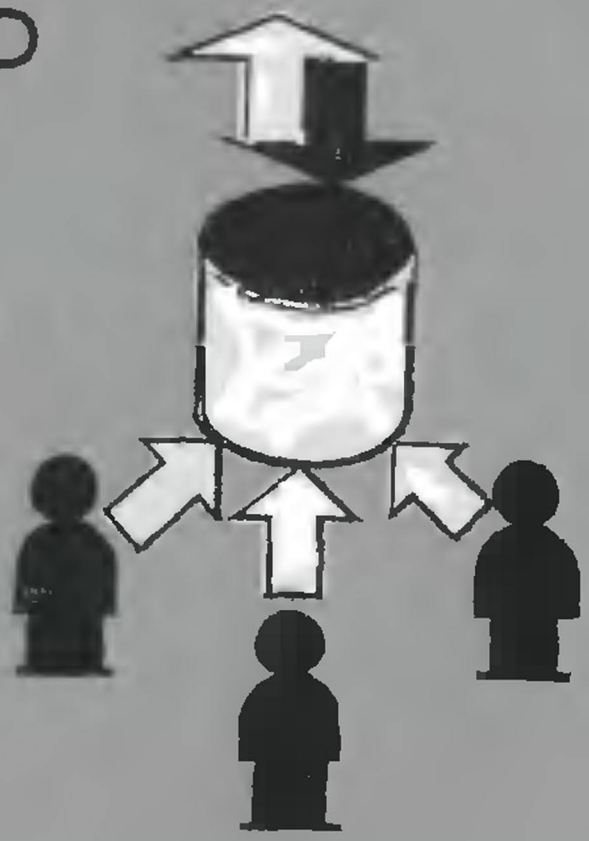
Т. ТИОРИ
ДЖ. ФРАЙ

ПРОЕКТИРОВАНИЕ СТРУКТУР БАЗ ДАННЫХ



ПРОЕКТИРОВАНИЕ СТРУКТУР БАЗ ДАННЫХ

DESIGN OF DATABASE STRUCTURES

Toby J. TEOREY
James P. FRY

The
University
of
Michigan

Prentice-Hall, Inc.,
Englewood Cliffs 1982

Т. ТИОРИ
ДЖ. ФРАЙ

ПРОЕКТИРОВАНИЕ СТРУКТУР БАЗ ДАННЫХ

В ДВУХ КНИГАХ

1

Перевод с английского
А. И. РОГОВСКОГО, канд. техн. наук В. И. ЧУЧКИНА

под редакцией
канд. техн. наук В. И. СКВОРЦОВА

МОСКВА · МИР · 1985

ББК 32.973.2
Т 32
УДК 681.3

Тиори Т., Фрай Дж.

Т 32 Проектирование структур баз данных: В 2-х кн. Кн. 1. Пер. с англ. — М.: Мир, 1985. — 287 с., ил.

Труд американских ученых посвящен проблеме проектирования баз данных. В русском переводе выпускается в 2-х книгах.

В книге 1 подробно описываются общие принципы, методы и средства проектирования баз данных на различных уровнях: концептуальном, реализации и физическом. Подход одинаково приемлем для сетевых, реляционных и иерархических баз данных. Большое внимание уделено обоснованию выбора эффективных проектных решений.

Для специалистов в области вычислительной техники.

Т $\frac{2405000000\text{-}197}{041(01)\text{-}85}$ 166-85, ч. 1

ББК 32.973.2
6Ф7.3

Редакция литературы по информатике и электронике

Предисловие редактора перевода

Проблемы проектирования автоматизированных информационных систем, основанных на концепции баз данных, являются в настоящее время объектом все возрастающего интереса широкого круга специалистов в области обработки данных. Согласно этой концепции, ядром информационной системы становятся данные, которые должны быть определенным образом организованы с целью адекватного отображения непрерывно меняющегося реального мира и эффективного удовлетворения информационных нужд пользователей системы. Проектирование баз данных, отвечающих указанной цели, требует применения определенной методологии. Эта методология может рассматриваться как совокупность методов и средств, последовательное применение которых обеспечивает разработку проекта баз данных, удовлетворяющего заданным целям проектирования. За последние несколько лет появилось большое количество публикаций, в которых излагаются теоретические и практические вопросы, касающиеся хотя и несомненно важных, но, как правило, отдельных аспектов проектирования баз данных. Исключение составляет книга Дж. Хаббарда «Автоматизированное проектирование баз данных» (пер. с англ. — М.: Мир, 1984). Однако ее основная направленность на проектирование баз данных, поддерживаемых СУБД типа IMS фирмы ИБМ, не позволила рассмотреть ряд проблем, возникающих на различных этапах проектирования.

Предлагаемая читателям монография является одной из первых, в которой с единых методологических позиций в рамках единой терминологии и столь подробно описаны общие принципы, методы и средства проектирования баз данных на концептуальном, логическом и физическом уровнях проектирования. Несомненным достоинством данной монографии является то, что авторы не ограничиваются изложением методологии поэтапного проектирования баз данных и примеров, иллюстрирующих ее применение, а детально рассматривают широкий спектр вопросов концептуального, логического и физического проектирования баз данных, уделяя при этом большое внимание обоснованию выбора эффективных проектных решений на всех этапах. Предложенная авторами методология основывается на общепринятых принципах проектирования баз данных, носит явно выраженную практическую направленность и доступна широкому кругу специалистов, занимающихся решением практических вопросов проектирования банков данных.

Отмечая особенности процесса проектирования баз данных, авторы впервые подчеркивают важность экспертной оценки проекта баз данных и описывает процедуру ее проведения. Заслуживает внимания тот факт, что с целью обеспечения адаптивности и гибкости баз данных предлагается разделять информацию, получаемую на этапе анализа предметной области, на две категории — информацию, описывающую естественные концептуальные связи между данными, и информацию, описывающую связи между данными, возникающие при обработке приложений пользователей. Влияние этих категорий информации на результат проектирования структуры баз данных различно: использование первой из них обеспечивает гибкость и адаптивность базы данных, использование второй — эффективность.

Этап концептуального проектирования авторы представляют как описание информационных требований и их синтез в первоначальный проект базы данных. Рассматриваются два подхода — объектное представление и модели-

рование сущностей — соответствующих нисходящей и восходящей методологиям проектирования. В первом из них большое внимание уделяется одному из основных вопросов концептуального проектирования — интеграции понятий. В связи с этим вводятся и подробно анализируются понятия абстракций агрегации и обобщения. Второй подход основан на введении понятия сущности, являющейся основным конструктивным элементом концептуальной модели предметной области, представляемой в виде диаграмм типа «сущность-связь».

Для обозначения этапа, связанного с построением СУБД-ориентированной схемы базы данных, авторы вводят термин, отличный от используемого в отечественной литературе, называя его этапом проектирования реализации. Предложенная последовательность шагов проектирования ориентирована на получение СУБД-ориентированной схемы базы данных, удовлетворяющей широкому диапазону предъявляемых к ней требований. Для оценки эффективности проектных решений используется система количественных оценок.

К числу достоинств данной монографии следует отнести также и то, что в ней дан сравнительный анализ эффективности широко используемых в настоящее время при проектировании физической организации баз данных первичных и вторичных методов доступа. Большой интерес представляет та часть книги, в которой рассмотрены способы и алгоритмы проектирования структур записей и методы адресации и поиска данных.

Авторы не ограничиваются только вопросами проектирования баз данных, но уделяют также внимание проблемам реорганизации баз данных, архитектуре систем управления распределенными базами данных и стратегиям распределения данных.

Не вызывает сомнений, что данная монография интересна не только специалистам в области обработки данных, но и разработчикам системного и прикладного программного обеспечения информационных систем. Много полезного из нее почерпнут преподаватели, аспиранты и студенты, специализирующиеся в области организации данных в вычислительных системах.

Перевод монографии выполнен А. И. Роговским (гл. 1—6, приложение А), кандидатом технических наук В. И. Чучкиным (гл. 7—12, приложение Б), Л. В. Осиповой (гл. 13—16, толковый словарь) и кандидатом технических наук М. Н. Петуховым (гл. 17, 18).

В. И. Скворцов

Посвящается Джульетте и Розмари

Предисловие

Цель этой книги состоит в том, чтобы выработать согласованную основу для многоуровневого проектирования баз данных, определить приемлемую методологию и описать общие принципы, средства и методы проектирования баз данных на каждом из уровней. Используется нисходящая методология проектирования, позволяющая на каждом шагу проводить оценку проектных решений и их пересмотр в случае необходимости. При этом на всех этапах проектирования применяется единая терминология. Задачей непосредственно процесса проектирования является построение структуры базы данных, которая адекватно отображает описываемую проблемную среду и может быть эффективно реализована с помощью существующих технических и программных средств. В то же время в процессе проектирования (в первую очередь на этапе концептуального проектирования) обеспечивается, насколько это возможно, независимость от конкретных системных средств.

Начиная с момента формулирования требований к системе баз данных и до определения спецификаций логической и физической структур баз данных, отвечающих этим требованиям, методология иллюстрируется подробными примерами. Подход одинаково приемлем для сетевых, реляционных и иерархических систем баз данных.

В процессе проектирования базы данных большинство шагов может быть выполнено с применением методов моделирования на ЭВМ и других средств, например, таких, как программный анализ требований. Границы диапазона этих средств могут меняться от способов разработки документации до сложных современных методологий проектирования. Наряду с широким использованием математических методов (имитационное моделирование, теория графов, оптимизация) в процессе проектирования находят применение и такие простые средства, как приближенные оценки значений предполагаемых характеристик эффективности функционирования баз данных. Рекомендации по применению тех или иных средств приводятся на каждом этапе проектирования.

Хотя главное внимание уделяется проектированию структуры баз данных, исследуется также ее влияние на эффективность выполнения программ. Вопросы целостности, восстановления и безопасности данных также решаются в сопоставлении с показателями эффективности.

Книга предназначена для специалистов в области анализа баз данных, проектировщиков, администраторов баз данных и прикладных программистов. Она может быть также использована для подготовки односеместрового курса лекций для студентов старших курсов или аспирантов первого года обучения. Материал книги носит учебный характер, он снабжен упражнениями и библиографией, терминологическим словарем, предметным указателем и списком сокращений. Предполагается, что читатель знаком со структурами данных, методами поиска и упорядочения информации и характеристиками основных систем управления базами данных. В книге нашли отражение как теоретические, так и практические вопросы, при этом постоянное внимание уделяется их взаимосвязи.

Порядок следования глав соответствует этапам жизненного цикла системы баз данных (гл. 1) и представляет материал в той последовательности, которой должен обычно следовать процесс проектирования. Часть I (гл. 1—3) является общей для понимания остальных глав, однако изложение каждого

этапа проектирования представлено в законченной форме и позволяет изучать их в отдельности. Например, краткий курс по физическому проектированию может включать гл. 1—3 и 9—16. В гл. 18 представлен обзор результатов проектирования баз данных в распределенных системах с целью постановки проблем для будущих исследований.

Анализ требований и концептуальное проектирование иллюстрируются одним общим примером (гл. 3 и 6). Проектирование реализации иллюстрируется отдельным примером (гл. 8), а обсуждение вопросов физического проектирования содержит небольшие отдельные примеры для демонстрации специфических приемов проектирования. Приведенные примеры показывают, что проектирование баз данных может быть выполнено вручную, однако они также указывают, где могут быть эффективно использованы средства с применением ЭВМ.

Энн Арбор

Тоби Дж. Тиори
Джеймс П. Фрай

Благодарности

Мы хотели бы выразить нашу признательность за рецензию рукописи и большинства примеров, которую выполнили Лерри Браун, Дженис Бубенко, Боб Куртис, Джефри Хоффер, Алан Мертен, Шамкант Навате, Донна Ранд, Марио Школьник, Деннис Северанс, Диана Смит, Джон Смит, Дик Вольц и др. В книгу вошла бо́льшая часть материалов курса, прочитанного на Летней технической конференции Мичиганского университета. На содержание книги оказали также большое влияние работы Симпозиума по проектированию баз данных (Нью-Йорк, 1978) и Рабочего совещания по проектированию баз данных (Нью-Орлеан, 1978). Мы признательны нашим коллегам: Сакти Хошу, Девиду Джефферсону, Полу Джонсу, Бобу Тейлору, Винсенту Ламу и Бингу Яо — за вклад в наш подход.

Многоопытные сотрудники Группы исследований информационных систем Мичиганского университета Эд Бирс, Девид Чен, Рик Кобб, К. Сундар Дас, Марк Депп, Дон Де-Смит, Альберто Гарсиа, Эрик Кинцер, Крис Мерил, Дон Новак, Лью Оберлендер, Дон Свортвут и Майк Виленс оказывали нам помощь в части, касающейся разработки программного обеспечения, оценки средств проектирования, а также в просмотре рукописи.

Мы благодарны Конни Аллен, Пэм Доуни, Кэрол Дан и Изен Гоулдинг за их помощь при подготовке рукописи.

Мы высоко оцениваем возможность пользоваться библиотекой и техническими средствами, предоставленными нам Отделом управляющих информационных систем Аризонского университета под руководством Джей Ньюнамейкер.

Упражнения Б5, Б6 и Б7 предложены Деннис Северанс из Мичиганского университета. Упражнение Б8 было предложено Уильямом Уайлером из организации «Голубой Крест/Голубой Щит», шт. Массачусетс, а проектное задание 3 в приложении А разработали Пол Хельман и Мерилин Мантей из Мичиганского университета. Проектное задание 2 заимствовано из работы [71].

Наконец, мы хотели бы выразить признательность за поддержку и поощрение со стороны наших семей, включая Юнис Л. и Томаса Ф. Тиори, Бесси Мей и Роберта У. Тиори, а также Энн Дж. и Палмера Е. Фрай.

Часть I

Введение

Глава 1. Системы баз данных

Проектирование интегрированных баз данных представляет собой трудоемкий, длительный и во многих случаях неформализуемый процесс. Это комплексная проблема, касающаяся в конечном счете не только вопросов обработки данных, но и всей организации в целом. Качество полученной в итоге структуры базы данных определяется общей методологией проектирования, используемыми на каждом шагу техническими приемами проектирования, обоснованностью информационных требований, а также готовностью людских и материальных ресурсов организации к проведению этой работы.

Построение основ методологии проектирования баз данных начинается в этой главе с описания стандартной терминологии, употребляемой в книге. Кроме того, мы рассмотрим некоторые из основных концепций данных с точки зрения их использования в организации, их описания в прикладных программах и системах управления базами данных, а также с точки зрения их хранения в вычислительных системах.

1.1. Данные и управление базами данных

Данные, согласно определению Уэбстера, есть «некоторый факт; то, на чем основан вывод или любая интеллектуальная система» [334]. Первичными компонентами данных являются цифры и символы естественного языка или их кодированное представление в виде строки двоичных битов. Наименьшей семантически значимой поименованной единицей данных является *элемент данных.* Совокупность взаимосвязанных элементов данных, рассматриваемая в прикладной программе как целое, называется *логической записью,* а набор записей одного типа — *файлом.* Хранение файлов может быть организовано с помощью электронных вычислительных машин (ЭВМ). Между логическими записями, отражающими точку зрения прикладного программиста, и хранимыми записями, отражающими способ хранения в устройстве и точку зрения системного программиста,

существует различие. Это будет постоянно подчеркиваться в последующем изложении.

Продолжающийся значительный рост использования ЭВМ в различных областях промышленности, в управлении и научных исследованиях привел к автоматизации обработки огромнейшего количества данных. В конце 1950-х — начале 1960-х годов коммерческие, правительственные и другие организации начали накапливать и хранить данные в виде файлов, доступных ЭВМ. По мере возникновения новых потребностей в хранении или обработке данных создавались все новые и новые файлы. Отдельные группы внутри организаций разрабатывали собственные приложения, накапливая и поддерживая необходимые данные в частных файлах. Прикладные программы разрабатывались с учетом имеющихся файлов данных и наоборот, при этом много информации неявным образом содержалось в организации взаимосвязи между программой и файлом.

Организации постепенно осознавали необходимость централизации управления данными и приложениями. Понимание этой необходимости приходило различными путями. Во-первых, руководители высшего уровня очень быстро обнаружили, что требуемую для принятия решений информацию не очень легко получить. Чтобы выполнить запрос на информацию, необходимо было написать прикладную программу, способную обработать несколько частных файлов, каждый со своим собственным форматом. Руководитель часто был вынужден отказываться от запроса из-за того, что за время, в течение которого информация могла быть получена, она становилась бесполезной, или из-за того, что ценность информации не соответствовала затратам на ее получение. Во-вторых, принятие решений сдерживалось отсутствием целостности данных. Отчеты, полученные с помощью ЭВМ, имели много расхождений по той причине, что логически идентичные элементы данных имели разные значения. Это было вызвано дублированием данных в частных файлах и непоследовательным проведением их обновления. Имелось также дублирование усилий по накоплению данных и выдаче отчетов. Вычислительные ресурсы, память и машинное время расходовались нерационально. И наконец, в-третьих, развитие технологии подошло к такому уровню, когда стало возможным проектировать, накапливать и обрабатывать большие наборы данных в вычислительной среде. В конце концов организации осознали значимость такого ресурса, как данные, и необходимость централизованного управления ими.

Таким образом, понятие базы данных было сформулировано только в недавние годы. *База данных* может быть определена как совокупность предназначенных для машинной обработки данных, которая служит для удовлетворения нужд многих поль-

зователей в рамках одной или нескольких организаций. Ключевым моментом является то, что база данных, будучи интегрированным средством, предназначена для использования всеми членами организации, которым необходима информация, содержащаяся в базе данных. Информация уже не скрыта в сочетании «файл-программа», она хранится явным образом в базе данных, которая может включать много различных типов логических записей. База данных ориентирована на интегрированные требования, а не на одну программу, как было с частными файлами данных.

Наиболее широко база данных используется в управленческой деятельности благодаря следующим свойствам:

1. *Скорость.* Вычислительная техника позволяет осуществить оперативный доступ к информации.
2. *Полная доступность.* Вся информация, содержащаяся в базе данных, доступна для использования.
3. *Гибкость.* Имеется возможность получать ответы на те вопросы, которые ранее оставались без ответа. Изменения в базу данных вносятся сравнительно легко.
4. *Целостность.* Уменьшилось дублирование данных, появилась возможность упорядочить проведение обновления, что привело к согласованности данных.

Однако наличие базы данных само по себе не разрешает полностью проблем организации в области обработки данных и принятия решений. Управление базой данных, являющейся достоянием многих пользователей внутри организации, должно осуществляться с пользой для всей организации и с точки зрения организации в целом, а не отдельных пользователей. Пользователи, естественно, стремятся развивать приложения для своих собственных целей, вовсе не интересуясь тем, какое влияние оказывают эти новые приложения на других пользователей. Без централизованного управления базой данных ее полезность со временем снижается.

Для решения проблемы регулирования и управления базами данных были развиты две новые концепции. Во-первых, программное обеспечение развивалось в направлении, обеспечивающем поддержание общего интерфейса между всеми пользователями и интегрированной базой данных. Общий интерфейс способствует обеспечению секретности и целостности данных. Пользователи не могут хранить информацию независимым образом, они должны использовать и обновлять данные в соответствии с требованиями организации. Программное обеспечение, известное как система управления базами данных, позволяет осуществлять контроль данных с помощью ЭВМ. *Система управления базами данных* (СУБД) представляет собой обобщенный инструмент для манипулирования базами данных.

Это осуществляется с помощью специального программного обеспечения для удовлетворения запросов, поддержания и анализа данных. Интерфейс СУБД с различными классами пользователей обычно обеспечивается широким диапазоном языков. В нем также предусматриваются соответствующие средства для проектирования и использования баз данных. СУБД может быть приобретена организацией через фирму, торгующую программным обеспечением, или разработана собственными силами.

Другой концепцией является концепция *администратора базы данных* (АБД). Под этим понятием подразумевается лицо (или группа лиц, возможно, целое штатное подразделение), на которое возложено управление средствами базы данных организации. АБД должен быть энергичной и способной личностью, организатором по призванию, желательно с техническим уклоном. Он должен уметь поддерживать взаимосвязи как с руководством высшего уровня, так и с пользователями, обрабатывающими данные, а также руководить штатом технических специалистов. Этот штат должен включать лиц, имеющих опыт работы в таких областях, как программное обеспечение СУБД, операционные системы, техническое обеспечение ЭВМ, прикладное программирование, системное проектирование. Важно также, чтобы в этот штат были включены лица, имеющие представление об организации и ее информационных потребностях. Персонал АБД должен уметь поддерживать хорошие отношения с другими группами, не входящими в отдел обработки данных.

Место АБД было определено тогда, когда организации осознали необходимость централизованного управления ресурсами данных, обработкой данных и другими аспектами, связанными с базой данных. Группы пользователей и отдельные пользователи должны обслуживаться всеми средствами исходя из целей и возможностей организации в целом. Поэтому АБД является ответственным за анализ потребностей пользователей, проектирование базы данных, ее внедрение, обновление или, если необходимо, реорганизацию базы данных, а также за консультацию и обучение пользователей [52, 102, 181, 208].

1.2. Уровни представления данных

В настоящее время признаны по меньшей мере три уровня абстракции для определения структуры базы данных: концептуальный (с позиции администратора предприятия), уровень реализации (с позиций прикладного программиста или конечного пользователя) и физический (с позиций системного программиста или системного аналитика). Существуют различия

во мнениях относительно необходимости определения бо́льшего числа уровней абстракции [182, 275], а также относительно адекватности представления данных только по вертикали [8], однако названные три основных уровня общепризнаны в большинстве дискуссий по представлениям данных. *Модель данных* — это представление данных и их взаимосвязей, описывающее понятия проблемной среды. Модели данных используются как для концептуального представления данных, так и представления реализации, но не физического представления, хотя различия между этими уровнями часто являются незначительными.

Концептуальное представление. На рис. 1.1 изображены три основных уровня абстракции и их первичные компоненты. Структура данных на концептуальном уровне называется концептуальной схемой или информационной структурой. Она является проблемно-ориентированной и системно-независимой, т. е. независимой от конкретной СУБД, операционной системы (ОС) и аппаратного обеспечения ЭВМ. В терминах трехуровневой модели данных ANSI/SPARC, которая является моделью оперирования данными (в противоположность нашей модели проектирования), понятия концептуальной схемы практически эквивалентны [8] (рис. 1.2). В действительности же существует много различных мнений относительно того, как должна выглядеть концептуальная схема и что она должна представлять; окончательного решения по этому вопросу пока еще нет. Предлагаемый подход по существу является более прагматичным, другие отличия будут обсуждены в гл. 4—6.

Концептуальная структура (или схема) состоит из: основных элементарных данных проблемной области (личности, факты), называемых *сущностями*; элементарных данных, описывающих сущности и называемых *атрибутами*; ассоциаций между экземплярами элементарных данных, называемых *связями*. Рис. 1.3 изображает это развернутое определение и иллюстрирует альтернативные решения проблемы в рамках концептуальной схемы. Обычно различают три типа (бинарных) связей между экземплярами сущностей:

- *Один-к-одному* (*1 : 1*). Покупатель имеет одну фамилию. Свойство единственности существует в обоих направлениях.
- *Один-ко-многим* (*1 : n*). Торговый агент обслуживает более одного покупателя, но каждый покупатель обслуживается одним торговым агентом.
- *Много-ко-многим* (*m : n*). Заказ состоит из многих товаров, а каждый товар может быть заказан многими покупателями.

Связи более высоких порядков обсуждались Коддом [85]. Во многих случаях они могут быть представлены в виде нескольких бинарных связей, но только при некоторых ограниче-

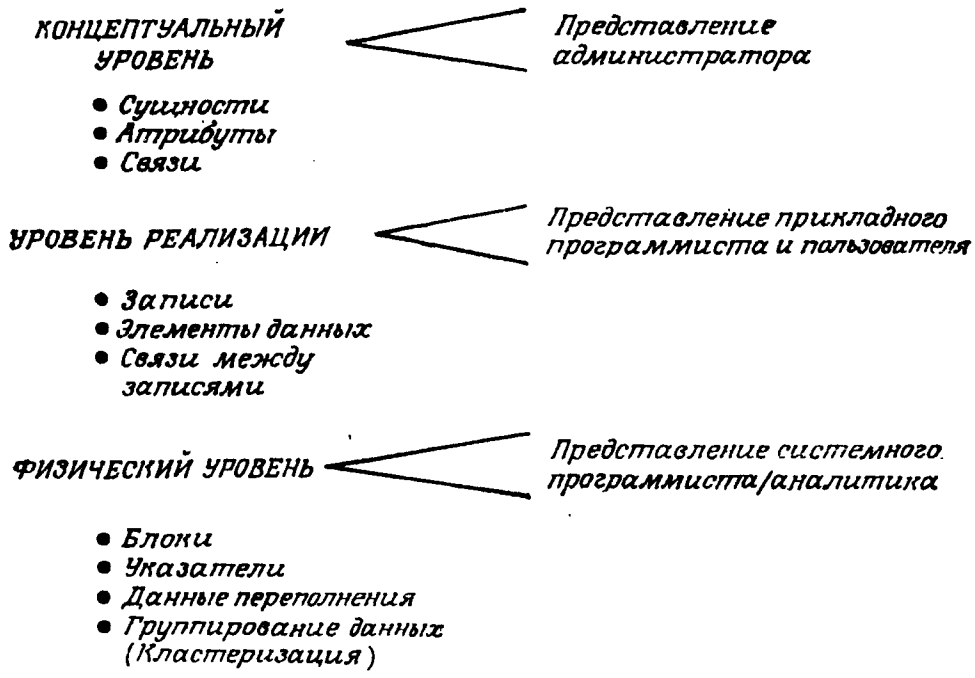


Рис. 1.1. Уровни абстракции данных.

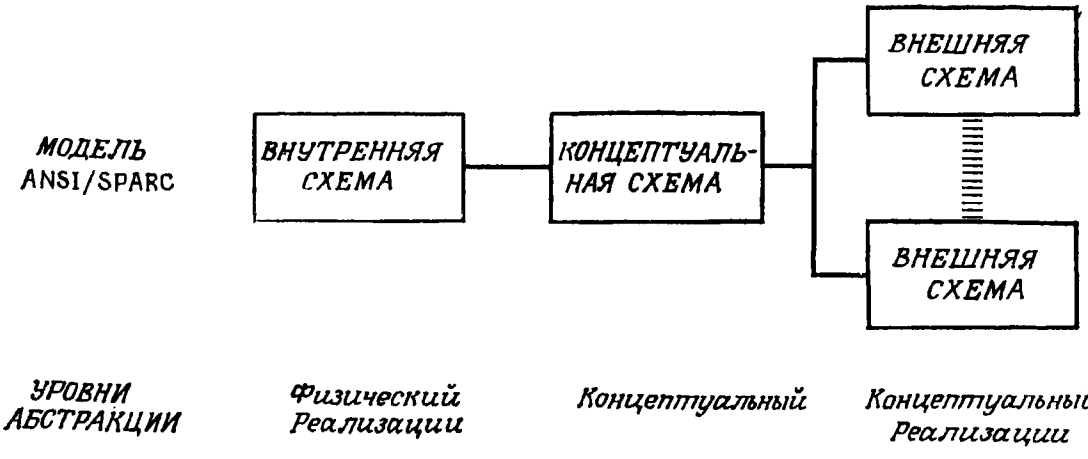


Рис. 1.2. Соответствие между моделью ANSI/SPARC и уровнями абстракции.

ниях. Большинство же связей проблемной области могут быть представлены простыми или составными бинарными связями.

Концептуальная схема должна поддерживать согласованность связей в пределах уровня детализации и временных ограничений. В вышеприведенном примере для связи типа $m:n$ подразумевается, что в соответствии с принятым в нем уровнем детализации значением родового термина «товар» является определенный тип товара, в то время как значением термина «тип товара» является конкретный товар определенного типа, отличающийся от других товаров порядковым номером. Заказ, состоящий из физических товаров, соотносится с товарами как

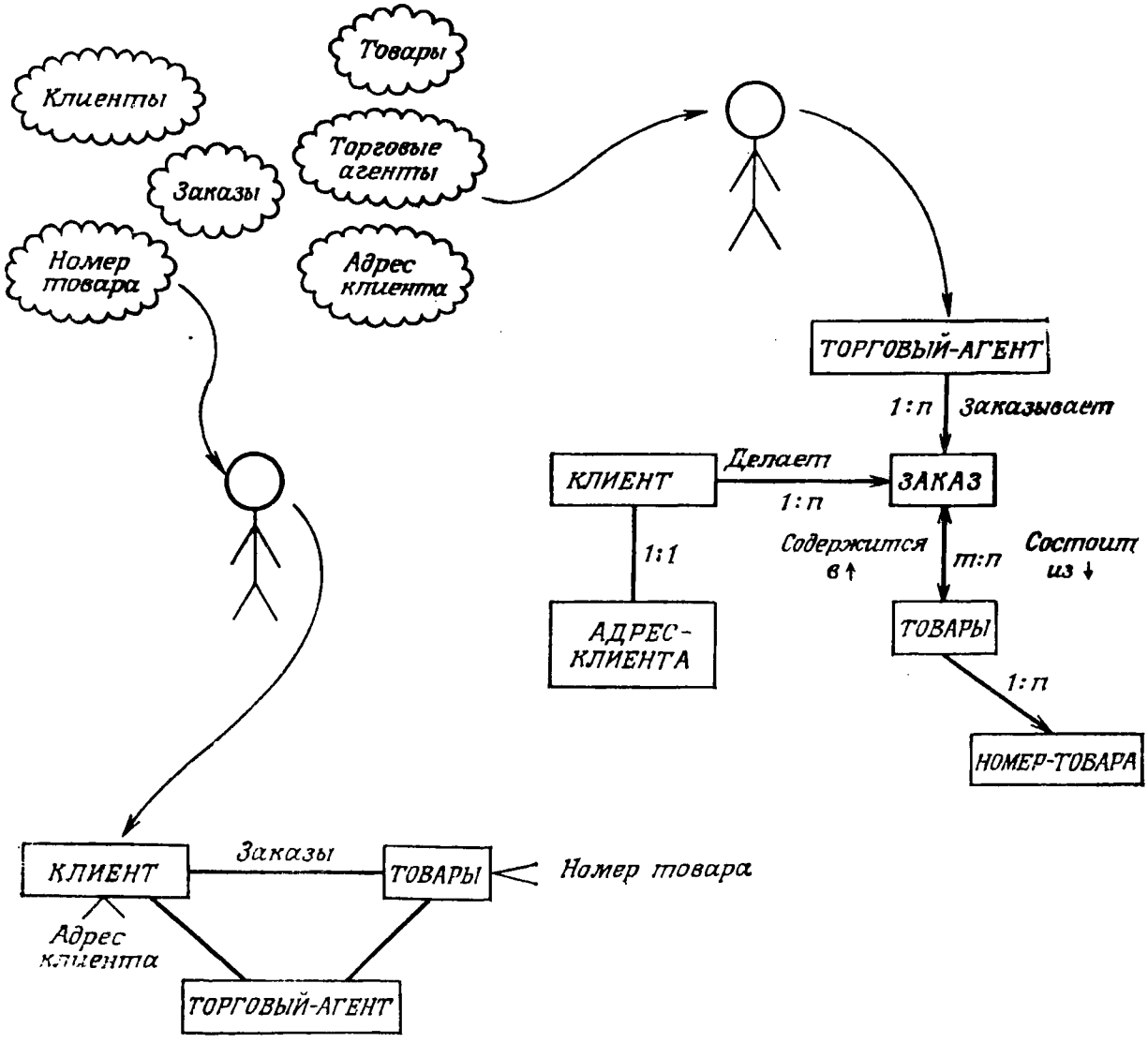


Рис. 1.3. Две концептуальные схемы для представления одной и той же проблемы. Очевидны различия в подходе и представлении данных.

1 : m, а не m : n. Согласованность во времени требует ответа на вопрос, сохраняется ли данная связь только в текущий момент времени, или она существует постоянно. В примере подразумевается, что связь между торговым агентом и покупателем является временной. Если она рассматривается в течение длительного периода времени, то она может стать связью типа m : n. Эта проблема встречается, например, в таких связях, как «супруги» или «название работы», тип которых может меняться во времени.

Представление реализации. Представление реализации состоит из (логических) записей, составляющих их элементов данных и взаимосвязей записей. Наиболее часто применяются три модели данных: иерархическая, сетевая и реляционная.

Иерархическое представление концептуальной схемы (рис. 1.3) показано на рис. 1.4. Оно состоит из нескольких типов записей, один из которых определен как корневой, или входной, тип записи. Каждый тип записи может состоять из нескольких элементарных типов (или полей), некоторые из них могут являться ключами, однозначно идентифицирующими каждую запись. Между типами записей в иерархии определена связь «один-ко-многим» (иногда «один-к-одному»), где запись, соответствующая элементу «один» указанной связи, определяется как исходная, а соответствующая элементу «много» — как порожденная.

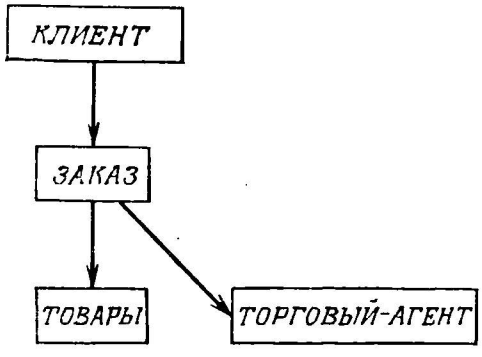


Рис. 1.4. Иерархическая структура базы данных.

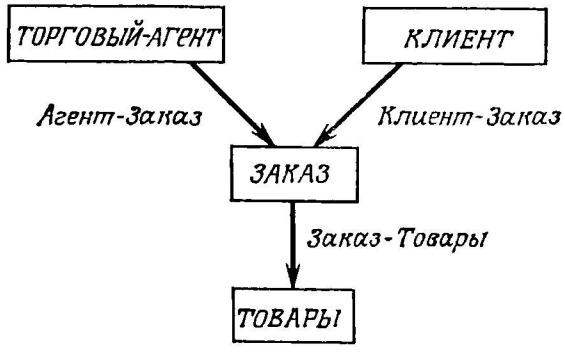


Рис. 1.5. Сетевая структура базы данных.

Запись может быть порожденной только в одной связи, это означает, что для каждой записи может существовать только одна исходная запись. Однако каждая запись может быть исходной во многих связях. Корневая запись может быть только исходной. Обычно между двумя типами записей может быть только одна связь. Иерархическая модель данных используется в системах IMS и System 2000, хотя соответствующие физические структуры этих систем совершенно различны.

Сетевая модель изображена на рис. 1.5. Она подобна иерархической модели, но является более общей в том смысле, что любая запись может входить в любое число именованных связей как исходная или порожденная или как то и другое. Поэтому здесь нет корневого узла, так как любая запись может быть определена как точка входа. На рис. 1.6 показано несколько типов возможных связей, включая множественные связи между двумя типами записей. Сетевая модель реализована в ряде систем, основанных на модели данных CODASYL (IDMS, DMS 1100, DBMS-10, IDS-II и т. д.), а также в других системах (TOTAL, DMSII). Еще несколько систем, относящихся к категории систем с инвертированными файлами, могут косвенно представлять сетевую концепцию (Model 204, DATACOM/DB, ADABAS и т. д.).

