого значения;
- количество возможных значений.

Если множественные значения возможны, то необходим дополнительный анализ корректности определения атрибута: нет ли здесь совмещения двух атрибутов в одном (и тогда создать два атрибута), или имеет место множественность, которую необходимо исключить в дальнейшем путем нормализации по 1НФ.

Если выявлен объект, для которого в домене-словаре нет возможного значения, необходимо также сделать дополнительный анализ, в результате которого возможны следующие решения:

- изменение состава словарей;
- добавление в словарь нового значения;
- добавление в словарь значения «прочее».

Для пояснения способов сокращенного представления словарей совпадающих, соподчиненных и пересекающихся атрибутов таблица заполнена гипотетическим примером. Такой подход вполне оправдан еще и потому, что для установления связуемости отношений необходимо будет определять сопоставимость атрибутов – претендентов на атрибуты связи. Это тем более целесообразно, потому как атрибуты, домены которых могут быть представлены словарем, являются основными претендентами на атрибуты связи.

Одновременно с анализом возможных значений атрибутов проводится анализ их сопоставимости:

$=$, если атрибуты совпадают (домены совпадают $D1 = D2$);

$<$, если атрибут первого отношения соподчинен атрибуту второго ($D1 \subset D2$);

$>$, если атрибут второго отношения соподчинен атрибуту первого ($D1 \supset D2$);

$\times$, если атрибуты сопоставимы ($D1 \cap D2 \neq \varnothing$);

$+$, если атрибуты совместимы ($D1 \cup D2$ имеет смысл в качестве домена исследуемой предметной области).

Анализ атрибутов на сопоставимость следует проводить в следующем порядке:

1. Прежде всего на сопоставимость проверяются все ключевые атрибуты, т. к. именно они являются основными атрибутами связи.
2. Далее наиболее вероятными претендентами на атрибуты связи, как уже отмечалось ранее, являются атрибуты, домены которых являются словарями и лишь иногда атрибутами связи могут быть числовые данные.

Важнейшим элементом формирования отношения является определение ключевых атрибутов, т. е. атрибутов, значение которых однозначно идентифицируют любой кортеж отношения (строку таблицы документа).

Вместе с тем ключевые атрибуты определяют смысл остальных атрибутов отношения: без использования ключевых атрибутов, значения неключевых теряют смысл.

Можно предложить следующие рекомендации по выявлению ключевых атрибутов. Ключевыми атрибутами, как правило, являются:

- боковик таблицы (крайний левый столбец);
- атрибут, выявленный при анализе названия таблицы (документа);
- первые атрибуты анкеты и анкетной части смешанного документа;
- атрибут, идентифицирующий конкретный объект, описание которого содержится в документе;
- атрибут, определяющий время, за либо на которое зафиксирована информация, содержащаяся в документе;
- атрибут, значение которого является классификационным признаком, определяющим некоторую сторону описания объекта или сферу его деятельности.

Следует иметь в виду, что все атрибуты также составляют ключ отношения, поэтому в качестве первичного ключа выбирается минимальное число атрибутов, при котором сохраняются свойства ключа. В отношении может быть более одного ключа (например, номер зачетной книжки), тогда в качестве первичного ключа выбирается наиболее компактная запись или атрибуты, в большей степени отражающие специфику ПрО.

2. **Уточнение множества исходных отношений** состоит из:

- свертка атрибутов в новое отношение;
- нормализация отношений;
- выявление отношений типа кодификатор.

2.1. Свертка подмножества атрибутов некоторого отношения в дополнительное отношение осуществляется в том случае, если в именах

атрибутов этого подмножества просматриваются значения одного классификатора. В результате свертки получаем следующие два отношения, связанные по типу иерархии:

- новое отношение, состоящее из ключа исходного отношения плюс атрибут, образованный на основе классификатора, полученного из названий атрибутов и также входящий в состав ключа этого отношения плюс неключевой атрибут (возможно несколько атрибутов), соответствующий значению свернутых атрибутов из исходного отношения;
- из исходного отношения исключается множество «свернутых» атрибутов, и это усеченное отношение будет старшим в связи с новым отношением, соответствующим «свернутым» атрибутам.

2.2. При нормализации отношения, не удовлетворяющего одной из нормальных форм, получаем два отношения, связанные по типу 1:M.

2.2.1. При нормализации отношения, не удовлетворяющего 1НФ по множественности значений атрибута, получаем два отношения, связанные по типу 1:M, причем:

- старшим становится исходное отношение за исключением атрибута с множественными значениями;
- подчиненным становится отношение, состоящее из ключа исходного и атрибута с множественными значениями, причем подчиненное отношение состоит только из ключевых атрибутов;
- получается простая структура типа **Иерархия**.

2.2.2. При нормализации отношения, не удовлетворяющего 2НФ, получаем два отношения, связанные по типу 1:M, причем:

- старшим становится отношение, содержащее неключевые атрибуты, зависящие от части ключа, и ту часть ключа, от которой они зависят;
- подчиненным становится исходное отношение, за исключением атрибутов, зависящих от части ключа;
- получается простая структура типа **Иерархия**.

2.2.3. При нормализации отношения, не удовлетворяющего 3НФ, получаем два отношения, связанные по типу 1:M, причем:

- старшим становится отношение, состоящее из неключевых атрибутов, транзитивно зависящих от ключа исходного отношения, плюс неключевой атрибут исходного отношения, от которого (через который) зависят упомянутые выше атрибуты;
- ключом старшего отношения становится атрибут, неключевой в исходном отношении, но от которого зависят неключевые атрибу-

ты, вызвавшие неудовлетворение исходного отношения 3-й нормальной форме;

- подчиненным является исходное отношение, за исключением атрибутов, транзитивно зависящих от ключа; неключевой атрибут, от которого зависят другие неключевые атрибуты, вызвавшие неудовлетворение третьей нормальной форме, остается в «подчиненном» отношении;
- получается простая структура типа **Комментарий**.

2.3. Выявление отношений типа **Кодификатор.**

В общем-то вопрос о кодировании значений атрибутов, т.е. о представлении информации в памяти компьютера относится к проектированию физической структуры БД. Однако кодификатор является дополнительным отношением, которое необходимо отобразить в концептуальной модели, поэтому решение о целесообразности кодирования атрибута необходимо и возможно принять уже на этапе определения КИМПО.

Полагаем, что использование кодификатора целесообразно, если объем некодированного представления больше, чем объем кодированного, что может быть представлено следующим выражением:

$$n l_t > n l_k + m(l_t + l_k), \tag{1}$$

где n – число кортежей в отношении; m – число элементов в словаре возможных значений (в кодификаторе); l_t – размер не кодированного (текстового) значения; l_k – размер кода.

Если принять $l_t = 5l_k$ (это заниженная граница, на самом деле соотношение от 5 до 10,) то

$$5\ nlk > nlk + 6\ mlk,$$
$$n/m > 6/4,$$

т. е. значения атрибута, домен которого есть словарь, целесообразно кодировать, если число кортежей в исходном отношении в полтора раза больше числа возможных значений атрибута.

Если в КИМПО есть несколько (S) атрибутов, определенных на одном и том же домене, то условие принимает вид:

$$l_t \sum_{i=1}^{s} n_i > l_k \sum_{i=1}^{s} n_i + m(l_t + l_k), \tag{2}$$

где n_i – количество кортежей в отношении, в который входит i-й атрибут положим,

$$n' = \frac{\sum_{i=1}^{s} n_i}{S}, \tag{3}$$

где n' – среднее количество кортежей в отношениях, в которых находятся кодируемые атрибуты, тогда:

$$l_t Sn' > l_k Sn' + m(l_t + l_k).$$

Если $l_t = 5l_k$, то

$$5l_k Sn' > l_k Sn' + 5l_k m + l_k m;$$

$$2n'S > 3m,$$

т. е. даже при двукратном использовании такого атрибута, как Ф.И.О., целесообразно использовать кодификатор (при условии, что среднее количество кортежей (рассчитываемое по формуле (3)) в отношениях, в которых находятся кодируемые атрибуты, больше, чем $\frac{3}{4}$ *кол-во кортежей в кодификаторе[2]).

3. Связывание отношений и простых структур в КИМПО

Предлагается единый алгоритм связывания в КИМПО отдельных отношений и простейших структур. Для этого связи между отношениями простейших структур предварительно фиксируются, а составляющие их отношения анализируются далее как автономные отношения в соответствии с приведенным ниже алгоритмом.

При построении алгоритма учитываем, что:

- в КИМПО преобладают иерархические зависимости;
- связь типа **Комментарий** – это связь через единственный атрибут;
- связь типа **Соединение** либо приводит к физическому соединению отношений, либо к наследованию связей каждого с другими отношениями;
- связь типа **Группировка** преобразуется в две связи типа **Иерархия** с некоторым, возможно виртуальным, отношением, ключ которого совпадает с пересечением ключей исходных отношений или с неключевым атрибутом пересечения.

Алгоритм построен таким образом, что некоторое отношение (будем называть его анализируемым) последовательно сравнивается с множеством других, уже включенных в базовую КИМПО отношений (будем называть их исследуемыми).

3.1. Содержательный (неформализованный) алгоритм связывания отдельных отношений в КИМПО

А. Предварительный этап.

---

[2] Значение коэффициента ($\frac{3}{4}$) рассчитано для случая $l_t = 5l_k$.

Отношения разбиваются на классы с равным числом атрибутов в ключе отношения.

Б. Связывание отношений первого класса с отношениями других классов.

Для каждого отношения первого класса (анализируемое отношение) проводим анализ на связуемость с отношениями других классов (исследуемые отношения) последовательно, начиная со второго класса.

1. Если ключ очередного исследуемого отношения является расширением ключа анализируемого отношения первого класса, то фиксируем между ними в таблице 6 связь типа **Иерархия**, в котором старшим является отношение из первого класса.
2. Если среди неключевых атрибутов исследуемого отношения есть атрибут, сопоставимый атрибуту ключа исследуемого отношения, то фиксируем между ними в таблице 6 связь типа **Комментарий**, в котором псевдостаршим является анализируемое отношение.

В. Связывание отношений i-го класса ($i \geq 2$) с отношениями $i + k$ класса.

Для каждого отношения i-го класса (анализируемого отношения) проводим анализ на связуемость с отношениями $i + 1$ класса (исследуемые отношения), затем с отношением $i + 2$ класса и т. д. Затем аналогично проводим анализ отношений $i + 1$ класса со всеми последующими классами, $i + 2$ класса и т. д.

Если ключ очередного исследуемого отношения является расширением ключа анализируемого, то фиксируем между ними связь типа иерархия, в котором старшим является анализируемое отношение.

Если ключи исследуемого и анализируемого отношений пересекаются (связь типа **Группировка**), то проверяется, установлена ли связь анализируемого с отношением, имеющим ключ, совпадающий с выявленным пересечением ключей. Если да, то аналогичная связь устанавливается и с исследуемым отношением, если нет, то необходимо сформировать аналогично ситуации в б) виртуальное отношение, зафиксировать в таблице 6 иерархическую связь с ним исходных отношений как со старшим, включить его в состав исследуемых и провести анализ его связуемости с другими отношениями в соответствии с в).

а) В каждом из блоков а), б), в) после проверки ключевых атрибутов проверяется наличие по крайней мере сопоставимых (соподчиненных, совпадающих) неключевых атрибутов в анализируемом и исследуемом отношениях и, если таковые есть, проверяется, есть ли в первом классе отношения с атрибутом ключа, сопоставимым с выявленным.

Если такое отношение есть, то с ним фиксируется связь типа **Комментарий** с псевдостаршим по отношению к исходным отношениям.

Если такового отношения в первом классе нет, то фиксируется виртуальное отношение с ключом, состоящим из одного атрибута, домен которого равен объединению доменов анализируемых атрибутов исходных отношений, фиксируется его связь с последними как псевдостаршего в комментарии и проводится его анализ с другими отношениями в соответствии с а).

б) Приведенный алгоритм выявляет все возможные связи, в то время как для наглядного отображения концептуальной модели следующие связи являются несущественными, загромождающими изображение:

- иерархические связи между несмежными элементами ветви в иерархии;
- совпадающие связи с другими отношениями тех отношений, которые связаны между собой по типу соединение.

Поэтому помечаем как несущественные (транзитивные) связи между отношениями, удовлетворяющими следующим условиям.

Если между отношением R1 с ключом К1 установлена связь типа **Иерархия** с отношением R2 с ключом (К1, К2), и связь отношения R2 с отношением R3 с ключом (К1, К2, К3) также связь типа **Иерархия** и между отношениями R1 и R3 также установлена связь типа **Иерархия**, то последняя может быть помечена как несущественная (транзитивная).

Если между отношением R1 и R2 установлена связь типа **Соединение**, то связи одного из них (соподчиненного или с меньшей мощностью домена) можно пометить как несущественные.

Следует заметить, что связи, несущественные с точки зрения иллюстрации концептуальной модели, могут быть обязательными при отображении концептуальной модели в модель БД конкретной СУБД (т. к. на основе связей формируются ограничения ссылочной целостности).

4.1.2. Модели, используемые в концептуальном проектировании

В настоящее время все существующие модели данных, используемые в концептуальном проектировании, можно отнести к двум семействам [16]:

- ER-модели;
- OR-модели.

Широкое применение этих моделей данных подтверждается фактами их использования в наиболее известных CASE-системах для информационного моделирования: Oracle Designer, Computer Associates All Fusion ERwin Data Modeler, Sybase Power Designer, IDS Prof. Scheer ARIS, Microsoft Visio и др.

4.1.2.1. ER-модель

ER-модель (Entity-Relationship – сущность-связь) опирается на понятия сущность, атрибут, связь и ПрО должна быть представлена как совокупность сущностей с атрибутами, между которыми установлены связи.

Под ***сущностью*** понимаем имеющее особый смысл, существующее в действительности или воображаемое явление или объект, информация о котором подлежит запоминанию или выяснению. Примерами сущности для ПрО вуз являются Абитуриент, Студент, Аспирант, Преподаватель, Помещение, Документ. Имя сущности может представлять тип или класс объекта, но не конкретное значение. Например, сущностью является Студент, которая представляет собой всех студентов вуза, а один из них, Иванов Иван Иванович является не сущностью, а конкретной реализацией этой сущности.

Атрибутом назовем любое свойство, позволяющее квалифицировать, идентифицировать, классифицировать, измерять сущность или выражать ее состояние либо любое описание объекта или явления. Атрибут служит для описания одной сущности и только той, под которой он подписан. Атрибут может иметь текстовую, числовую, графическую форму, он может быть получен в результате функционирования органов чувств (осязания, обоняния и т. п.). Поскольку нас интересует обработка информации, сконцентрируем внимание на текстовых и числовых атрибутах. Например, атрибутами сущности Студент являются *ФИО, номер группы, номер зачетной книжки, дата зачисления* и др.

Иногда атрибут может стать сущностью, если он представляет самостоятельный объект или явление со своими собственными связями и атрибутами (например, атрибут *адрес* у сущности Личность, представленный в виде текстового поля, может быть трансформирован в сущность Адрес с атрибутами *страна, регион, населенный пункт, улица, номер дома, квартира*).

Связью является поименованное отношение, имеющее место между двумя сущностями. Такая связь является бинарной в том смысле, что она имеет место между ровно двумя поименованными сущностями или же имеет вид отношения сущности к самой себе. Каждая связь имеет два конца, каждый из которых обладает:

- именем;
- степенью / мощностью;
- признаком обязательности.

Эти свойства используются для характеристики связи по отношению к каждой из участвующей в ней сторон.

Каждая сущность должна иметь ***уникальный идентификатор***, т. е. должна быть уникально определена: каждый экземпляр (вхождение) сущности должен иметь ясное и недвусмысленное определение, позволяющее отличать его от других экземпляров (вхождений) той же сущности. Уникальным идентификатором может быть атрибут, комбинация атрибутов, комбинация связей или атрибутов и связей. Например, уникальным идентификатором для сущности Студент является атрибут *№ зачетной книжки*.

Каждый атрибут должен быть определен на конкретном ***домене***. Доменом называется совокупность правил проверки соответствия, форматных ограничений и других свойств (характеристик), присущих группе атрибутов.

Например:

- список значений;
- диапазон;
- уточненный перечень значений или диапазон;
- любая комбинация из вышеперечисленного.

Атрибуты из одного домена подчиняются общему набору ограничений. Домены не принято представлять на схемах.

На сегодняшний день существуют несколько *нотаций* для представления ER-моделей:

1. Нотация Чена: сущность изображается прямоугольником, атрибут – овалом, соединенным со своей сущностью (идентифицирующий атрибут подчеркнут), а связь – ромбом, соединенным со связываемыми сущностями. Вид линии в месте соединения с сущностью определяет кардинальность связи («воронья лапка» – M, «крест» – 1). Имена сущности, атрибута и связи располагаются внутри их изображений (рис. 4.7).

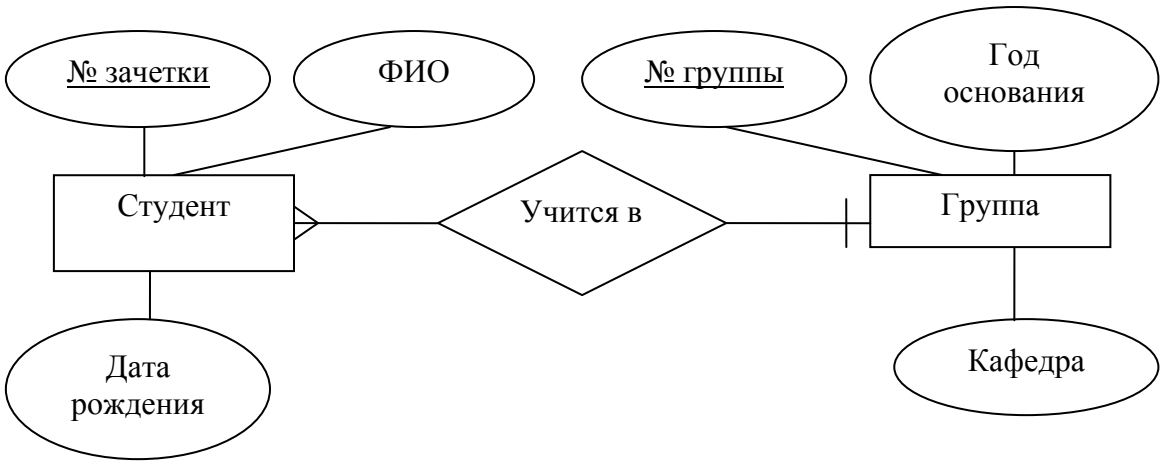


Рис. 4.7. Пример ER-модели в нотации Чена

2. Нотация Мартина: сущность изображается прямоугольником, внутри которого указано ее имя жирным шрифтом и список ее атрибутов (идентифицирующий атрибут подчеркнут), а связь – линией, название которой располагается над ней и ее вид в месте соединения с сущностью определяет кардинальность связи («воронья лапка» – M, «крест» – 1) (рис. 4.8).

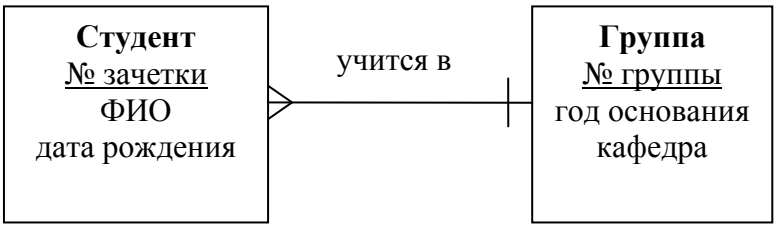


Рис. 4.8. Пример ER-модели в нотации Мартина

3. Нотация Баркера: сущность изображается прямоугольником, внутри которого указано ее имя жирным шрифтом и список ее атрибутов (перед идентифицирующим атрибутом стоит #), а связь – линией, название которой располагается над ней и ее вид в месте соединения с сущностью определяет кардинальность связи («воронья лапка» – M, отсутствие – 1) (рис. 4.9).