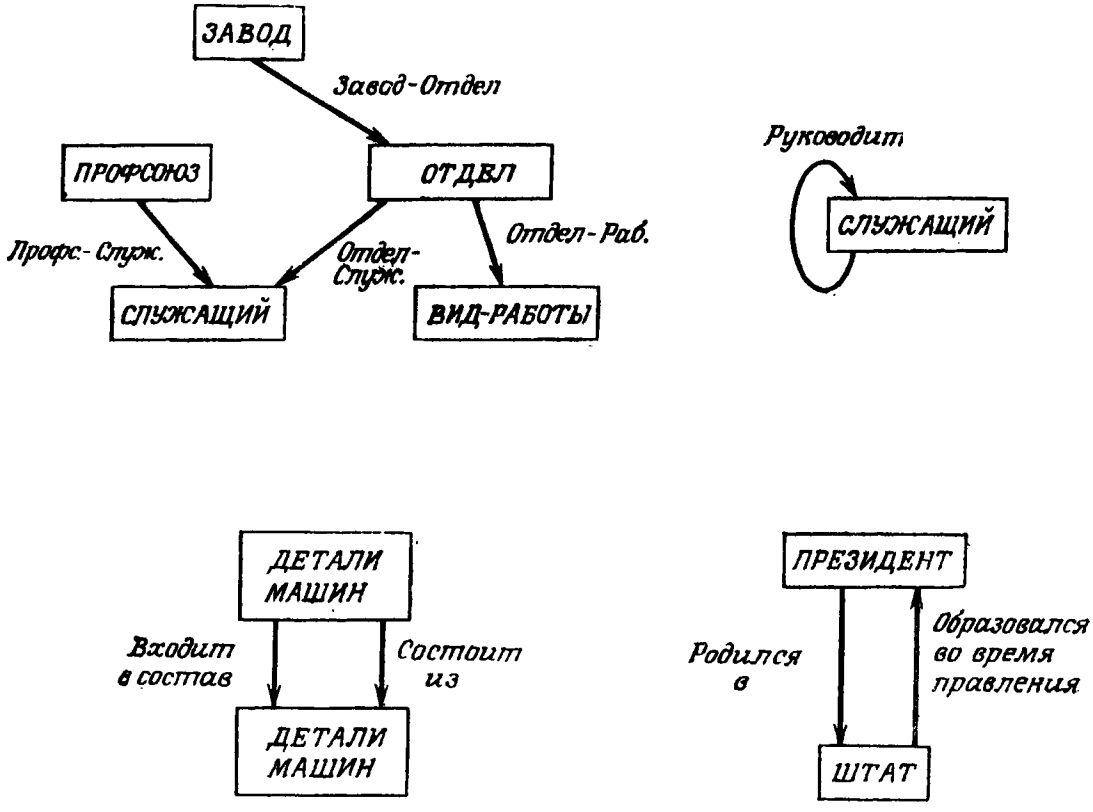
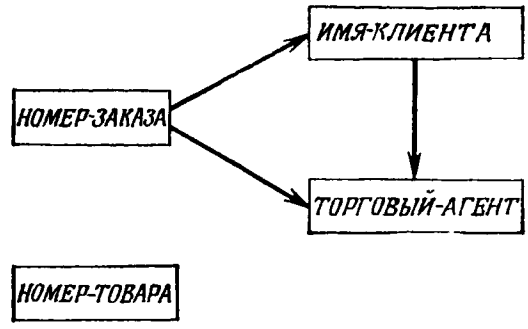


Рис. 1.6. Конфигурации сетевой базы данных.

Реляционная модель данных (рис. 1.7) является концепцией, которая легка для понимания и имеет много возможных приложений. Реляционная база данных состоит из набора «плоских файлов», или таблиц, называемых *отношениями*, которые в свою очередь включают *кортежи* (экземпляры записей) и *атрибуты* (элементарные типы), значения которых выбираются из простого *домена*. Связи между отношениями неявно определены на перекрывающихся доменах. Используя общую информацию, например такую, как уникальный ключ исходной записи для поиска ее порожденных записей, можно проектировать преобразования между основными моделями данных. В реляционных базах данных связь типа «исходный — порожденный» реализуется путем определения домена в порожденной записи, содержащей ключевой элементарный тип исходной записи. Однако реляционная модель предоставляет гораздо большие возможности по сравнению с простым преобразованием данных. Она обладает математическими свойствами, которые являются полезными при определении языков манипулирования данными, а также нормальных форм различных уровней. Нормализация отношений обеспечивает целостность и повышает эффективность использования базы данных в целом. На рис. 1.8 проил-



а

Клиент	ИМЯ-КЛИЕНТА	ТОРГОВЫЙ-АГЕНТ

Заказ	НОМЕР-ЗАКАЗА	ТОРГОВЫЙ-АГЕНТ

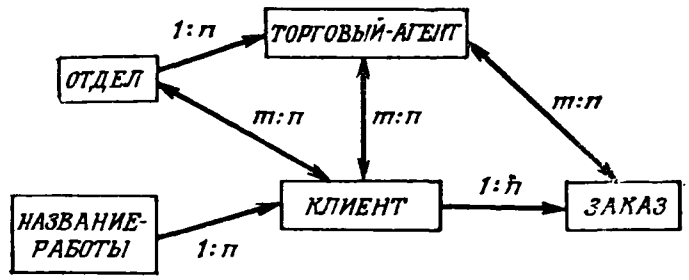
Товар	НОМЕР-ТОВАРА	НОМЕР-ЗАКАЗА

б

Рис. 1.7. Реляционная структура базы данных.
а — диаграмма функциональной зависимости (1НФ); *б* — отношения (3НФ).

люстрированы три основные нормальные формы, однако хорошо известны нормальные формы четвертого и даже более высоких порядков [113—115]. В настоящее время уже разработано много реляционных систем, и можно ожидать в скором времени, что некоторые из них потеснят на коммерческом рынке системы других типов.

Физическое представление. Фрагмент физической структуры показан на рис. 1.9. Основными компонентами физической базы данных являются физические блоки, хранимые записи, указатели, данные переполнения и промежутки между блоками. Взаимосвязи между хранимыми записями, возникающие в результате их группирования (объединения в кластеры) или использования индексных структур, могут также рассматриваться как часть физической структуры.



а

НОМЕР-ОТДЕЛА	ТОРГОВЫЙ-АГЕНТ	ИМЯ-КЛИЕНТА	НАЗВАНИЕ-РАБОТЫ	НОМЕР-ЗАКАЗА

Единственное отношение (файл)

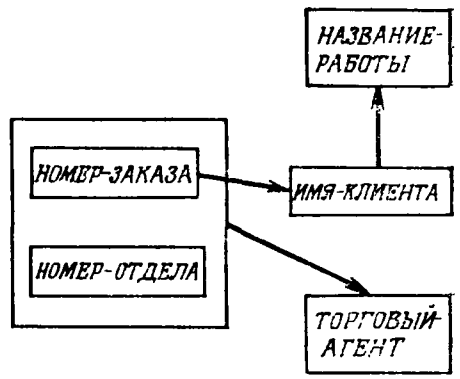
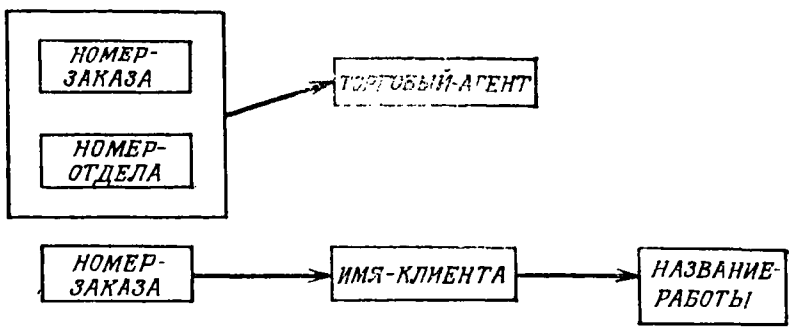
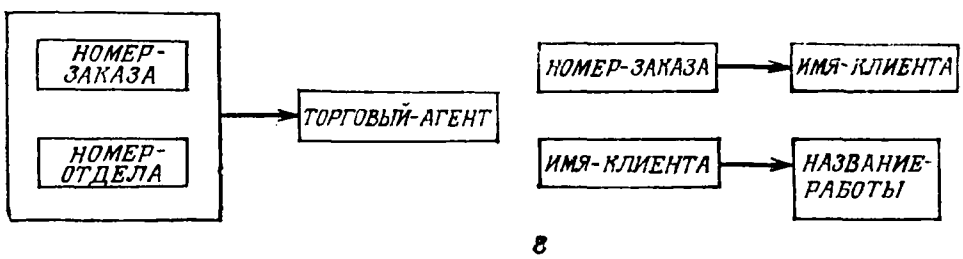


Диаграмма функциональной зависимости

б



в



г

Рис. 1.8. Первая, вторая и третья нормальные формы реляционных баз данных.

а — ненормализованная база данных — сетевая структура, включающая все возможные отношения, в том числе и избыточные; *б* — первая нормальная форма (1НФ); *в* — вторая нормальная форма (2НФ) — неполная зависимость исключена из 1НФ; *г* — третья нормальная форма (3НФ) — транзитивная зависимость исключена из 2НФ.

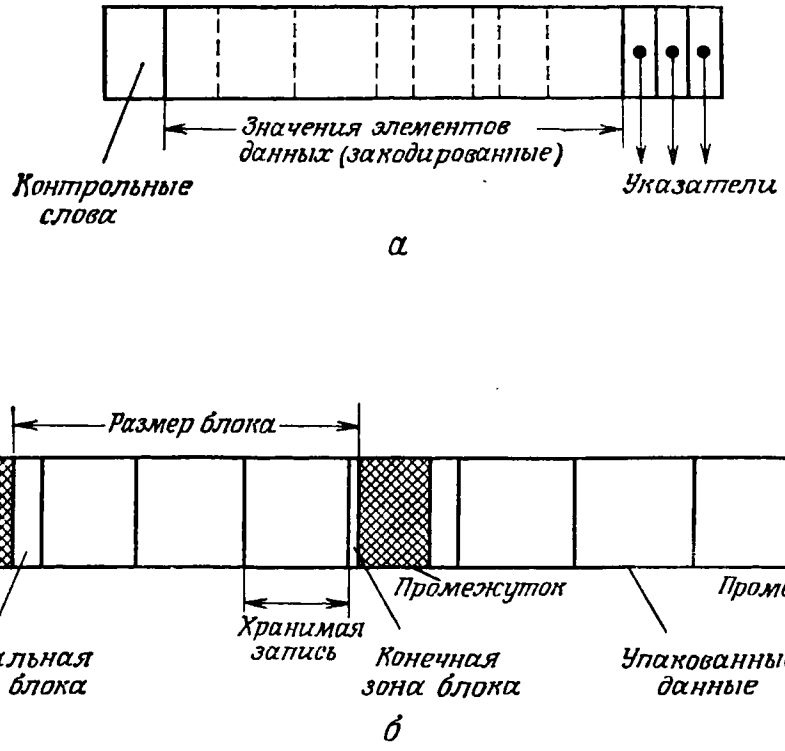


Рис. 1.9. Физическая структура базы данных.
а — хранимая запись; *б* — физические блоки.

1.3. Проблемы проектирования баз данных

Объединение программного обеспечения СУБД, прикладного программного обеспечения, реализованной базы данных, операционной системы и аппаратных средств в одну систему для информационного обслуживания пользователей известно под названием *система баз данных.* Хотя технология применения СУБД, операционных систем и прикладных программ хорошо известна, необходимо уделить внимание эффективному использованию этих средств с различными структурами баз данных. Так, главная проблема, стоящая перед администратором баз данных, заключается не в том, использовать ли конкретную технологию, а в том, как использовать ее наиболее эффективно [2]. Эта проблема может быть сформулирована в виде нескольких вопросов, возникающих в течение жизненного цикла приложения:

1. Что представляют собой требования пользователей и в какой форме они могут быть выражены?
2. Как эти требования могут быть преобразованы в эффективную структуру базы данных?
3. Как часто и каким образом структура базы данных должна перестраиваться в соответствии с новыми и/или изменяющимися требованиями?

Процесс разработки структуры базы данных в соответствии с требованиями пользователей называется *проектированием базы данных*. Многие практики утверждают, что процесс проектирования базы данных состоит по крайней мере из двух отдельных шагов: проектирования логической структуры базы данных, которая поддерживается СУБД и описывает представление пользователя о данных, и выбора физической структуры, которая включает представление данных или кодирование, методы доступа и физическое группирование (кластеризацию) данных. Однако общая структура процесса проектирования, за исключением логического и физического описания, пока еще не определена достаточно хорошо, и даже вопрос о разграничении логического и физического описаний требует тщательного обсуждения. Нам хотелось бы избежать этой путаницы путем более точного определения каждого этапа процесса проектирования.

В использующейся сегодня технологии проектирования баз данных можно обнаружить многое из тех методов, которые применялись при проектировании файлов с одним типом записи. Проектирование файлов ориентировано в первую очередь на конкретную прикладную программу, поскольку состав и структура данных зависит от приложения, использующего эти данные. Появление систем управления базами данных заставило пересмотреть акценты в подходе к проектированию как данных, так и программ. Концепция интегрированной базы данных, обслуживающей многих пользователей, явилась прямым результатом тех возможностей полного структурирования данных, которые предоставляет СУБД. Данные могут теперь рассматриваться как общий ресурс организации, а не как приложение к программе, и соответственно они должны иметь ориентацию на интегрированные требования, а не на отдельную программу.

Достижение приемлемого для всех пользователей уровня эксплуатационных характеристик базы данных является сложной задачей. Проектировщик баз данных должен постоянно помнить о стоимости различных услуг, предоставляемых пользователям одной или нескольких интегрированных баз данных. Ожидаемая экономия памяти и широкое использование баз данных в деятельности организации должны сопровождаться критическим анализом потенциального снижения качества обслуживания некоторых пользователей. Этого необходимо избегать, если возможно. Приемлемые эксплуатационные характеристики для всех пользователей — вот, что должно быть целью.

Другим аспектом функционирования базы данных является ее гибкость. Базы данных, тесно привязанные к текущим приложениям, могут иметь слишком ограниченную сферу применения в других подобных организациях. Быстрое изменение тре-

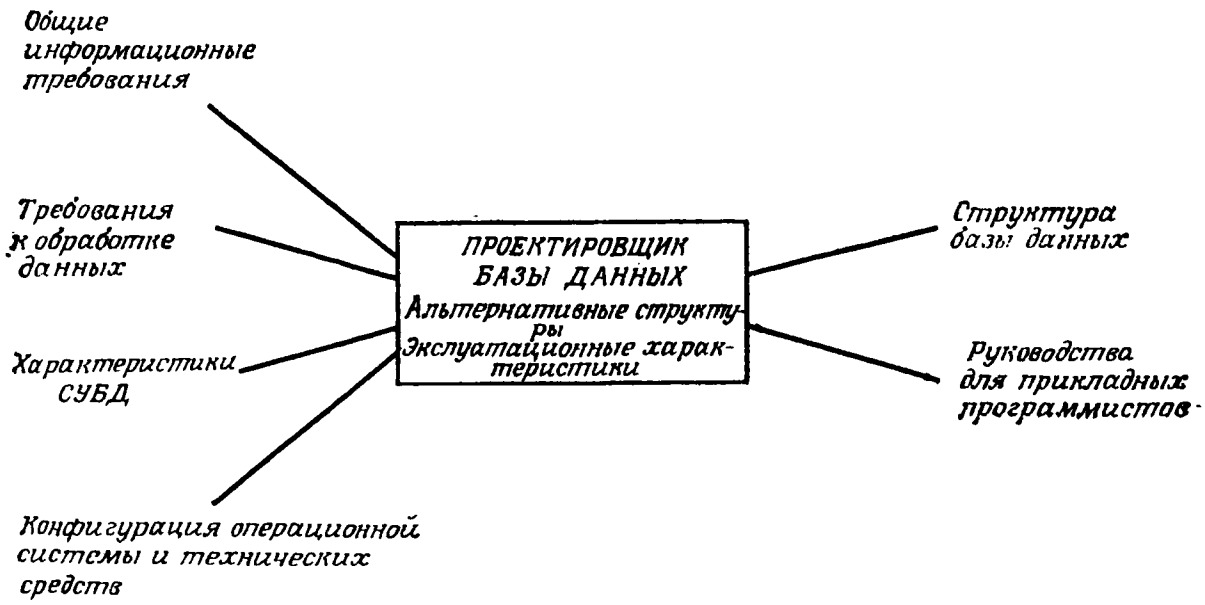


Рис. 1.10. Основные компоненты процесса проектирования базы данных.

бований и введение новых типов элементов данных могут иметь следствием повышение стоимости сопровождения программ, размножение временных файлов и сортировок, а также снижение производительности системы. Необходимо проведение осмысленного общего процесса проектирования базы данных для обеспечения как интеграции, так и гибкости.

Основные компоненты процесса проектирования базы данных показаны на рис. 1.10. Общие информационные требования включают формулирование целей системы баз данных, определение элементарных данных, включаемых в базу данных, и описание использования элементарных данных в организациях. Эти требования не связаны с каким-либо конкретным приложением, поэтому проектирование структуры баз данных, основанное на таких требованиях, необходимо применять в первую очередь для долгосрочных баз данных, обладающих адаптивностью к изменяющимся приложениям.

Требования к обработке данных состоят из трех характерных компонентов: специфические элементы данных, требуемые для каждого приложения, объем данных (число экземпляров элементов данных) и частоты обработки данных для каждого приложения (число обработок приложения в единицу времени). Каждый из этих компонентов является очень важным для отдельных фаз или этапов процесса проектирования базы данных. Проектировщик использует также и другие компоненты, такие, как характеристики (или ограничения) СУБД или конфигурация операционной системы и технических средств.

На проектировщика баз данных возложено также определение эксплуатационных характеристик и ограничений. Обычно

эксплуатационные ограничения рассматриваются как часть системных требований, а эксплуатационные характеристики, используемые проектировщиком, могут быть получены из этих ограничений. Типичные ограничения включают верхние пределы времени отклика на запросы, время восстановления после системных сбоев, специальные данные, необходимые для обеспечения требований безопасности и целостности. Особые эксплуатационные характеристики, используемые для оценки окончательной структуры, в дополнение к требованиям на время отклика могут включать стоимость обновления, хранения и реорганизации.

Двумя основными результатами процесса проектирования базы данных являются полная структура базы данных (включая так называемые логические и физические компоненты) и руководства для прикладных программистов, основанные на структуре базы данных и требованиях к обработке данных. В целом эти результаты могут рассматриваться как спецификации для реализации базы данных.

1.4. Жизненный цикл системы баз данных

Жизненный цикл системы баз данных представляет собой концепцию, в рамках которой полезно и удобно рассматривать развитие системы баз данных во времени. Эта концепция создает хорошие предпосылки для регламентирования функций администратора баз данных. Жизненный цикл системы баз данных делится на две отдельные фазы: фазу анализа и проектирования и фазу эксплуатации. В течение первой фазы происходит сбор требований пользователей и проектирование базы данных, в течение второй — машинная реализация и использование. С точки зрения проектировщика и пользователя можно детализировать содержание работ, выполняемых в течение этих фаз жизненного цикла системы баз данных. В этом смысле указанные фазы включают следующие этапы:

- Фаза анализа и проектирования
  1. Формулирование и анализ требований.
  2. Концептуальное проектирование.
  3. Проектирование реализации.
  4. Физическое проектирование.
- Фаза реализации и функционирования базы данных
  1. Реализация базы данных.
  2. Анализ функционирования и поддержка.
  3. Модификация и адаптация.

Каждый этап фаз жизненного цикла описан ниже. Здесь читатель может отметить, что при таком подходе игнорируются два важных вопроса проектирования: выбор системных технических средств и системного программного обеспечения. Эти вопросы выходят за рамки настоящего обсуждения.

1.4.1. Фаза анализа и проектирования

Формулирование и анализ требований являются, вероятно, наименее изученным, наиболее трудным и длительным по времени этапом процесса проектирования. Однако он является наиболее важным, так как на нем основано большинство последующих проектных решений, в результате чего он оказывает возрастающее влияние на другие этапы проектирования. Основной задачей является сбор требований, предъявляемых к содержанию и процессу обработки данных всеми известными и потенциальными пользователями базы данных. Анализ требований обеспечивает согласованность целей пользователей, а также согласованность их представлений об информационном потоке организации.

Концептуальное проектирование имеет целью построение независимой от СУБД информационной структуры путем объединения информационных требований пользователей. Результат концептуального проектирования называется также концептуальной схемой, поскольку она является представлением точки зрения пользователей на предметную область и не зависит ни от программного обеспечения СУБД, ни от технических решений.

Проектирование реализации состоит из двух компонентов: проектирование базы данных и проектирование программ. Структурой базы данных, полученной в результате проектирования реализации, является СУБД-ориентированное описание данных или схема, обычно выраженная в терминах языка описания данных. Она называется также логической структурой данных. Если описание данных включает физические параметры (например, области, размеры страниц и т. д.), выбор соответствующих значений этих характеристик откладывается до этапа физического проектирования. Проектирование программного обеспечения имеет целью создание структурированных программ, использующих включающий (базовый) язык программирования и язык манипулирования данными СУБД. Результатом являются функциональные спецификации программных модулей и набор возможных запросов к базе данных.

В недавнем прошлом второй и третий этапы обычно рассматривались как логическое проектирование, но, так как это часто приводило к путанице в литературе, мы исключаем термин «логическое проектирование» из нашего лексикона, заменяя его более точными терминами «концептуальное проектирование» и «проектирование реализации».

Физическое проектирование, так же как и проектирование реализации, состоит из двух компонентов: выбор физической структуры базы данных и окончательная отладка программных модулей, определенных на предыдущем этапе. Результатом

физического проектирования является полностью готовая к внедрению структура базы данных. Например, в системе CODASYL физическая структура базы данных может включать реализацию наборов с помощью списков указателей, смежные области для различных типов записей и методы доступа по ключам. Программный компонент подразумевает разработку структурированных программ на языке манипулирования данными для заданной логической структуры базы данных. Результатом этого этапа является набор реализуемых алгоритмов.

Мы кратко описали четыре этапа фазы анализа и проектирования. Необходимо четко представлять, что поверхностный анализ или недостаточно обоснованное решение, принятые на этой фазе, приведут к неправильной реализации и в конечном итоге к несоответствующей первоначальному замыслу системе.

1.4.2. Фаза реализации и функционирования базы данных

Реализация базы данных подразумевает создание базы данных и прикладных программ на основе результатов трех главных этапов проектирования базы данных, а также загрузку базы данных. Задача загрузки базы данных часто упускается из вида, однако она требует много усилий. Имеющиеся данные необходимо преобразовывать из существующей формы представления логической и физической структуры в новую форму, соответствующую результатам проектирования базы данных. Разработка прикладных программ тесно связана с выбором включающего (базового) языка программирования, с логической структурой базы данных и в меньшей степени с физической структурой. Целью этого этапа является создание надежных и эффективных программ доступа к базе данных, удовлетворяющих требованиям пользователей к обработке данных.

Анализ функционирования и поддержка используются для сбора (регистрации) и статистической обработки данных о функционировании системы. Эта информация позволяет выявить степень обоснованности требований пользователей, а также «узкие места» в процессе эксплуатации с целью пересмотра системы в будущем. Поддержка базы данных должна обеспечивать ее целостность и эффективное восстановление после сбоев.

Модернизация и адаптация предусматривают внесение в реализованный проект изменений, возникающих вследствие появления новых требований, анализа функционирования системы или анализа мнений пользователей о работе системы. Целью этого этапа является оптимизация функционирования существующей системы путем реорганизации базы данных и/или внесения изменений в программное обеспечение. Реорганизация

базы данных является широким понятием, используемым для описания любых изменений в логической или физической структуре базы данных. Изменения могут варьироваться в пределах от инвертирования связей (реструктурирования) до перестройки физической структуры (реформатирования).

И наконец, программная адаптация подразумевает процесс модификации прикладных программ в случае проведения реструктуризации базы данных. Обычно эта проблема возникает вследствие отсутствия логической независимости данных [121, 123, 125], а модификации включают новые последовательности команд манипулирования данными и замену переменных для подсчета числа логических записей в случае изменения коэффициента блокирования.

1.5. Заключение

Недостатки, присущие файловым системам, и отсутствие централизации в управлении данными привели к развитию концепции интегрированной базы данных. Вместе с более сложной структурой данных появилась необходимость в системах управления базами данных. Однако программное обеспечение СУБД предназначено только для определения, загрузки и доступа к данным, структура которых заранее установлена, и, следовательно, оно не связано с процессом проектирования базы данных.

Главные полномочия по проектированию, внедрению и сопровождению базы данных принадлежат администратору базы данных. Функция проектирования базы данных является сложным творческим процессом, который имеет много аспектов: методология общего подхода, многоуровневое представление данных, начиная от представления непрофессионального пользователя и кончая представлением системного программиста, аналитические средства проектирования и анализ структур баз данных, методы и средства документирования, используемые как в процессе проектирования, так и во время функционирования базы данных.

Организационной основой проектирования и анализа является жизненный цикл системы баз данных и этапы проектирования, реализации и реорганизации. Хотя основные фазы проектирования базы данных определены достаточно четко, выбор отдельных этапов, их последовательности, итеративных приближений и характеристик функционирования зависит от предпочтений проектировщика, позволяя в общем случае проявить необходимую гибкость в этом творческом процессе.

Глава 2. Процесс проектирования баз данных

2.1. Концепция методологии проектирования

Метод — это упорядоченная логическая процедура для выполнения определенной задачи. Методология — система методов, применяемых в научных исследованиях для обоснования результатов. В терминах баз данных *методология проектирования* может рассматриваться как совокупность инструментов и средств, применяемых для последовательной разработки проекта структуры баз данных [333]. Поскольку система баз данных состоит из программ и данных, методология проектирования баз данных рассматривается как неотъемлемая часть общей методологии систем программного обеспечения.

Что является целью «хорошей» методологии проектирования баз данных? Во-первых, должна быть создана приемлемая структура базы данных в разумные сроки и при разумных затратах. Приемлемой базой данных является такая, которая соответствует целям пользователей (т. е. высокоэффективная и адаптируемая к возникающим нуждам обработки, обладающая свойствами целостности, безопасности и т. д.), поддерживает системные ограничения (например, ограничения памяти) и может быть представлена в виде сравнительно простой модели данных для облегчения ее восприятия и понимания пользователем. Во-вторых, методология должна быть достаточно общей и гибкой, доступной разработчикам с различным опытом проектирования, использующим различные модели данных и программное обеспечение систем управления базами данных. И наконец, методология должна быть воспроизводимой, т. е. такой, чтобы два разработчика (или две программы), применяя методологию для решения одной и той же проблемы, получили бы одинаковые или примерно одинаковые результаты.

Определить цели сравнительно просто, намного проще, чем оценить критерии их достижения. Легко проверить, например, воспроизводимость, но гораздо сложнее ее достичь. Только постепенное совершенствование отдельных средств проектирования до такого уровня, что они могут быть обобщены и автоматизированы, может служить окончательным доказательством воспроизводимости. Гибкость может быть продемонстрирована путем экспериментов (тестирования) на различных моделях данных с использованием имеющегося опыта проектирования. Оценка окончательной структуры базы данных предполагает

использование научных методов постановки проблем, формирование гипотез, экспериментальное проектирование, сбор данных, интерпретацию результатов и подготовку заключения как по каждому отдельному средству проектирования, так и по всей методологии в целом. Аналитические модели структур баз данных могут быть оценены с помощью тестовых данных на действующих системах. Оценка же методологии носит скорее условный характер, поскольку для достижения приемлемого уровня качества функционирования необходимо выполнить большое количество тестовых испытаний при различных условиях.

Для достижения указанных выше целей методология проектирования баз данных должна включать следующие основные компоненты [243]:

1. Процесс проектирования, состоящий из серии этапов, на каждом из которых необходимо сделать выбор из нескольких альтернативных решений.
2. Методики выполнения требуемых в процессе проектирования расчетов и критерии оценки альтернативных решений на каждом этапе.
3. Информационные требования в качестве исходных данных для процесса проектирования как в целом, так и на каждом этапе.
4. Средства описания исходных данных и представления результатов каждого этапа проектирования.

Рассмотрим каждый из этих компонентов более подробно, используя концепцию жизненного цикла системы баз данных. Следует отметить, что оставшаяся часть книги посвящена детальному рассмотрению основных предложенных этапов и анализу используемых средств проектирования.

Процесс проектирования

При проектировании баз данных можно воспользоваться широко известными методами проектирования программного обеспечения [38, 333]. В частности, полностью применимо к структурам баз данных нисходящее проектирование с последовательными итерациями. На начальной стадии концептуальная модель, представляющая элементы данных и взаимосвязи предметной области, последовательно преобразуется в СУБД-ориентированную структуру базы данных. Процесс проектирования хорошо структурирован, так как каждый его этап завершается четко определенным результатом, а также потому, что допускает итеративное повторение предыдущих этапов в случае, если полученный результат не соответствует требованиям пользователей или системным ограничениям, либо если накладываются дополнительные требования. В общем случае это позволяет проектировщику пересматривать проектные решения с любого

предыдущего этапа. На практике стоимость проектирования при этом значительно возрастает, особенно если проектные решения пересматриваются после того, как некоторые из них уже реализованы; поэтому итерации наиболее эффективны на этапах, предшествующих этапу реализации. Хотя итерация может значительно уменьшить общую стоимость разработки базы данных, она затрудняет достижение одной из целей методологии проектирования, а именно воспроизводимости. Тем не менее на сегодняшний день итерация является наиболее необходимым и полезным средством (рис. 2.1), которое может применяться на любом этапе процесса проектирования базы данных.

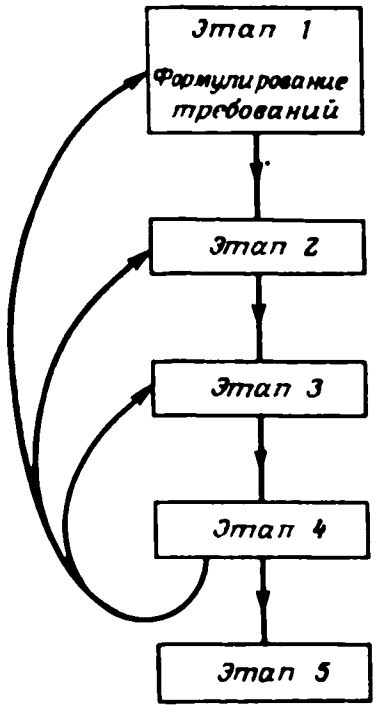


Рис. 2.1. Прямой и итеративной варианты (для этапа —4) в исходящей методологии.

Тесно связана с процессом проектирования многошаговая *методология экспертной оценки проекта* или *проектной экспертизы*, включающая такие методы, как сквозной структурный контроль структуры базы данных, а также прикладного программного обеспечения. Проектная экспертиза может особенно широко применяться в системах баз данных [42] с использованием стратегии, предложенной для более общих информационных систем [162, 355]. Цель экспертизы — обнаружить ошибки системного проектирования и исправить их на более ранних этапах жизненного цикла, чтобы уменьшить стоимость разработки системы. Экспертная бригада подбирается из опытного персонала и включает представителей групп администратора баз данных, разработки программного обеспечения и специалистов в каждой прикладной области, на которые ориентирована система. Каждая бригада экспертов выбирает руководителя, на которого возлагаются полномочия назначать и проводить обсуждения. Обычно рекомендуемая продолжительность обсуждения — от одного до полутора часов, однако, если обсуждение проходит безрезультатно, руководитель вправе отменить его и назначить снова.

Обычно проектная экспертиза проводится не менее четырех раз в течение жизненного цикла, а именно:

1. После анализа требований и проектирования информационной структуры, т. е. после концептуального проектирования.
2. После детального проектирования системы.
3. После реализации, но до начала эксплуатации системы.
4. После начала эксплуатации, когда уже собрана содержа-

тельная информация об эксплуатационных характеристиках.

Каждая последующая экспертиза требует более подробной документации или данных тестирования, но форма проведения обсуждений остается одной и той же. Документация распространяется среди членов экспертной бригады за несколько дней до обсуждения, поэтому каждый из них может заранее подготовить вопросы и альтернативные предложения относительно рассматриваемых структур баз данных. Во время обсуждения делается формальное представление использовавшихся в процессе проектирования подходов, рассматриваются замечания, сделанные экспертами как в устной, так и в письменной форме. Вторая экспертиза чаще всего не укладывается в регламент одного обсуждения. Если система большая, основные ее компоненты могут рассматриваться по отдельности. Третья и четвертая экспертизы имеют преимущество, заключающееся в том, что обычная документация дополняется статистическими данными. Отклонения от предполагаемых характеристик производительности или требований служат предметом для последующего анализа и обсуждения.

Средства проектирования и оценочные критерии

Каждый этап процесса проектирования характеризуется набором методов проектирования и критериями оценки альтернативных решений. Несмотря на то что методы проектирования могут носить аналитический, эвристический или процедурный характер, любой метод, который может быть реализован в виде программы, становится *инструментальным средством проектирования* и, как таковой, он практически не подвержен влиянию стиля проектирования.

Оценочные критерии, как средство проектирования, необходимы для выбора рациональной структуры базы данных среди нескольких альтернативных возможностей. Можно сказать, что большинство проблем и неудач при проектировании баз данных возникают из-за нечеткого представления того, что понимается под оптимальным проектированием баз данных. В настоящее время, а также в ближайшее будущее неопределенность при выборе критериев будет оставаться наиболее слабым звеном в проектировании баз данных.

Трудности в определении критериев выбора альтернативных решений вызваны в основном двумя факторами. Первая проблема заключается в том, что может быть построено практически бесконечное число различных структур баз данных, удовлетворяющих одному и тому же множеству системных требований. Критерии выбора должны позволять дифференцировать все имеющиеся в данный момент альтернативы. Описание и идентификация альтернатив должны быть такими, чтобы не

упустить из рассмотрения те из них, которые приводят к лучшим решениям. Вторая проблема состоит в том, что альтернативы чрезвычайно трудно поддаются оценке. Существует много признаков оптимальности, являющихся неизмеряемыми свойствами, которые трудно выразить в количественном представлении или в виде целевой функции. Поэтому оценочные критерии следует делить на количественные и качественные. Некоторые критерии, сгруппированные в такие категории, показаны на рис. 2.2.

Критерии оценки базы данных

Количественные	Качественные
• Время отклика на запрос	• Гибкость
• Стоимость обновления	• Адаптивность
• Стоимость памяти	• Понимаемость проекта для новых пользователей
• Время, затраченное на создание	• Совместимость с другими системами
• Стоимость реорганизации	• Возможность конвертирования для использований в другой вычислительной среде
	• Возможность восстановления/рестарта
	• Возможность членения или расширения структуры

Рис. 2.2.