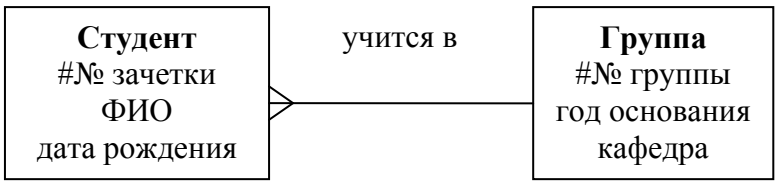


Рис. 4.9. Пример ER-модели в нотации Баркера

4. Нотация IDEF1X: сущность изображается прямоугольником, атрибут – овалом, соединенным со своей сущностью, а связь – ромбом, соединенным со связываемыми сущностями. Имена сущности, атрибута и связи располагаются внутри их изображений (рис. 4.10).

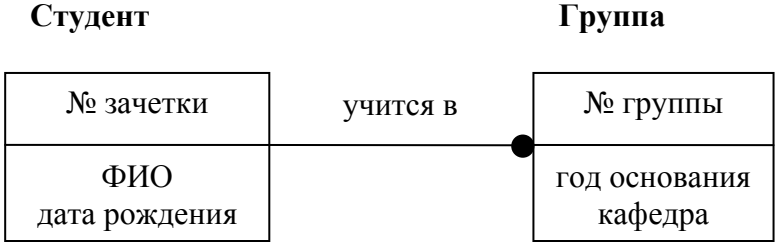


Рис. 4.10. Пример ER-модели в нотации IDEF1X

5. Нотация Бахмана: сущность изображается таблицей из одного столбца, столбцы которой являются атрибутами сущности (идентифицирующий атрибут выделен жирным шрифтом), а связь – стрелкой, соединяющей таблицы, направление которой указано на стороне M (рис. 4.11).

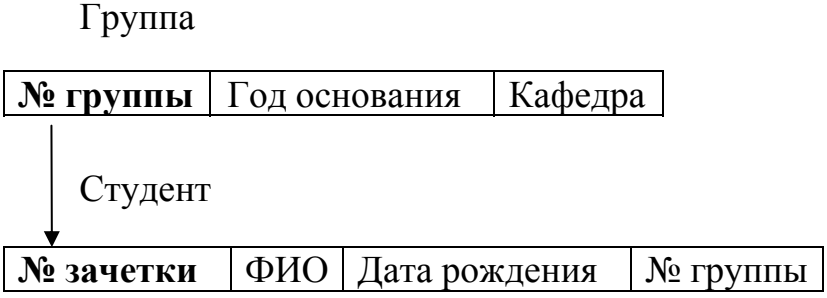


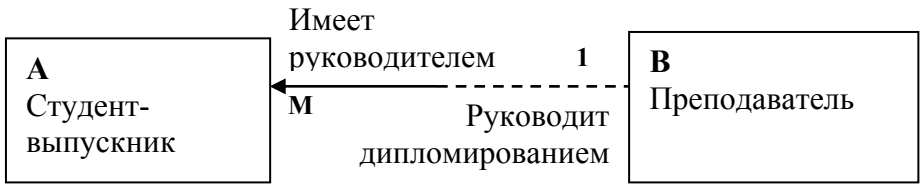
Рис. 4.11. Пример ER-модели в нотации Бахмана

Допустимые виды связей в ER-модели

Рассмотрим, какие возможны виды связей в ER-модели в зависимости от степени мощности и обязательности связей, полученные в методике Р. Баркера CASE*Method [11].

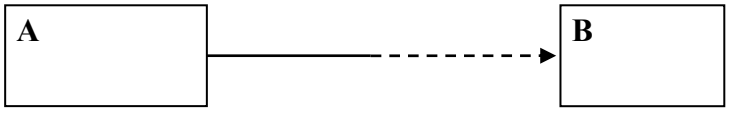
- **Многие к одному - 1:M или M:1**
  - ***Обязательность на одном конце с необязательностью на другом.***

Рассмотрим пример:



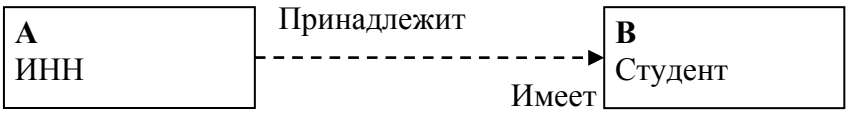
Это наиболее часто встречающаяся форма связи, которая предполагает, что каждое и любой экземпляр сущности A может существовать только в контексте одного (и только одного) экземпляра сущности B. С другой стороны, экземпляры B могут существовать как в связи с экземплярами A, так и без нее.

Противоположная ситуация:



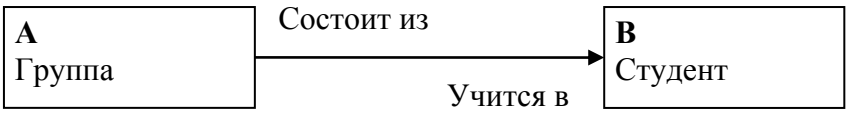
Это редко используемая конструкция и, вероятнее всего, имеет место, когда A представляет собой некоторое придуманное понятие, всегда включающее в себя точный набор вхождений B. При этом экземпляры B могут уже существовать сами по себе (при ближайшем рассмотрении эти связи зачастую оказываются связями типа «многие ко многим»).

- ***Необязательность на обоих концах.***



Применяется редко. Как A, так и B могут существовать без связи между ними.

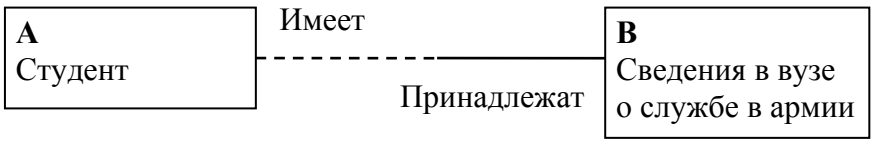
- ***Обязательность на обоих концах.***



Достаточно сильная конструкция, предполагающая, что экземпляр сущности B не может быть создан без одновременного создания одного связанного с ним экземпляра сущности A.

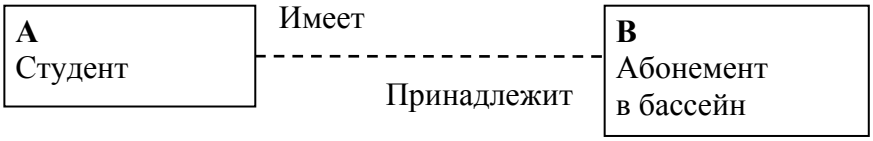
➢ **Один к одному - 1:1**

- ***Обязательность на одном конце с необязательностью на другом.***



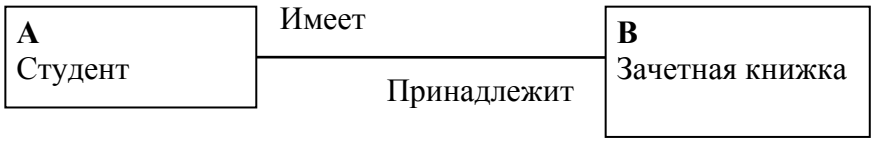
Используется редко.

- ***Необязательность на обоих концах.***



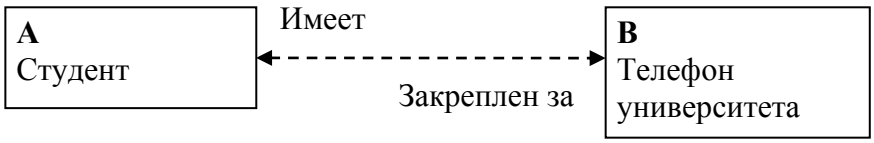
Используется редко.

- ***Обязательность на обоих концах.***



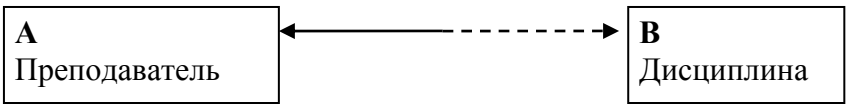
Крайне редко (почти всегда ошибочно). При ближайшем рассмотрении связи типа «один к одному» почти всегда оказывается, что A и B представляют собой в действительности разные подмножества одного и того же предмета или разные точки зрения на него, просто имеющие отличные имена и по-разному описанные связи и атрибуты.

- **Многие ко многим – N : M**
  - ***Необязательность на обоих концах.***



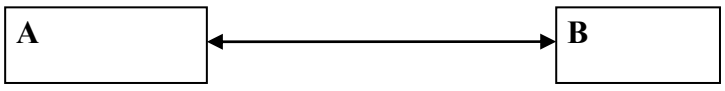
Такая конструкция часто имеет место в начале этапа анализа и означает связь, либо понятую не до конца и требующую дополнительного разрешения, либо отражающую простое коллективное отношение – двунаправленный список.

  - ***Обязательность на одном конце с необязательностью на другом.***



Применяется редко. Такие связи всегда подлежат дальнейшей детализации. Например, Преподаватель–Дисциплина, когда преподаватель должен проводить занятия по одной или нескольким дисциплинам, а по дисциплине могут проводить занятия один или несколько преподавателей или не проводиться совсем.

  - ***Обязательность на обоих концах.***



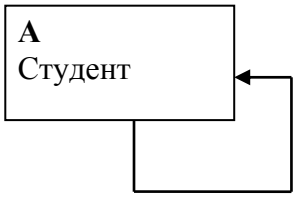
В принципе невозможна. Такая связь означала бы, что ни один из экземпляров А не может существовать без В и наоборот. На деле каждая подобная конструкция всегда оказывается ошибочной.

Рекурсивные связи – петля

- **Многие к одному М:1**
  - ***Обязательность на одном конце с необязательностью на другом.***

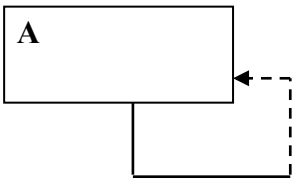
Связь Изделие–Изделие: может состоять из других компонентов-изделий.

  - ***Обязательность на обоих концах.***



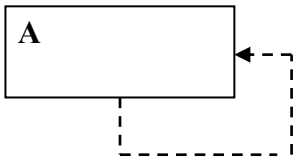
Связь Студент–Староста: студент должен иметь старосту, который является также студентом, а староста должен иметь студентов.

  - ***Необязательность на одном конце с обязательностью на другом.***



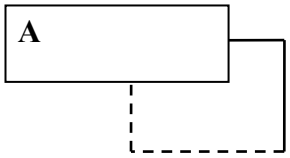
В принципе невозможна.

  - ***Необязательность на обоих концах.***



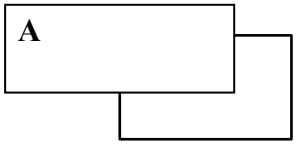
Довольно часто. Иногда называется «необязательное свиное ухо». Отражает наличие простой иерархии с любым числом уровней (организационная иерархия, классификация продуктов, структура рынка и т. п.).

- **Один к одному 1:1**
  - ***Обязательность на одном конце с необязательностью на другом.***



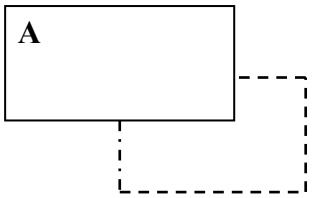
В принципе невозможна.

  - ***Обязательность на обоих концах.***



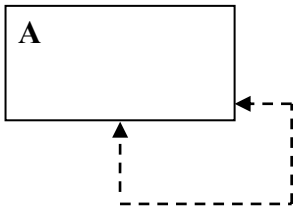
В принципе невозможна.

  - ***Необязательность на обоих концах.***



Редко, но имеет место. Отражает связи альтернативного типа. Например, Сотрудник–Преподаватель, когда сотрудник может быть преподавателем и наоборот.

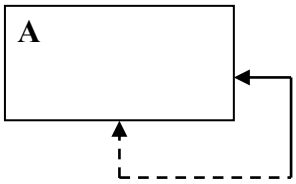
- **Многие ко многим**
  - ***Необязательность на обоих концах.***



Имеет место на ранних этапах проектирования. Часто отражает структуру «перечня материалов» (взаимная вложенность компонент).

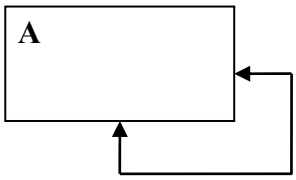
Например, каждая компонента может состоять из одной и более (других) компонент, и каждая компонента может использоваться в одной и более (других) компонентов.

- ***Обязательность на одном конце с необязательностью на другом.***



В принципе невозможна.

- ***Обязательность на обоих концах.***



В принципе невозможна.

4.1.2.2. OR-модель

OR-модель (Object-Role – объект-роль) рассматривает ПрО как совокупность элементарных фактов и описывает реальный мир в терминах *объектов*, которые играют определенные *роли* (участвуют в связях) по определенным правилам (в соответствии с определенными ограничениями). В отличие от ER-моделей в этой модели понятие атрибута явно не используется, а наличие атрибута сущности в ER-модели будет соответствовать связи между 2-мя объектами (атрибут становится самостоятельным объектом). В этой модели количество объектов в одной связи также неограничено. Пример OR-модели представлен на рис. 4.12.

Графическая нотация для OR-модели следующая. Объект-сущность (Студент) изображается поименованным овалом, свойство объекта (ФИО) – пунктирным поименованным овалом, связь (обучение студента в группе) – поименованным прямоугольником, разделенным на столько частей, сколько объектов участвует в связи, т. е. количество связей у одного объекта равно количеству его ролей. Идентифицирующее свойство (№ зачетки) может быть указано в круглых скобках после названия объекта или выделено в отдельный объект со связью. Черная точка возле объекта означает обязательность роли этого объекта в связи (группа должна обучаться по какой-нибудь специальности). Стрелка над ролью объекта в связи означает уникальность участия объекта в связи по отношению к другому объекту в этой связи (каждый студент учится только в одной группе). Стрелка над всеми ролями в одной связи говорит о том, что связь между объектами в этой связи M:N (кафедра обеспечива-

ет подготовку по нескольким специальностям; специальность может обеспечиваться несколькими кафедрами). Уникальность нескольких ролей объекта отображается соединяющей эти роли линией с кругом, внутри которого стоит U (uniqueness – уникальность) (студент с одной фамилией может учиться только в одной группе).

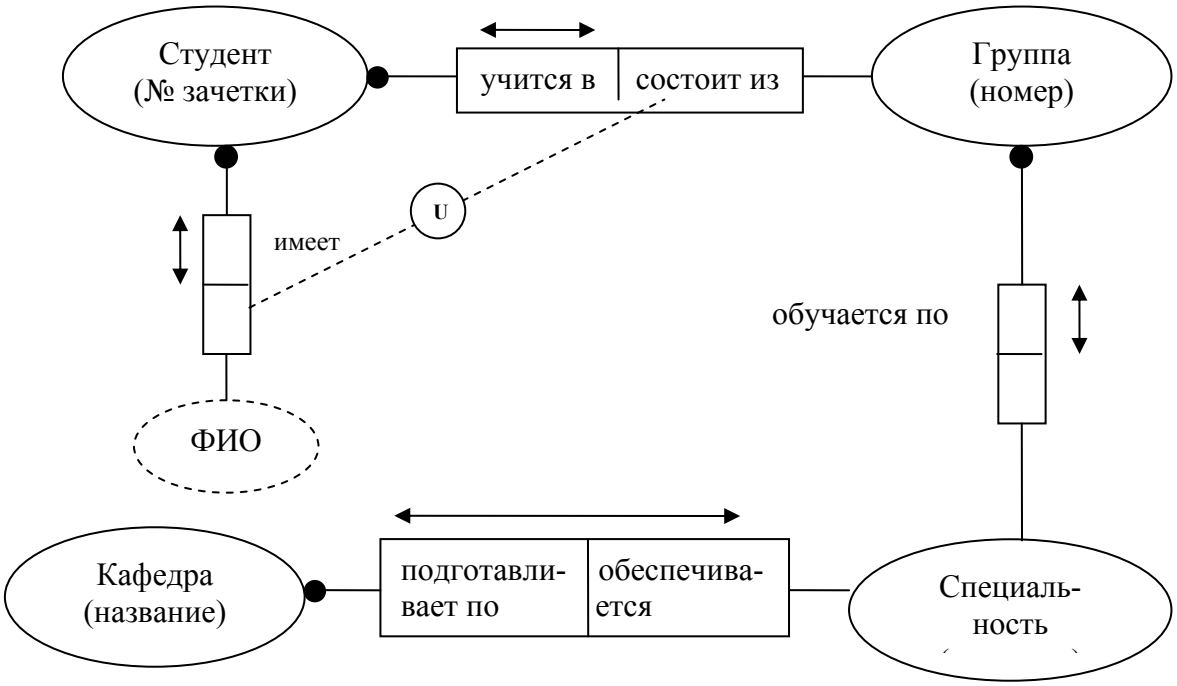


Рис. 4.12. Пример OR-модели

OR-модели, используя универсальный подход представления ПрО в виде объектов и их ролей в связях друг с другом, являются более выразительными моделями по сравнению с ER-моделями. Однако только потому, что одна модель является более выразительной, чем другая, не дает ей превосходства; среды разной выразительности моделирования являются разными инструментами для различных целей. При составлении модели всегда имеются компромиссы: ER-модели являются более компактными и в результате легко читаемыми, чем более выразительные и масштабными OR-моделями. В настоящее время для построения концептуальной схемы предпочитают пользоваться ER-моделями. Следует отметить, что построив одну модель (ER-модель), можно без особого труда получить из нее другую модель (OR-модель), и наоборот. Поскольку атрибут ER-модели преобразуется в объект OR-модели и наоборот, объект OR-модели, который играет только одну роль и связан с одним объектом, является атрибутом той сущности в ER-модели, с которой он связан.

4.2. Выбор СУБД

Выбор СУБД представляет собой сложную многопараметрическую задачу и является одним из важных этапов при разработке приложений БД. Выбранный программный продукт должен удовлетворять как текущим, так и будущим потребностям предприятия, при этом следует учитывать финансовые затраты на приобретение необходимого оборудования, самой системы, разработку необходимого программного обеспечения на ее основе, а также обучение персонала.

В настоящее время существуют некоторые требования, точнее, критерии при выборе СУБД, и приводится их классификация. Очевидно, наиболее простой подход при выборе СУБД основан на оценке того, в какой мере существующие системы удовлетворяют основным требованиям создаваемого проекта ИС. Более сложным и дорогостоящим вариантом является создание испытательного проекта на основе нескольких СУБД и последующий выбор наиболее подходящего из кандидатов. Но и в этом случае необходимо ограничивать круг возможных систем, опираясь на некоторые критерии отбора. В результате выделяют следующие группы критериев [17]:

- моделирование данных;
- особенности архитектуры и функциональные возможности;
- контроль работы системы;
- особенности разработки приложений;
- производительность;
- надежность;
- требования к рабочей среде;
- смешанные критерии.

Далее рассмотрим каждую из этих групп критериев.

➢ **Моделирование данных**

- *Используемая модель данных.* На сегодняшний день существуют следующие модели данных: иерархическая, сетевая, реляционная, объектно-ориентированная. Вопрос об использовании той или иной модели должен решаться на начальном этапе проектирования ИС.
- *Триггеры и хранимые процедуры.* Триггер – программа БД, автоматически вызываемая всякий раз при вставке, изменении или удалении строки таблицы. Триггеры обеспечивают проверку любых изменений на корректность, прежде чем эти изменения будут приняты. Хранимая процедура – программа, которая хранится на сервере и может вызываться клиентом. Поскольку хранимые процедуры

выполняются непосредственно на сервере БД, обеспечивается более высокое быстродействие, нежели при выполнении тех же операций средствами клиента БД. В различных программных продуктах для реализации триггеров и хранимых процедур используются различные инструменты.

- *Средства поиска.* Некоторые современные системы имеют встроенные дополнительные средства контекстного поиска.
- *Предусмотренные типы данных.* Здесь следует учесть два фактически независимых критерия: базовые или основные типы данных, заложенные в систему, и наличие возможности расширения типов. В то время как отклонения базовых наборов типов данных у современных систем от некоего стандартного обычно невелики, механизмы расширения типов данных в системах того или иного производителя существенно различаются.
- *Реализация языка запросов.* Все современные системы совместимы со стандартным языком доступа к данным SQL-92, однако многие из них реализуют те или иные расширения данного стандарта.