Одна из трудностей в оценке альтернатив заключается в том, что критерии обладают свойством чувствительности и время действия различных критериев различно. Критерии эффективности обычно являются краткосрочными, поскольку они чрезвычайно чувствительны к проводимым изменениям, в то же время такие понятия, как конвертируемость и адаптивность, требуют более длительной перспективы для рассмотрения и менее чувствительны к кратковременным флуктуациям среды.

Информационные требования

Информационные требования проходят через весь процесс поэтапного совершенствования баз данных. Часто бывает полезным разделить информационные требования, используемые в процессе проектирования, на информацию, формирующую концептуальное структурное представление, и информацию, формирующую концептуальное прикладное представление [176].

Информация, отнесенная к *концептуальному структурному представлению* (ISP — information structure perspective), описывает естественные концептуальные связи всех данных в базе данных. Она не связана ни с конкретным способом обработки, ни с конкретным приложением. ISP-информация отображает прообразы реального мира в сущности и атрибуты, а взаимо-

связи между прообразами реального мира — в связи между элементами данных. Информационные требования типа ISP всегда имеют место в любой организации, хотя их не всегда легко выявить и описать количественно. Пример информации такого типа показан на рис. 2.3.

Информация, отнесенная к *концептуальному прикладному представлению* (UP — usage perspective), определяет требования организации к обработке данных. Она описывает (в отличие от естественных концептуальных связей и данных) данные и связи, используемые в приложениях. Эта информация отображает

ISP-информация

Информация	Пример
• Описание сущности	
Наименование	Служащий
Мощность	100
• Описание атрибута	
Наименование	Номер страхового полиса
Длина	9
Диапазон значений	0—999 999 999
Вероятность существования	1,0
Коэффициент повторения	1
• Описание связи	
Наименование	Работает в
Определяемые сущности	Служащий, отдел
Мощность	100
Отображение	1 : 1
Вероятность существования	0,95

Рис. 2.3.

требования к обработке данных текущих приложений и предполагаемые требования планируемых в будущем приложений. Иногда бывает невозможно оценить использование данных в приложениях, поскольку они могут содержать произвольные запросы или являться непредвиденными. Пример информации типа UP показан на рис. 2.4.

Между этими двумя представлениями существует значительное пересечение. Используемые в приложениях элементы данных, входящие в состав UP-информации, представляют собой соответствующее подмножество элементов данных, содержащихся в ISP-информации при условии, что набор ISP-информации является полным и согласованным.

База данных, спроектированная на основе требований, содержащихся в ISP-информации, значительно отличается от базы данных, полученной исходя из требований типа UP. Использование независимых от конкретных приложений элементов данных типа ISP имеет при проектировании базы данных

особое значение. Использование ISP-информации гарантирует то, что структура базы данных обеспечит поддержку любых текущих и будущих приложений, поскольку элементы данных, входящие в состав UP-информации, являются подмножеством набора элементов данных типа ISP. Вследствие этого ISP-информация обеспечивает основу для обработки неформализованных, изменяющихся и неизвестных запросов и приложений (приложений, для которых невозможно заранее определить требования к данным). Поскольку предполагается, что база

UP-информация

Информация	Пример
• Описание процесса	
Наименование	Платежная ведомость
Частота применения	Еженедельно
Вероятность применения	1,0
Приоритет	Высший
Требуемые данные	Служащий, табель учета, оклад
Объем данных	100 служащих
• Оператор	
Оператор	НАЙТИ
Критерий поиска	Служащий
Кол-во поисковых образов	Все
Используемые ассоциации	Признак занятости
Вероятность события	1,0
Вероятность использования поисков образа	0,95

Рис. 2.4.

данных предназначена обеспечивать различные приложения в течение длительного времени, такая адаптивность является весьма желательной.

Информация типа UP иначе воздействует на процесс проектирования. Основывая проектирование на текущих и предвидимых приложениях, можно достичь высокоэффективного решения, так как в этом случае структура базы данных может отражать наиболее часто используемые пути доступа. В прошлом, во времена программно-ориентированного проектирования, это имело еще большее значение.

Каждый из этих наборов требований воздействует на результаты проектирования в разных направлениях: ISP обеспечивает гибкость и адаптивность, а UP — эффективность. Причина, по которой UP-ориентированное проектирование было так популярно, заключается в том, что выгоды такого проектирова-

ния проявляются незамедлительно и легко поддаются измерению. Однако польза, получаемая за счет гибкости и адаптивности, может быть намного больше, поскольку проектные решения, ориентированные на достижение сиюминутной эффективности, более чувствительны ко всякого рода изменениям как в процессе обработки данных, так и в их структуре. Такие структуры подвержены более частой реорганизации с целью поддержания эффективности и полноты базы данных. Выгоды от гибкости более долговременны, но их трудно выразить в количественном отношении. ISP-ориентированное проектирование чувствительно к изменениям реального мира, которые определяют требования к данным и время жизни которых обычно намного больше, чем прикладных программ.

Желание достичь и гибкости, и эффективности привело к формулированию методологии проектирования, использующей как ISP-, так и UP-информацию [177, 243]. В общем случае ISP-информация должна использоваться для построения первоначальной информационной структуры, а UP-информация — для ее совершенствования с целью повышения эффективности обработки данных. Таким образом, польза включения ISP-информации вполне очевидна: при сравнительно низкой стоимости выполнения известных формализованных приложений повышается гибкость системы за счет возможности обработки неформализованных и незапланированных приложений. Возможно, что эти новые и неформализованные приложения будут обрабатываться не столь эффективно, если их пути доступа будут значительно отличаться от путей доступа текущих приложений. Однако это все же лучше, чем не иметь такой возможности вообще. Признание этого факта помогло значительно увеличить потенциальные возможности систем баз данных. Без включения ISP-информации база данных обеспечивала бы только простое объединение данных нескольких пользователей, ограниченных созданием своих собственных файлов.

Средства описания

И спецификации требований, и структуры баз данных на всех этапах проектирования должны быть представлены в виде простой модели или комплекса связанных моделей, понятных как конечному пользователю данных, так и прикладному программисту, а также системным аналитикам и программистам. Пошаговое совершенствование основной структуры требует согласованности в представлении данных и при манипулировании ими на всех уровнях.

Существуют три основных класса описательных средств, использующихся в методологии проектирования. Первым из них является язык описания данных ЯОД, входящий в состав СУБД

и использующийся для описания конечного результата процесса проектирования реализации. Один из прототипов ЯОД для терминологии и определений некоторой обобщенной СУБД описан в ANSI/SPARC Interim Report [8].

Ко второму классу относится описание исходной информации, которое будет обсуждаться в гл. 3. Все средства сбора и анализа информации, используемые в настоящее время, подобны в том, что они обеспечивают форматы для спецификации как информации типа ISP, так и типа UP, а также выполняют основные проверки согласованности данных. Однако идентификация синонимов, омонимов и тавтологий элементов данных и их связей трудно поддается автоматизации и, вероятно, будет оставаться предметом деятельности человека с использованием программного обеспечения в качестве вспомогательного средства.

Третий класс описательных средств используется для представления результатов промежуточных этапов и является промежуточным между ЯОД и описанием исходной информации. Основным результатом является диаграмма сущностей (рис. 1.3), которая играет роль мостика между концептуальным проектированием и проектированием реализации. Многие методы проектирования используют нормализованное реляционное представление, которое признано весьма полезным средством.

2.2. Основные этапы проектирования баз данных

Основные цели проектирования баз данных заключаются в том, чтобы обеспечить пользователей точными данными, необходимыми для исполнения ими своих служебных обязанностей, а также в том, чтобы обеспечить доступ к данным за приемлемое время. Для достижения первой цели необходимо, чтобы элементы базы данных наиболее полным образом отвечали потребностям пользователей организаций, основанным на их общих целях, внутренней организационной структуре и предполагаемых требованиях доступа к данным. Достижение второй цели (т. е. производительности) требует, чтобы структура базы данных, полученная в результате процесса проектирования, обеспечивала достаточно быстрый доступ к данным с тем, чтобы те, кто нуждается в этих данных, могли выполнять свои служебные обязанности достаточно эффективно. Однако производительность имеет на разных этапах проектирования различные аспекты. Если целью ранних этапов проектирования является разработка гибкой структуры базы данных, адаптируемой к изменениям пользовательской среды, то на последующих этапах можно акцентировать внимание на достижении оптимальности функционирования при известных требованиях обработки.

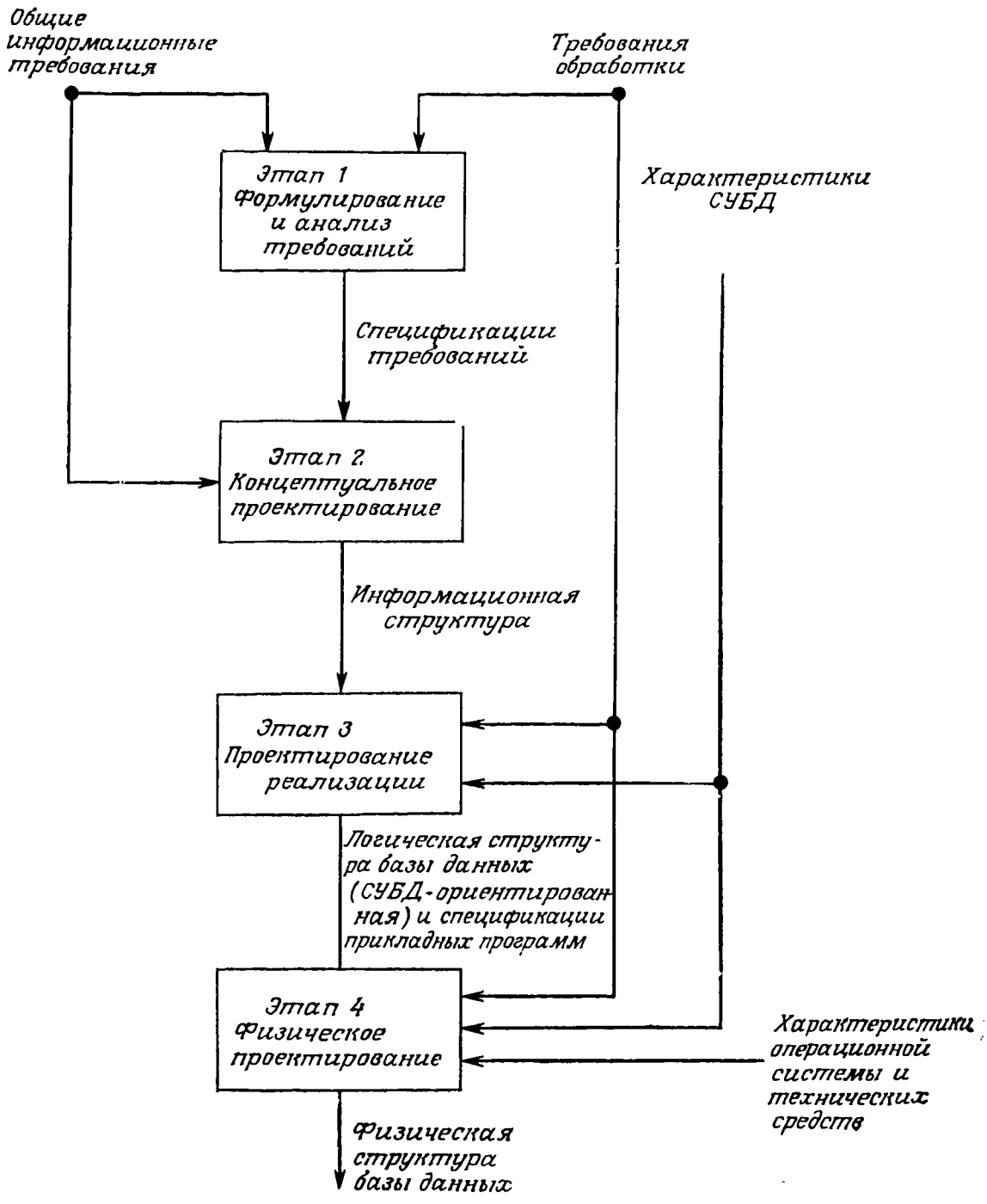


Рис. 2.5. Основные этапы проектирования баз данных.

Методология поэтапного проектирования баз данных в настоящий момент может быть определена как построение структуры базы данных на основе множества информационных и процедурных требований пользователей. Развитие методологии проектирования баз данных продолжается, и в настоящее время не существует общего соглашения относительно того, какой

подход является наилучшим. Поэтому практическая методология должна основываться на общепринятых принципах и не зависеть от узконаправленных представлений и предположений. В этом смысле предлагаемые ниже этапы проектирования представляют основные принципы, вытекающие из общеизвестных в настоящее время методологий. Особое внимание уделяется методологиям, развитым в работах Шеппарда [282, 266], Кана [176, 121], Тозера [325], Чена [72], Бубенко [46] и Герритцена [130, 131]. Эти этапы можно рассматривать как общие директивы или контрольные точки для администратора или проектировщика баз данных, но никоим образом как формальный набор правил, охватывающий все возможные ситуации, возникающие в процессе проектирования. Общая схема взаимодействия этапов проектирования показана на рис. 2.5. Следует отметить, что последовательность этапов соответствует жизненному циклу систем баз данных, обсуждавшемуся в гл. 1.

Этап 1. Формулирование и анализ требований

На этом этапе устанавливаются цели организации, определяются специфичные требования к базе данных, вытекающие из этих целей или сформулированные непосредственно управляющим персоналом организации. Эти требования документируются в форме, доступной как конечному пользователю, так и проектировщику базы данных. Обычно при этом используется методика интервьюирования персонала различных уровней управления и ведущих специалистов организации, участвующих в процессах производства, обслуживания и обработки данных. В результате собеседований определяются информационные потоки, отображающие указанные процессы и их взаимодействие, а также вырабатывается однозначное представление о семантике информационной модели. Специфичные цели и требования к базе данных необходимо определить на самом высоком уровне организации. Собранные и документированные требования должны включать ограничения, обусловливающие безопасность, надежность, уровень достигнутой технологии, а также политические и бюрократические ограничения. Должна быть также оценена политика организации в отношении персонала, управленческой деятельности и перспектив роста организации. Документирование может быть выполнено вручную с помощью языков представления требований или с помощью средств автоматизации. Может оказаться полезным на этапе документирования отделить требования, зависящие от процесса обработки данных (т. е. связанные со специфичными приложениями), от требований, не зависящих от процесса обработки [176].

Этап 2. Концептуальное проектирование

Этап концептуального проектирования связан с описанием и синтезом разнообразных информационных требований пользователей в первоначальный проект баз данных. Результатом этого этапа является высокоуровневое представление информационных требований, например, такое как диаграмма «сущность-связь». Основу этой диаграммы составляет набор сущностей, который представляет или моделирует определенную совокупность сведений, специфицированную в требованиях. Сущности могут быть описаны атрибутами, позволяющими детализировать свойства сущности. Один или несколько атрибутов могут служить идентификатором для обозначения отдельных экземпляров сущности. Связи между сущностями отображают функциональные аспекты информации, представленной сущностями [69, 72].

Существует несколько подходов к построению диаграмм типа «сущность-связь». Общим для всех подходов является набор из четырех основных проектных решений или шагов:

1. Определение сущностей.
2. Определение атрибутов сущностей.
3. Идентификация ключевых атрибутов сущностей.
4. Определение связей между сущностями.

Хотя существует некоторая общность мнений о необходимости этих шагов для построения информационной структуры, тем не менее нет общего соглашения относительно порядка их выполнения. Главное, что здесь необходимо, это чтобы первоначальная информационная структура была хотя бы частично независимой от процесса обработки, что позволит обеспечить гибкость процесса проектирования.

Подход к концептуальному проектированию обычно предполагает, что рассматривается представление одного-единственного пользователя. При этом возможно предположить, что таким единственным пользователем является администратор или проектировщик базы данных, который понимает требования всех пользователей и объединяет эти требования в полный набор согласованных спецификаций. Альтернативным является подход, в котором «интеграция представлений» выполняется как часть процесса концептуального проектирования [238, 348, 350, 352].

Моделирование частных представлений. Требования каждого пользователя анализируются и представляются в некоторой общей форме. Объекты и события, ассоциированные с представлением каждого пользователя, моделируются множеством сущностей, атрибутов и связей между сущностями.

Интеграция частных представлений. Требования отдельных пользователей, определенные в стандартной форме, такой, как диаграммы «сущность-связь» [69, 72, 297], сливаются в единое глобальное представление. При этом должны быть выявлены и ликвидированы противоречивые и избыточные данные. Если в окончательном варианте системы предусмотрено использование механизма подсхем, то результаты моделирования частных представлений могут быть использованы для проектирования подсхем, а результат интеграции частных представлений после соответствующей обработки может быть преобразован в схему.

Этап 3. Проектирование реализации

После построения первоначальной информационной структуры следует ее уточнение и совершенствование. Главной целью этапа проектирования реализации является создание СУБД-ориентированной схемы с использованием в качестве исходных данных результатов концептуального проектирования и требований обработки (UP-информации). Хотя вопрос точного определения границы между концептуальным проектированием и проектированием реализации является открытым, равно как и вопрос о взаимосвязи между логическим и физическим проектированием, тем не менее отмечаемые здесь различия являются важными для развития практической поэтапной методологии.

Вначале анализируется содержание требований обработки данных. Формат локальных информационных структур, подлежащих обработке, соответствует формату первоначальной структуры, полученной на этапе концептуального проектирования. После того как представлены все виды обработки, первоначальная структура может быть объединена со всеми локальными структурами в новую информационную структуру. Затем, используя знания, полученные в процессе пересмотра и объединения информационных структур, учитывая связи обрабатываемых данных и характеристики типов записей, допускаемых СУБД, можно сформировать предварительные типы записей.

Построенная таким образом логическая структура базы данных (схема) может быть оценена количественно с помощью таких характеристик, как число обращений к логическим записям, объем обрабатываемых в каждом приложении данных, общий объем хранимых данных. Эти оценки помогут приблизительно определить эффективность функционирования физической базы данных в терминах затрачиваемого на обработку времени и требуемой физической памяти. Оценка эффективности логической структуры базы данных, а также некоторых других характеристик проектирования, таких, как целостность и безопасность, описывается в гл. 7.

И наконец, схема может быть усовершенствована с целью достижения большей эффективности. В некоторых случаях, заглядывая вперед, полезно оценить такие свойства СУБД, как возможность индексирования, перемешивания (хеширования), существования точек входа в систему. Однако, если дальнейшее совершенствование схемы не может быть выполнено без изменения информационного содержания, ее проектирование на этом заканчивается. Все усовершенствованные варианты подвергаются такой же оценке, что и первоначальная схема.

Этап 4. Физическое проектирование

Практика проектирования физической базы данных прошла несколько этапов своего развития, начиная от проектирования файлов и кончая проектированием физических структур интегрированных баз данных в рамках существующих ныне СУБД. Эта практика включает различные известные методы хранения структур, поисковые механизмы, сегментацию записей. К счастью, современные СУБД в основном используют физические структуры и методы доступа, опирающиеся на технологию проектирования файлов, что позволяет использовать многое из старой методики.

Можно разделить основные проектные решения в физическом проектировании по меньшей мере на три категории.

Проектирование формата хранимых записей. Сюда включаются все виды представления и сжатия данных в хранимых записях. Сюда же относится распределение элементов данных записи по различным участкам физической памяти в зависимости от размеров и характеристик их использования.

Анализ и проектирование кластеров. Кластеризация включает размещение экземпляров записей в смежных участках физической памяти, распределение по различным устройствам внешней памяти, выбор размеров блоков с целью эффективной выборки.

Проектирование путей доступа. Сюда включаются такие параметры, от которых в значительной степени зависит стоимость доступа при поиске и обновлении данных (например, логическое упорядочение записей, выбор указателей, методы доступа, техника обработки переполнений). Отметим, что эти характеристики имеют важное значение и на этапе 3, из чего следует, что вопросы проектирования реализации и физического проектирования пересекаются.

Техническое описание прикладных программ и ограничений целостности и безопасности также является общей частью как физического проектирования, так и проектирования реализации. Однако порядок, в котором должны рассматриваться

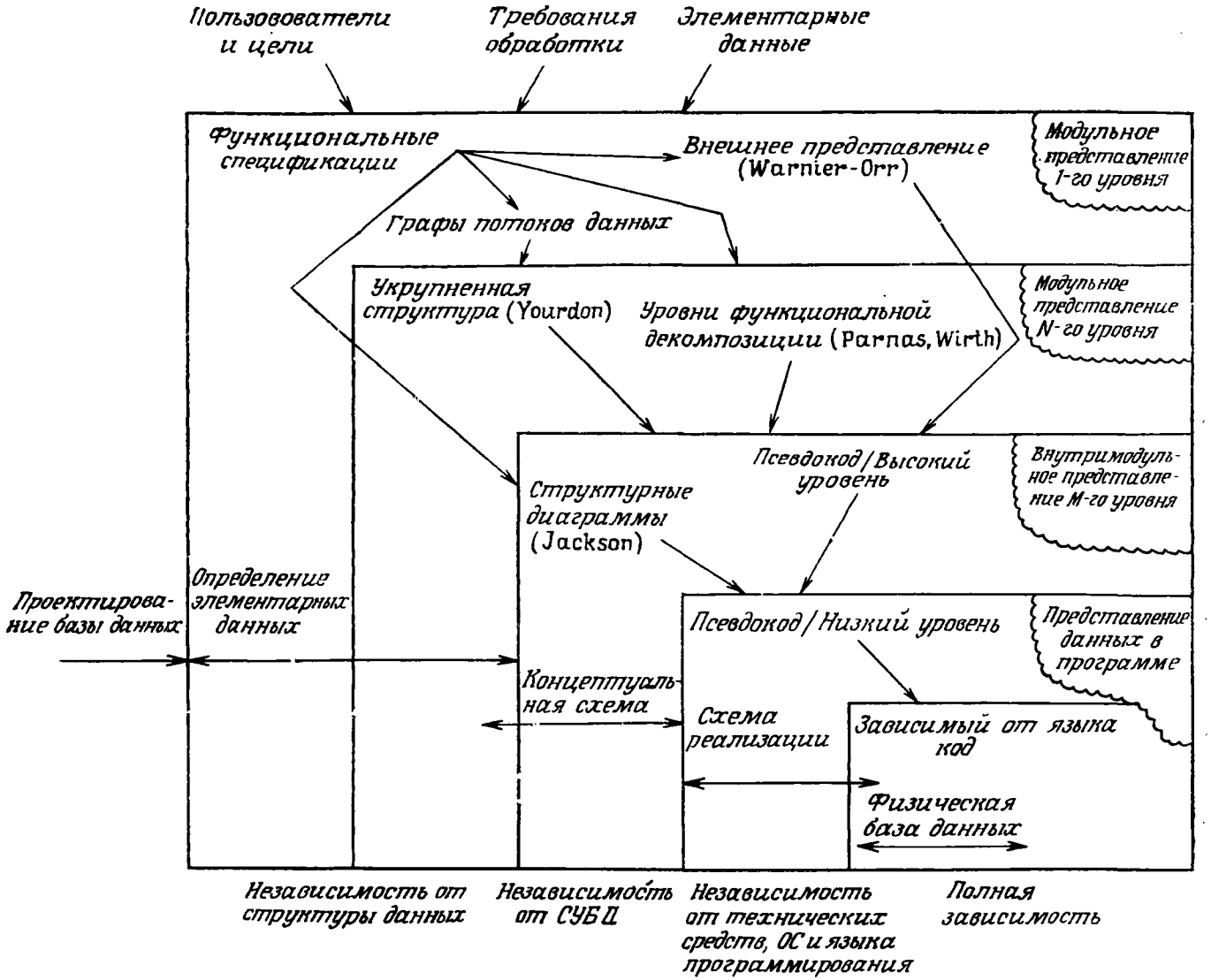


Рис. 2.6. Соответствие между этапами развития программного обеспечения и базы данных.

основные проектные решения на этапе физического проектирования, не столь важен. В большинстве случаев легче всего оценить на этом уровне время обработки ввода-вывода, являющееся одним из главных компонентов времени, затрачиваемого на выполнение приложений. Минимизация времени обработки ввода-вывода и затрат памяти является одной из главных целей этого этапа. Важным вопросом является также минимизация процессорного времени, однако это определяется в большей степени путем надлежащего проектирования прикладных программ, нежели проектированием структуры базы данных.

Проектирование системного программного обеспечения, включающего проектирование программ и базы данных в качестве своих компонентов, претерпело быструю эволюцию в соответствии с принципами структурного анализа, проектирования и технологии программирования, предложенными Дейкстрой и др. [109, 227, 250, 340, 341] и реализованными в виде коммерческих систем [51, 129, 173, 248, 260, 356]. Опыт показывает, что как программное обеспечение, так и базы данных в своем развитии и поэтапном совершенствовании проходят параллельными (а иногда пересекающимися) путями и что на всех этапах проектирования они оказывают влияние друг на друга при выборе проектных решений. Хотя основное внимание в данной книге уделяется проектированию структур баз данных, важно постоянно помнить о том влиянии, которое оказывает очередное улушение структуры базы данных или, например, новые ограничения целостности на структуру программ и их эффективность.

Основные взаимосвязи между программным обеспечением и базами данных, возникающие в процессе проектирования систем, показаны на рис. 2.6. Эти взаимосвязи определяют многие известные средства документирования, используемые в настоящее время для описания программ и структур данных. Самый внешний прямоугольник определяет наиболее абстрактное (наивысшего уровня) представление, используемое на ранних этапах проектирования, а внутренний — наименее абстрактное (низшего уровня или системно-зависимое) представление, используемое на этапе реализации системы. Стрелками обозначено взаимное влияние компонентов друг на друга.

2.3. Вопросы проектирования

Основными проблемами в процессе проектирования баз данных являются целостность, согласованность, восстанавливаемость, безопасность, эффективность, а также последствия предполагаемого роста базы данных в будущем. Эти проблемы рассматриваются в последующих разделах, где уделяется также внимание их взаимодействию с другими проблемами.

Указанные проблемы можно сгруппировать следующим образом [206]:

1. Целостность, согласованность, восстанавливаемость.
2. Безопасность.
3. Эффективность, рост, размер, эксплуатационные ограничения.

Это разделение на группы отображает степени свободы проектировщика баз данных. Первая группа обычно отображает требования пользователей и состоит из «жестких» ограничений, в рамках которых должен работать проектировщик. Вторая и третья группы обычно предоставляют проектировщику больше свободы для удовлетворения этих требований. Решение проблем третьей группы является в значительной степени (но не полностью) прерогативой проектировщика, когда уже найдено возможное решение, удовлетворяющее требованиям пользователей.

Вопросы эффективности должны быть разрешены как можно раньше, с целью дальнейшего беспрепятственного развития системы. Если, например, эти вопросы не принимались во внимание в процессе разработки схемы реализации, они должны быть рассмотрены где-либо еще (в операционной системе, в процедурах обработки баз данных на уровне приложений и т. д.). Децентрализация при решении этих вопросов приводит к возникновению проблем координации, гибкости и организации при реализации системы.

2.3.1. Целостность, согласованность и восстанавливаемость

База данных обладает свойством *целостности*, если она удовлетворяет некоторым определенным ограничениям значений данных и сохраняет это свойство при всех модификациях (замена, добавление или удаление) базы данных [206, 330]. В качестве примера рассмотрим ограничение целостности «зарплата служащего не может быть больше зарплаты его руководителя», определенное для базы данных служащих. Чтобы ввести это ограничение, необходимо сверить зарплату каждого служащего с зарплатой его (или ее) руководителя. Если это ограничение не удовлетворяется для какого-либо служащего, то запись о данном служащем удаляется из базы данных. В дальнейшем это ограничение должно контролироваться всякий раз, когда в базу данных добавляется запись о служащем или модифицируется поле, в котором содержатся сведения о зарплате служащего. Из смысла этого ограничения следует, что оно не должно проверяться при операциях удаления записей из базы данных. Существуют такие ограничения, которые необходимо проверять либо во время операций обновления, либо удаления, либо в обоих случаях.

База данных обладает свойством *согласованности* по отношению к некоторой совокупности пользователей, если в любой момент времени база данных реагирует на их запросы одинаковым образом (т. е. все пользователи на заданный ими конкретный запрос получают одинаковый ответ). В качестве примера рассмотрим файл служащих, используемый в системе с разделением времени совместно несколькими пользователями. Предположим, что администратор базы данных корректирует сведения о зарплате, а в то же самое время ревизор, используя базу данных, готовит отчет о бюджете. Если администратор не имеет возможности блокировать доступ к базе данных (т. е. исключить всех других пользователей) на все время выполнения транзакции, ревизор может получить неправильный отчет. С другой стороны, если возможность блокирования доступна как администратору, так и ревизору и используется надлежащим образом, то при этом оба достигнут согласованных результатов. Существует много способов поддержания согласованности базы данных.

Целостность и согласованность — важные свойства базы данных и поэтому являются особо важными факторами для проектирования базы данных. Ниже перечислены их основные параметры:

1. *Целостность*
   - а. Ограничения.
   - б. Правила применения ограничений.
   - в. Правила обработки данных при нарушении ограничений целостности.
   - г. Эффективность применения ограничений.
2. *Согласованность*
   - а. Методы модификации данных.
     1) Промежуток времени между двумя модификациями.
     2) Время на выполнение модификации.
     3) Домены, на которые оказывает влияние модификация.
     4) Алгоритм модификации данных.
   - б. Число пользователей базы данных в момент проведения модификации.

Восстанавливаемость — это запроектированная возможность восстановления с помощью СУБД целостности базы данных после любого сбоя системы. Процесс проектирования такой возможности включает разработку систем проверки адекватности баз данных, предназначенных для того, чтобы избежать потерь транзакций и данных. В случае сбоя системы основной задачей является восстановление в течение приемлемого времени существовавшего до сбоя состояния транзакций и данных. Время восстановления может быть определено в требованиях пользо-

вателя как количественная характеристика. Однако такая характеристика должа включать четкое определение того, кем или чем будет инициироваться процесс восстановления: транзакцией, СУБД, операционной системой или оператором пульта управления. Это, конечно, будет зависеть от источника сбоя, которым могут являться транзакции, операционная система или технические средства.

Взаимодействие с другими проблемами проектирования

Реализация методов поддержания целостности и согласованности оказывает главное влияние на эффективность СУБД. Разумный подход при их реализации может сократить связанные с этим накладные расходы в любой СУБД и, следовательно, повысить эффективность. Если, например, поддержание согласованности спроектировано таким образом, что блокировки ограничены только затрагиваемыми при модификации доменами, то это окажет влияние на время отклика только для небольшого числа пользователей и, следовательно, повысит эффективность всей системы.

В общем же случае поддержание целостности и согласованности увеличивает время отклика. Разумные методы и алгоритмы могут только минимизировать это увеличение. Если можно спланировать работу системы таким образом, что поддержание целостности и согласованности осуществляется в то время, когда система не занята обслуживанием пользователей, то рост времени отклика может быть снижен до нуля. Поэтому влияние этих компонентов на эффективность системы зависит исключительно от методов реализации. Чтобы оценить влияние поддержания целостности на эффективность, можно определить частоту и удельную стоимость процесса поддержания целостности в терминах вычислительных ресурсов (например, процессорного времени и времени ввода-вывода) и рассматривать программное обеспечение целостности как одно из приложений наряду с приложениями конечных пользователей. Поддержание согласованности, восстанавливаемости и безопасности может быть оценено таким же способом.

Существует непосредственная зависимость между восстанавливаемостью и другой проблемой, а именно производительностью. Чем сложнее и быстрее проходит процесс восстановления, тем больше накладные расходы. Поисковые операции могут восстанавливаться быстрее и надежнее, в то время как транзакции обновления восстанавливаются гораздо медленнее. Второстепенной, но не менее важной является зависимость между производительностью, размерами и ростом базы данных. Можно определить независимо восстанавливаемые подсхемы так, что транзакции смогут обновлять только одну

подсхему. В зависимости от проблемы это может привести к более (или менее) гибкой системе в отношении роста: требование поддержания независимости пользователей друг от друга может привести как к повышению, так и к снижению гибкости системы. После расчленения система может стать больше по размерам, однако возможно, что благодаря хорошо организованным интерфейсам ее будет легче поддерживать.

2.3.2. Безопасность

Под *безопасностью* понимается защита данных от преднамеренного или непреднамеренного доступа, модификации или разрушения. Основной задачей поддержания безопасности базы данных является запрещение несанкционированного доступа к данным при минимальных затратах. Поэтому контроль доступа является наиболее важной проблемой реализации. Он может быть осуществлен посредством подсхем, которые позволяют отдельным пользователям иметь дело только с той частью логической структуры базы данных, которая им необходима. Внутри такой структуры можно усовершенствовать контроль доступа (только читать, писать, включить и т. д.) с помощью механизма блокирования. Большинство СУБД позволяют блокировать как всю базу данных целиком, так и отдельные записи и элементы данных. Выбор объектов блокирования может оказать значительное влияние на эффективность. Блокирование легче всего осуществить на уровне базы данных, однако это может лишить пользователей возможности доступа ко многим нечувствительным и срочно необходимым данным. Блокирование на уровне элементов данных представляет наибольшую гибкость, но значительно увеличивает накладные расходы при обработке элементов данных.

Наиболее приемлемым, по-видимому, будет решение, позволяющее без излишних накладных расходов на обработку добиться обеспечения безопасности на более низком уровне детализации данных. Поэтому четкое определение ограничений безопасности наряду с требованиями производительности (время отклика и т. д.) является решающим для достижения приемлемого компромисса при проектировании схемы базы данных.

При проектировании и реализации мер безопасности необходимо также принимать во внимание способы реализации и алгоритмы, используемые для обеспечения целостности и согласованности. Необходимо проектировать систему таким образом, чтобы поддержка целостности возлагалась на тех пользователей, которые имеют право на модификацию базы данных, в то время как для остальных пользователей это не является необходимым.

2.3.3. Эффективность

Использование вычислительных ресурсов и затраченное на выполнение приложений базы данных время (или время отклика) являются основными аспектами эффективности при проектировании СУБД-ориентированной логической структуры (схемы) базы данных. В зависимости от соотношения этих аспектов можно выделить три класса задач:

1. Проектируемая схема должна обеспечить выполнение каждого приложения за минимальное время. Основу составляют приложения, выполняющиеся в реальном масштабе времени. Время отклика для терминальных пользователей в большой степени зависит от количества запросов к внешней памяти. Проектируемые будущие приложения и рост базы данных должны также оцениваться в терминах приемлемого времени отклика.

2. Проектируемая схема должна использовать минимальное количество внешней памяти для хранения базы данных. Это ограничение называется *ограничением размера.* Кроме того, при выполнении приложений требуется ограничить до минимума объем обмена данных между внешней и основной памятью.

3. Приложения должны выполняться за минимальное время при минимальном использовании основной памяти.

Прежде чем перейти к более подробному обсуждению вопросов эффективности, необходимо остановиться на том, почему эффективность операций с базой данных используется для оценки структуры базы данных. Во-первых, почти каждая реальная база данных ограничена в размерах, а ее приложения — максимально допустимым временем отклика. Во-вторых, хотя другие аспекты проектирования (целостность, согласованность, восстанавливаемость, безопасность) и могут быть достигнуты намного легче, если игнорировать требование эффективности, однако это может привести к такому неэффективному решению, что оно станет неприемлемым.

С точки зрения эффективности проектирование схемы может рассматриваться как проблема оптимизации. Эта проблема заключается в том, чтобы построить такую СУБД-ориентированную схему, которая:

1. Содержится в пространстве решений, определяемом ограничениями целостности, согласованности, восстанавливаемости и безопасности, а также некоторыми ограничениями СУБД.

2. Является оптимальной в смысле эффективности в пределах указанных ограничений.

Эта точка зрения неявно ставит эффективность в зависимость от ограничений целостности, согласованности, восстанавливаемости и безопасности. В пользу этого могут быть предло-

жены два аргумента. Один состоит в том, что главной целью схемы является обеспечение глобального представления базы данных. Другой аргумент заключается в том, что схема не определяет физическую структуру базы данных настолько детально, чтобы можно было осуществить доскональный анализ производительности системы.

Соотношение между эффективностью и другими вопросами проектирования обнаруживается также при проектировании эффективного поиска, эффективного обновления и поддержания ограничений согласованности. Например, в то время как использование избыточности в схеме может уменьшить время отклика при поиске, стоимость обновления при этом повышается, а кроме того, возникают проблемы согласованности. Вдобавок к этому избыточность может привести к увеличению требуемой внешней памяти для базы данных. В общем случае имеет смысл для обеспечения быстрого поиска несколько увеличить расходы на обновление. Тем не менее, отдавая предпочтение поиску или обновлению, следует помнить, что это является решающим фактором для определения эффективности.

Возможно, что наиболее серьезным препятствием для анализа производительности является отсутствие адекватных характеристик производительности физической структуры в терминах схемы. Обычно используются следующие характеристики: число доступных логических записей, объем данных, передаваемых между основной и внешней памятью, размеры требуемой внешней памяти при условии, что размеры хранимых записей равны размерам логических записей. Хотя эти три характеристики могут быть очень полезными, они не всегда адекватны, особенно если физическая структура отличается от схемы.

Глава 3. Формулирование и анализ требований

Донна Л. С. Ранд

Когда появились первые приложения, использующие базы данных, они были сравнительно простыми, а базы данных — небольшими. Поэтому проектировщики имели дело в первую очередь с оптимизацией некоторых физических параметров баз данных (например, размер блока или метод доступа). В этих условиях для проектирования базы данных обычно требовалась некоторая статистическая информация (такая, как частота обращения к базе данных, объем данных и т. д.). В настоящее время приложения баз данных стали намного шире и сложнее, причем несколько различных приложений могут использовать одну и ту же интегрированную базу данных.

В этих новых условиях проектирование базы данных, поддерживающей все приложения, становится очень сложной задачей. Проектирование невозможно без соответствующего предварительного анализа информации. Поэтому в процессе проектирования баз данных анализ требований выдвинут на первый план.

В этой главе на примерах, взятых из деятельности коммерческих компаний, исследуются цели и методы формулирования и анализа информационных требований. При этом используется концептуальное структурное представление (ISP), описанное в гл. 2, так как в большинстве случаев все выявленные информационные потребности организации независимы от какого-либо конкретного приложения. Здесь можно возразить, что в «информационных потребностях» организации неявно отображены данные, используемые в приложениях, и поэтому «информационные потребности» подразумевают также данные концептуального прикладного представления (UP). Поэтому еще раз подчеркнем, что ISP-информация отображает модель организации, представленную данными, загружаемыми в базу данных и рассчитанными на долговременное использование, в то время как UP-информация представляет данные, требуемые непосредственно в конкретных отчетах, произвольных запросах и процедурах обновления, выполняемых вручную или с помощью ЭВМ. В идеальном случае UP-данные являются соответствующим подмножеством данных типа ISP, однако на практике не все приложения могут быть предусмотрены заранее, поэтому возможность их возникновения должна быть предусмотрена на ранних этапах проектирования.

3.1. Введение в анализ требований

Анализ требований включает, кроме всего прочего, анализ потребностей организации без учета каких-либо ограничений, кроме тех, которые встречаются в ее обычной повседневной деятельности. Например, регулярная публикация компанией отчета о своих доходах должна выполняться независимо от процедур или систем, используемых для выполнения этих действий. Аналогично этому принятие компанией решения автоматизировать этот процесс также не зависит ни от конкретной ЭВМ, ни от используемой базы данных. Глобальные решения обычно принимаются высшим руководством, которое определяет цели организации и стратегию для достижения этих целей.

После того как высшее руководство определило политику, намеченные стратегии должны быть выполнены средним звеном управления. Скорее всего именно среднее звено будет определять, какая ЭВМ и какая система являются подходящими для конкретных приложений. Например, будет ли система представлять собой распределенную вычислительную сеть или это будет единственная ЭВМ при штаб-квартире компании. Предполагается, что такие решения будут приниматься средним звеном после консультаций с высшим руководством и с его одобрения. (Несомненно также и то, что во многих случаях подобные решения могут приниматься в соответствии с ранее достигнутой договоренностью.) Системный аналитик, используя представляемые сотрудниками функциональных подразделений дополнительные сведения, может полностью определить потребности в данных и построить соответствующую структуру базы данных.

Для этого системному аналитику необходимо, во-первых, наличие методологии, обеспечивающей получение содержательной информации, требуемой для проектирования базы данных, и, во-вторых, наличие средств, позволяющих сравнительно легко и достаточно четко описать приложения. Этим двум вопросам в данной главе уделяется основное внимание.

Анализ требований охватывает широкий круг пользователей, начиная от клерков, выполняющих простые операции (например, выписка счетов), до руководителей высшего уровня, ответственных за формулирование общих целей и стратегий организации. Поэтому основное внимание концентрируется на первостепенном вопросе, а именно на взаимодействии людей, выполняющих в организации различные функции, имеющих различные цели и различное представление об информационной системе. Однако необходимо ясно представлять, что анализ требований очень тесно связан с другими этапами проек-

тирования. Первичные метаданные (описание и определение данных) необходимы для построения и уточнения концептуальной модели, используемой в дальнейшем в качестве общей основы для сбора и анализа других метаданных. Без такой основы собранные метаданные едва ли будут достаточно надежными: количество метаданных может оказаться больше, чем необходимо, а их структура может оказаться недостаточно эффективной для выявления ошибок.

Как указывалось в гл. 2, основной исходной информацией процесса проектирования базы данных являются информационные требования, включающие ограничения, существующие в организации, элементы данных и способ обработки данных. В данной главе предполагается, что приложения определяются независимо друг от друга либо аналитиками, либо специалистами конкретной области. Так, служащие расчетного отдела, которые вероятнее всего являются бухгалтерами по профессии, будут определять приложения по расчету зарплаты, а специалисты в области управления кадрами определят приложения, связанные с обеспечением функционирования отдела кадров. Каждое подразделение решает свои задачи вне каких-либо внешних ограничений. Это означает, что каждая отдельная задача обработки данных будет определена в рамках локального представления данных без учета представлений пользователей других приложений.

3.1.1. Необходимость общего анализа требований

Необходимые аналитику элементы данных и их связи трудно получить из анализа только основных принципов коммерческой или иной деятельности организации, поскольку между отдельными организациями имеется слишком много различий. Основные принципы обеспечивают построение схемы лишь в самом общем виде, в то время как информация, представляемая пользователями, находящимися на всех уровнях организации, даёт возможность эту схему детализировать. Элементы данных и их связи, которые будут уточняться на последующих этапах проектирования базы данных, должны быть определены во время общего анализа требований. Здесь же должны быть если не разрешены, то хотя бы обнаружены все конфликтные ситуации. Часто разные отделы используют различные наименования для обозначения одних и тех же объектов или понятий, и наоборот, разные по сути объекты могут называться одинаково. Поэтому проблема заключается в установлении взаимопонимания с пользователями с целью выработки предварительного общего представления о данных и способах их обработки, которое обеспечит надежность результатов на последующих этапах проектирования. Такое общее представление мо-

жет быть выработано только совместно с пользователями во время общего анализа требований. Это представление не всегда является прообразом окончательной структуры базы данных, оно лишь устанавливает формальную основу для накопления метаданных. Интеграция представлений осуществляется позднее [206].

Не следует недооценивать проблему достижения взаимопонимания с пользователями. Необходимо очень тщательно подходить к вопросу выбора соответствующей модели данных и языка манипулирования данными на каждом пользовательском уровне (например, для среднего звена управления такой моделью может являться блок-схема информационных потоков, а для высшего звена — иерархическая схема управления). Разработчик должен уметь поддерживать общение с пользователем в терминах и выражениях, понятных пользователю, чтобы ограниченным набором простых предложений выразить основной смысл утверждений пользователя. Этим подчеркивается важность итераций на различных этапах проектирования, а также отличие этапа общего анализа требований от других этапов проектирования базы данных, на которых не поддерживается прямой контакт с пользователями.

Можно показать, что элементы данных, связи между ними, а также способы их обработки могут быть получены непосредственно в результате анализа действующей системы. Однако это требует предварительного проведения двух дополнительных этапов: во-первых, необходимо из множества видов деятельности выделить виды функциональной деятельности, а во-вторых, отделить коммерческую или производственную деятельность (подверженную наименьшим изменениям при реализации новой информационной системы) от видов деятельности, связанных с контролем и планированием (которые могут претерпеть значительные изменения в связи с реализацией новой информационной системы). Выполнение этих двух этапов также требует самого тесного взаимодействия с пользователями.

3.1.2. Этапы формулирования и анализа требований

Рассматриваемая в данной главе методология включает три основных подэтапа в рамках жизненного цикла системы баз данных:

- *Этап 1.1.* Определение сферы применения базы данных как в настоящем, так и в будущем.
- *Этап 1.2.* Сбор информации об использовании данных.
- *Этап 1.3.* Преобразование собранной информации в форму, удобную для проведения анализа.

Каждый из этих этапов подробно описан в последующих разделах.

3.2. Определение сферы применения

В идеальном случае сфера применения базы данных должна определяться независимо от любой прикладной задачи и охватывать все функциональные службы организации. Однако руководство редко одобряет выделение довольно больших средств на разработку логической или физической базы данных, которая охватывает всю организацию в целом. Обычно базы данных разрабатываются и внедряются позадачно. Средства обработки данных, полученные в результате такого подхода, очень похожи на средства обработки файлов, к которым добавлено дорогостоящее и сложное программное обеспечение СУБД. Поэтому основное внимание будет уделено методике, которая позволяет определить потенциальную сферу применения баз данных для выбранных прикладных задач и одновременно минимизировать возможность создания ограниченной файловой системы.

Предположив, что для выбранной существующей системы необходимо создать одну или несколько баз данных, необходимо определить, какие функциональные задачи организации вне сферы применения баз данных должны быть дополнительно рассмотрены в проектном решении. Лучшим источником такой информации является *информационная схема* организации, если она имеется. Хотя содержание схем может меняться в широких пределах, они должны включать определение текущей и будущей стратегий информационного обеспечения руководства, указание сферы применения для каждой системы, определение зависимостей между основными системами (автоматизируемой, неавтоматизируемой) и группами данных. Используя подобную схему в качестве основы, можно определить функциональные задачи, которые следует рассмотреть в рамках проекта. На рис. 3.1 показана схема информационных потоков фрагмента модели, описывающей торговую компанию [129]. Если предположить, что база данных разработана для задачи учета текущих счетов, то эта схема может быть использована, во-первых, для идентификации основных групп данных, требуемых при учете текущих счетов (а именно данных о заказах и клиентах), а во-вторых, для идентификации других приложений, требующих по крайней мере некоторые из этих данных (а именно задачи обработки заказов и задачи управления кредитами). Сфера применения концептуального проекта для задачи учета текущих счетов должна включать функциональные задачи учета текущих счетов, обработки заказов и управления кредитами, а также другие задачи, существующие в реальных системах.

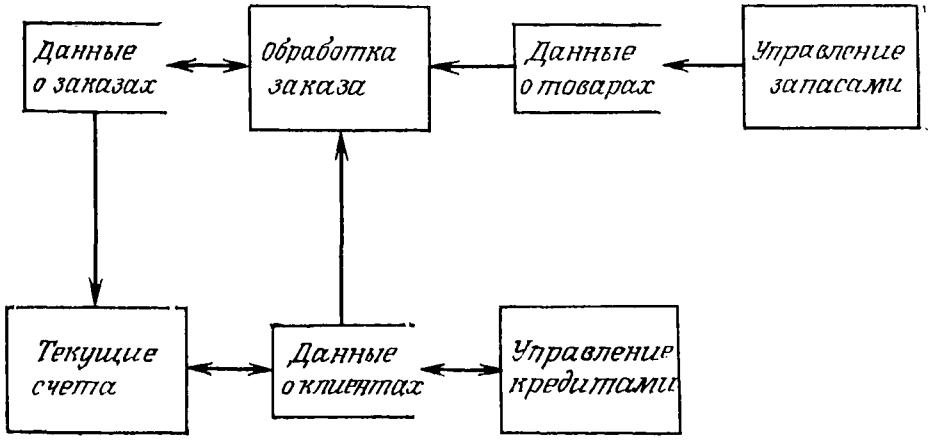


Рис. 3.1. Фрагмент информационной схемы и системных зависимостей.

Если организация не имеет информационной схемы или эта схема не содержит диаграммы зависимостей данных и задач, определение сферы применения проекта возлагается на разработчика. Для этого необходимо во время собеседований с персоналом функциональных служб, включенных в проект, выяснить их отношение к остальной части организации. В выявленных таким образом службах необходимо провести дополнительные собеседования с целью определения совместно используемых данных. Другим фактором, который необходимо принимать во внимание при определении сферы применения, являются возможные в будущем изменения в деятельности организации. Такие изменения могут включать расширение торговой деятельности (например, появление новых товаров, рынков сбыта и т. п.), наиболее важные изменения в проведении торговых операций (например, централизация или децентрализация закупок и складирования), а также регулирование коммерческой деятельности со стороны правительственных органов. Каждое из таких возможных изменений должно быть тщательно проанализировано с целью определения возможных изменений в составе данных, их взаимосвязи и использовании. Если обнаружится, что будущие изменения могут оказать воздействие на базу данных, сфера применения проекта должна быть расширена с тем, чтобы учесть влияние этих возможных изменений.

3.3. Сбор информации об использовании данных

Как только сфера применения базы данных определена, можно приступать к сбору сведений об использовании данных. Прежде чем начать обсуждение методов сбора и документирования такой информации, целесообразно выявить связь между

данными и организацией, в которой эти данные создаются и используются. После этого будет легче понять, почему некоторая информация собирается и документируется самым тщательным образом, в то время как другая информация документируется не регулярно или не собирается вообще. Для этого рассмотрим некоторую гипотетическую организацию и изменения, которым она может подвергнуться с течением времени.

Предположим, что организация была основана двумя братьями в начале 20-х годов для производства и продажи гаек и болтов:

- Один из братьев закупал сырье, изготавливал продукцию и отгружал или доставлял ее клиентам.
- Другой брат принимал заказы клиентов, собирал деньги за проданный товар и вел книги.
- Предприятие размещалось в сарае, принадлежащем братьям.
- За исключением случайных мальчишек-посыльных, других работников у компании не было.

Теперь допустим, что компания продолжает свое дело, но она значительно расширилась:

- Компания получила признание, число служащих составляет более 4000 человек.
- Компания имеет два предприятия внутри страны и несколько контор по сбыту.
- Предприятия характеризуются высокой степенью автоматизации производственных процессов.
- Прием заказов, планирование операций и учет производства осуществляется с помощью ЭВМ.

То, что в деятельности компании произошли изменения, является очевидным: объем продукции значительно возрос, производственный процесс автоматизирован, прием и обработку заказов осуществляет ЭВМ. Не представляет труда, анализируя количественные и качественные стороны каждого из этих изменений, выявить связи между данными в организации. Однако такая точка зрения может привести к заблуждению, поскольку многое осталось неизменным (табл. 3.1):

- Производственные функции предприятия (т. е. сбыт, производство продукции, обработка счетов и т. п.) остались теми же, что и в начале 20-х годов.
- Связи между производственными функциями также остались фактически неизменными (т. е. заказ должен быть принят прежде, чем отгружен товар, выписан счет, сбалансирован текущий счет и т. п.).
- Каждая производственная функция создает или использует относительно неизменный набор данных (т. е. имя клиен-

Таблица 3.1. **Изменяющиеся и постоянные факторы в деятельности организации**

Изменяющиеся	Почти постоянные
Как выполняются функции (т. е. технология выполнения)	Функции, характеризующие деятельность (принятие заказов, ведение счетов и т. д.)
Количество продуктов и рынков сбыта	Вид деятельности
Количество служащих	Связи между различными производственными функциями
Информация, необходимая для управления	
Законодательство: налоги, социальное страхование, охрана труда	Данные, требуемые для выполнения основных производственных функций

та, номер текущего счета, номер расчетного счета, дата выписки расчетного счета и сумма, предъявленная к оплате).

Отсюда можно вывести два основных заключения:

1. Если род деятельности организации не меняется, она продолжает выполнять те же самые производственные функции и использовать в основном те же самые данные о производстве. Следовательно, стабильная база данных может быть создана только в том случае, если логическая структура базы данных основана на связях, образованных производственными функциями предприятия.

2. Потребности руководства в контроле и планировании будут изменяться в соответствии с тем, «как» выполняются функции, но не «что» должно быть выполнено («как» может означать, например, «автоматизированным способом», в то же время «что» может означать «прием заказов»). Кроме того, функции контроля и планирования всегда связаны с производственными функциями предприятия. Следовательно, если структура базы данных основана на связях, образованных производственными функциями, то почти все данные, необходимые для удовлетворения меняющихся потребностей руководства, будут обеспечены.

В соответствии с вышесказанным информацию об использовании данных можно разделить на два вида: информацию, связанную с производственными функциями, и информацию, связанную с функциями управления. Способы сбора этих видов информации коренным образом отличаются друг от друга.

3.3.1. Производственные функции предприятия

Собранная информация о производственных функциях предприятия и об использовании данных является основной исходной информацией процесса проектирования базы данных. Поэтому достоверность и полнота собранной информации являются существенным фактором. Для достижения этого необходимо, чтобы методика собеседований и техника документирования были строго формализованы.

Прежде чем начать обсуждение того, как проводить собеседования, необходимо определить, с кем их проводить. Поскольку целью собеседований является выявление производственных функций, лучше всего начать с определения тех производственных подразделений (в рамках сферы применения проекта), где выполняются наиболее существенные для данного рода деятельности функции. Например, перевозка грузов, продажа, производство товаров являются функциями важными в коммерческой деятельности. С другой стороны, такие функции, как управление запасами, анализ рынка, проверка отчетности, являются характерными для управленческой, но не производственной деятельности. Как только выявлены подразделения, выполняющие производственные функции, можно наметить кандидатуры для собеседований. Лучшим способом является рассылка каждому руководителю подразделения вопросников примерно такого содержания:

• Наименование работы для каждого из подчиненных ему (или ей) лиц.

Таблица 3.2. **Примеры наименований работы, функций и целей**

Наименование работы	Функция	Цель
Комплектация заказов [1])	Комплектовать товары со склада в соответствии с заказом	Выполнять заказы клиентов
Планирование запасов товаров [1])	Покупать товары с целью пополнения запасов	Поддерживать запасы товаров
Управление запасами товаров [2])	Определять оптимальное количество товаров и время для их закупки	Минимизировать капиталовложения при создании запаса товаров
Прием заказов [1])	Выписать заказ	Принять заказ клиента

[1]) Основная функция организации.
[2]) Управляющая функция, которая не учитывается в данный момент.

• Производственные функции, выполняемые на каждом рабочем месте.

• Краткое описание целей выполняемой работы.

Как только будут получены ответы, необходимо составить список всех наименований работ, выполняемых функций и целей (табл. 3.2). Затем необходимо рассмотреть каждую из работ и отнести ее к одному из видов деятельности: производственной или управленческой. После этого совместно с руководителем необходимо по каждой из работ отобрать для собеседований двух человек.

3.3.2. Проведение собеседований

Собеседование преследует три цели:

• Идентификация каждой производственной функции.

• Идентификация данных, требуемых для выполнения этих функций.

• Идентификация явных и неявных правил, определяющих, когда и как выполняется каждая функция.

Далее описана методика проведения собеседований, позволяющая собрать (а также документировать) необходимую информацию.

Каждое собеседование следует начинать с просьбы к собеседнику подробно описать функции или задачи, выполняемые им в течение рабочего дня. Когда собеседник опишет основные действия, решения и взаимосвязи, их следует задокументировать в виде блок-схемы (рис. 3.2). Впоследствии можно использовать эту блок-схему в качестве средства обратной связи для проверки того, что производственные функции и их последовательность сформулированы правильно. Затем следует повторить эту же процедуру для функций, выполняемых еженедельно, ежемесячно, ежеквартально и ежегодно.

После составления блок-схемы следует определить, какие документы, файлы или неформализованные ссылки используются при выполнении каждой функции, и записать все это в отдельный список. Результатом этих действий для блок-схемы, показанной на рис. 3.2, будет следующий информационный список:

1. Заказ.
2. Книга товаров.
3. Вызов склада (номер товара, заказанное количество, имеется в наличии, предполагаемая дата отгрузки).
4. Вызов торгового агента (номер товара, не имеющегося в наличии, номер заменяющего товара).
5. Журнал клиентуры.
6. Прейскурант.

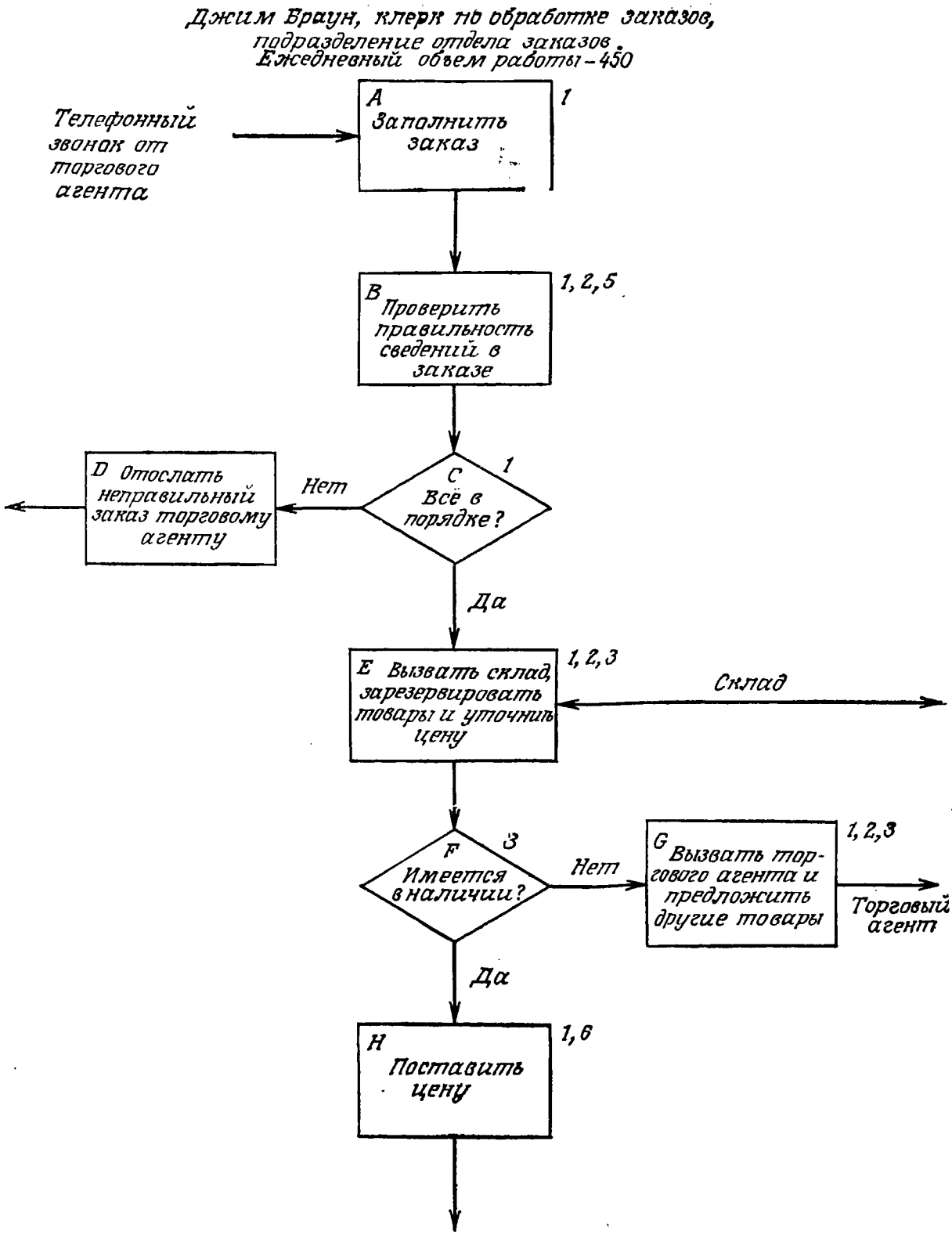


Рис. 3.2. Блок-схема выполнения производственных функций.

После того как собеседник понял содержание блок-схемы и согласился с ним, необходимо обсудить каждое отдельное действие (например, «Заполнить заказ»), каждое решение (например, «Имеется в наличии?») и каждую точку взаимодействия (например, «Вызвать склад»), чтобы определить требуемые для этого конкретные документы. Используя информационный список, необходимо поставить соответствующий номер документа (например, 2 для «Книги товаров») рядом с каждым графическим символом блок-схемы (рис. 3.2). Затем необходимо получить образец каждого обсуждаемого документа. Если документ не предусмотрен, как, например, в случае телефонного вызова, необходимо поместить рядом с графическим символом блок-схемы, обозначающим ссылку, обозначение вызова (см. «Вызов склада» в рассматриваемом примере).

И наконец, необходимо сопоставить каждый элемент данных, используемый или создаваемый в каждом документе, с каждой функцией на блок-схеме. Для этого надо обозначить каждый графический символ на блок-схеме уникальным идентификатором (например, A, B, C и т. д. на рис. 3.2). Далее следует составить список используемых во всех документах элементов данных и поместить рядом с каждым элементом данных идентификаторы функций, связанных с каждым документом, в которых используется этот элемент данных.

Не существует специальной формы для документирования явных и неявных правил, определяющих, когда и как выполняется каждый шаг блок-схемы (например, «торговый агент может передавать заказы только на склады своего округа» и т. д.). Важно то, чтобы эти правила были зарегистрированы. Они могут быть записаны или на самой блок-схеме, или на отдельном листе.

3.3.3. Собеседования с руководством о функциях управления

Другой вид информации, требуемый для построения концептуальной структуры базы данных, включает информацию, необходимую для более глубокого понимания коммерческой политики организации, определение функций контроля и планирования и требуемых для их выполнения данных, а также предположения о возможных изменениях в деятельности организации в будущем. Эта информация выявляется в собеседованиях с высшим и средним руководством. Поскольку содержание этой информации может меняться в зависимости от типа организации и персонала, не существует строгого формализма при документировании этой информации. Однако с целью уменьшения искажений и потерь информации рекомендуется записать беседу на магнитофон, если возможно, а затем

расшифровать. Если магнитофонная запись невозможна, надо тщательно запротоколировать беседу.

В качестве собеседника, представляющего руководство, следует выбирать такое лицо, в компетенцию которого входит определение целей и задач предприятия, формулирование стратегий достижения этих целей и управление выполнением планов реализации этих стратегий. Целью бесед является получение общего представления по следующим пунктам:

- Основные компоненты деятельности и их взаимодействие друг с другом.
- Внутренняя среда (т. е. организационная структура, расположение и т. д.).
- Внешняя среда, которая прямо или косвенно влияет на деятельность предприятия (законодательные органы, рынки сбыта и т. д.).
- Явная или подразумеваемая коммерческая политика, которая определяет поведение организации. (*Замечание*: часть этой информации может быть получена во время обсуждения внутренней и внешней среды.)
- Информация, требуемая для планирования деятельности (привести пример, если возможно).
- Предполагаемые изменения, которые могут влиять на род или сферу деятельности, либо на способы ведения деятельности.

Собеседниками, представляющими среднее звено, могут быть лица, непосредственно ответственные за одну или несколько областей деятельности. Цель этих собеседований заключается в том, чтобы получить более полное понимание следующих вопросов:

- Взаимодействие между различными областями деятельности.
- Правила и политика, определяющие повседневную деятельность.
- Виды информации, используемой для контроля и оценки функционирования.
- Возможное влияние предполагаемых изменений на производственную деятельность.

3.4. Преобразование информационных требований

Процесс преобразования информации, собранной во время собеседований, в форму, используемую при методологическом анализе, включает пять основных шагов:

1. Составление списка всех используемых и создаваемых элементов данных.
2. Определение производственных задач организации, их характеристик и используемых в них данных.

3. Определение задач управления, их характеристик и используемых данных.

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

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

Ниже эти шаги рассмотрены в деталях.

3.4.1. Идентификация элементов данных

При составлении списка всех элементов данных, создаваемых и используемых при выполнении производственных функций, зафиксированных в результате собеседований в виде блок-схем, наиболее часто встречающейся проблемой является проблема определения избыточности элементов данных. Одним из способов решения этой проблемы является выделение отдельных элементов данных из полученных документов и распределение их по родо-видовым спискам (таким, как даты, количества, запасы, наименования и т. д.). При этом достигаются две основные цели. Во-первых, процесс распределения данных по видам уменьшает возможность дублирования элементов данных. Во-вторых, вопросы определения каждого элемента данных могут быть разрешены с учетом контекста (т. е. документов и функций), в котором эти данные рассматриваются.

Как только составлены родо-видовые списки и решены вопросы избыточности, каждый элемент данных следует поместить в словарь данных, назначить ему уникальный идентификатор и составить описание (табл. 3.3).

Таблица 3.3. **Пример списка элементов данных**

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

3.4.2. Идентификация производственных задач

После того как составлен словарь элементов данных, можно приступить к анализу блок-схемы, полученной во время проведения собеседований, и определить связи между производственными задачами и данными. Как отмечалось выше,

производственные функции организации являются той основой, на которой может быть определено множество относительно стабильных связей данных. Для этого необходимо определить самый низкий уровень деятельности, который многократно использует уникальный набор данных. Такой уровень деятельности называется *задачей* и определяется следующим образом:

• Уникальная единица деятельности, состоящая из набора последовательно выполняемых шагов.

• Все шаги направлены на достижение одной и той же цели.

• На каждом шаге создается или используется один и тот же набор данных.

Связь между задачей и данными можно определить как уникальную связь, возникающую между элементами данных при их использовании для выполнения определенной задачи. Эти связи имеют первостепенное значение для концептуального анализа проекта. Поэтому очень важно, чтобы они были определены самым тщательным образом.

Процесс определения связей между задачами и данными начинается с анализа блок-схемы, построенной во время собеседований. Для анализа каждой блок-схемы и деления ее на задачи следует применять следующие правила:

• Задача должна выполняться в пределах одной функциональной области деятельности (т. е. задачи всегда определяются в рамках блок-схемы и никогда не принадлежат двум или более блок-схемам одновременно).

• Каждая задача должна состоять из набора последовательно выполняемых шагов или последовательно размещаемых символов на блок-схеме. Если встречается графический символ, обозначающий принятие решения (символы C и F на блок-схеме рис. 3.3), и одна из ветвей включает новое действие, предыдущая задача заканчивается и начинается новая (см. рис. 3.3, задачи 2 и 3).

• Каждый шаг внутри задачи должен выполняться в пределах одних и тех же временных ограничений (например, если между двумя шагами проходит значительное количество времени, как для символов E, F и G на рис. 3.3, необходимо определить новую задачу).

• Каждый шаг внутри задачи должен использовать один и тот же набор данных. (*Замечание*: подразумевается использование тех же самых элементов данных, но не номеров на блок-схеме, являющихся ссылками.) Если на каком-то шаге задачи создаются данные, которые используются на следующем шаге, они могут рассматриваться как один и тот же набор данных.

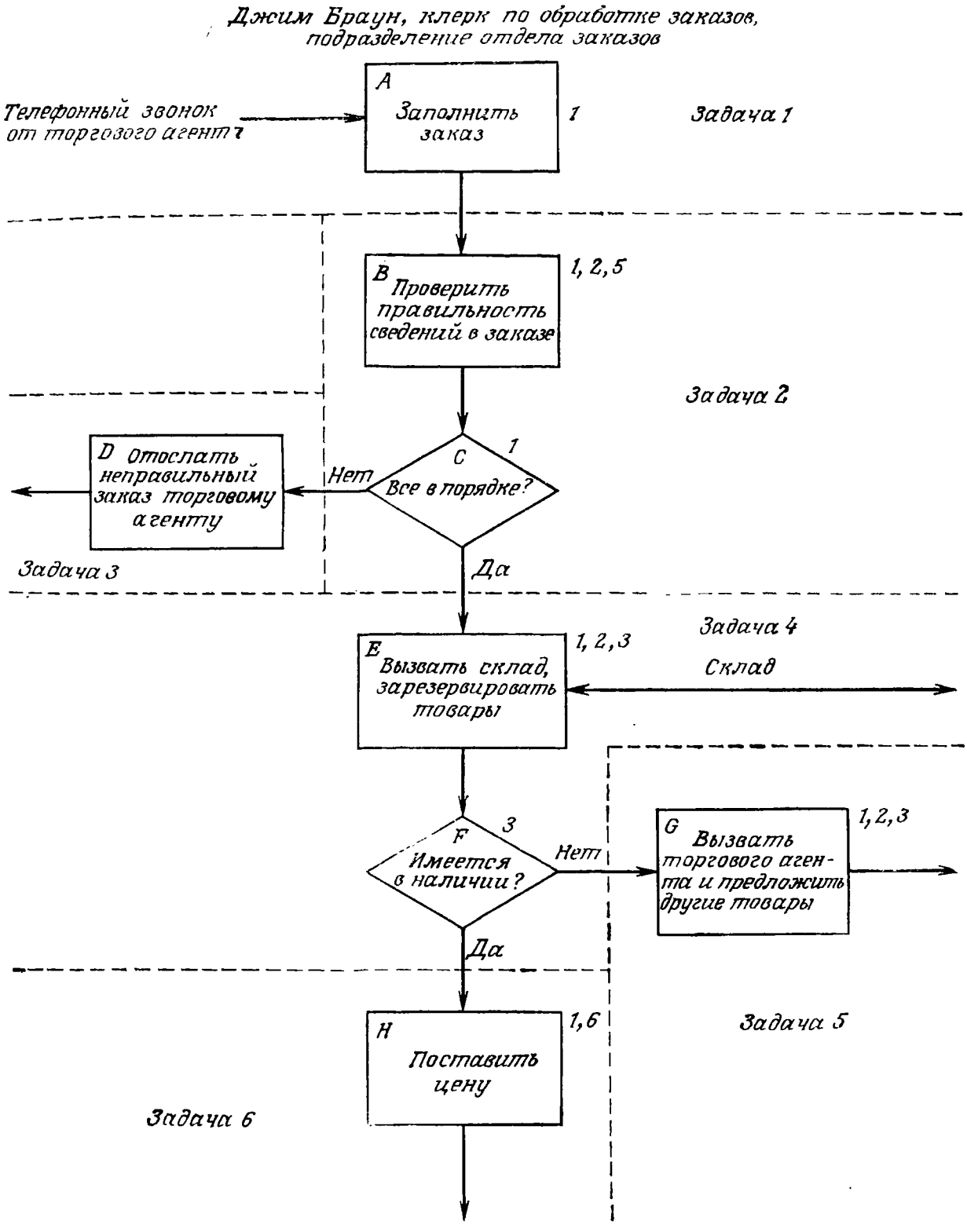


Рис. 3.3. Разбиение блок-схемы на задачи.