➢ **Особенности архитектуры и функциональные возможности**

- *Мобильность.* Это независимость системы от среды, в которой она работает. Средой в данном случае является как аппаратура, так и ПО (операционная система).
- *Масштабируемость.* При выборе СУБД необходимо учитывать, сможет ли данная система соответствовать росту ИС, причем рост может проявляться в увеличении числа пользователей, объема хранимых данных и объеме обрабатываемой информации.
- *Распределенность.* Основной причиной применения информационных систем на основе БД является стремление объединить взгляды на всю информацию организации. Самый простой и надежный подход – централизация хранения и обработки данных на одном сервере. Однако различные системы имеют разные возможности управления распределенными БД.
- *Сетевые возможности.* Многие системы позволяют использовать широкий диапазон сетевых протоколов и служб для работы и администрирования.

➢ **Контроль работы системы**

- *Контроль использования памяти компьютера.* Система может иметь возможность управления использованием как оперативной памяти, так и дискового пространства. Во втором случае это может выражаться, например, в сжатии БД или удалении избыточных файлов.

- *Автонастройка.* Многие современные системы включают в себя возможности самоконфигурирования, которые, как правило, опираются на результаты работы сервисов самодиагностики производительности. Данная возможность позволяет выявить слабые места конфигурации системы и автоматически настроить ее на максимальную производительность.

➢ **Особенности разработки приложений**

- Многие производители СУБД выпускают также средства разработки приложений для своих систем. Как правило, эти средства позволяют наилучшим образом реализовать все возможности сервера, поэтому при анализе СУБД стоит обратить внимание также и на возможности средств разработки приложений.
- *Средства проектирования.* Некоторые системы имеют средства автоматического проектирования как БД, так и прикладных программ. Причем, средства проектирования различных производителей могут существенно различаться.
- *Многоязыковая поддержка.* Поддержка большого количества национальных языков расширяет область применения системы и приложений, построенных на ее основе.
- *Возможности разработки Web-приложений.* При разработке различных приложений зачастую возникает необходимость использовать возможности среды Internet. Средства разработки некоторых производителей имеют большой набор инструментов для построения приложений под Web.
- *Поддерживаемые языки программирования.* Широкий спектр используемых языков программирования повышает доступность системы для разработчиков, а также может существенно повлиять на быстродействие и функциональность создаваемых приложений.

➢ **Производительность**

- *Рейтинг TPC* (Transactions per Cent – количество транзакций в процентом отношении). Для тестирования производительности применяются различные средства, и существует множество тестовых рейтингов. Одним из самых популярных и объективных является TPC-анализ производительности систем. Фактически TPC анализ рассматривает композицию СУБД и аппаратуры, на которой эта СУБД работает. Показатель TPC – это отношение количества запросов, обрабатываемых за некий промежуток времени, к стоимости всей системы.
- *Возможности параллельной архитектуры.* Для обеспечения параллельной обработки данных существует как минимум два подхо-

да: распараллеливание обработки последовательности запросов на несколько процессоров либо использование нескольких компьютеров-клиентов, работающих с одной БД, которые объединяют в так называемый параллельный сервер.

- *Возможности оптимизирования запросов.* При использовании непроцедурных языков запросов их выполнение может быть неоптимальным. Поэтому необходимо произвести процесс оптимизации запросов, т. е. выбрать такой способ выполнения, когда по начальному представлению запроса путем его синтаксических и семантических преобразований вырабатывается процедурный план выполнения запроса, наиболее оптимальный при существующих в БД управляющих структурах.

➢ **Надежность**

- Понятие надежности системы имеет много смыслов – это и сохранность информации независящая от любых сбоев, и безотказность работы системы в любых условиях, и обеспечение защиты данных от несанкционированного доступа.
- *Восстановление после сбоев.* При возникновении программных или аппаратных сбоев целостность, да и работоспособность всей системы может быть нарушена. От того, как эффективно спланирован механизм восстановления после сбоев, зависит жизнеспособность системы.
- *Резервное копирование.* В результате аппаратного сбоя может быть частично поврежден или выведен из строя носитель информации и тогда восстановление данных невозможно, если не было предусмотрено резервное копирование БД или ее части. Резервное копирование спасает и в ситуациях, когда происходит логический сбой системы, например, при ошибочном удалении таблиц. Существует множество механизмов резервирования данных (хранение одной или более копий всей БД, хранение копии ее части, копирование логической структуры и т. д.). Зачастую в систему закладывается возможность использования нескольких таких механизмов.
- *Откат изменений.* При выполнении транзакции применяется простое правило – либо транзакция выполняется полностью, либо не выполняется вообще. Это означает, что в случае сбоев, все результаты не доведенных до конца транзакций должны быть аннулированы. Механизм отката может иметь различное быстродействие и эффективность.
- *Многоуровневая система защиты.* ИС организации почти всегда включает в себя секретную информацию, поэтому для предотвра-

щения несанкционированного доступа используется служба идентификации пользователей. Уровень защиты может быть различным. Кроме непосредственной идентификации пользователей при входе в систему может использоваться также механизм шифрования данных при передаче по линиям связи.

➢ **Требования к рабочей среде**

- *Поддерживаемые аппаратные платформы.*
- *Минимальные требования к оборудованию.*
- *Максимальный размер адресуемой памяти.* Поскольку почти все современные системы используют свою файловую систему, немаловажным фактором является то, какой максимальный объем физической памяти они могут использовать.
- *Операционные системы*, под управлением которых способна работать СУБД.

➢ **Прочие критерии**

- *Качество и полнота документации.* Для любой системы не последнюю роль играет наличие подробной и качественной документации.
- *Локализованность.* Возможность использования национальных языков является еще одним преимуществом выбранной системы.
- *Модель формирования стоимости.* Как правило, производители СУБД используют определенные модели формирования стоимости. Например, стоимость одного и того же продукта может существенно изменяться в зависимости от того, сколько пользователей будет с ним работать.
- *Стабильность производителя.*
- *Распространенность СУБД.*

Четкий и глубокий сравнительный анализ на основании вышеперечисленных критериев поможет рационально выбрать подходящую систему для конкретного проекта, и затраченные усилия не будут напрасными.

4.3. Проектирование физической структуры БД

На этапе проектирования физической структуры БД решается вопрос о способе реализации БД, но после того, как будет определена целевая СУБД. Целью этого этапа является описание способа физической реализации логической модели, полученной на предыдущем этапе, в среде выбранной СУБД.

Для реляционной СУБД должны быть создан набор таблиц и ограничений, представленный в реляционной модели данных. Для физиче-

ской модели также должны быть определены структуры хранения данных и методы доступа к ним, обеспечивающие требуемую производительность системы.

Физическая модель фактически представляет собой ddl[3]-скрипт по созданию объектов в БД: таблиц, описанных в реляционной модели данных, др. объектов (представлений, триггеров, процедур, функций и др.) для реализации необходимых бизнес-правил в среде СУБД. Пример физической модели для создания таблицы Личность представлен ниже:

```
CREATE TABLE Личность
(
  фамилия VARCHAR2(100) NOT NULL,
  имя VARCHAR2(80) NOT NULL,
  отчество VARCHAR2(90) NOT NULL,
  дата_рождения DATE,
  место рождения VARCHAR2(100),
  пол VARCHAR2(10) NOT NULL,
  национальность VARCHAR2(30)
);
```

4.4. CASE-средства, используемые при проектировании БД

На сегодняшний день рынок CASE-средств, используемых для проектирования БД, представлен достаточно широко, среди популярных продуктов можно выделить следующие: Computer Associates AllFusion ERwin Data Modeler, Oracle Designer, Microsoft Office Visio, Sybase Power Designer, IDS Prof. Scheer ARIS и др.

Рассмотрим два мощных средства, позволяющих выполнить весь процесс проектирования БД от построения концептуальной схемы данных ПрО до создания физической модели БД: Computer Associates AllFusion ERwin Data Modeler и Oracle Designer.

ERwin Data Modeler

- ✓ ERwin – это графический инструментарий для моделирования данных, основной целью которого является поддержка процесса описания бизнес-правил и требований к информации при создании логических и физических моделей данных.
- ✓ ERwin является мощным и достаточно простым в использовании средством проектирования реляционных БД, завоевавшим широкое признание и популярность.

---

[3] ddl (data definition language) – язык определения данных, используемый для создания, изменения и удаления объектов БД

- ✓ На протяжении всего процесса – от логического моделирования требований к информации и бизнес-правилам, которые определяют БД, до оптимизации физической модели в соответствии с заданными характеристиками – ERwin позволяет наглядно отобразить структуру и основные элементы БД.
- ✓ ERwin – это не просто средство, облегчающее проектирование, но и инструмент разработки, способный автоматически создавать таблицы и генерировать тексты хранимых процедур и триггеров для многих популярных СУБД.
- ✓ Однако необходимо понимать, что сам по себе ERwin – это по большому счету лишь удобное средство «рисования» с множеством дополнительных возможностей, позволяющее зафиксировать и наглядно представить то, что спроектировал пользователь. Использование ERwin (и стандарта IDEF1X в его рамках) само по себе не обеспечивает получение оптимальной схемы БД. Поэтому для эффективного использования данного инструмента необходимо хорошее знание теории проектирования реляционных БД.
- ✓ В ERwin предусмотрено два типа моделей: логическая и физическая. Логическая модель – это абстрактный взгляд на данные, это КИМПО. Логическая модель данных является универсальной в том смысле что, она никак не связана с конкретной реализацией для выбранной СУБД. Физическая модель – модель БД для конкретной СУБД, которая фактически является отображением системного каталога БД. В физической модели содержится уже информация не об объектах ПрО, а об объектах БД. Одной и той же логической модели могут соответствовать несколько разных физических моделей.
- ✓ ERwin поддерживает стандарт моделирования IDEF1X (Integration DEFinition for Information Modeling). IDEF1X является методом для разработки реляционных БД и использует условный синтаксис, специально разработанный для удобного построения концептуальной схемы. IDEF1X подразумевает использование различных уровней моделирования при проектировании системы.

➢ *Логической модели* в IDEF1X соответствуют три уровня моделирования, отличающиеся по глубине представления информации о данных:

- Диаграмма сущность-связь (Entity-Relationship Diagram (ERD)) – это модель данных верхнего уровня, которая включает сущности и взаимосвязи, отражающие основные бизнес-правила предметной области (рис. 4.13).