• Каждый шаг внутри задачи должен быть обязательно выполнен. Из этого правила имеется одно исключение, которое может быть показано на блок-схеме. Если блок-схема содержит принятие решения таким образом, как показано на рис. 3.4, все ветви после принятия решения должны быть

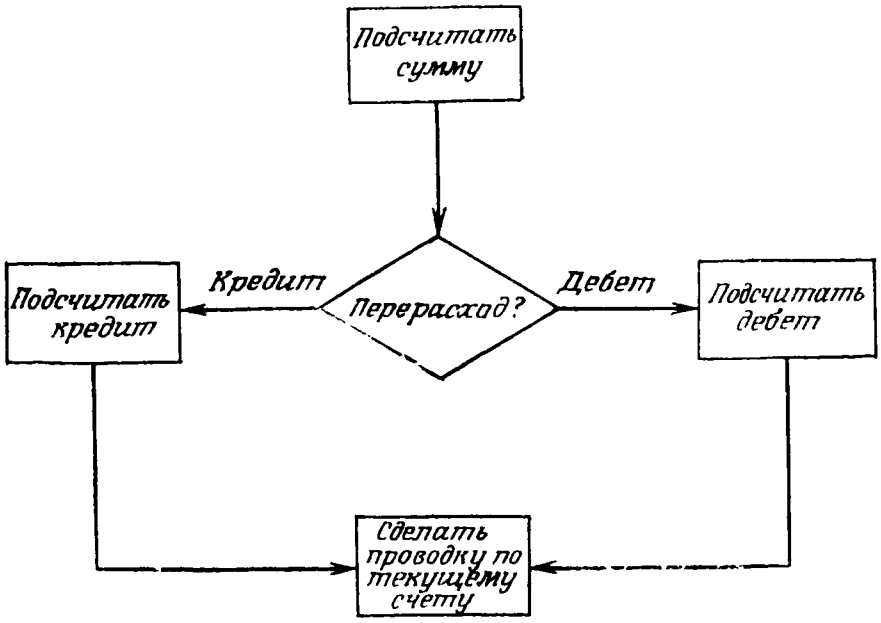


Рис. 3.4. Пример принятия решения по двум ветвям, которые должны быть включены в одну задачу.

включены в одну задачу, так как ни одна из ветвей не включает ни новых целей, ни новых данных, ни новых шагов.

Руководствуясь этими правилами, можно выделить на блок-схеме, показанной на рис. 3.3, следующие шесть задач:

• *Задача 1* состоит только из шага A, так как шаг B имеет другую цель и использует другой набор данных.

• *Задача 2* включает шаги B и C, поскольку они используют один и тот же набор данных и преследуют одну общую цель.

• *Задача 3* состоит только из шага D, поскольку он представляет отдельную ветвь решения, которая выполняется не всегда.

• *Задача 4* состоит из шагов E и F, так как оба шага преследуют одну и ту же цель и используют общий набор данных.

• *Задача 5* состоит только из шага G, поскольку он выполняется не всегда и имеет цель, отличную от целей шагов E, F и H.

• *Задача 6* состоит из шага H, поскольку он использует новый набор данных.

После того как все блок-схемы проанализированы и разделены на задачи, необходимо сравнить между собой блок-схемы, полученные в результате дублирующих собеседований, на совпадение задач. (*Замечание*: для собеседований в каждом отделе было выбрано по два человека, выполняющих одинаковую работу.) Если обнаружено несовпадение (обычно так оно и случается), надо сравнить отдельные графические символы блок-схемы:

1. Если блок-схемы отличаются только уровнем детализации, необходимо выбрать ту, которая лучше описывает работу в целом.
2. Если обнаружено несовпадение функций, надо проверить документы, соответствующие каждой функции (и сопутствующие замечания). В некоторых случаях одно и то же наименование работы может скрывать за собой различные функции, зависящие либо от служебного положения, либо от компетенции. В таких случаях надо выделить отдельные задачи и добавить их в список задач.
3. Если причину несовпадений трудно объяснить, необходимо обратиться к руководителю отдела с вопросом, как он понимает эти различия. (*Замечание*: не следует обсуждать с ним результаты собеседований, поскольку это может нарушить конфиденциальность и доверие, возникшее в процессе беседы.)

Когда все подобные ситуации будут разрешены, можно приступать к документированию связей между задачами и данными. И поскольку весьма желательно, чтобы были определены только неизбыточные задачи, следует упорядочить все блок-схемы отдела или организации. Упорядочение повышает вероятность того, что избыточные задачи будут обнаружены в процессе документирования. Этот процесс включает следующие действия (табл. 3.4):

- Каждой задаче должен быть присвоен цифровой идентификатор.
- Каждая задача должна быть кратко описана повелительной формой типа «глагол-дополнение», например «Принять заказ», «Резервировать заказанный товар» и т. п.
- Каждая задача должна быть отнесена либо к производственным задачам, либо к управленческим. (*Замечание:* за исключением особых случаев все задачи на блок-схемах являются производственными.)
- Для каждой задачи следует определить частоту ее выполнения и объем, отметив это в заголовке блок-схемы.
- Каждая задача должна быть отнесена к конкретной функциональной области деятельности организации, как, например, обработка заказов, производство товаров и т. п. (*Замечание:* не следует использовать наименования организации или отде-

Таблица 3.4. **Пример документирования связей между задачами и данными**

Номер задачи	Определение задачи	Тип	Частота	Объем	Отдел	Элементы данных
1	Выписать заказ	Производств.	Ежедневно	2 000	Прием заказов	1, 45, 50, 67
2	Проверить заказ	»	»	2 000	То же	4—17, 22, 27, 50, 117 120—124, 202
3	Выписать сообщение об ошибке	»	»	25	»	4—10, 200—204
4	Резервировать товар на складе	»	»	8 000	»	7—9, 62, 65
5	Запросить альтернативные товары	»	»	50	»	2, 7—9, 15
6	Справиться о цене	»	»	7 950	»	15, 62, 65, 75

ла, так как они не могут представлять функции, выполняемые двумя или более подразделениями.)

Поскольку объем информации для описания элементов данных и связей между задачами и данными бывает достаточно велик, для хранения, учета и обновления такой информации следует использовать автоматизированную систему, например автоматизированный словарь данных.

3.4.3. Идентификация задач управления

Задачи управления определяются путем анализа результатов собеседований с руководством и выделения областей деятельности, относящихся к функциям контроля и планирования. В качестве примера рассмотрим выдержку из протокола беседы с управляющим бригады лесорубов:

«Мой хозяин оценивает мою работу в соответствии с выработкой моих лесорубов. Каждый день я даю им заполнить карточки, из которых можно узнать, сколько и какой величины повалено деревьев и сколько из них поломано. Мой секретарь собирает эти карточки и ведет учет на каждую *смену,* так что я могу проверить, как они работают. Таким образом я могу установить и устранить причину, если *выработка* падает».

Анализ этой выдержки показывает, что управляющий выполняет задачу контроля «Учет выработки смен лесорубов». При выполнении этой задачи управляющий использует следующие элементы данных: дни работы, количество поваленных деревьев, размеры деревьев, количество поломанных деревьев, идентификатор смены.

«Мой хозяин оценивает мою работу в соответствии с выработкой моих лесорубов. Каждый *день* я даю им заполнить карточки, из которых можно узнать, *сколько повалено деревьев, какой они величины* и *сколько поломано.* Мой секретарь собирает эти карточки и ведет *учет на каждую смену,* так что я могу проверить, как они работают. Таким образом я могу установить и устранить причину, если выработка падает».

Задача контроля, полученная в результате этого анализа, должна быть добавлена к списку производственных задач с пометкой «Управленческая» в столбце «Тип задачи». Описанный выше процесс повторяется для всех протоколов собеседований с руководством, в которых выявлены задачи планирования и контроля. Если во время этого процесса обнаруживаются новые элементы данных, они добавляются к словарям данных так, чтобы было можно ссылаться на них с помощью цифровых идентификаторов.

3.4.4. *Идентификация текущих и будущих правил поведения, определяющих политику организации*

Процесс идентификации линий поведения и будущих изменений очень похож на процесс идентификации задач управления, поскольку и то и другое выявляется во время собеседований. В результате этого процесса должны быть получены два списка: список правил поведения и список возможных изменений.

Правила поведения определяют образ действий организации или то, как различные части организации соотносятся друг с другом. При анализе результатов собеседований необходимо обращать внимание на те предложения, в которых устанавливаются параметры, указывающие, как должно быть выполнено то или иное действие. Например, следующая выдержка из беседы содержит две линии поведения:

«Мы являемся децентрализованной организацией. Все *торговые агенты* связаны с *отдельными округами* и могут предлагать товары только из *торговой базы* своего округа. Кроме того, каждый *клиент обслуживается по округам*. В соответствии с этим клиент X может заключить одну *торговую сделку* в округе 1, а другую — в округе 2. Мы выбрали такую организацию из-за конкуренции, которая значительно меняется от округа к округу».

Анализируя подчеркнутые части текста, можно определить следующие линии поведения:

- Торговые агенты распределены по округам и могут продавать только те товары, которые имеются на торговой базе одного округа.
- Клиенты обслуживаются окружными торговыми базами, при этом каждый клиент может иметь различные торговые сделки в нескольких округах.

Будущие изменения включают все, что может влиять на сферу деятельности, текущую политику или связи организации с внешним окружением (законодательные органы и т. п.). Для идентификации изменений необходимо просмотреть протокол собеседования и выделить те фразы, которые подразумевают изменения и возможные последствия этих изменений. Например:

«В настоящее время мы имеем отдел закупок на каждой торговой базе. Мы делаем это для того, чтобы каждый управляющий мог контролировать общую картину, отражающую прибыль и убытки. Однако если цены поднимутся, то нам было бы выгоднее производить *централизованные закупки товаров крупными партиями*. Это — радикальное изменение, а вы знаете, как медленно иногда разворачивается руководство».

Возможное в будущем изменение «Централизованные закупки крупных партий товаров вместо децентрализованных» должно быть добавлено к списку изменений.

На этом завершается этап формулирования требований при проектировании баз данных. Описание терминологии и методов концептуального проектирования представлено в последующих трех главах.

3.5. Средства для формулирования и документирования требований

Здесь рассматриваются две проблемы, связанные с подготовкой персонала и инструментальными средствами. Подготовка специалистов по информационным системам (которые являются связующим звеном между пользователями и техническим персоналом, участвующим в проектировании баз данных) должна быть такой, чтобы они могли успешно осуществлять взаимосвязь с людьми на различных уровнях организации и извлекать соответствующую информацию. Проблемы взаимосвязи возрастают не только за счет неточностей и качественных (а не количественных) особенностей естественного языка, но также и потому, что неотъемлемой частью общих требований являются неформальные правила и линии поведения или такие вещи, о которых может знать каждый, кроме того, кто проводит собеседование. Необходимо, чтобы эти неопределенные правила и линии поведения были выражены явным образом, а правильность их толкования проверена пользователями. Для того чтобы различные лица, не являющиеся специалистами в области баз данных, могли четко определить свои прикладные задачи, необходимы инструментальные средства. Успешное решение этих проблем зависит от выбора подходящей модели данных и методологии.

Модель данных должна быть настолько простой и «естественной», чтобы она легко воспринималась конечными пользователями, являющимися специалистами в других областях. Поэтому модель, приемлемая для проектирования схемы реализации, здесь может оказаться неподходящей. Необходимо отметить, что модель, не имеющая математического обоснования, по некоторым аспектам подходит больше. Причина заключается в том, что, несмотря на желательность определения конечными пользователями всех представлений данных, способов их обработки и другой информации, наиболее важная роль принадлежит системным аналитикам и проектировщикам. Предполагается, что они смогут очистить информацию перед тем, как подвергнуть ее дальнейшей обработке в процессе

проектирования.

Из-за недостатка удовлетворительных методологий и инструментальных средств, из-за того, что основным методом является общение с людьми, общий анализ требований является, несомненно, весьма трудоемким и не свободным от ошибок процессом, требующим большого количества итераций для достижения приемлемого результата. Методы и средства, основанные на программировании, часто являются или слишком общими, чтобы их можно было успешно применять, или ориентированными на обработку, а не на использование данных. Существует очевидная потребность в методах и инструментальных средствах, которые могли бы использоваться при проектировании как баз данных, так и программного обеспечения. Необходимо, чтобы эти методы и средства учитывали реальные особенности информационных систем, в которых неполнота и изменение требований являются скорее правилом, нежели исключением. Основу проектирования должно составлять разумное сочетание точных требований и приближенных оценок.

Сложным, но полезным вычислительным средством является язык постановки и описания проблем PSL/PSA (Problem Statement Language/Problem Statement Analyzer) [150, 315—317]. Пользователь описывает проблему (на языке описания проблем PSL), как набор объектов предполагаемой системы баз данных, атрибутов этих объектов и связей между объектами. Возможно также описание объектов в виде комментариев, которые хранятся как часть определения объектов. Существует несколько типов объектов. Объекты типа ENTITY (СУЩНОСТЬ) и RELATION (ОТНОШЕНИЕ), соединенные связью RELATED (ОТНОСИТСЯ), могут быть использованы для представления информационной структуры рассматриваемой базы данных. В базе данных производственной компании, например, классы сущностей ПРЕДПРИЯТИЕ и СЛУЖАЩИЙ могут быть представлены PSL-объектами типа ENTITY. Между этими объектами типа ENTITY возможно существование отношения, представленного объектом типа RELATION и именуемого «РАБОТАЕТ-НА» (или некоторым другим словосочетанием). Связность объекта РАБОТАЕТ-НА может быть определена как 1 к 100 000, поскольку каждый конкретный служащий работает только на одном предприятии и максимум 100 000 служащих одновременно работают на конкретном предприятии. Для каждого из объектов типа ENTITY и RELATION может быть определено более подробное описание, включающее описание хранимых данных. Спецификации использования могут быть определены с помощью объектов типа PROCESS (ОБРАБОТКА), INPUT (ВХОД), OUTPUT (ВЫХОД) и др.

Объект типа PROCESS является транзакцией или сообщением проектируемой системы.

Информация, содержащая описание проектируемой системы, хранится в базе данных PSL/PSA. Предусмотрены средства для проверки согласованности и печати различных сообщений о хранящейся информации.

Другой системой анализа требований, которая может быть использована в качестве системы учета требований баз данных, является система CASCADE [1]. Она может быть использована подобно системе PSL/PSA. Информация, которую необходимо хранить в базе данных, представляется в виде связанных информационных объектов, объектов типа информационных множеств и сообщений. Прикладные программы представляются в виде объектов типа обработки и связаны с объектами описания информации, которыми они манипулируют.

Подобно PSA, система CASCADE хранит в базе данных описание проектируемой системы и может выдавать по требованию документацию в виде сообщений.

Словари данных

На этапах анализа требований, логического и физического проектирования, а также на этапах реализации жизненного цикла баз данных часто применяется каталогизация. Результатом этапа формулирования и анализа требований является множество выявленных данных и требований обработки, которые служат входной информацией для остальных этапов фазы анализа и проектирования. В настоящее время нет удовлетворительного программного обеспечения для документирования требований обработки. Однако имеется несколько пакетов программ, называемых словарями/справочниками данных, для документирования требований к данным. Словари/справочники данных (DD/D — data dictionary/directory) служат для централизованного управления информацией о базе данных. Их непосредственной функцией является хранение и поиск этих метаданных (например, неформализованных определений данных), а также выдача сообщений для администратора базы данных.

В последние годы пакеты DD/D появились на рынках сбыта. Пока еще ни один из пакетов DD/D не предусматривает выполнения всех возлагаемых на них функций, кроме того, необходимо расширить содержимое этих пакетов, включив в них программы (процессы обработки). Несмотря на это, предполагается, что пакеты DD/D предназначены для выполнения следующих функций:

1. *Хранение определений данных.* Хранение символических и закодированных определений данных, предназначенных как для машинной, так и ручной обработки.

2. *Справочные функции.* Выдача сообщений об определениях и связях элементов данных как для администратора баз данных, так и для пользователя.

3. *Генерация процедур контроля баз данных.* Создание программ контроля достоверности входных данных, данных тестирования программ и программ выборки данных.

4. *Обеспечение безопасности и целостности.* Контроль доступа к системе DD/D, обеспечение сохранности определений данных в соответствии с правилами и соглашениями, принятыми в организации.

5. *Сбор статистики об использовании.* Регистрация и хранение информации об использовании каждого элемента базы данных и программ обращения к этим данным.

6. *Обработка определений данных.* Автоматическое кодирование определений данных для систем управления базами данных (например, DBD и PSB для системы IMS), переопределение существующих элементов, добавление новых элементов.

Средства для конечных пользователей

Поскольку те, кто пользуется инструментальными средствами, могут быть специалистами в любой прикладной области, но при этом не быть посвященными в тонкости вычислительных систем и их функциональных возможностей, средства должны быть такими, чтобы их использование не требовало специальных знаний о вычислительных системах. В некоторых случаях может оказаться полезным знакомство с технологией программирования [38, 147, 259, 260, 333]. В общем случае рекомендуется выбрать самый высокий уровень представления, обеспечивающий адекватность информации для целей проектирования базы данных. В существующих работах по проектированию баз данных набор информационных требований организации и их анализ редко применяются в качестве основы для разработки инструментальных средств.

Поскольку необходимо описывать процессы, связанные с ними представления данных и соответствующую информацию для последующего анализа, средства описания должны обеспечить точность описываемой информации, но в то же время не слишком ограничивать ее. Например, несмотря на то что спецификации, написанные на одном из существующих языков программирования, таком, как Кобол или PL/1, могут быть очень точными, возможно искажение прикладного представления: первоначальные намерения могут быть так замаскированы, что их практически невозможно будет обнаружить, особен-

но с помощью автоматизированных средств. Здесь необходимо иметь язык очень высокого уровня, основанный на модели, являющейся простой и понятной пользователю, но при этом полностью независимой от какой-либо системы.

Имея такие средства, конечный пользователь сможет точно и правильно описать все прикладные задачи, так что системные аналитики и проектировщики баз данных смогут легко объединить эту информацию с информацией, полученной во время собеседований. Что еще более важно, информация, описанная с помощью таких средств, может быть легко обработана на ЭВМ и может служить входной информацией в других задачах. Возникает вопрос: можно ли разработать такие средства? Наблюдая за прогрессом в развитии таких высокоуровневых языков для обработки баз данных, как QBE, SEQUEL, QUEL, CONVERT и др. [64, 290, 304, 357], можно надеяться, что высокоуровневые языки прикладных спецификаций, обладающие требуемыми характеристиками, также могут быть созданы.

Решению этой проблемы в будущем могут значительно способствовать следующие факторы: 1) расширение роли администратора баз данных, как промежуточного звена между пользователями и специалистами в области вычислительной техники; 2) развитие общих информационных моделей организации; 3) расширение использования средств вычислительной техники руководящим звеном, в частности в коммерческих школах для высшей администрации; 4) повышенное внимание, которое стали уделять специалисты по электронно-вычислительной технике человеческим факторам в противовес эффективности использования ЭВМ; 5) развитие других средств технологии программного обеспечения, применимых в рассматриваемой области. Эти факторы совместно с развитием средств проектирования баз данных могут помочь в решении проблем, стоящих сегодня перед аналитиками и проектировщиками баз данных.

Часть II

Концептуальное проектирование

Глава 4. Концептуальное моделирование данных

Как подчеркивалось в гл. 2, при проектировании баз данных можно выделить четыре фазы: анализ требований, концептуальное проектирование, проектирование реализации и физическое проектирование. Вопросы анализа требований обсуждались в гл. 3, а следующие три главы будут посвящены концептуальному проектированию.

4.1. Основы концептуального проектирования

Концептуальное проектирование оперирует информацией, независимой от любой фактической реализации (т. е. от любой конкретной системы технического или программного обеспечения). Цель концептуального проектирования именно в том и состоит, чтобы представить информацию в доступной пользователю форме, не зависящей от спецификаций системы, но реализуемой несколькими системами.

Представление информации

В последние годы предложено значительное количество средств проектирования баз данных и описания информационных требований. Однако большинство из этих средств касаются только части проблемы и неудобны для использования. Камнем преткновения является связь между способами представления информационных требований пользователей и первыми этапами методологии проектирования.

Приемлемый механизм представления должен отображать точку зрения и опыт пользователя. Проблема заключается в том, что механизм представления, ориентированный на пользователя, не всегда годится для базы данных. В то же время механизм представления, ориентированный на проектирование, заставляет пользователя искать способы, позволяющие представлять информацию в форме, удобной для обработки. Во многих случаях такие формы представления могут содержать в замаскированном виде важные проектные решения, которые

сами по себе не различаются методологией проектирования. Все это ограничивает сферу и возможности процесса проектирования.

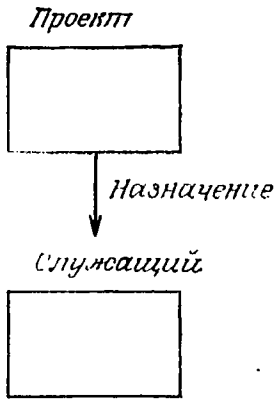
Обоснование механизма высокоуровневого представления

В большинстве случаев пользователи описывают свои информационные требования в терминах сущностей, атрибутов и связей (диаграммы типа «сущность-связь», или ER-диаграммы, будут обсуждаться в разд. 4.3) или в терминах записей, элементов и наборов, используя язык описания данных СУБД. Очевидно, что, если пользователь при описании своих информационных требований вместо высокоуровневого представления в форме ER-диаграмм ограничен языком описания данных низкого уровня, большое количество потенциальных возможностей проектирования может быть не реализовано. В то же время, несмотря на то что ER-диаграммы являются высокоуровневым представлением, они не дают однозначного решения при определении пользователями своих требований в такой форме. Например, на рис. 4.1 показано, что для представления факта назначения служащего для работы над заданным проектом могут быть использованы по крайней мере три различные структуры.

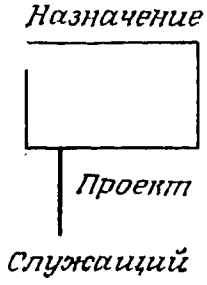
В зависимости от точки зрения пользователя назначение может быть представлено как связь (1), сущность (2) или атрибут (3). Вполне вероятно, что в подобной ситуации несколько пользователей могут описать одни и те же факты разными способами.

Одним из наиболее трудных аспектов при разработке систем, удовлетворяющих требованиям многих пользователей, является интеграция представлений пользователей. Это является сложной задачей даже тогда, когда приходится иметь дело с каким-либо одним классом структур, например только сущностей или только связей. Трудности возрастают вдвое, если для представления одних и тех же концепций используются различные структуры. Таким образом, для проектировщика имеются два основных довода в пользу применения в процессе проектирования абстракций высокого уровня. Во-первых, сущности, атрибуты и связи не всегда распознаются явным образом и поэтому проектные решения часто носят размытый (нечеткий) характер. Во-вторых, применение общей высокоуровневой структуры представления для концептуальных информационных структур позволяет упростить проблему проверки согласованности.

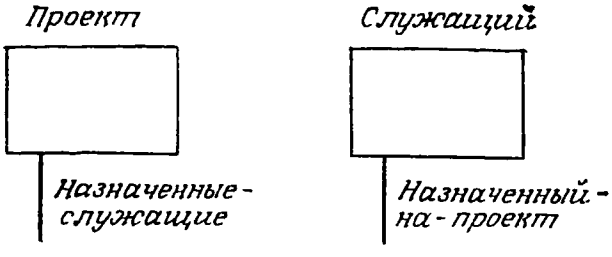
Необходимо остановиться еще на одном вопросе. Будет ли приемлемым для пользователей такой механизм представления, который не делает различий между сущностями, атрибу-



(1) Сущность проект
Сущность служащий
Связь назначение
между проектом и
служащим



(2) Сущность назначение
Атрибут проект
Атрибут служащий



(3) Сущность проект
Атрибут назначенные-
служащие
Сущность служащий
Атрибут назначенный-
на-проект

Рис. 4.1. Альтернативное представление элементов данных.

тами и связями? Мы надеемся, что именно такое представление будет для пользователей предпочтительнее. Интуитивно это может показаться противоестественным тем из нас, кто рассматривает данный вопрос с позиций проектировщика. Однако мы часто забываем, что пользователей вовсе не интересует то, как хранится информация. Их интересуют только ответы на поставленные ими вопросы. Известно, что пользователи не имеют ни малейшего желания разбираться в физической структуре базы данных, если только это не связано с решением их непосредственных задач. То же самое можно сказать относительно ЯОД и уровней абстракции. Только *содержание* информации пред-

ставляет интерес для пользователей, информационная *структура* не имеет для них никакого значения. Это не означает, что, разрабатывая прикладные программы, пользователи-программисты не будут обращаться к структуре базы данных. Однако при проектировании базы данных прикладные программисты являются не единственными, кто определяет информационное содержание базы данных. Желательно, чтобы это были конечные пользователи, которых не интересует, *как* проектируется база данных, но которые лучше понимают, *что* им необходимо.

Подход к проектированию

Мы будем рассматривать концептуальное проектирование с двух точек зрения: объектного представления и моделирования сущностей. На рис. 4.2 показаны уровни представления информации для этих двух подходов.

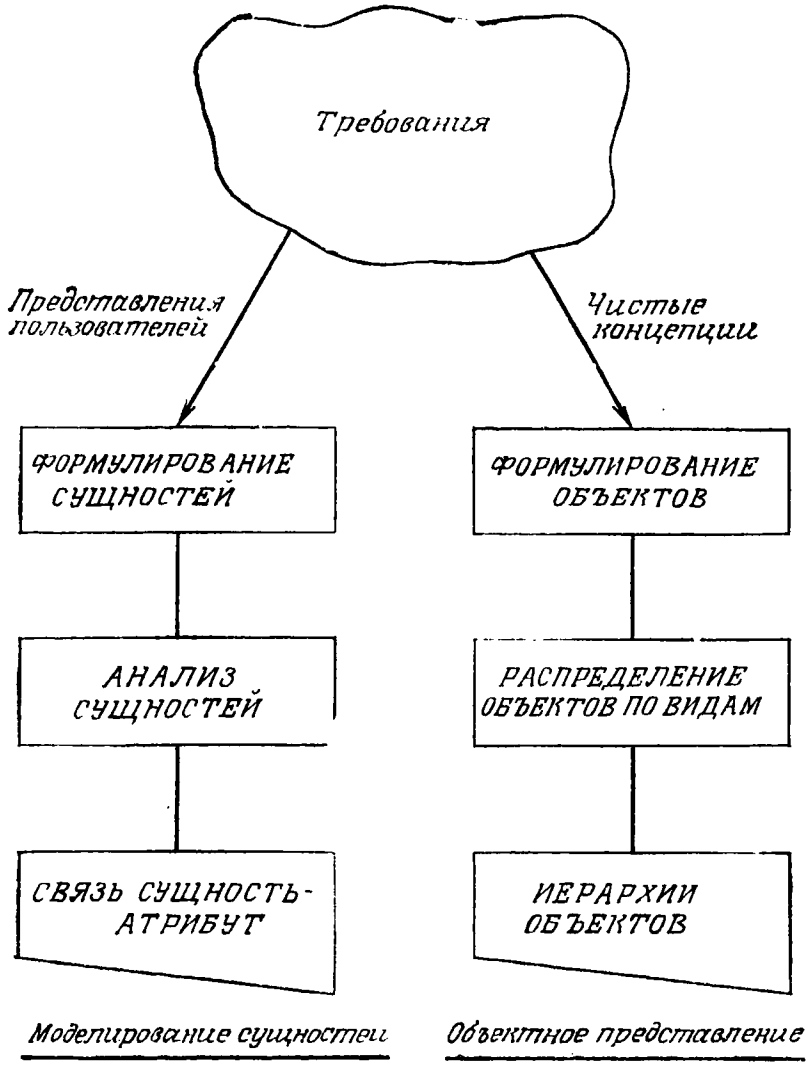


Рис. 4.2. Два представления концептуального проектирования.

Первый подход включает формулирование, определение и интеграцию объектов высокого уровня, используемых для построения модели. Основное внимание при этом уделяется интеграции понятий (концепций), представляющих объекты. Попросту говоря, технология этого подхода заключается в объединении элементов, которые в свою очередь могут состоять из отдельных частей. Эти элементы рассматриваются как объекты, объединенные в виде иерархии объектов. Основными вопросами, решаемыми при этом подходе, являются следующие:

1. Что понимается под объектами?
2. Каково контекстное содержание этих объектов?
3. Каковы описательные свойства каждого объекта?
4. Каковы идентификационные свойства каждого объекта?

Второй подход к концептуальному проектированию, моделирование сущностей, заключается в моделировании и интеграции представлений пользователей в терминах диаграмм сущностей. Техника построения диаграмм сущностей, являясь в основном неформализованной, имеет конечным результатом спецификацию сущностей, атрибутов и связей.

С представлением информационных требований пользователей при проектировании информационной структуры связаны четыре основных проектных решения:

1. Определение сущностей.
2. Определение атрибутов сущностей.
3. Определение ключевых атрибутов сущностей.
4. Определение связей между сущностями.

4.2. Объектное представление

Концептуальный подход к информационным системам берет свое начало с публикации рабочей группы по базам данных ANSI/SPARC [8]. Несмотря на то что этой проблеме было уделено много внимания в литературе [116, 242, 275], до сих пор не существует общего соглашения ни по терминологии, ни в отношении основных аксиом. Не вдаваясь в подробности дебатов по данному вопросу, мы сконцентрируем внимание на методах представления информации, развитых в работе Смита и Смит [294] и исследованиях Новака [243].

Этот подход основан на идеализации уровня концептуального проектирования. Он предполагает существование «чистого» множества понятий (концепций), которые должны быть интегрированы. Они являются «чистыми» в том смысле, что каждое понятие заключает в себе отличное от других значение, смысл которого доступен проектировщику.

В этом подходе основной упор сделан на главном вопросе концептуального проектирования — интеграции понятий. Для

правильного понимания этого подхода необходимо ввести понятие абстракции, которое часто используется для мысленного представления или описания сложных структур проблемной области.

Абстракция

Словарное определение термина «абстрактный» таково: «не связанный с каким-либо конкретным явлением или предметом; трудный для понимания, идеализированный» [334]. Это принципиальное определение служит фундаментальной предпосылкой того, что абстракции являются основой, при помощи которой человечество воспринимает сложные системы и управляет ими. Абстракция представляет собой совокупность деталей конкретного предмета (явления, понятия), которая может быть соответствующим образом рассмотрена и поименована как целое. Например, совокупность напечатанных страниц, собранных вместе, называется книгой; почтовым адресом называется ссылка на физическое строение, которое состоит из помещений. Абстракции существуют в двух формах: абстракция состояния системы (т. е. абстрактные объекты) и абстракция преобразования системы (т. е. абстрактные действия).

Для целей проектирования баз данных будут использоваться абстрактные объекты или понятия (концепции). Целью концептуального проектирования является определение взаимосвязанной структуры абстрактных объектов. Обычно база данных состоит не из независимых объектов, при этом объекты соотносятся друг с другом двумя способами: как класс или как совокупность. В соответствии с этим существует два способа формализации объектов: агрегация и обобщение. *Агрегация* формирует объект как связь между другими объектами. *Обобщение* формирует объект из класса других объектов.

Агрегация

В математическом смысле понятие агрегации соответствует понятию декартова произведения. Объекты в этом случае формируются как связь между другими объектами.

На рис. 4.3 приведено несколько примеров, которые иллюстрируют идею агрегации. В первом случае связь между четырьмя объектами ЧЕЛОВЕК, КОМНАТА, ГОСТИНИЦА и ДАТА выражается через объект БРОНИРОВАНИЕ. Этим самым выражается тот факт, что человек бронирует номер в гостинице на определенную дату. В этой конкретной агрегации наименования индивидуальных объектов отбрасываются и связь именуется как целое.

Агрегацию можно рассматривать по-другому, как именную форму глагола в связи. Например, БРОНИРОВАНИЕ является именной формой глагола «бронировать». Однако это годится не во всех случаях, как, например, для агрегации «преподаватель читает предмет в течение семестра», для которой наименование КУРС подходит лучше, чем ЧТЕНИЕ. Поэтому наименования агрегатных объектов надо выбирать очень тщательно, поскольку они заключают в себе семантику агрегации. Важно

Агрегация

Связь между объектами	Агрегатный объект
Человек бронирует номер в гостинице на определенную дату	БРОНИРОВАНИЕ
Преподаватель читает предмет в течение семестра	КУРС (ЧТЕНИЕ)
Автомашина перевозит груз от места загрузки до места назначения	ПЕРЕВОЗКА (РЕЙС)

Рис. 4.3.

отметить, что каждый компонент объекта является не множеством объектов, а *простым* объектом. Например, ПРЕПОДАВАТЕЛЬ и ПРЕДМЕТ являются допустимыми компонентами КУРСА, однако множество СТУДЕНТОВ на КУРСЕ не является допустимым компонентом.

Обобщение

Рис. 4.4 иллюстрирует идею обобщения. Например, класс объектов {СОБАКА, КОШКА, СЛОН} может быть обобщен в объект ЖИВОТНОЕ. В этом обобщении подчеркивается общая

Обобщение

Класс объектов	Родовой объект
{СОБАКА, КОШКА, СЛОН}	ЖИВОТНОЕ
{ПОВОЗКА, ТЯГАЧ, ВЕЛОСИПЕД, ...}	ДОРОЖНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ
{ЯБЛОКО, БАНАН, ГРУША}	ФРУКТ

Рис. 4.4.

природа объектов СОБАКА, КОШКА и СЛОН, а их индивидуальные отличия (например, собака лает, слон имеет хобот) игнорируются. В общем случае, если класс $\{O_1, \ldots, O_n\}$ может быть обобщен в O, говорят, что O_i является *категорией* (относится к категории) O. Например, СОБАКА является категорией (относится к категории) ЖИВОТНЫХ. Необходимо отме-

тить, что невозможно определить, является ли O обобщением $\{O_1, \ldots, O_n\}$, пока не назван каждый объект. Именно эти наименования объектов являются носителями семантики обобщения.

Агрегация и обобщение не являются взаимоисключающими категориями. Агрегатный объект может быть обобщением некоторого класса объектов, в этом случае он называется родовым объектом. Родовой объект в свою очередь может являться агрегацией некоторой связи между объектами, в этом случае он называется агрегатным объектом. В общем случае каждый объект может быть и агрегатным, и родовым. Однако некоторые компоненты или категории объекта могут не представлять интереса и поэтому они не отображаются в концептуальной модели. Объект называется *первичным*, если ни компоненты, ни категории объекта не представляют интереса.