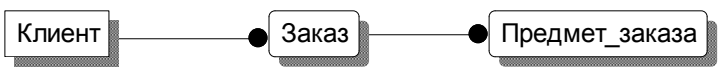


Рис. 4.13. Фрагмент диаграммы сущность-связь

- Модель данных, основанная на ключах (Key-Based Model (KBM)) – более подробное представление данных, включает описание сущностей и первичных ключей (рис. 4.14).

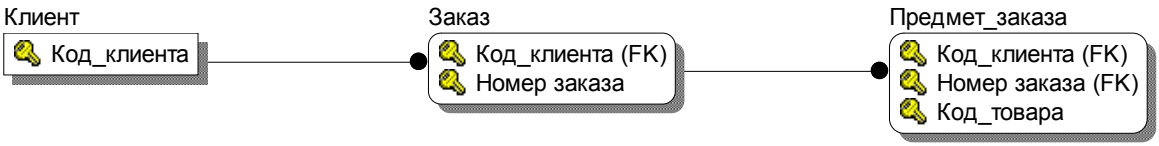


Рис. 4.14. Фрагмент модели данных, основанной на ключах

- Полная атрибутивная модель (Fully-Attributed Model (FAM)) – наиболее детальное представление структуры данных, представляет данные в третьей нормальной форме и включает все сущности, атрибуты и связи (рис. 4.15).

Рис. 4.15. Фрагмент полной атрибутивной модели

➢ *Физическая модель* в IDEF1X представлена двумя уровнями:

- Трансформационная модель (Transformation Model (TM)) – модель данных, соответствующая конкретной СУБД. Структуры данных оптимизируются исходя из возможностей СУБД, объемов данных, предполагаемых схем доступа и интенсивности использования данных (рис. 4.16).

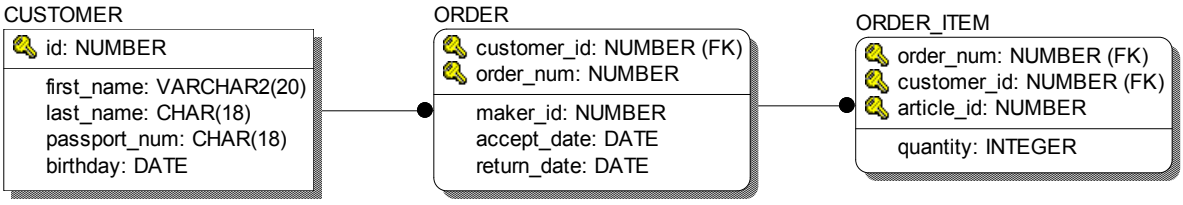


Рис. 4.16. Фрагмент трансформационной модели

- Модель СУБД (DBMS Model) – содержит определения объектов физической структуры БД в схеме СУБД или системном каталоге БД, напрямую генерируется из TM. ERwin непосредственно поддерживает эту модель путем генерации схемы БД. Фактически эта модель представляет собой текст DDL-предложений на языке SQL:

```
CREATE TABLE CUSTOMER (
    id              NUMBER NOT NULL,
    first_name      VARCHAR2(20) NULL,
    last_name       CHAR(18) NULL,
    passport_num    CHAR(18) NULL,
    birthday        DATE NULL,
    PRIMARY KEY (id));
CREATE TABLE ORDER (
    id              NUMBER NOT NULL,
    order_num       NUMBER NOT NULL,
    maker_id        NUMBER NULL,
    accept_date     DATE NULL,
    return_date     DATE NULL,
    PRIMARY KEY (id, order_num),
    FOREIGN KEY (id)
            REFERENCES CUSTOMER (id));
CREATE TABLE ORDER_ITEM (
    id              NUMBER NOT NULL,
    order_num       NUMBER NOT NULL,
    article_id      NUMBER NOT NULL,
    quantity        INTEGER NULL,
    PRIMARY KEY (order_num, id, article_id),
    FOREIGN KEY (id, order_num)
            REFERENCES ORDER (id, order_num));
```

Таким образом, процесс разработки БД в ERwin фактически представляет собой последовательное продвижение по уровням моделирования от диаграммы сущность-связь до генерации кода, необходимого для физического создания БД.

Oracle Designer

Oracle Designer – это сложное приложение, реализованное в среде СУБД Oracle, состоящее из двух частей: серверной и клиентской. Серверная сторона Oracle Designer представляет собой репозитарий (хранилище метаданных) и API по управлению репозитарием. Каждый раз, когда выполняется операция добавления, сохранения, удаления и т. д. в одном из инструментов клиентской стороны, изменения посредством API заносятся в репозитарные таблицы. Для проектирования БД серверная часть не нужна, а необходимо использовать клиентскую часть Oracle Designer. Клиентская сторона продукта – это набор инструментов, включающий несколько диаграммеров, навигаторов, утилит преобразования и генераторов, а также утилиты, поддерживающие все этапы разработки системы.

Для проектирования БД используются следующие инструменты Oracle Designer:

- ✓ Entity-Relationship Diagrammer – для проектирования КИМПО и построения диаграммы сущность–связь.
- ✓ Design Editor – для построения реляционной модели ПрО.

Фрагмент реляционная модели ПрО вуз представлен на диаграмме, построенной в Design Editor, на рис. 4.17.

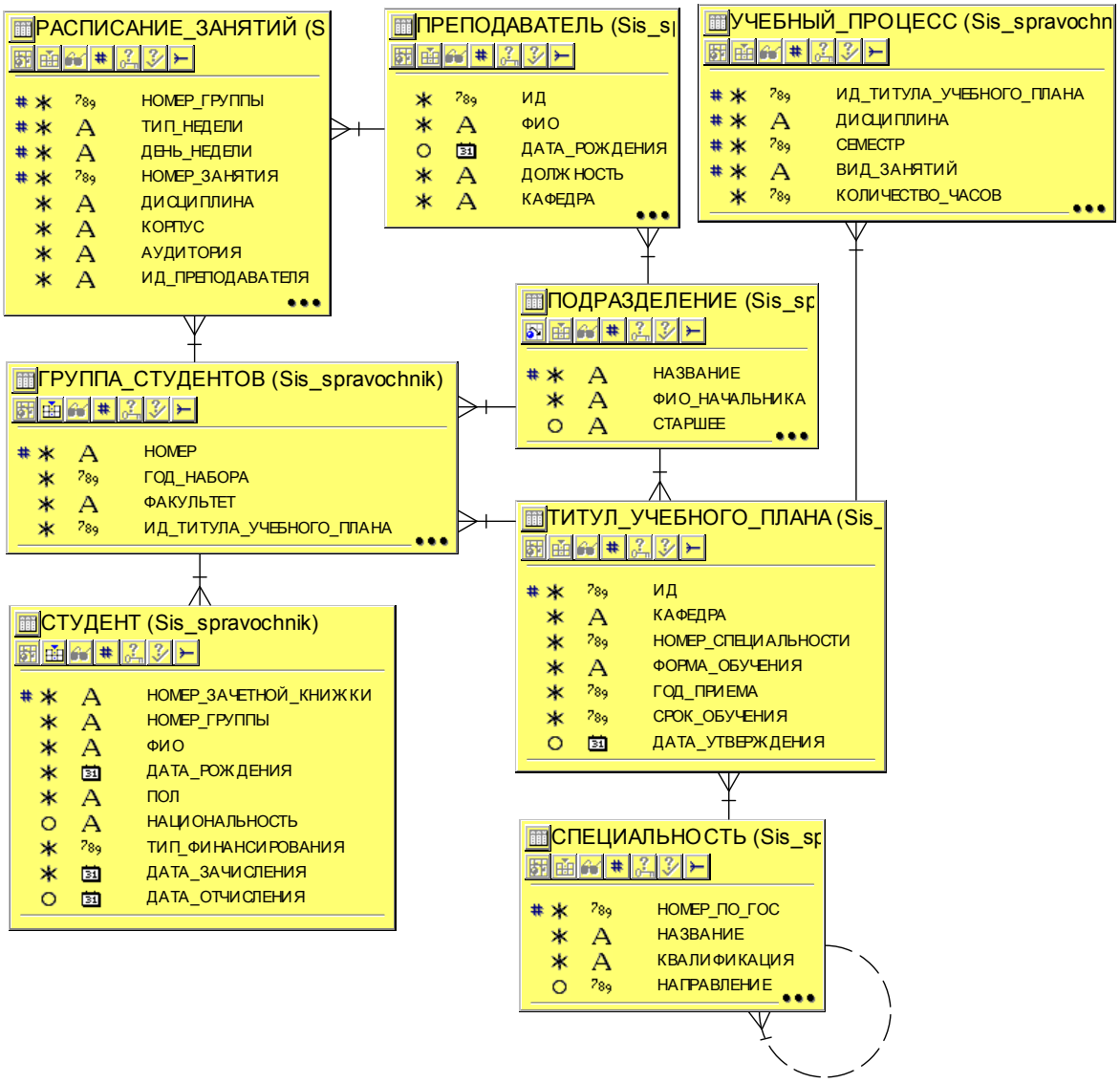


Рис. 4.17. Пример реляционной модели, построенной в Design Editor

Для получения физической структуры БД в Oracle Designer используется утилита Generate Database from Server Model, которая создает ddl_файл для генерации объектов БД.

4.5. Пример проектирования БД

Рассмотрим на примере весь процесс проектирования БД от построения концептуальной схемы данных ПрО до создания физической модели БД на основе имеющегося описания ПрО «Отдел кадров в части трудоустройства». Описание включает в себя информационных потребностей пользователей и основных ограничений к ПрО.

В отделе кадров предприятия накапливается и обрабатывается анкетная информация о работниках: ФИО, дата, место рождения, адрес, паспортные данные, национальность, гражданство, знание иностранных языков, дети (ФИО, дата рождения), образование (учебное заведение, год окончания, № диплома, специальность, квалификация), история работы (дата поступления, название организации, должность, дата увольнения).

При трудоустройстве работник заключает контракт с предприятием, который включает информацию: ФИО, подразделение, должность, условия работы: длительность рабочего дня (число часов либо ненормированный), место работы, система оплаты (оклад, сдельная, повременная) и размер оплаты, требования к работнику, его адрес, паспортные данные. Условия контракта могут изменяться с течением времени по соглашению сторон. Все изменения должны быть зафиксированы и сохранены.