Иерархия абстракций

Если несколько раз последовательно применить к некоторым объектам обобщение или агрегацию, образуется иерархия объектов. Рис. 4.5 иллюстрирует иерархию обобщения различных категорий транспортных средств. Класс {VW — ТЯГАЧ, FORD —ТЯГАЧ, GM — ТЯГАЧ} обобщается в ТЯГАЧ. Класс {ТЯГАЧ, ЛАЙНЕР, САМОЛЕТ} обобщается в МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ. Класс {ДОРОЖНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ, МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ, ВОЗДУШНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ} обобщается в СРЕДСТВО ПЕРЕДВИЖЕНИЯ. Объект МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ заштрихован, так как он используется в нескольких последующих примерах.

Рассматривая структуру иерархии обобщения, можно отметить две важные особенности. Во-первых, некоторые категории могут одновременно принадлежать нескольким объектам. Например, ТЯГАЧ одновременно является и ДОРОЖНЫМ СРЕДСТВОМ ПЕРЕДВИЖЕНИЯ, и МОТОРНЫМ СРЕДСТВОМ ПЕРЕДВИЖЕНИЯ, а САМОЛЕТ является и МОТОРНЫМ СРЕДСТВОМ ПЕРЕДВИЖЕНИЯ, и одновременно ВОЗДУШНЫМ СРЕДСТВОМ ПЕРЕДВИЖЕНИЯ. Во-вторых, иерархия обобщения может иметь не один, а несколько корней. Иерархия на рис. 4.5 имеет только один корень, однако если этот корень удалить, то получится иерархия с тремя корнями: ДОРОЖНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ, МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ и ВОЗДУШНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ.

Согласно требованию агрегации, между каждым объектом и его компонентами существует функциональное $(1 : n)$ соответствие. Например, на рис. 4.6 каждая ПЕРЕВОЗКА опреде-

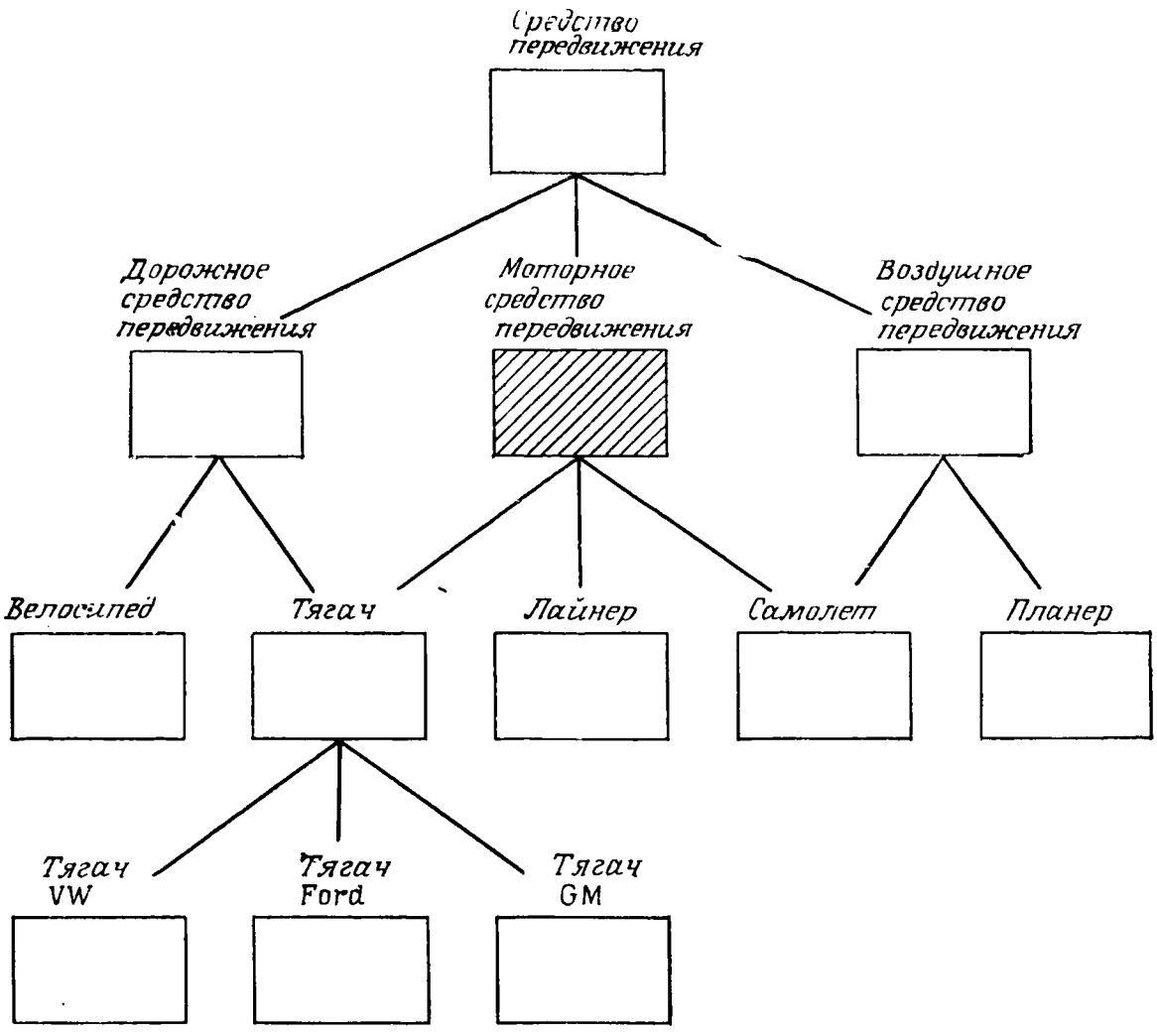


Рис. 4.5. Иерархия обобщения.

ляет единственное МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ, единственный ПУНКТ НАЗНАЧЕНИЯ и единственный ГРУЗ. Это характерное свойство иерархии агрегации отличает ее от иерархий других типов. В частности, функциональное соответствие может быть использовано для того, чтобы определить, который из двух рассматриваемых объектов является компонентом другого. Например, МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ не может быть компонентом ИЗГОТОВИТЕЛЯ, поскольку изготовитель производит много моторных средств передвижения.

Сравнивая рис. 4.5 и рис. 4.6, интересно отметить различные структурные контексты, в которых появляется объект МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ. На рис. 4.5 показаны только категории — МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ является категорией СРЕДСТВА ПЕРЕДВИЖЕНИЯ, но

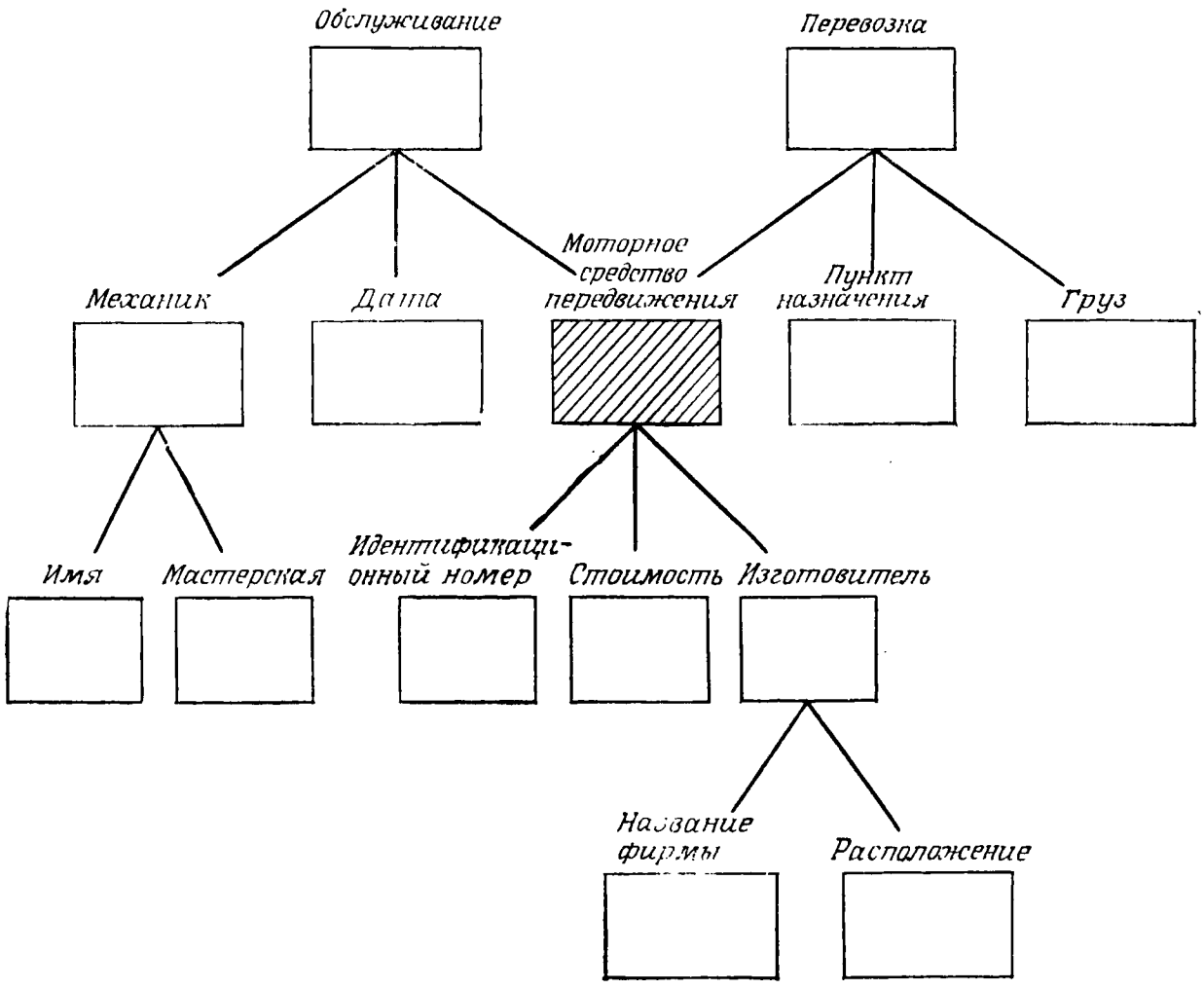


Рис. 4.6. Иерархия агрегации.

не компонентом. На рис. 4.6 показаны только компоненты — СТОИМОСТЬ является компонентом МОТОРНОГО СРЕДСТВА ПЕРЕДВИЖЕНИЯ, но не категорией. Ребрам иерархий агрегации и обобщения может быть дана следующая интерпретация: ребро иерархии обобщения можно прочитать как «является» («is-a»), а ребро иерархии агрегации как «является частью» («is-part-of»). Например, МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ «является» СРЕДСТВОМ ПЕРЕДВИЖЕНИЯ, а СТОИМОСТЬ «является частью» МОТОРНОГО СРЕДСТВА ПЕРЕДВИЖЕНИЯ.

Концептуальная модель должна включать иерархии абстракции и обобщения для всех концепций (понятий), необходимых для поддержки прикладных представлений. В принципе иерархия агрегации и иерархия обобщения в концептуальной модели могут быть определены отдельно одна от другой. Однако можно достичь большей модульности, определяя структуру каждого объекта как единого блока. Чтобы отделить в этой

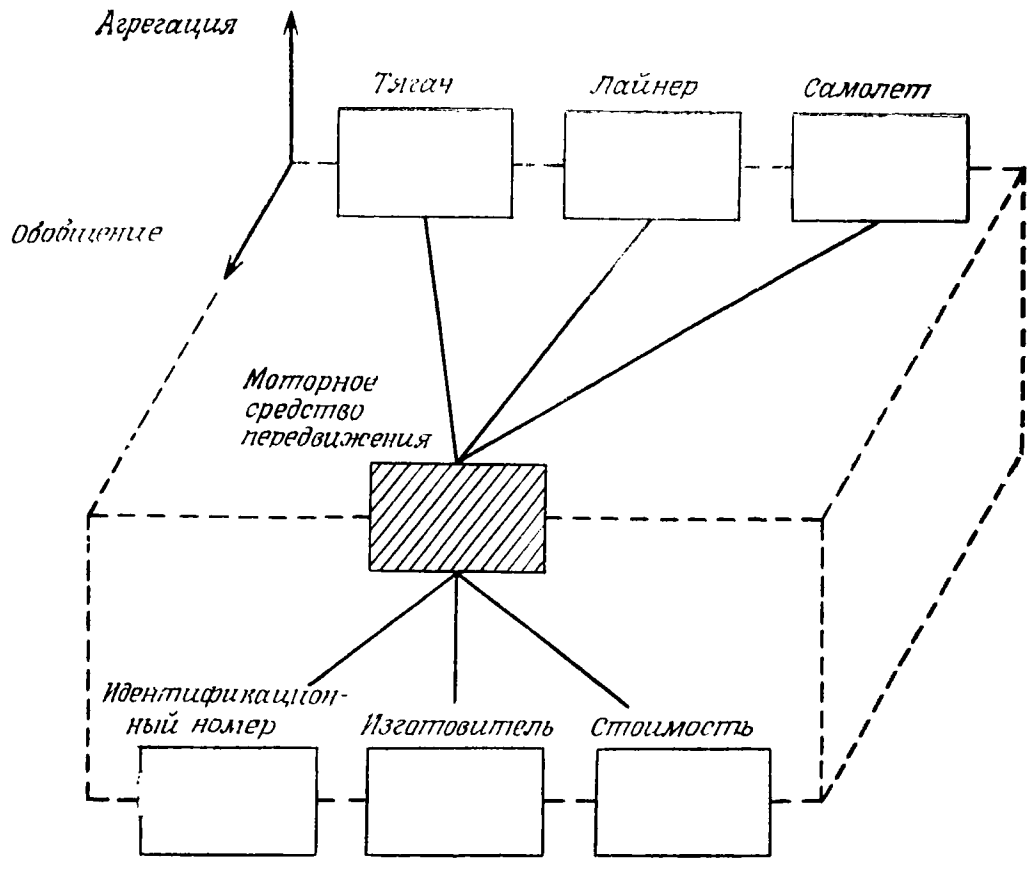


Рис. 4.7. Объединение иерархий обобщения и агрегации.

структуре компоненты от категорий, полезно разместить компоненты в плоскости страницы, а категории — в перпендикулярной плоскости. Пример структуры МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ, построенной таким образом, приведен на рис. 4.7.

Такие трехмерные диаграммы могут быть рекурсивно расширены включением в них структур компонентов и категорий. Часто эти диаграммы являются наиболее приемлемым средством построения концептуальных структур, и поэтому они будут использоваться в последующих разделах. Однако, поскольку любые два объекта могут быть (в принципе) обобщены или агрегированы, концептуальные модели не всегда удается точно привязать к трехмерной системе. Некоторые примеры подобных ситуаций будут рассмотрены позднее.

Представления пользователя

На рис. 4.6 показана иерархия агрегации, которая включает МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ. Действуя сверху вниз, можно описать эту иерархию. ОБСЛУЖИВАНИЕ имеет три компонента: МЕХАНИК, который выполняет обслу-

живание, ДАТА обслуживания и обслуживаемое МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ. Необходимо отметить, что МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ является одновременно компонентом объектов ОБСЛУЖИВАНИЕ и ПЕРЕВОЗКА. МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ имеет свои собственные компоненты: ИДЕНТИФИКАЦИОННЫЙ НОМЕР, СТОИМОСТЬ и ИЗГОТОВИТЕЛЬ. Значение компонентов МЕХАНИК и ИЗГОТОВИТЕЛЬ понятно из их названий.

Приведенное описание выполнено безотносительно к точке зрения какого-либо пользователя. Представим, однако, что мы рассматриваем модель с точки зрения пользователя, которого интересует моторное средство передвижения. Такой пользователь может рассматривать МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ как сущность, ОБСЛУЖИВАНИЕ — как связь (т. е. механик обслуживает моторное средство передвижения в определенный день), ПЕРЕВОЗКА — как связь (т. е. моторное средство передвижения перевозит груз по назначению), а ИДЕНТИФИКАЦИОННЫЙ НОМЕР, СТОИМОСТЬ и ИЗГОТОВИТЕЛЬ — как компоненты (рис. 4.8, *а*). С другой стороны, пользователь, которого интересуют изготовители, будет считать ИЗГОТОВИТЕЛЬ сущностью, МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ — связью (т. е. изготовитель определяет стоимость и идентификационный номер), а НАЗВАНИЕ ФИРМЫ и МЕСТО НАХОЖДЕНИЯ — компонентами (рис. 4.8, *б*). Наконец, пользователь, интересующийся обслуживанием, как таковым (например, управляющий обслуживанием), будет считать МЕХАНИК, ДАТА и МОТОРНОЕ СРЕДСТВО ПЕРЕДВИЖЕНИЯ компонентами (рис. 4.8, *в*).

Из предыдущих рассуждений может быть выведено важное заключение, а именно: нельзя заранее жестко интерпретировать объект как сущность, связь или компоненту. Такая интерпретация может быть определена только с точки зрения конкретного пользователя. Если пользователь рассматривает объект как сущность, то его «исходные» объекты становятся связями, а «порожденные» объекты — компонентами. Различные пользователи могут выбрать различные объекты в качестве сущностей. Представление концептуальной модели, требующее жесткой интерпретации, искусственно ограничивает интеграцию понятий (концепций). Это искусственное ограничение обычно проявляется через усложнение структуры и некорректное обновление.

4.3. Моделирование сущностей

Другим подходом, с точки зрения которого может быть рассмотрено концептуальное проектирование, является моделирование сущностей. Этот подход является, по-видимому, наиболее

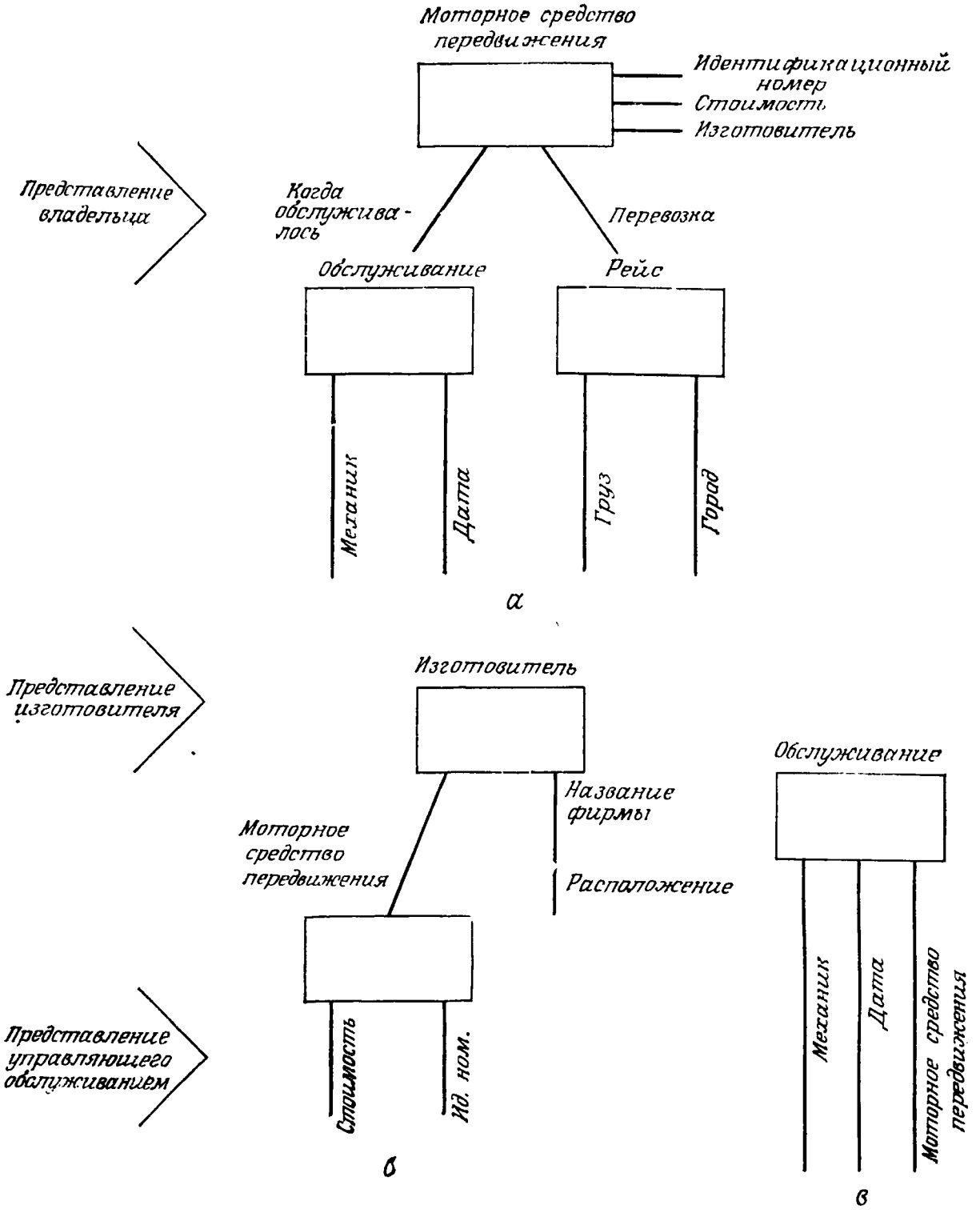


Рис. 4.8. Различные концептуальные представления системы средств передвижения.

широко известным и практикуемым из всех подходов. Он берет свое начало со времени первых попыток использования систем управления базами данных в середине 60-х годов и впервые появился в работе Бахмана в 1969 г. в связи с использованием диаграмм структур данных. Были предложены графические обозначения, состоящие из блоков и стрелок, для описания информационной структуры в системе управления базами данных IDS. Формализация и расширение этих обозначений, предпринятые Сенко и его коллегами в 1973 г., привели к развитию модели набора сущностей. Продолжил работу в этом направлении Холл [141], а затем Чен [68] ввел модель «сущность-связь», являющуюся расширением диаграмм структур данных, и развил методологию проектирования. Совсем недавно понятие категории сущностей было введено Бахманом и Дейа [19] в ролевой модели и Палмером [249] в модели типов наборов записей.

Для представления информации в модели «сущность-связь» конструктивными элементами модели служат сущности, атрибуты и связи. Основным конструктивным элементом является сущность. Пользователь описывает интересующие его объекты предметной области с помощью сущностей, затем определяет свойства сущностей, используя атрибуты, и, наконец, описывает соответствия между сущностями, используя связи.

Сущности

Сущность представляет собой основное содержание того явления или процесса, о котором необходимо собрать информацию, она является узловой точкой сбора информации. В качестве сущности может выступать личность, место или вещь, информацию о которых необходимо хранить. Необходимо различать такие понятия, как тип сущности и экземпляр сущности. Понятие *тип сущности* относится к набору однородных предметов или вещей, выступающему как целое. *Экземпляр сущности* относится к конкретной вещи в наборе. Например, типом сущности может быть СЛУЖАЩИЙ, а экземпляром сущности — Джон Джонс, Билл Смит, Джейн Джонс и т. д.

Атрибуты

Средством, с помощью которого определяются свойства сущностей, являются атрибуты. *Атрибут* — это поименованная характеристика сущности. Его наименование должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различных типов сущностей (например, ЦВЕТ может быть определен для многих сущностей — ЛИЧНОСТЬ, СРЕДСТВО ПЕРЕДВИЖЕНИЯ и т. д.). Хотя сущности существуют сами по себе, атрибуты используются для определения

того, какая информация должна быть собрана о сущности. Примерами атрибутов для сущности СЛУЖАЩИЙ являются ИМЯ, АДРЕС, ДАТА-РОЖДЕНИЯ и т. д. Здесь также существует основное различие между типом и экземпляром. Тип атрибута ДАТА-РОЖДЕНИЯ имеет много экземпляров или значений: 1 мая 1940 года, 29 сентября 1971 года и т. д. Однако каждому экземпляру сущности присваивается только одно значение атрибута.

Атрибут имеет следующие характеристики:

1. *Наименование.* Уникальное обозначение атрибута.
2. *Описание.* Повествовательное изложение смысла атрибута.
3. *Роль.* Конкретное использование атрибута.

Атрибут может быть использован в любой роли, описанной ниже.

Наиболее часто встречающейся ролью атрибута является описание свойства сущности (т. е. описание информации, представляющей интерес). Другой важной ролью является идентификация сущности, когда атрибут может использоваться для однозначного распознавания экземпляров сущности. Например, атрибут СЛУЖЕБНЫЙ НОМЕР, имеющий уникальный набор значений, позволяет отличать друг от друга экземпляры сущности СЛУЖАЩИЙ, даже если несколько служащих имеют одно и то же имя. Среди других ролей атрибута необходимо отметить: 1) представление связей между сущностями; 2) использование в процессе получения других выводимых величин; 3) обеспечение информацией, которая используется в особых случаях [19], например диапазон значений домена/типа, вероятность существования, количество экземпляров, длина, единица измерения.

Связи

Под связями понимаются ассоциации между одинаковыми или различными типами сущностей. Сущности соотносятся в предметной области между собой, а механизм связей используется для отображения этого соответствия в модели. Для понимания практического использования механизма связей важными являются следующие характеристики: наименование связи, степень ассоциативности, избирательность, однозначность, время существования и идентификатор.

Связь имеет *наименование.* Выбор наименования заключает в себе определенный смысл. Например, связь БЫТЬ-ЗАПОЛНЕННЫМ предоставляет пользователю некоторую информацию. Она содержит также определенную направленность, например «дом заполнен людьми, домашними животными и грызунами». Связь между сущностями СЛУЖАЩИЕ и ПРОФЕССИИ может указывать направление от конкретного служащего к набору

профессий, которыми он владеет. С другой стороны, СЛУЖАЩИЕ-С-ПРОФЕССИЕЙ указывает другое направление связи, ассоциируя каждую профессию со служащими, владеющими этой профессией.

Графические обозначения

Кроме наименования, существует некоторое количество дополнительных характеристик связи, которые могут быть легко объяснены с помощью графических обозначений. Тип сущности изображается прямоугольником, внутри которого указывается наименование сущности. Каждая связь изображается дугой

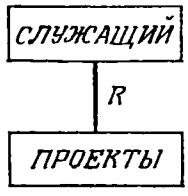


Рис. 4.9. Связь *R* между двумя сущностями.

или линией между двумя сущностями (прямоугольниками). Дуге присваивается наименование, которое является наименованием связи. На рис. 4.9 изображена связь между двумя сущностями СЛУЖАЩИЕ и ПРОЕКТЫ.

Характеристики ассоциативности/однозначности

Характеристика однозначности связи обозначает степень ассоциации типов сущностей [119]. Если экземпляр сущности Z соотносится *не более чем с одним* экземпляром другой сущности (Y), связь является однозначной по данному типу сущности. В математических обозначениях такая связь представляется функцией: любому заданному образу Z ставится в соответствие единственный образ Y[1)].

Ассоциативность связи определяется через однозначность в обратном направлении относительно Z. Вообще говоря, существуют четыре возможных типа ассоциаций сущности Z с сущностью Y: один-к-одному, один-ко-многим, много-к-одному и много-ко-многим.

Если связь однозначна в обоих направлениях (т. е. в направлении обоих типов сущностей), она является связью *один-к-одному*, как показано на рис. 4.10, *а*. Если связь является однозначной по X, то между X и Y существует ассоциация *один-ко-многим*, что также показано на рис. 4.10, б. Если однозначность

[1)] Из определения однозначности следует, что существуют некоторые $z \in Z$, не соотносящиеся ни с одним из образов $y \in Y$, что видно из рис. 4.10, *а*. Такое определение противоречит общепринятому определению функции. — *Прим. ред.*

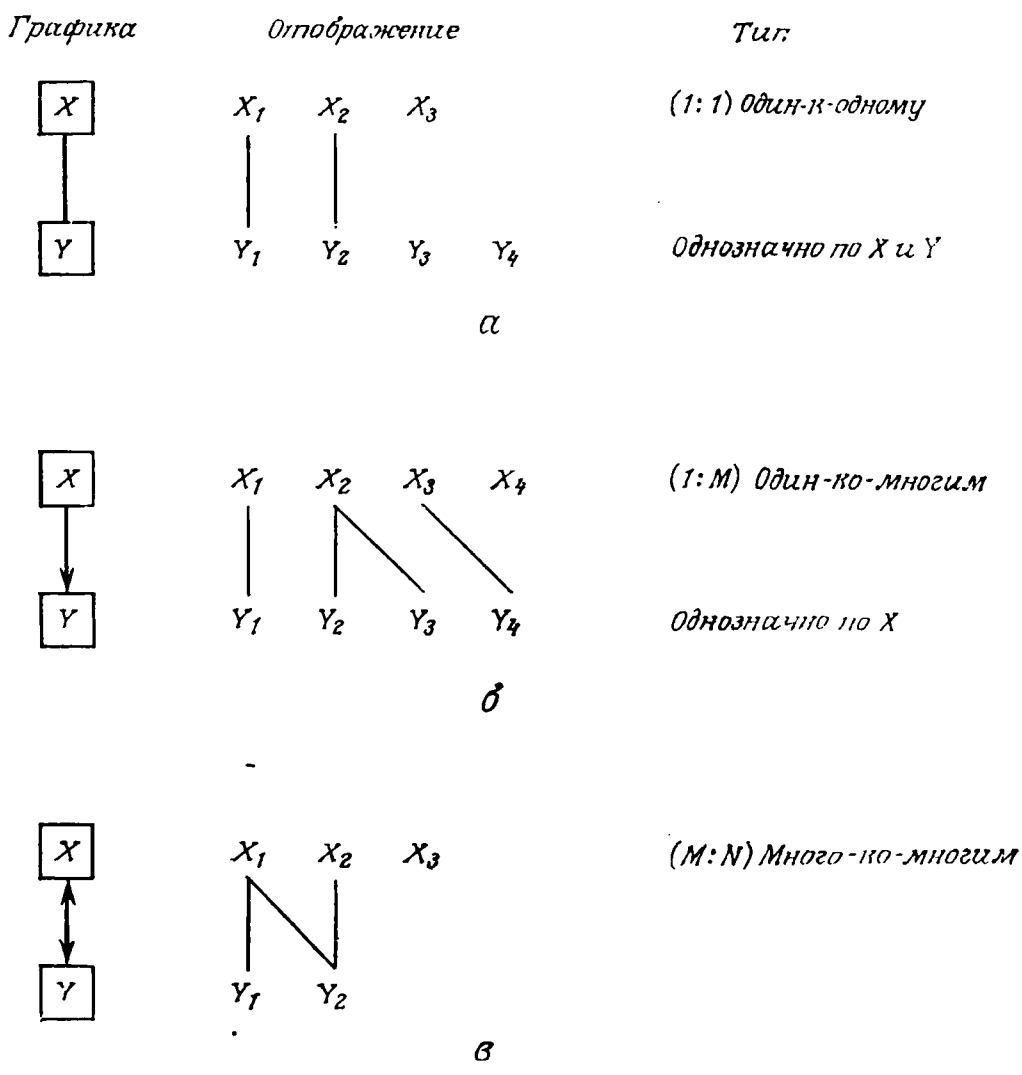


Рис. 4.10. Однозначность связей.

отсутствует, между X и Y существует ассоциация *много-ко-многим* (рис. 4.10, *в*).

Для целей проектирования полезно знать кардинальность, или размерность связи. Например, что подразумевается под связью один-ко-многим, 1 : 3 или 1 : 1000? Если кардинальность не может быть определена точно, разумным приближением является частотное распределение. Полезно также знать степень отклонения от среднего значения распределения, например, возрастает или убывает число экземпляров со временем.

Характеристика избирательности (полноты, зависимости)

Характеристика избирательности связи определяет правила членства экземпляров сущности в связи [249]:

1. *Необязательная связь.* Существование обеих сущностей в связи не зависит от связи.
2. *Возможная связь.* Существование одной из сущностей в связи зависит от связи.
3. *Условная связь.* Существование одной из сущностей определяется булевским условием. Условная связь является специальным случаем возможной связи. В случае условной связи на диаграммах рядом со стрелкой указывается условие существования.
4. *Обязательная связь.* Существование обеих сущностей зависит от связи.

Рис. 4.11 иллюстрирует примеры этих характеристик. Характеристики избирательности играют очень важную роль для поддержания целостности и согласованности базы данных.

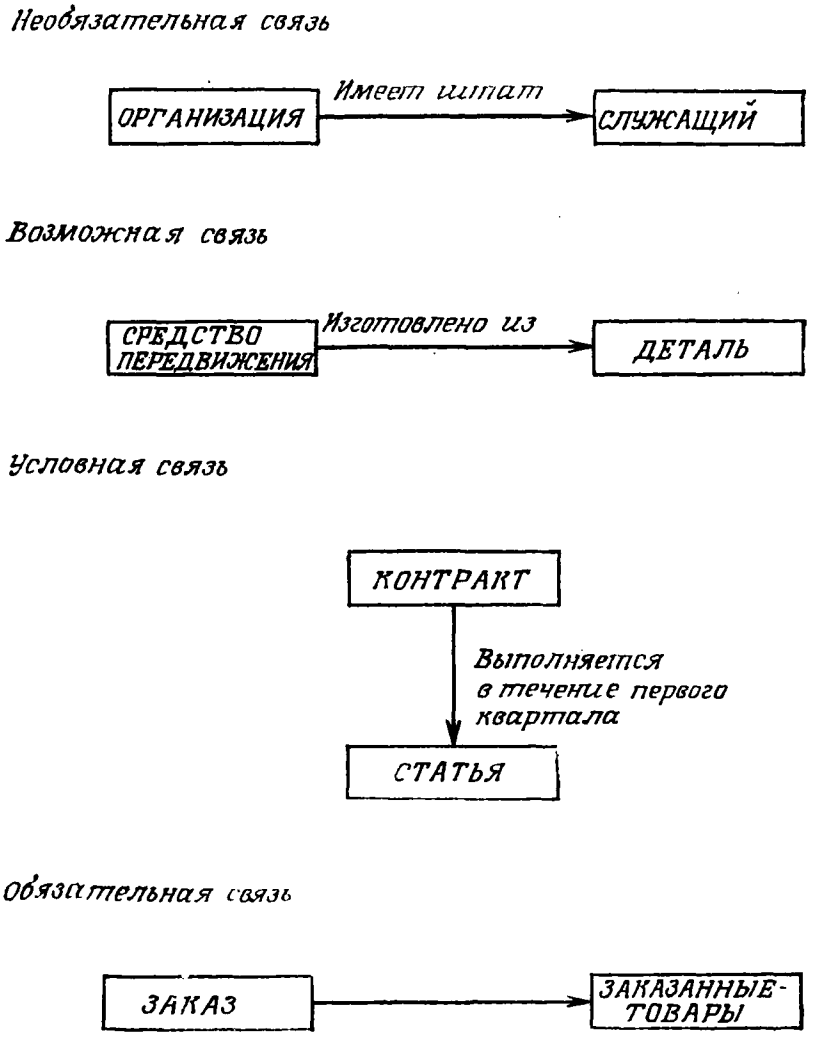


Рис. 4.11. Характеристики избирательности связей.