Кроме того, в отделе кадров накапливается информация о награждениях и взысканиях, повышениях квалификации, совмещении должностей.

Используя классический интеграционный подход для выявления основных сущностей ПрО, анализируя имеющуюся выше информацию, получим следующие результаты.

Во-первых, однозначно можно сказать, что есть сущность **Работник** с атрибутами о персональных сведениях *(ФИО, дата рождения, место рождения, национальность)*.

Если место рождения представлять детально, то получим следующий вариант сущности **Работник1** (*ФИО, дата рождения, страна рождения, регион рождения, район рождения, тип населенного пункта рождения, название населенного пункта рождения*). Таким образом, для всех атрибутов, по которым необходимо детальная информация, не создаются отдельные сущности, а в той же сущности этот атрибут заменяется на те атрибуты, которые необходимо заполнить.

Далее определяем сущности, которые могут иметь несколько экземпляров для одного работника.

Определяя сущность **Паспортные данные**, полагаем, что человек может иметь несколько паспортов, но в одной стране не может быть двух паспортов с одинаковыми серией и номером, даже в совокупности с уже недействительными. Заметим, что атрибут *ID работника* не является ключевым, т. к. один паспорт не может принадлежать нескольким личностям (*ID работника* однозначно для конкретного значения совокупности атрибутов *Страна, выдавшая паспорт, Серия паспорта, Номер паспорта*, составляющих ключ сущности). Здесь полагаем, что атрибут *кем выдан* не требует детализации, при этом следует помнить, что в реляционной СУБД в этом случае возникнут сложности в выборке записей, например, по датам выдачи паспортов. В противном случае потребуется вместо атрибута *кем выдан* определить атрибуты, детально описывающие информацию о том, кем выдан паспорт.

Паспортные данные (*ID работника, Тип паспорта, Страна, выдавшая паспорт, Серия паспорта, Номер паспорта, Кем выдан, Дата выдачи, Действителен до*).

Полагаем, что у работника может быть только один адрес регистрации и только один адрес проживания и что работник может проживать и быть зарегистрированным в разных городах.

Адрес работника (ID работника, Тип адреса, Город, Улица, Номер дома, Номер квартиры)

Полагая, что одна личность может быть гражданином нескольких стран, определяем сущность **Гражданство**.

Гражданство (*ID работника, Страна гражданства, Дата предоставления*).

Если необходима история гражданства, то необходимо добавить атрибут *Дата лишения* и сделать его, либо атрибут *Дата предоставления* – ключевым (добавить и в ключ).

Аналогично определяем сущности **Знание иностранных языков** и **Дети**.

Знание иностранных языков (*ID работника, Язык, Степень владения*).

Дети (*ID работника, ФИО ребенка, Дата рождения ребенка*).

Определяя сущность **Образование**, полагаем, что не может быть двух дипломов с одинаковым номером.

Образование (*ID работника, Учебное заведение, Год окончания, Номер диплома, Специальность, Квалификация*).

Далее анализируем часть описания ПрО, связанной с историей работы. Полагаем, что работник мог совмещать должности в одной организации и несколько раз (в разные периоды), быть на одной и той же должности, а также одновременно работать в нескольких организациях.

Получаем сущность **История работы** (ID работника, Название организации, Должность, Дата поступления, Дата увольнения).

Далее анализируем часть описания, связанную с контрактом.

Полагаем, что работник может иметь **только один основной контракт** (*ID работника* не входит в ключ), все другие трудовые соглашения и изменения осуществляются через дополнительные соглашения к основному контракту. Требования к работнику в описании не формализованы и требуют дальнейшего анализа.

Основной контракт (ID работника, № контракта, Дата заключения контракта, Подразделение, Должность, Длительность рабочего дня, Рабочее место, Система оплаты, Размер оплаты, Требования к работнику).

Поскольку не известна структура контракта, а в рамках одного дополнительного соглашения могут одновременно поменяться значения нескольких атрибутов, структура дополнительного соглашения во многом повторяет структуру основного.

Дополнительное соглашение (ID работника, № контракта, Дата заключения контракта, Дата дополнительного соглашения к основному контракту, Подразделение, Должность, Длительность рабочего дня, Рабочее место, Система оплаты, Размер оплаты, Требования к работнику).

Кроме того, в отделе кадров накапливается информация о награждениях и взысканиях, повышениях квалификации, совмещении должностей.

Полагаем, что работник может получать награду/поощрения одного типа неоднократно.

Награждение (ID работника, Тип награды, Дата награждения).

Поощрение (ID работника, Тип поощрения, Дата поощрения).

Так как работник может проходить квалификацию несколько раз в одном и том же учебном заведении, поэтому необходимо учитывать дату начала для каждого повышения квалификации.

Повышение квалификации (ID работника, Учебное заведение, Дата начала, Дата окончания, Специальность, Номер удостоверения).

Построенная концептуальная схема ПрО «Отдел кадров» в нотации IDEF1X представлена на рис. 4.18.