Характеристика однозначности

С помощью однозначности связей могут быть описаны дополнительные системные ограничения, как показано на рис. 4.12.

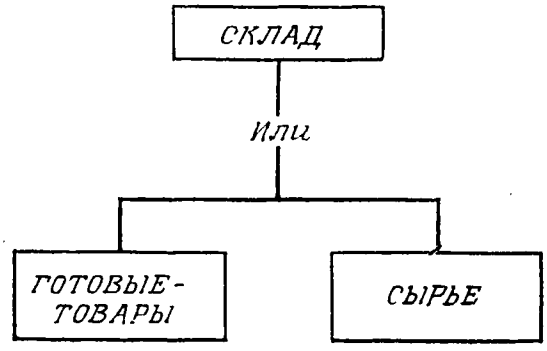


Рис. 4.12. Связь типа или/или.

Здесь характеристики однозначности используются для отображения того факта, что на складе могут храниться либо готовые товары, либо сырье, но не то и другое вместе.

Использование связи для идентификации сущности

Еще одной характеристикой является случай, когда связь используется для идентификации сущностей. Как уже отмечалось при обсуждении сущностей, наиболее употребительным способом идентификации является использование одного или более атрибутов сущности, значения которых позволяют однозначно отличить один экземпляр сущности от другого. Связи используются для идентификации сущностей двумя способами:

1. *В комбинации с атрибутами.* В этом случае значения атрибутов не являются уникальными внутри базы данных, и поэтому используется «владелец» связи для однозначного определения экземпляра сущности, являющегося членом связи. Значения атрибутов уникальны в пределах набора всех экземпляров, относящихся к сущности — «владельцу».

2. *Две или более связи.* В составной связи подчиненная сущность существует только тогда, когда имеются два соответствующих экземпляра исходных связей. В результате «владельцы» связи определяют существование подчиненной сущности. Часто в этой ситуации имеются данные пересечения.

4.4. Методологии концептуального проектирования

Рассмотрев основные принципы и понятия избранных подходов к концептуальному проектированию, можно перейти к выбору методологии применения этих принципов при проектировании концептуальных информационных структур. Различные подходы к концептуальному проектированию баз данных мож-

но отнести к одной из двух методологий проектирования: анализу сущностей и синтезу атрибутов. По-другому эти методологии можно не слишком строго охарактеризовать, как нисходящую и восходящую соответственно.

4.4.1. Анализ сущностей

Анализ сущностей является нисходящим подходом, который делит процесс проектирования на четыре стадии: моделирование представлений, объединение представлений, составление и анализ схемы и физическое проектирование. Для концептуального проектирования представляют интерес только две первые стадии.

Моделирование представлений

Под стадией моделирования представлений понимается моделирование информации, требуемой для базы данных, с точки зрения, отображающей различные аспекты деятельности организации. Видами или типами представлений являются: общее представление (с точки зрения организации в целом), прикладное представление, информационное представление и представление событий. Каждому типу представления соответствует особый вид информации:

- *Цели/ограничения организации.* Производственные или коммерческие цели организации, выраженные в терминах требуемой информации. Различные информационные ограничения, такие, как требования отчетности, необходимость проверки, управляющие воздействия, а также меры по обеспечению секретности.
- *Обрабатываемая информация.* Информация о существующих и будущих приложениях. Сюда включаются все предполагаемые виды обработки базы данных.
- *Информационные структуры и связи.* Моделирование общих информационных связей: существующих элементов данных, агрегаций элементов данных и связей, необходимых для отображения деятельности организации или предприятия. Этим обеспечивается основа для специальных и незапланированных запросов.
- *События и планирование.* Сроки представления информации, различных отчетов, сроки решения прикладных задач. Сюда включается как незапланированная (специальная), так и заранее определенная информационная деятельность организации.

Моделирование представлений заключается в фактическом сборе информации на различных уровнях организации в

соответствии с четырьмя видами представлений. Оно включает входную информацию от исполнителей, руководящего состава и конечных пользователей.

Объединение представлений

Процесс объединения представлений заключается в интеграции различных представлений, полученных на предыдущей стадии, в единое для всей организации концептуальное представление информации и требований обработки данных. Концептуальное представление отображается высокоуровневой диаграммой в виде информационной структуры. Интегрированное представление и диаграмма информационной структуры составляют основу подхода к управлению базами данных. Поэтому на практике разработка концептуального представления является существенной, если не самой важной частью процесса проектирования. В предыдущих работах этот этап часто опускался, а если и были попытки, то или в урезанном виде, или направленные на решение узких специальных вопросов.

Основная цель объединения представлений заключается в идентификации и выделении общих аспектов различных представлений, а также в обнаружении и разрешении их основных противоречий. Этот процесс включает анализ и принятие решений на нескольких уровнях:

- *Несогласованность наименований.* Идентификация синонимов и омонимов среди элементов данных.
- *Несогласованность идентификации.* Различная идентификация одних и тех же типов сущностей (например, служащие могут однозначно идентифицироваться номером страхового полиса в одном приложении и номером служащего — в другом).
- *Несогласованность агрегации.* Ограничение различных групп элементов на структурном уровне или операций над значениями элементов на уровне экземпляров (например, означает ли «Суммарные закупки» суммарные для округа, страны и т. д.).
- *Дополняющие подмножества.* Распознавание взаимодополняющих друг друга подмножеств данных, таких, как «служащие, работающие неполный рабочий день», «служащие, работающие полный рабочий день» и «уволенные служащие».
- *Противоречивость требований обновления.* Обнаружение несогласованных правил добавления/исключения среди различных представлений пользователей.
- *Противоречивость ограничений целостности.* Идентификация различий в правилах поддержания целостности данных. Например, каждый новый проект создает новый экземпляр сущности СЛУЖАЩИЙ, вызывая тем самым дублирование.

Подробное описание последовательности шагов будет сделано в гл. 5.

4.4.2. Синтез атрибутов

Эта методология называется восходящей, так как она начинается с синтеза атрибутов самого нижнего уровня, из которых затем формируются сущности и связи верхнего уровня. Различают четыре стадии для данной методологии: классификация атрибутов, композиция сущностей, формулирование связей и графическое представление.

Классификация элементов данных

Результатом анализа требований, описанного в гл. 3, является полный список элементов данных, используемых в различных задачах организации. Эти элементы данных с помощью эвристических правил классифицируются по типам атрибутов, а также в отношении членства в сущности. Они делятся по принадлежности к одному из двух типов атрибутов. Идентифицирующий атрибут — это такой элемент данных, который уникален во всех задачах и среди других элементов данных и поэтому может быть использован для доступа к другим атрибутам. Атрибут, не являющийся идентификатором, зависит от идентифицирующего атрибута при доступе и не может существовать (в полном смысле слова) обособленно. По определению сущность (идентификатор) является его «владельцем».

Композиция сущностей

Атрибуты принадлежат сущностям двух типов: уникальным сущностям и неуникальным сущностям (зависимым). Уникальные сущности существуют сами по себе, имея по меньшей мере один идентифицирующий атрибут. Их существование не зависит ни от каких-либо атрибутов, ни от других сущностей. Существование неуникальных сущностей, естественно, зависит от других сущностей, а их смысловое значение — от других атрибутов.

Формулирование связей

На этой стадии, кроме двух типов сущностей, необходимо использовать также другую информацию, такую, как политика организации, а также информацию, полученную в результате собеседований. Эта информация используется для определения связей между типами сущностей и для определения дополнительных атрибутов сущностей.

Графическое представление

На конечной стадии вышеописанные атрибуты, сущности и связи оформляются графически в терминах модели «сущность-связь». Используется следующая последовательность действий: изобразить все уникальные и неуникальные сущности, изобразить графически все связи между сущностями, представить перекрестные связи между сущностями.

Подробное описание последовательности шагов методологии синтеза атрибутов будет сделано в гл. 6.

Глава 5. Формулирование и анализ сущностей

5.1. Введение

Гл. 4 была посвящена представлению информации для фазы концептуального проектирования. Были предложены две схемы моделирования: высокоуровневое объектное представление и традиционное представление на основе сущностей. В данной главе описывается методология формулирования концептуальных схем с использованием традиционного подхода. В гл. 4 были развиты основные конструкции для моделирования, а здесь описываются этапы проектирования концептуальной схемы с использованием этих конструкций.

В концептуальном проектировании различаются две стадии: моделирование проектных представлений и объединение представлений [1]. Несмотря на то что имеется большое число исследований, посвященных концептуальному проектированию, конкретных результатов получено мало. В дополнение к работам по моделированию, на которые есть ссылки в гл. 4, можно отметить следующие исследования, оказавшие влияние на подход, развитый в данной главе: по моделированию представлений [238], по интеграции представлений [237]. Необходимо также отметить подход к анализу данных, развитый в работах [98] и [249].

5.1.1. Цели и область проектирования

Началом всякого проектирования независимо от методологии является определение области применения проекта. Обычно приходится искать компромисс, поскольку слишком обширная область ведет к нечеткому и сложному проектированию, а слишком узкая — к дроблению данных и снижению уровня интеграции базы данных. В идеале область применения должна включать все соответствующие функциональные службы организации. Однако руководство обычно не слишком одобрительно относится к огромным расходам и значительным затратам времени, необходимым для решения такой большой задачи. В результате

[1] Это следует из материалов Семинара по проектированию баз данных в Нью-Орлеане, 1978 г. [206], идеи которых восходят к работам Симпозиума по проектированию баз данных 1978 г. [352], неофициального семинара по исследованиям фирмы IBM в Сан-Хосе и работам Группы исследования систем баз данных Мичиганского университета [243].

в очень больших организациях возможно появление нескольких баз данных. К сожалению, существует крайний (так называемый позадачный) подход к этому явлению, который приводит к микроскопическим базам данных на каждое приложение. В итоге получается неинтегрированная разобщенная структура базы данных.

Чтобы достичь компромисса в этом вопросе, необходимо учесть технические возможности организации, ее готовность к внедрению базы данных, выраженную в наличии денежных и людских ресурсов, сложность моделируемых данных и степень их интеграции. В идеальном случае выбранная область применения должна быть знакома проектировщику и иметь явные связи с другими базами данных.

Для решения этой проблемы полезно составить общую информационную схему организации. Как только будут определены функциональные подразделения организации, будет относительно легко при помощи анализа информационных потоков провести декомпозицию общей информационной схемы на базы данных приемлемых размеров.

5.1.2. Определение проектных представлений

Другим способом упрощения процесса концептуального проектирования является разбиение множества используемых в процессе проектирования данных на составные части. С этой целью, а также для выявления некоторой специфичной и уникальной информации предлагается рассматривать информационные потребности организации с точки зрения нескольких определенных аспектов. Этот подход является расширением методики, предложенной в работах Кана [176] и Чена [69]. В нем используются четыре представления: обобщенное, прикладное, информационное и представление событий. Каждое представление определяет различные точки зрения на информационные требования.

Обобщенное представление

Обобщенное представление отображает точку зрения высшего и среднего руководства на информационные потребности организации. Оно основано на представлении о функционировании организации и ее структуре. Это представление является очень важным, так как из него вытекают многие окончательные проектные решения.

Прикладное представление

Прикладное представление отображает те процессы обработки данных, которые должны выполняться для достижения стоящих перед организацией целей. Оно включает отчеты, которые

необходимо подготовить, транзакции, а также обычно выполняемую текущую обработку данных.

Информационное представление

Информационное представление описывает общие информационные связи, необходимые для принятия решений, и долгосрочные информационные требования. Оно включает специальные запросы пользователей, перспективные информационные схемы и общие требования руководства.

Представление событий

Представление событий описывает требования, связанные с планированием или определенными моментами времени. Оно включает такие факты, как время начала выполнения транзакций или время завершения обработки данных. Например, «отчет XYZ — по пятницам» или «приложение ABC выполняется раз в две недели».

5.2. Моделирование проектных представлений

Данная стадия методологии концептуального проектирования заключается в моделировании и соответствующем отображении информации, необходимой для описания четырех основных проектных представлений. Такое отображение информации, основанное на локальных или индивидуальных представлениях, необходимо для описания организации, пользователей, приложений и управления.

5.2.1. Конструктивные элементы модели

В гл. 4 для описания каждого из представлений были выбраны в качестве конструктивных элементов моделирования сущности, атрибуты и связи. Напомним, что *сущность* представляет интересующий нас объект — личность, место, предмет, событие,— о котором собирается информация. *Связь* — это ассоциация между экземплярами двух или более сущностей. *Атрибут* — свойство, которое хочет отметить проектировщик. Основой модели «сущность-атрибут-связь» является графическое изображение, состоящее из прямоугольников и стрелок. Прямоугольники используются для изображения сущностей, а стрелки — для направленных связей. Атрибуты изображаются линиями (без стрелок), выходящими из прямоугольников, как показано на рис. 5.1. Для изображения связи между сущностями ТОРГОВЫЙ-АГЕНТ и КЛИЕНТ необходимо изобразить стрелку, направленную от сущности ТОРГОВЫЙ-АГЕНТ к сущности КЛИЕНТ, указывающую, что торговый агент обслуживает нескольких клиентов.

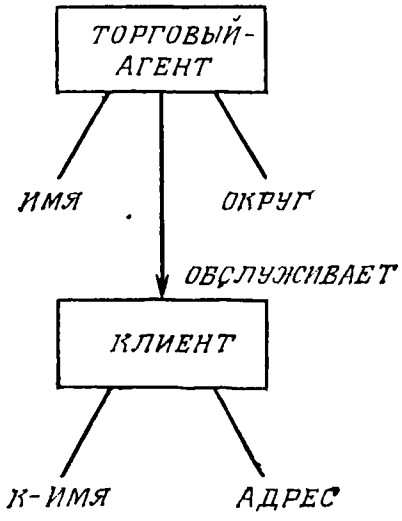


Рис. 5.1. Изображение сущностей, атрибутов и связей на информационных диаграммах.

Атрибуты ИМЯ и ОКРУГ описывают торгового агента, а атрибуты К-ИМЯ и АДРЕС служат для описания сущности КЛИЕНТ.

Основные предпосылки/правила

Ниже приводятся общие правила, служащие основанием для моделирования проектных представлений.

- Проектные представления моделируются с помощью трех типов конструктивных элементов: сущностей, атрибутов и связей.
- Каждый компонент информации в проектном представлении изображается одним и только одним конструктивным элементом.

5.2.2. Формулирование проектных представлений

Моделирование проектных представлений состоит из последовательности шагов, завершающейся моделью локального представления. Описанная последовательность относится ко всем проектным представлениям.

Шаг 1. Идентификация локальных представлений

Для каждого установленного выше проектного представления существует набор «подпредставлений» или локальных представлений. Они могут соответствовать независимым областям данных, относящимся к функциональным областям. Например, в системе приема заказов в качестве локальных представлений могли бы выступать такие функциональные задачи, как прием заказов, расчеты с клиентами, история заказа, новые изделия. Выбор локального представления зависит от конкретного проектного представления и от масштаба функциональной области.

Минимальная зависимость от других локальных представлений или минимальное взаимодействие с ними, а также степень управляемости являются факторами, которые следует принимать во внимание при формулировании локальных представлений. Локальное представление, основанное на этих факторах, может описывать приложения, функциональные области и даже проектные представления в целом.

Шаг 2. Формулирование сущностей

Для каждого локального представления могут быть сформулированы сущности, требуемые для описания этого локального представления. На этом шаге проектировщик встречается с двумя важными проблемами. Первая из них связана с распознаванием различных категорий сущности. Она разрешается с помощью концепции типа или роли. Например, содержание сущности СЛУЖАЩИЙ может быть разделено по категориям типов служащих: водитель грузовика, секретарь, инженер. На этой стадии концептуального проектирования важно выявить релевантные типы и представить каждый из них в виде обособленной сущности. Обобщение этих типов в родовую сущность СЛУЖАЩИЙ рассматривается на следующей стадии концептуального проектирования — стадии объединения представлений (см. разд. 5.3).

Вторая проблема заключается в использовании сущности в качестве конструктивного элемента. Часто некоторая порция информации может быть представлена как атрибут, сущность или связь. Например, тот факт, что двое служащих находятся в семейных отношениях, может быть выражен сущностью СЕМЬЯ, связью ЖЕНАТ-НА (ЗАМУЖЕМ-ЗА) или атрибутом СУПРУГ(А). В таких случаях проектировщик должен руководствоваться двумя правилами. Первое правило состоит в том, чтобы использовать ту конструкцию, которая кажется более естественной: если конструкция выбрана не очень удачно, это обязательно выявится на последующих этапах проектирования. Второе правило включает ранее сделанное предположение о том, что в локальном представлении для моделирования порции информации должна использоваться одна и только одна конструкция. Необходимо избегать избыточности в использовании конструктивных элементов.

В отношении числа сущностей, используемых в локальном представлении, можно применить правило из теории информации: «магическое число семь плюс минус два» [226]. Это правило констатирует, что число фактов (информационных кластеров), которыми человек может одновременно управлять, равно примерно семи. Применяя это правило к процессу проектирования баз данных, можно сказать, что число сущностей в

локальном представлении должно быть не более девяти, скорее всего шесть-семь. Если это не соблюдается, то возможно, что область применения локального представления слишком обширна.

Другое соображение, которое необходимо отметить, касается выбора наименований сущности. Поскольку сущность представляет информационный факт, этому факту должно быть дано четкое наименование (название). Это является важным также и для стадии объединения представлений, где придется иметь дело с синонимами и омонимами. Если понятия будут иметь расплывчатые наименования, то процесс интеграции и объединения представлений будет также расплывчатым.

Шаг 3. Выбор идентифицирующего атрибута для каждой сущности

Хотя определенная совокупность атрибутов и может служить основой для выделения сущностей, самым важным атрибутом на данном шаге процесса проектирования является идентификатор, служащий для однозначного распознавания отдельных экземпляров сущности. Идентификатор сущности может состоять из одного или нескольких атрибутов, набор значений которых уникален. Это является важным и для стадии объединения представлений, так как значения идентифицирующего атрибута находятся в однозначном соответствии с экземплярами сущности. Поэтому две сущности с одинаковыми идентификаторами могут оказаться в некоторой степени избыточными. Это зависит, однако, от описательных атрибутов сущности и от уроня обобщения.

Шаг 4. Спецификация связей

На этом шаге локальное представление дополняется информацией, полученной в результате определения ассоциаций между экземплярами сущностей. Как указывалось в гл. 4, имеется несколько типов связей: необязательная, возможная, обязательная, однозначная и условная. Одна из неформальных процедур для этого шага заключается в попарном объединении между собой всех сущностей, содержащихся в данном представлении [98]. Для каждой пары сущностей необходимо провести исследование, заключающееся в получении ответа на вопрос: могут ли быть использованы обе сущности в одной и той же транзакции или можно ли задать содержательный вопрос, включающий обе сущности? Если ответ положительный, определить тип связи, который необходим для формирования ассоциации. Затем определить, какие связи являются наиболее важ-

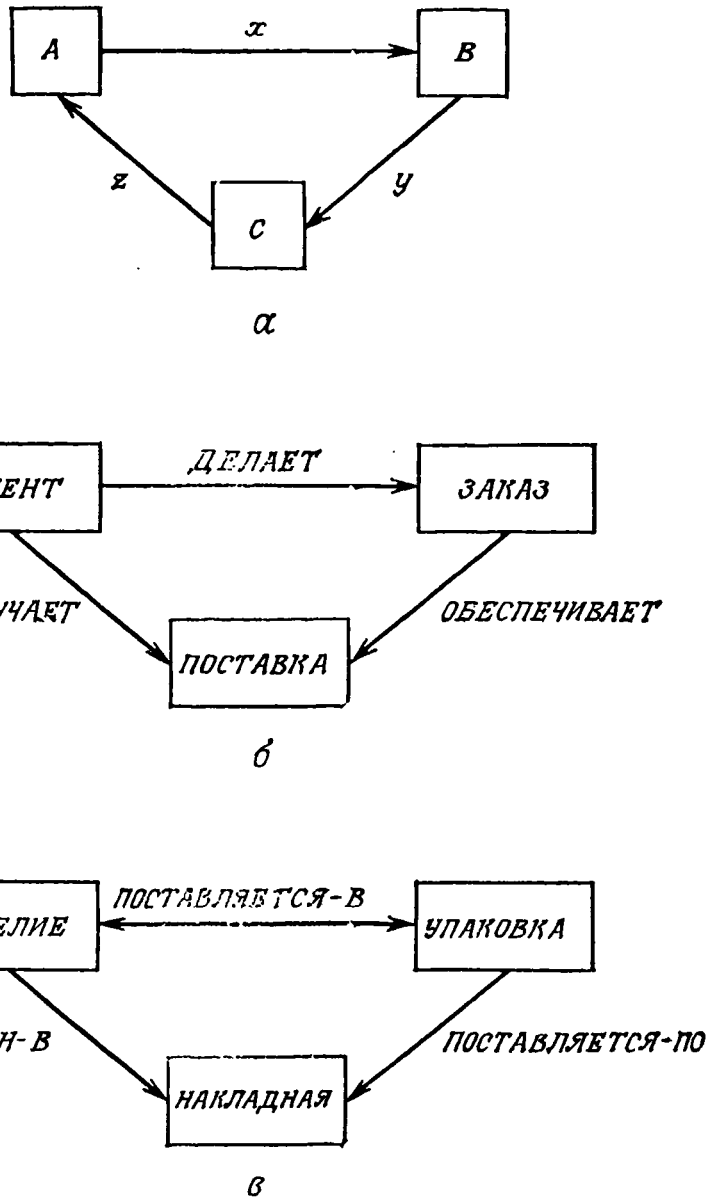


Рис. 5.2. Анализ избыточных связей.

ными и какие — избыточными. К сожалению, это может быть выполнено только с учетом детального рассмотрения соответствующего проектного представления. Например, на рис. 5.2, *а* показано, что одна из связей *x*, *y*, *z* между сущностями *A*, *B* и *C* может оказаться избыточной. Как показано на рис. 5.2, *б* связь ПОЛУЧАЕТ обычно бывает избыточной, поскольку ПОСТАВКА не может быть выполнена, пока не сделан ЗАКАЗ. Однако связь ПОСТАВЛЯЕТСЯ-В на рис. 5.2, *в* ассоциирует все возможные комбинации ИЗДЕЛИЙ и УПАКОВОК. В соответствии с этим определяется сущность НАКЛАДНАЯ в связях ЗАКАЗАН-В и ПОСТАВЛЯЕТСЯ-ПО [180].

Шаг 5. Добавление описательных атрибутов к сущностям

Атрибуты могут быть разделены на два класса: те, которые служат для идентификации экземпляров сущности, и те, которые описывают свойства сущности. На последнем шаге моделирования локального представления к ранее определенным сущностям добавляются описательные атрибуты. Следствием того, что идентифицирующие атрибуты служат для полной идентификации экземпляров сущности, является обязательная зависимость (т. е. функциональная зависимость) описательных атрибутов от идентификаторов сущности. Другим следствием является то, что в качестве описательных атрибутов сущности могут использоваться только однозначные атрибуты. В терминах отношений это означает, что сущности находятся в первой нормальной форме и что все повторяющиеся атрибуты исключены. Рассмотрим, например, сущность КЛИЕНТ, которая идентифицируется атрибутами НОМЕР и ИМЯ и описывается атрибутами ПЛАТЕЖНЫЙ-БАЛАНС и АДРЕС, при этом АДРЕС включает УЛИЦУ, ГОРОД, ШТАТ и ПОЧТОВЫЙ-КОД. В случае, если КЛИЕНТ размещается в нескольких местах, принцип нормализации нарушается тем, что допускаются повторяющиеся группы. В таком случае можно образовать другую сущность, например РАЗМЕЩЕНИЕ, содержащую атрибуты УЛИЦА, ГОРОД, ШТАТ и ПОЧТОВЫЙ-КОД. При этом возникает ассоциация между сущностями КЛИЕНТ и РАЗМЕЩЕНИЕ, которая именуется с помощью связи АДРЕС-КЛИЕНТА.

Другим вопросом, на котором следует остановиться, является случай, когда один атрибут используется несколькими сущностями. В этом случае атрибут является избыточным, и необходимо установить какое-либо правило, в соответствии с которым атрибут может быть назначен только одной сущности. Одним из таких правил является подсчет частоты использования данного атрибута вместе с идентификатором при обработке транзакций.

Мы полностью описали пять шагов моделирования локального представления, которые должны быть проделаны для каждого из основных проектных представлений. Не претендуя ни в коей мере на научность, эти процедуры предлагают последовательность проектирования, которая успешно применяется некоторыми опытными проектировщиками. Анализ согласованности, целостности и возможности интеграции будет выполнен позднее.

5.3. Объединение представлений пользователей

Теперь, когда каждое представление сформулировано и описано с помощью диаграмм «сущность-связь», следующей стадией процесса концептуального проектирования является объе-

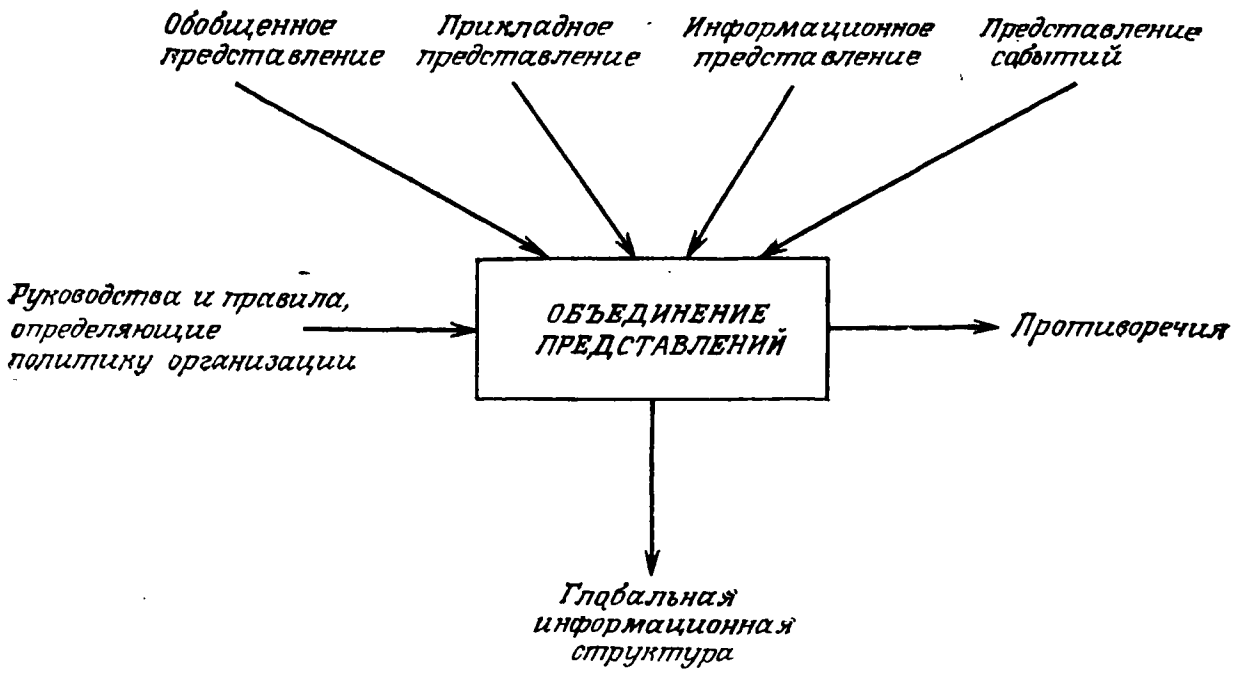


Рис. 5.3. Модель объединения представлений.

динение этих представлений в единую глобальную информационную структуру. Эта структура не только взаимно соответствует каждому представлению, но представляет также интегрированную базу данных в сжатой форме.

Желательно создать такую глобальную информационную структуру, которая вобрала бы в себя представления пользователей и руководства относительно производственной и коммерческой деятельности, а также ограничений политики организации.

Как показано на рис. 5.3, глобальная информационная структура образуется из входных представлений различных уровней — обобщенного, прикладного, информационного, представления событий, а также правил и руководств, определяющих политику организации. Входные представления содержат информацию, необходимую для обеспечения коммерческой деятельности, а политические руководства (правила) описывают законодательные и правительственные ограничения в области коммерческой деятельности. Все эти входные представления описаны в виде информационных диаграмм типа «сущность-связь» в соответствии с пошаговой процедурой, изложенной в разд. 5.2.

Основной результат процесса объединения представлений — глобальная информационная структура. Насколько это возможно, она является интеграцией обобщенного, прикладного, информационного представлений и представления событий. Вспомогательные результаты, включающие неразрешенные противоречия и неинтегрируемые компоненты, показывают достигнутую

степень интеграции. Естественно, что цель процесса объединения состоит в минимизации такого выхода. Хотя данная модель явным образом не отражает итеративности процесса объединения представлений, он часто является таковым из-за необходимости устранения ограничений и противоречий.

5.3.1. Понятия и принципы объединения

Существуют три основополагающие концепции объединения проектных представлений: идентичность, агрегация и обобщение. Различные способы, с помощью которых компоненты этих концепций могут быть использованы в локальных представлениях пользователей, образуют типы взаимосвязей, которые описываются в разд. 5.3.2.

Идентичность

Идентичность является наиболее простой из всех трех концепций. Говорят, что два или более элементов являются идентичными, если они имеют одинаковое семантическое значение. Для идентичных элементов не является, однако, обязательным иметь одинаковое синтаксическое представление. Другим способом описания отношения идентичности является объявление двух или более элементов синонимами.

Простота концепции идентичности не определяет простоты установления синонимии понятий. Из-за неадекватности методов представления данных понимание семантики данных весьма ограничено. Обычно требуется глубокое понимание области деятельности пользователей, чтобы установить существование идентичности понятий. Задание соответствия между описаниями элементов и экземплярами этих элементов в предметной области затруднительно, поскольку образы описательных элементов в совокупности могут составлять некоторое подмножество возможных образов. В этом случае трудно определить, является ли описание элемента слишком расплывчатым или же образы действительно представляют подмножество описания элемента. Более того, образы двух описаний элементов, которые являются кандидатами в отношение идентичности, могут образовать пересечение вместо объединения. В таких случаях бывает трудно решить, одно или оба описания элементов способны покрыть объединение этих образов. Эти условия определяют существование скорее подобных, а не идентичных элементов. Решение вопроса, может ли подобие заменить идентичность или какое из двух отношений элементов необходимо применять на деле, требует глубокого и детального понимания области деятельности пользователей и их потребностей в данных.

Агрегация

Понятие агрегации так же является очень простым и часто встречающимся на практике, как и понятие идентичности. Как указывалось в гл. 4, агрегация соответствует концепции, позволяющей рассматривать связь между элементами, как новый элемент более высокого уровня. Например, элемент СЛУЖАЩИЙ может быть воспринят как агрегация элементов ИМЯ, НОМЕР-СТРАХОВОГО-ПОЛИСА и АДРЕС (рис. 5.4).

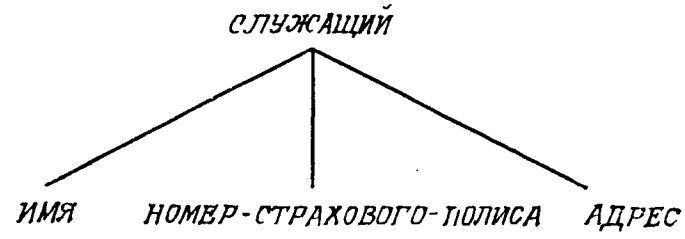


Рис. 5.4. Агрегация.

Многие агрегации могут быть легко идентифицированы, поскольку основные модели данных включают синтаксис, позволяющий представить агрегацию.

Обобщение

Это понятие является наиболее трудным для восприятия и может быть легко спутано с агрегацией. В то время как агрегация может быть представлена в виде составных частей, образующих некоторое «целое», обобщение связано только с «целыми». Оно относится к типу абстракции, в которой группа подобных элементов воспринимается как родовой элемент, при этом различия между отдельными элементами опускаются (см. гл. 4). Например, СЛУЖАЩИЙ может быть воспринят как обобщение групп ФАБРИЧНЫЙ-СЛУЖАЩИЙ, КОНТОРСКИЙ-СЛУЖАЩИЙ и АДМИНИСТРАТОР. Образ любого из этих трех типов является одновременно образом обобщенного элемента СЛУЖАЩИЙ. Здесь применяется ссылка на подобие (идентичность) элементов из предыдущей секции. Во многих случаях оказывается, что подобные элементы не могут рассматриваться как идентичные, однако могут быть представлены в виде обобщения.

Агрегация и обобщение являются очень похожими по структуре и применению концепциями и поэтому могут быть легко спутаны, поскольку каждый элемент может одновременно участвовать и в агрегации, и в обобщении (рис. 5.5). Размерность агрегации может определяться размерностью обобщения, и наоборот. На рис. 5.5 показано, например, что каждый образ элемента АДМИНИСТРАТОР является одновременно агрегацией элементов ИМЯ, НОМЕР-СТРАХОВОГО-ПОЛИСА и АДРЕС,

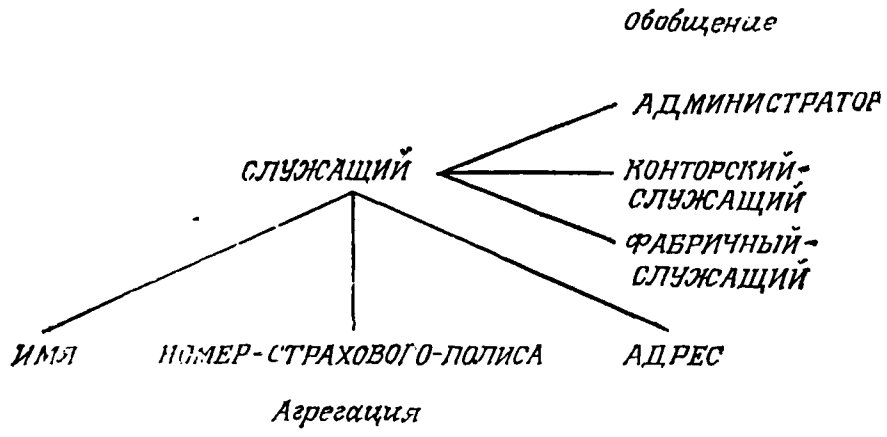


Рис. 5.5. Агрегация и обобщение.

поскольку все характеристики обобщенного элемента СЛУЖАЩИЙ применимы также и к более специфичным элементам, которые представляет СЛУЖАЩИЙ.

5.3.2. Типы объединения представлений

Имеются три типа объединения, каждый из которых основан на одной из вышеназванных концепций. Эти концепции могут комбинироваться различными способами для построения любого типа связи между объектами (элементами) в различных представлениях пользователей. Вначале обсуждается каждый тип объединения в отдельности. Затем будет проведено обсуждение и приведен пример комплексного объединения (которое включает более одного типа объединения).