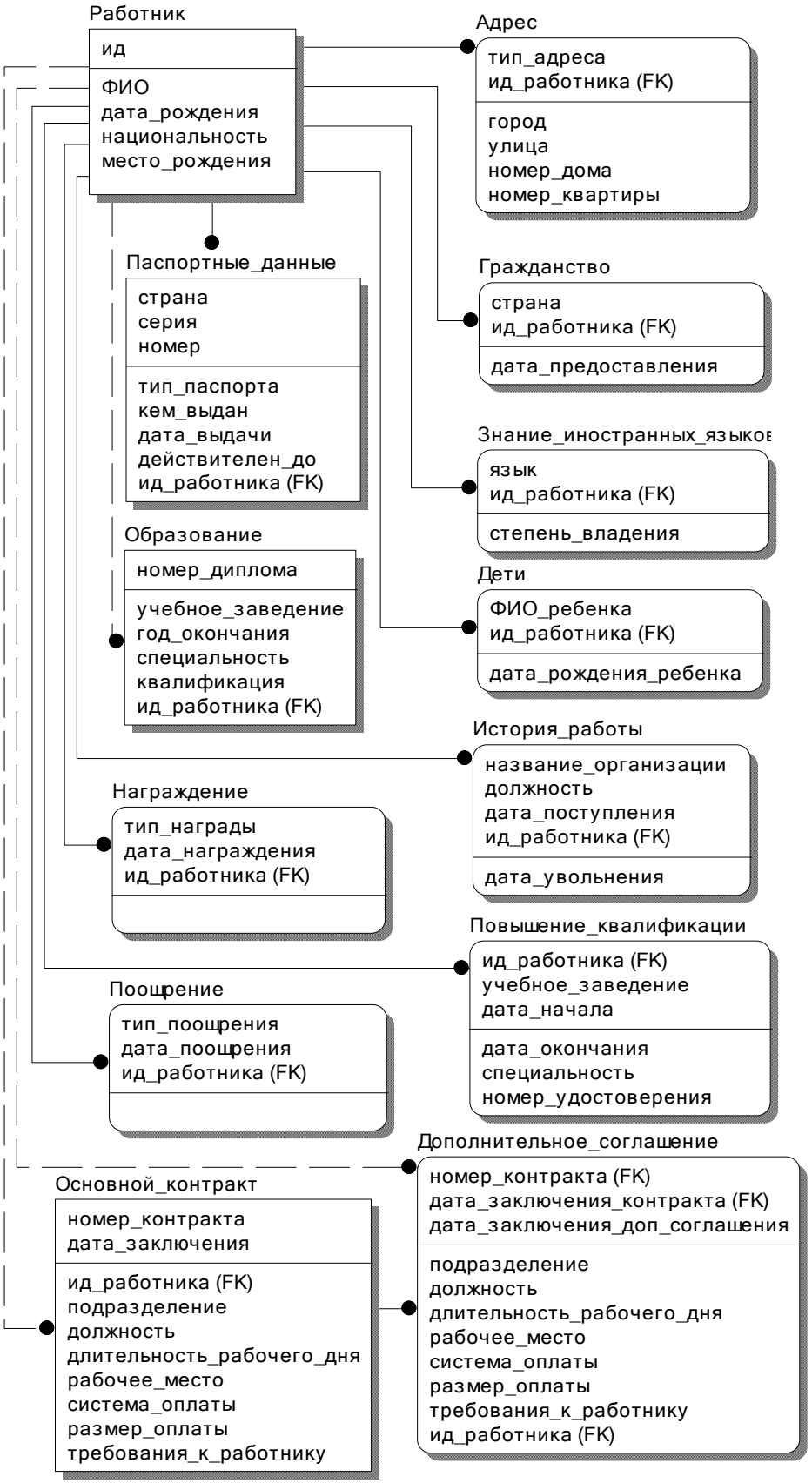


Рис. 4.18. Концептуальная схема ПрО «Отдел кадров» в нотации IDEF1X

Ниже приведен листинг ddl-файла, запустив который, выполнится скрипт по созданию физической структуры БД для спроектированной выше концептуальной схемы ПрО «Отдел кадров».

```
CREATE TABLE Работник
(
        ид                  NUMBER(4)  NOT NULL PRIMARY KEY,
        ФИО                 VARCHAR2(60)  NOT NULL,
        дата_рождения       DATE  NOT NULL,
        национальность      VARCHAR2(30)  NOT NULL,
        место_рождения      VARCHAR2(100)  NOT NULL
);

CREATE TABLE Паспортные_данные
(
        страна              VARCHAR2(50)  NOT NULL,
        серия               VARCHAR2(10)  NOT NULL,
        номер               VARCHAR2(20)  NOT NULL,
        тип_паспорта        VARCHAR2(15)  NOT NULL,
        ид_работника        VARCHAR2(18)  NOT NULL,
        кем_выдан           VARCHAR2(30)  NOT NULL,
        дата_выдачи         DATE NOT NULL,
        действителен_до     DATE  NULL,
        PRIMARY KEY (страна, серия, номер)
);

CREATE TABLE Адрес
(
        ид_работника        CHAR(18)  NOT NULL,
        тип_адреса          VARCHAR2(20)  NOT NULL,
        город               VARCHAR2(30)  NOT NULL,
        улица               VARCHAR2(30)  NOT NULL,
        номер_дома          VARCHAR2(5)  NOT NULL,
        номер_квартиры      VARCHAR2(5)  NOT NULL,
        PRIMARY KEY (ид_работника, тип_адреса)
);

CREATE TABLE Гражданство
(
        ид_работника        CHAR(18)  NOT NULL,
        страна              VARCHAR2(50)  NOT NULL,
        дата_предоставления  DATE  NOT NULL,
        PRIMARY KEY (ид_работника, страна)
);

CREATE TABLE Знание_иностранных_языков
(
        ид_работника        VARCHAR2(18)  NOT NULL,
        язык                VARCHAR2(20)  NOT NULL,
        степень_владения    VARCHAR2(20)  NOT NULL,
        PRIMARY KEY (ид_работника, язык)
);

CREATE TABLE Дети
(
        ид_работника        CHAR(18)  NOT NULL,
```

```
        ФИО_ребенка     VARCHAR(100)  NOT NULL,
        дата_рождения_ребенка  DATE  NOT NULL,
        PRIMARY KEY (ид_работника, ФИО_ребенка)
);

CREATE TABLE Образование
(
        номер_диплома   VARCHAR2(20)  NOT NULL PRIMARY KEY,
        ид_работника     VARCHAR2(18)  NOT NULL,
        учебное_заведение  VARCHAR2(50)  NOT NULL,
        год_окончания    NUMBER(4)  NOT NULL,
        специальность    VARCHAR2(30)  NOT NULL,
        квалификация     VARCHAR2(30)  NOT NULL
);

CREATE TABLE История_работы
(
        ид_работника     VARCHAR2(18)  NOT NULL,
        название_организации  VARCHAR2(50)  NOT NULL,
        должность        VARCHAR2(30)  NOT NULL,
        дата_поступления  DATE  NOT NULL,
        дата_увольнения          DATE  NULL,
        PRIMARY KEY (ид_работника, название_организации, должность,
        дата_поступления)
);

CREATE TABLE Основной_контракт
(
        номер_контракта          VARCHAR2(10)  NOT NULL,
        дата_заключения          DATE  NOT NULL,
        ид_работника     VARCHAR2(18)  NOT NULL,
        подразделение    VARCHAR2(50)  NOT NULL,
        должность        VARCHAR2(30)  NOT NULL,
        длительность_рабочего_дня  DATE  NOT NULL,
        рабочее_место    VARCHAR2(30)  NOT NULL,
        система_оплаты   VARCHAR2(10)  NOT NULL,
        размер_оплаты    NUMBER(10,2)  NOT NULL,
        требования_к_работнику  VARCHAR2(500)  NOT NULL,
        PRIMARY KEY (номер_контракта, дата_заключения)
);

CREATE TABLE Дополнительное_соглашение
(
        номер_контракта          VARCHAR2(10)  NOT NULL,
        дата_заключения_контракта  DATE  NOT NULL,
        дата_заключения_доп_соглашения  DATE  NOT NULL
        ид_работника     VARCHAR2(18)  NOT NULL,
        подразделение    VARCHAR2(50)  NOT NULL,
        должность        VARCHAR2(30)  NOT NULL,
        длительность_рабочего_дня  DATE  NOT NULL,
        рабочее_место    VARCHAR2(30)  NOT NULL,
        система_оплаты   VARCHAR2(10)  NOT NULL,
        размер_оплаты    NUMBER(10,2)  NOT NULL,
        требования_к_работнику  VARCHAR2(500)  NOT NULL,
        PRIMARY KEY (номер_контракта, дата_заключения_контракта,
        дата_заключения_доп_соглашения)
);
```

```
CREATE TABLE Награждение
(
        ид_работника      VARCHAR2(18)  NOT NULL,
        тип_награды       VARCHAR2(30)  NOT NULL,
        дата_награждения  DATE  NOT NULL,
        PRIMARY KEY (ид_работника, тип_награды, дата_награждения)
);

CREATE TABLE Поощрение
(
        ид_работника      VARCHAR2(18)  NOT NULL,
        тип_поощрения   VARCHAR2(30)  NOT NULL,
        дата_поощрения            DATE  NOT NULL,
        PRIMARY KEY (ид_работника, тип_поощрения, дата_поощрения)
);

CREATE TABLE Повышение_квалификации
(
        ид_работника      VARCHAR2(18)  NOT NULL,
        учебное_заведение     VARCHAR2(50)  NOT NULL,
        дата_начала                 DATE NOT NULL,
        дата_окончания   NUMBER(4)  NULL,
        специальность     VARCHAR2(30)  NOT NULL,
        номер_удостоверения  VARCHAR2(10)  NOT NULL,
        PRIMARY KEY (ид_работника, учебное_заведение, дата_начала)
);

ALTER TABLE Паспортные_данные
        ADD FOREIGN KEY (ид_работника) REFERENCES Работник(ид);
ALTER TABLE Адрес
        ADD FOREIGN KEY (ид_работника) REFERENCES Работник(ид);
ALTER TABLE Гражданство
        ADD FOREIGN KEY (ид_работника) REFERENCES Работник(ид);
ALTER TABLE Знание_иностранных_языков
        ADD FOREIGN KEY (ид_работника) REFERENCES Работник(ид);
ALTER TABLE Дети
        ADD FOREIGN KEY (ид_работника) REFERENCES Работник(ид);
ALTER TABLE Образование
        ADD FOREIGN KEY (ид_работника) REFERENCES Работник(ид);
ALTER TABLE История_работы
        ADD FOREIGN KEY (ид_работника) REFERENCES Работник(ид);
ALTER TABLE Основной_контракт
        ADD FOREIGN KEY (ид_работника) REFERENCES Работник(ид);
ALTER TABLE Дополнительное_соглашение
        ADD FOREIGN KEY (ид_работника) REFERENCES Работник(ид);
ALTER TABLE Дополнительное_соглашение
        ADD FOREIGN KEY (номер_контракта, дата_заключения_контракта)
        REFERENCES Основной_контракт(номер_контракта, дата_заключения);
ALTER TABLE Награждение
        ADD FOREIGN KEY (ид_работника) REFERENCES Работник(ид);
ALTER TABLE Поощрение
        ADD FOREIGN KEY (ид_работника) REFERENCES Работник(ид);
ALTER TABLE Повышение_квалификации
        ADD FOREIGN KEY (ид_работника) REFERENCES Работник(ид);
/
```

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Мартин Дж. Организация баз данных в вычислительных системах: пер. с англ. – 2-е изд., доп. – М.: Мир, 1980. – 662 с.
2. Дейт К. Дж. Введение в системы баз данных: пер. с англ. – 6-е изд. – К.; М.; СПб.: Издательский дом «Вильямс», 2000. – 848 с.
3. Олле Т.В. Предложения КОДАСИЛ по управлению базами данных: пер. с англ. / пер. В.И. Филиппов; С.М. Кругова // Финансы и статистика, 1981. – 286 с.
4. Godd E.F. A Relation Model of Data for Large Shared Data Banks. Communications of the ACM 13:6, 1970. – P. 377–387.
5. History of IBM // http://www-03.ibm.com/ibm/history/
6. Andy Oppel. Databases Demystified. – San Francisco, CA: McGraw-Hill Osborne Media., 2006. – P. 90–91. – ISBN 0-07-225364-9
7. ISO/IEC 9075-11:2008: Information and Definition Schemas (SQL/Schemata) // http://www.iso.org/iso/home.html
8. O'Reilly Network. An Interview with Chris Date by Tony Williams // http://www.oreillynet.com/lpt/a/6060
9. Oracle Database SQL Reference 10*g* – Oracle Corporation, Release 2 (10.2), 2005.
10. ANSI Standard SQL Joins, by Jonathan Gennick: журнал Oracle Magazine, no.6, 2001 (/oramag/oracle/01-nov/o61ansi.html)
11. Barker R. Case*Method – Entity Relationship Modelling / R. Barker. – Addison Wesley Professional, Great Britain, 1990.
12. Перегудов Ф.И., Тарасенко Ф.П. Основы системного анализа: учебник. 2-е изд., доп. – Томск: Изд-во НТЛ, 1997. – 396 с.: ил.
13. Основы системного подхода и их приложение к разработке территориальных автоматизированных систем управления / под ред. Ф.И. Перегудова. – Томск: Изд-во Томского ун-та, 1976. – 243 с.
14. Перегудов Ф.И., Сагатовский В.Н., Ямпольский В.З., Кочнев Л.В. Принципы декомпозиции целей и методика построения целей в системах организационного управления // Кибернетика и вуз. – 1975. – Вып. 8. – С. 3–20.
15. Чудинов И.Л. Интеграционный подход к концептуальному проектированию информационной базы единой информационной среды / И.Л. Чудинов, И.В. Исаев // Материалы шестой Всероссийской научно-технической конференции «Теоретические и прикладные вопросы современных информационных технологий». – Улан-Удэ, 2005.
16. Kim Y.-G. Comparing Data Modeling Formalisms / Y.-G. Kim, S.T. March // Communications of the ACM. – 1995. – Vol. 38. – No. 6. – P. 103–115.
17. Аносов А. Критерии выбора СУБД при создании информационных систем [Электронный ресурс]. – URL: www.interface.ru

Учебное издание

ЧУДИНОВ Игорь Леонидович
ОСИПОВА Виктория Викторовна

БАЗЫ ДАННЫХ

Учебное пособие

Научный редактор
доктор технических наук,
профессор В.А. Силич

Редактор *Е.Л. Тен*
Компьютерная верстка и дизайн обложки
О.Ю. Аршинова

Подписано к печати 02.11.2012. Формат 60x84/16. Бумага «Снегурочка».
Печать XEROX. Усл.печ.л. 8,14. Уч.-изд.л. 7,36.
Заказ 1312-12. Тираж 100 экз.

Национальный исследовательский Томский политехнический университет
Система менеджмента качества
Издательства Томского политехнического университета сертифицирована
NATIONAL QUALITY ASSURANCE по стандарту BS EN ISO 9001:2008

ИЗДАТЕЛЬСТВО ТПУ. 634050, г. Томск, пр. Ленина, 30
Тел./факс: 8(3822)56-35-35, www.tpu.ru