Прежде чем начать обсуждение, необходимо сделать два важных предположения. Первое из них заключается в том, что каждое представление пользователя и набор требований являются полными и согласованными. Это означает, что определены все необходимые объекты (полнота) и что не требуется объединения внутри представлений пользователей. Иными словами, предполагается, что представления пользователей удовлетворяют следующим условиям:

1. Множество объектов является полным с точки зрения потребностей пользователя.
2. Все объекты имеют уникальные наименования (омонимы не допускаются).
3. Синонимы отсутствуют (не требуется объединения идентичных элементов).

Допускаются и даже желательны неявные объединения агрегированных и обобщенных объектов. Предположение полноты/согласованности служит для упрощения процесса объединения отдельных представлений путем исключения проверок

согласованности внутри каждого представления пользователей.

Другое предположение делается из соображений простоты и удобства. Предполагается, что будет проводиться только попарное объединение. Это означает, что одновременно будет производиться объединение представлений только двух пользователей. Это позволит облегчить поиск альтернативных комбинаций объединения, а также исключает любые возможные проблемы, связанные с очередностью объединения при n-арном объединении. Кроме того, оказывается, что в большинстве случаев более эффективным является объединение представлений на основе бинарного метода. Это обсуждается в разд. 5.3.3.

Объединение идентичности

Объединение идентичности основано на ранее введенном понятии идентичности. Два объекта могут быть идентичными с учетом дополнительной возможности иметь идентичные наименования. Следует избегать омонимов, а также подобных, но не идентичных объектов. Подобие может быть лучше выражено с использованием агрегации и обобщения.

Для проверки согласованности результата объединения и представлений пользователей может быть предложено следующее правило: *если объект из представления одного пользователя идентичен объекту из представления другого пользователя, ни один из этих объектов не должен в дальнейшем принимать участие в каком-либо другом объединении идентичности между этими двумя представлениями.* Это справедливо, поскольку предполагается, что каждый объект является уникальным в контексте своего локального представления.

Обозначения типов объединения, введенные здесь, будут использоваться и в дальнейшем. Пусть представление первого пользователя состоит из объектов A_{1-N}, а представление второго пользователя — из объектов B_{1-N}. На рис. 5.6 показан процесс объединения идентичности. Глобальный объект $G_{(AB)}$ является результатом объединения идентичности объектов $A_{(X)}$ и $B_{(X)}$. В данном случае $A_{(X)}$ приравнивается $B_{(X)}$ и $G_{(AB)}$.

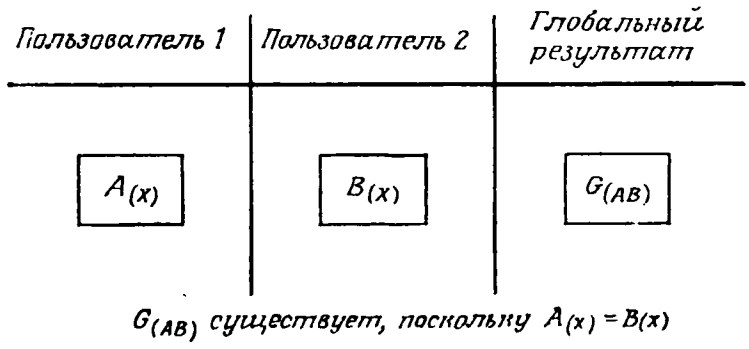


Рис. 5.6. Объединение идентичности.

Объединение агрегаций

Агрегация может встретиться в одной из двух форм. Отличие между ними заключается в том, что один пользователь может определить агрегатный объект как целое, а другой — в виде составных частей. В качестве примера рассмотрим ситуацию, показанную на рис. 5.7. Пусть первый пользователь

Пользователь 1	Пользователь 2
(представление кладовщика)	(представление продавца)
КОЛЕСА	ВЕЛОСИПЕДЫ
РАМЫ	
СИДЕНИЯ	
РУЛИ	

Рис. 5.7. Агрегация представлений двух пользователей.

определил несколько объектов, не устанавливая какой-либо связи между ними. Это не является ошибкой или упущением с его стороны. Информационные потребности этого пользователя не включают необходимость явного объединения между этими объектами. Возможно, существуют ассоциации или группы, включающие один или несколько из этих объектов. Однако эти связи не представляют объект более высокого уровня (агрегированный объект).

Пусть второй пользователь определил один из объектов так, что он является агрегацией объектов первого пользователя. Чтобы конкретизировать пример, допустим, что первый пользователь представляет функции кладовщика сборочных деталей на фабрике велосипедов, а второй пользователь — функции сбыта собранных велосипедов.

Объединение позволяет слить оба представления, не выражая явным образом тот факт, что ВЕЛОСИПЕД является агрегацией частей, перечисленных в списке. Однако это не слишком повышает возможность совместного использования данных и равносильно физическому слиянию данных, в котором различные данные хранятся в одном и том же файле, но логически никак не соотносятся друг с другом. Более сильным и более предпочтительным способом является логическое слияние данных, осуществляемое как на физическом, так и на концептуальном уровнях с учетом семантики и определений данных. Это повышает возможность совместного использования данных, так как проектировщик системы может организовать ссылки и пути доступа между представлениями пользователей, оптимизируя таким образом содержание базы данных и обеспечивая ее долгосрочную гибкость для удовлетворения информационных потребностей.

Более сложным случаем агрегации является такой, когда ни один из пользователей до конца не определил все составные части некоего «целого». На рис. 5.8 приведен пример, в котором рассматриваются функции двух кладовщиков, у одного из которых хранятся основные, незаменяемые детали (РАМЫ, КОЛЕСА), а у другого — детали, которые могут быть поставлены по требованию покупателя (СИДЕНИЯ, РУЛИ). Этот тип агрегации является более трудным для рассмотрения, поскольку ни в одном из представлений не определен в явном виде объект ВЕЛОСИПЕД.

Пользователь 1	Пользователь 2
(незаменяемые детали)	(заменяемые детали)
КОЛЕСА	СИДЕНИЯ
РАМЫ	РУЛИ

Рис. 5.8. Агрегация составных частей.

Возможно возражение, что поскольку ни один из пользователей не определил агрегированный объект ВЕЛОСИПЕД, то процесс объединения, в результате которого появляется глобальный объект, не имеет смысла. Это возражение игнорирует возможность проявления синергетических эффектов [1] среди определений данных. В каждом отдельном представлении перечисленные объекты необходимы, однако достаточных условий для создания объекта ВЕЛОСИПЕД не имеется. Вместе же эти объекты создают достаточные условия для того, чтобы объект ВЕЛОСИПЕД, полученный в результате логического слияния данных, связал между собой два набора деталей. Если не создавать агрегированного объекта, будет получено только физическое слияние.

На рис. 5.9 показаны способы объединения для двух рассмотренных типов агрегации. Пример агрегации первого типа показан на рис. 5.7, а второго — на рис. 5.8. Агрегация второго типа использует «виртуальный» объединенный объект, который создается в процессе агрегации. Глобальный результат состоит из глобального объекта и списка глобально-специфичных объектов из первоначальных представлений. Стрелки могут быть истолкованы, как «является агрегацией». В терминах множеств объекты отдельного представления являются подмножеством, возможно несобственным, глобального объекта. Если необходимо и позволяют условия, к объединению двух

[1] Синергизм (от греческого synergos — совместно действующий) — явление, когда общий результат превосходит сумму отдельных эффектов. — *Прим. ред.*

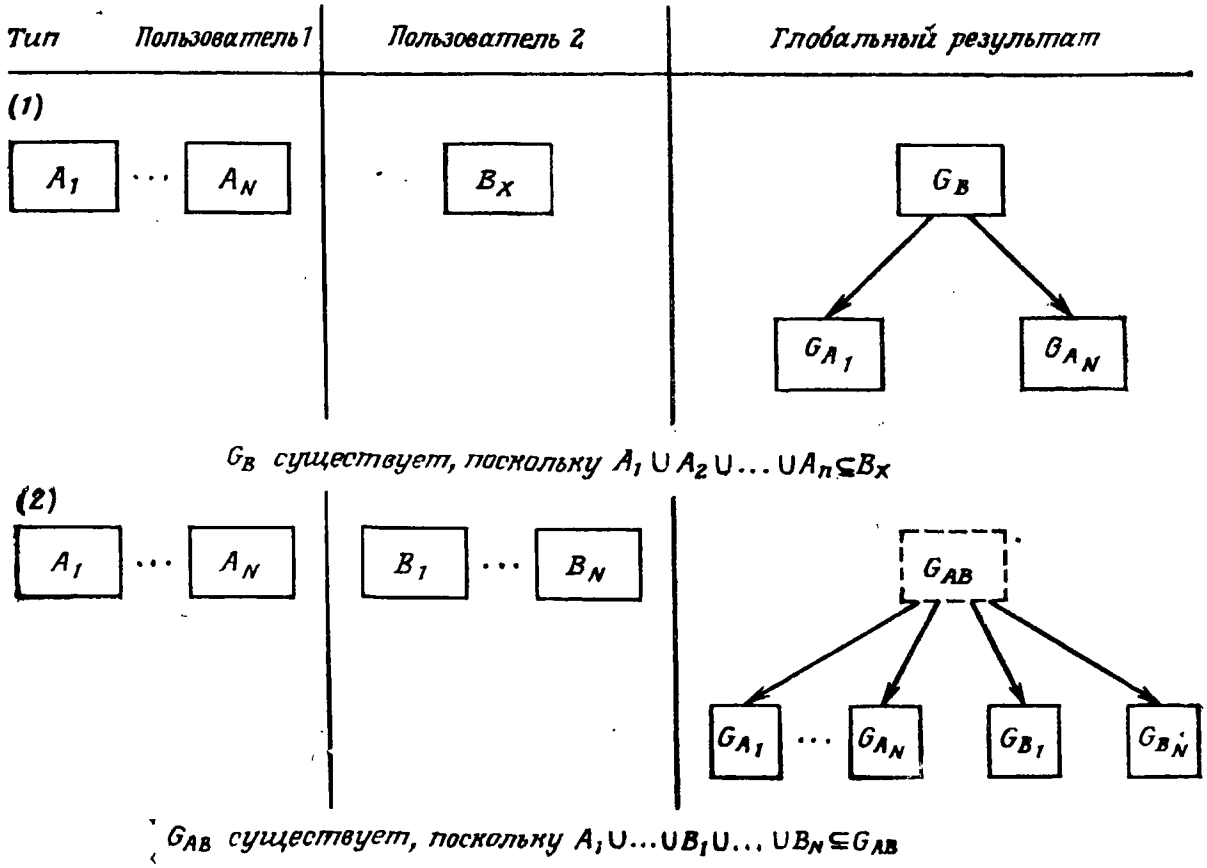


Рис. 5.9. Объединение агрегаций типа 1 и 2.

представлений другими пользователями могут быть добавлены другие объекты, например СПИЦЫ и ЦЕПИ.

Объединение обобщений

Так же как и агрегация, обобщение может встречаться в двух формах. Их основное отличие заключается в том, определен ли хотя бы одним из пользователей обобщенный или родовой объект или нет. Те же доводы относительно совместного использования логических данных и возможности проявления синергизма, которые приводились для агрегации, могут быть использованы и для обобщения. Два примера на рис. 5.10 и 5.11 иллюстрируют два возможных типа обобщения.

В первом случае один пользователь определил подмножества образа понятия «велосипеды», в то время как другой определил родовой объект ВЕЛОСИПЕДЫ. Во втором примере ни один из пользователей не определил родового объекта ВЕЛОСИПЕДЫ. Хотя концептуально возможно представить родовой объект, состоящий из одного или двух образов, однако маловероятно, чтобы это встретилось на практике. Установить нали-

Пользователь 1	Пользователь 2
ДВУХКОЛЕСНЫЕ-ВЕЛОСИПЕДЫ ТРЕХКОЛЕСНЫЕ-ВЕЛОСИПЕДЫ ТАНДЕМЫ	ВЕЛОСИПЕДЫ

Рис. 5.10. Обобщение — пример 1.

Пользователь 1	Пользователь 2
ДВУХКОЛЕСНЫЕ-ВЕЛОСИПЕДЫ ТАНДЕМЫ	ТРЕХКОЛЕСНЫЕ-ВЕЛОСИПЕДЫ

Рис. 5.11. Обобщение — пример 2.

чие родовой связи между специфичными типами объектов возможно только при сопоставлении нескольких образов.

Возможно несколько способов представления результатов обобщения. Одним из таких способов является введение объек-

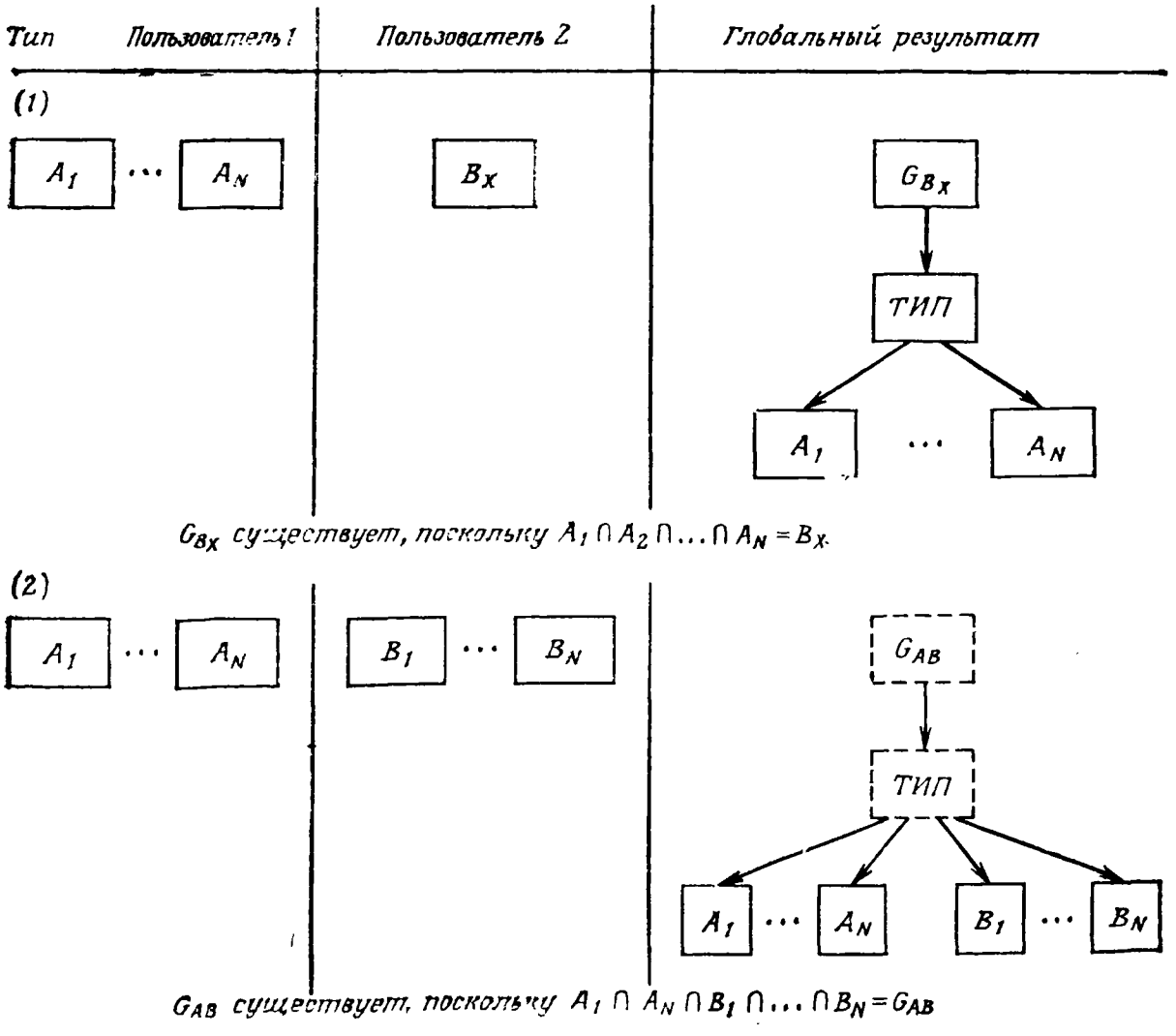


Рис. 5.12. Объединение обобщений типа 1 и 2.

та ТИП, так как объекты, составляющие родовой объект, являются во многом подобными, а по некоторым аспектам — идентичными. В первом примере родовой объект ВЕЛОСИПЕДЫ может быть описан объектом ТИП-ВЕЛОСИПЕДА, который может иметь коды или другие величины в качестве значений, позволяющие делать различие между двухколесными велосипедами, трехколесными и тандемами.

Другой способ включает связь списка специфичных объектов непосредственно с родовым объектом или через объект ТИП. Выбор способа реализации зависит от того, насколько тот или иной способ служит целям пользователя и проектировщика. На рис. 5.12 показан процесс обобщения с использованием объекта ТИП, а также с использованием списка специфичных объектов, связанных с объектом ТИП. Родовой объект может рассматриваться как пересечение определений специфичных объектов. В этом — одно из существенных отличий от агрегации, поскольку агрегированный объект можно представить как объединение определений специфичных объектов.

Комплексные семантические связи

До сих пор каждый тип объединения рассматривался обособленно. Однако каждый объект отдельного представления и каждый глобальный объект могут одновременно участвовать в различных типах представлений. Кроме того, не было сделано никаких оговорок относительно явных или неявных агрегаций и обобщений, которые могут встретиться в рамках отдельного локального представления. Путем комбинирования типов объединений могут быть представлены сложные и глубокие связи. Таким образом, можно надеяться, что если не все, то большинство семантических связей может быть представлено некоторой комбинацией первичных объединений.

Лучше всего проиллюстрировать это на примере. На рис. 5.13 показаны два представления пользователей о велосипедах. В данном случае трудно провести четкое разграничение между этими представлениями: налицо расхождения и частичные совпадения определений объектов и уровней абстракции. Например, уровень представления первого пользователя ТРЕХКОЛЕСНЫЕ-ВЕЛОСИПЕДЫ и ДВУХКОЛЕСНЫЕ-ВЕЛОСИПЕДЫ является более низким по сравнению со вторым пользователем, который рассматривает родовой объект ВЕЛОСИПЕДЫ. Оба пользователя считают объекты состоящими из частей, представленных понятиями различных уровней абстракции. Первый пользователь имеет более детализированное представление объекта, в то же время более обобщенно представляет объект КОЛЕСА (ср. СПИЦЫ и ОБОДЫ у второго пользователя). Кроме того, второй пользователь определил дополни-

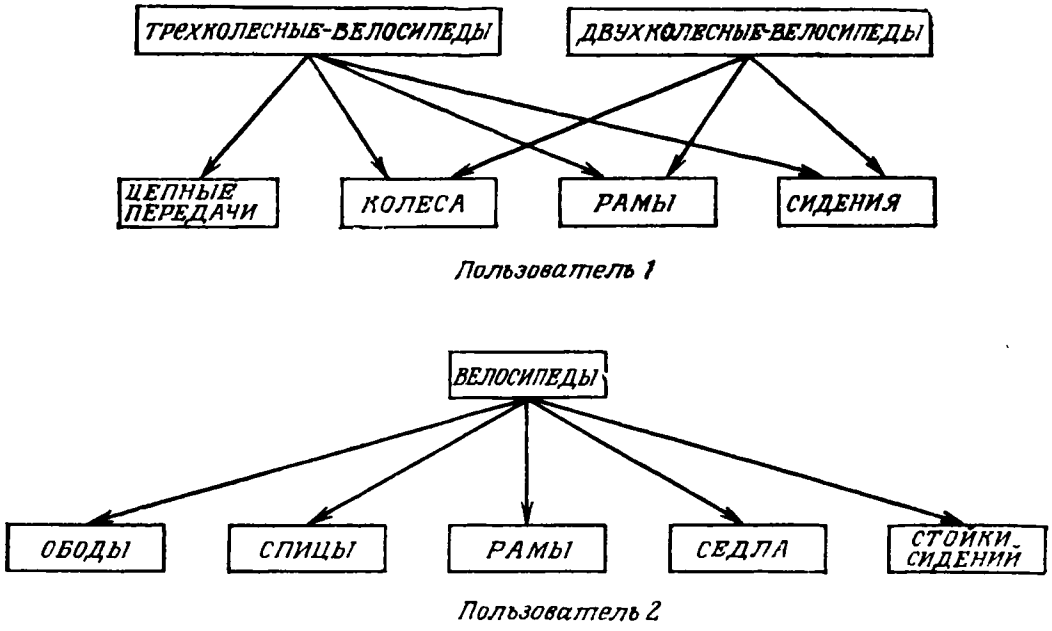


Рис. 5.13. Пример представлений пользователей.

тельный элемент, СТОЙКУ-СИДЕНИЯ, который не соотносится ни с одним объектом в представлении первого пользователя. В этих примерах оба представления можно определить прежде всего как связи агрегатного типа. В более общем случае могут часто встречаться связи, не являющиеся ни агрегацией, ни обобщением. Существует много связей, представляющих процессы, процедуры, условные ассоциации. Ради упрощения эти связи на представленных здесь диаграммах опущены.

На рис. 5.13 показана довольно типичная ситуация, отражающая высокую степень совпадения определений объектов и связей локальных представлений и наглядно показывающая несовпадение уровней абстракции агрегации и обобщения. Рис. 5.14 показывает, как должен выглядеть конечный результат, объединяющий оба представления. В объединенном представлении содержится вся специфичная информация пользователей, а также дополнительная информация, созданная в результате логического слияния данных.

Это объединенное представление может быть легко построено и документировано с помощью основных типов объединения, как показано на рис. 5.15. Если глобальные связи заданы и определены соответствующими выражениями (пунктирные линии на рис. 5.15), построить диаграмму, показанную на рис. 5.14, сравнительно просто. Связывая глобальные объекты с объектами пользователей с помощью несложного алгоритма, можно построить полную глобальную диаграмму. Рассмотрим в качестве примера приближенную процедуру обобщения ти-

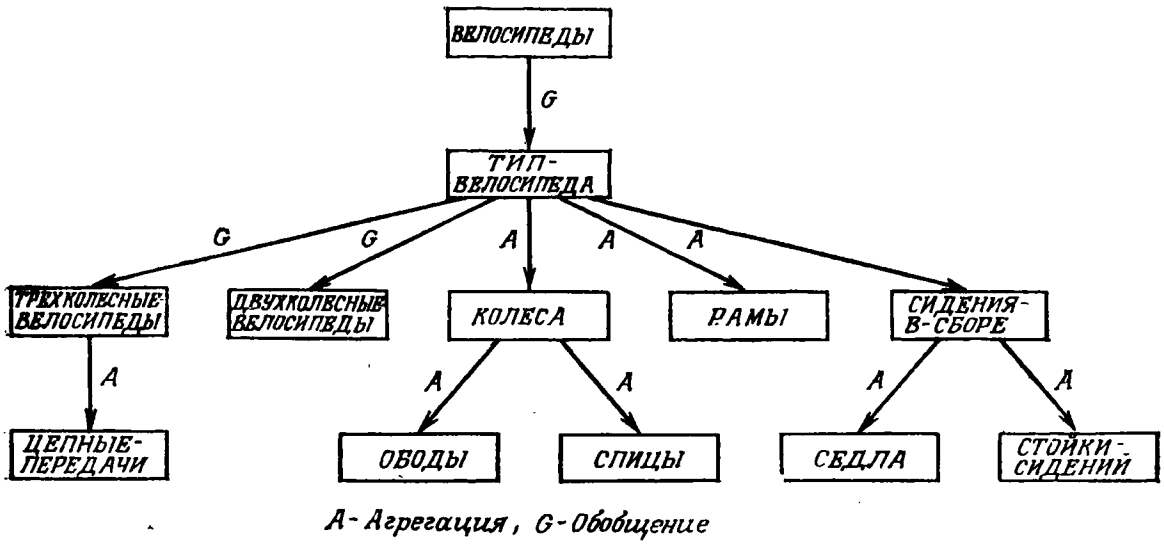


Рис. 5.14. Пример возможной спецификации объединения.

па 1 объектов ДВУХКОЛЕСНЫЕ-ВЕЛОСИПЕДЫ (1) и ТРЕХКОЛЕСНЫЕ-ВЕЛОСИПЕДЫ (1) в ВЕЛОСИПЕДЫ (2).

1. Для обобщения типа 1 [(1), рис. 5.15] построить глобальный объект для родового объекта пользователя *x* [ВЕЛОСИПЕДЫ (2)].

2. Для каждого объекта пользователя *y*, являющегося специфичным по отношению к объектам пользователя *x* [ТРЕХКОЛЕСНЫЕ-ВЕЛОСИПЕДЫ (1) и ДВУХКОЛЕСНЫЕ-ВЕЛОСИПЕДЫ (1)], построить соответствующий глобальный объект.

3. Связать глобальный объект, созданный на шаге 2, с глобальным объектом, который является «типовым» для родового объекта (ТИП-ВЕЛОСИПЕДА).

4. Для каждого объекта пользователя *y*, который связан со *всеми* родовыми объектами пользователя *y* в агрегатном контексте, построить соответствующие глобальные объекты и связать их с глобальным типовым объектом [связать КОЛЕСА (1), РАМЫ (1) и СИДЕНИЯ (1) с глобальным типовым объектом ТИП-ВЕЛОСИПЕДА].

5. Для каждого объекта пользователя *y*, который связан *не* со всеми родовыми объектами пользователя *y* в агрегатном контексте, построить соответствующие глобальные объекты и связать их с глобальным эквивалентом исходного агрегированного объекта (создать глобальный объект ЦЕПНЫЕ-ПЕРЕДАЧИ и связать его с глобальным объектом ТРЕХКОЛЕСНЫЕ-ВЕЛОСИПЕДЫ).

На основе предложенной процедуры можно разработать алгоритм, который позволит, использовав в качестве исходного любой набор утверждений объединения, создать полную согла-

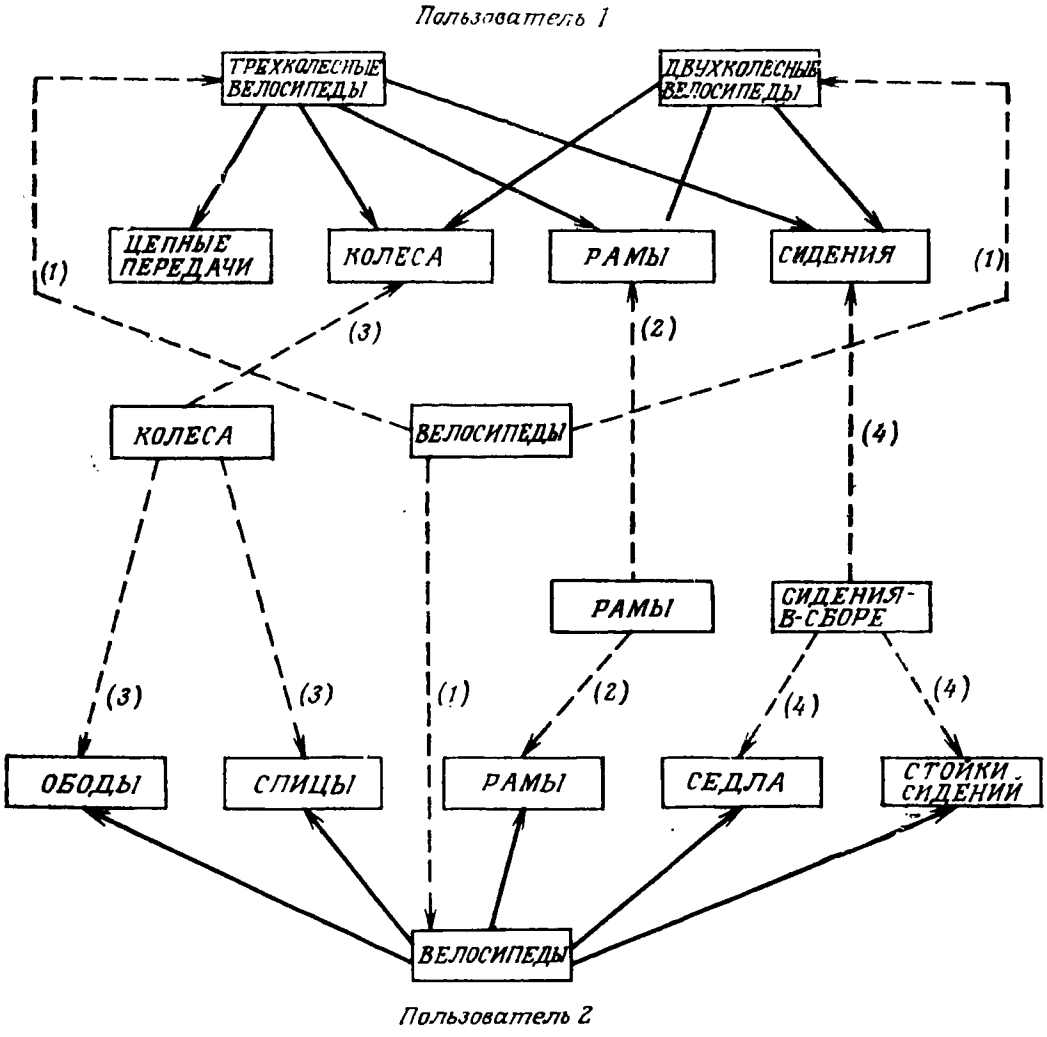


Рис. 5.15. Пример комплексного объединения.

Типы объединения

1. Объект ВЕЛОСИПЕДЫ (2)* является обобщением типа 1 объектов ТРЕХКОЛЕСНЫЕ-ВЕЛОСИПЕДЫ (1) и ДВУХКОЛЕСНЫЕ-ВЕЛОСИПЕДЫ (1).
2. Объект РАМЫ (1) идентичен объекту РАМЫ (2).
3. Объект КОЛЕСА (1) является агрегацией типа 1 объектов ОБОДЫ (2) и СПИЦЫ (2).
4. Объект СИДЕНИЕ-В-СБОРЕ (2) является агрегацией типа 2 объектов СИДЕНИЯ (1), СЕДЛА (1) и СТОЙКИ-СИДЕНИЙ (2).

* — номер пользователя.

сованную глобальную спецификацию. Необходимо подчеркнуть, что в этих предложениях содержится только описание модели объединения и средств представления различных типов подобия между объектами пользователей. Не было сделано никаких попыток автоматизировать идентификацию образов этих типов объединений в представлениях пользователей. Ограниченность семантических представлений препятствует любым попыткам подменить в ближайшем будущем человеческий фактор каким-либо алгоритмом.

5.3.3. Процесс объединения

До сих пор с целью упрощения мы имели дело только с попарными объединениями. В большинстве реальных ситуаций в процесс проектирования баз данных вовлекается огромное количество пользователей. Как в таком случае должен проходить процесс объединения?

Одна из альтернатив состоит в том, чтобы выполнить одно n-арное объединение за один шаг. В принципе это выполнимо, но не желательно. Это было бы слишком сложно как в алгоритмическом, так и концептуальном смысле и привело бы к огромному количеству ошибок и упущений. Кроме того, это сопровождалось бы большими временны́ми затратами, поскольку установление идентичности только для одного объекта пользователя потребовало бы проверки всех объектов остальных пользователей. При большом количестве пользователей и объектов это было бы в высшей степени неэффективно. Можно использовать эвристические способности человека для целенаправленного поиска, однако в очень больших системах это выходит за пределы возможностей одного человека.

Очевидной альтернативой является выполнение n-арного объединения, когда n много меньше числа пользователей. Можно показать, что бинарное (попарное) объединение является логическим завершением уменьшения значения n. В конечном результате процесс объединения будет представляться в виде бинарного дерева, в котором вершины будут обозначать представления пользователей, корень — окончательное глобальное представление, а промежуточные вершины — некоторые частичные объединения.

С концептуальной точки зрения это проще, поскольку каждый отдельный процесс включает только два представления, и поэтому более вероятно, что администратор баз данных или кто-то другой, кто координирует процесс объединения, будет лучше понимать семантику этих представлений пользователей. Этот подход особенно удобен для структур с иерархической организацией, которые обычно используются в коммерческой деятельности, так как персонал нижнего уровня может определять объединения на нижних уровнях, для которых необходима детализация, а персонал высшего звена может направлять объединение на высшем уровне, где наиболее очевидными являются абстракции агрегации и обобщения. Общая схема такого процесса показана на рис. 5.16.

С точки зрения эффективности форма бинарного объединения в виде дерева является важной потому, что при этом существует упорядоченность терминальных вершин, соответствующих представлениям пользователей. Преимущество бинар-

ного объединения перед n-арным состоит также в том, что результат любого бинарного объединения N объектов с M объектами будет содержать $N + M - X$ объектов, где X соответствует количеству совпадающих определений объектов в разных представлениях. При n-арном объединении значение X может быть бóльшим, однако это преимущество теряется из-за очень низкой скорости сравнения объектов на ранних стадиях объединения.

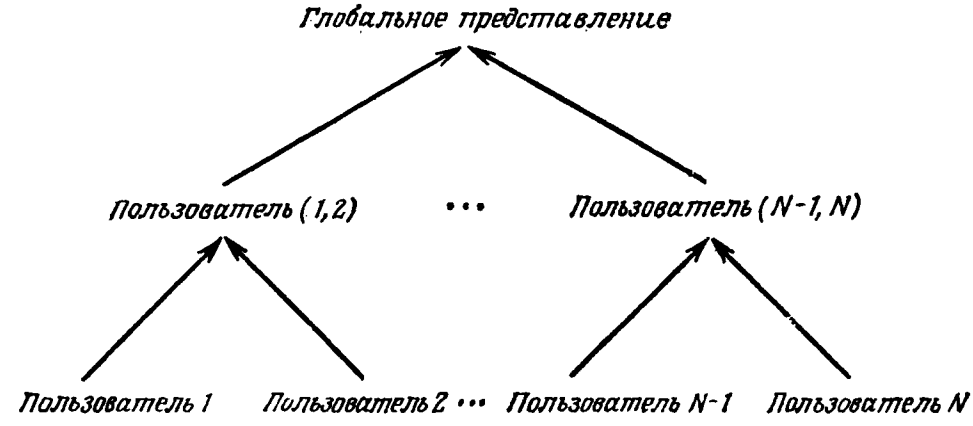


Рис. 5.16. Бинарное объединение.

Можно добиться выигрыша за счет симметризации дерева бинарных объединений. При этом минимизируется число сравниваемых объектов двух представлений, объединяемых на промежуточных шагах. Эта концепция иллюстрируется на рис. 5.17. Несимметричная структура объединения приводит к тому, что на каждом шаге рассматривается новое необъединенное представление, что приводит к уменьшению значения фактора X и увеличению числа сравнений на каждом шаге, поскольку оно пропорционально произведению числа объектов, полученных на предыдущих шагах.

Важность упорядочения вершин дерева состоит в том, что при соответствующем группировании начальных и промежуточных вершин можно максимизировать степень совпадения представлений, увеличивая в результате значение фактора X и минимизируя таким образом число объектов, рассматриваемых на последующих шагах процесса объединения. Другими словами, имеет смысл, насколько это возможно, начинать процесс объединения с представлений пользователей тех функциональных подразделений, которые в процессе функционирования имеют тесное взаимодействие. В большинстве организаций это соответствует иерархической структуре, которая эквивалентна структуре дерева. Тщательная подготовка на фазе планирования может минимизировать затраты усилий и времени в процессе объединения представлений.

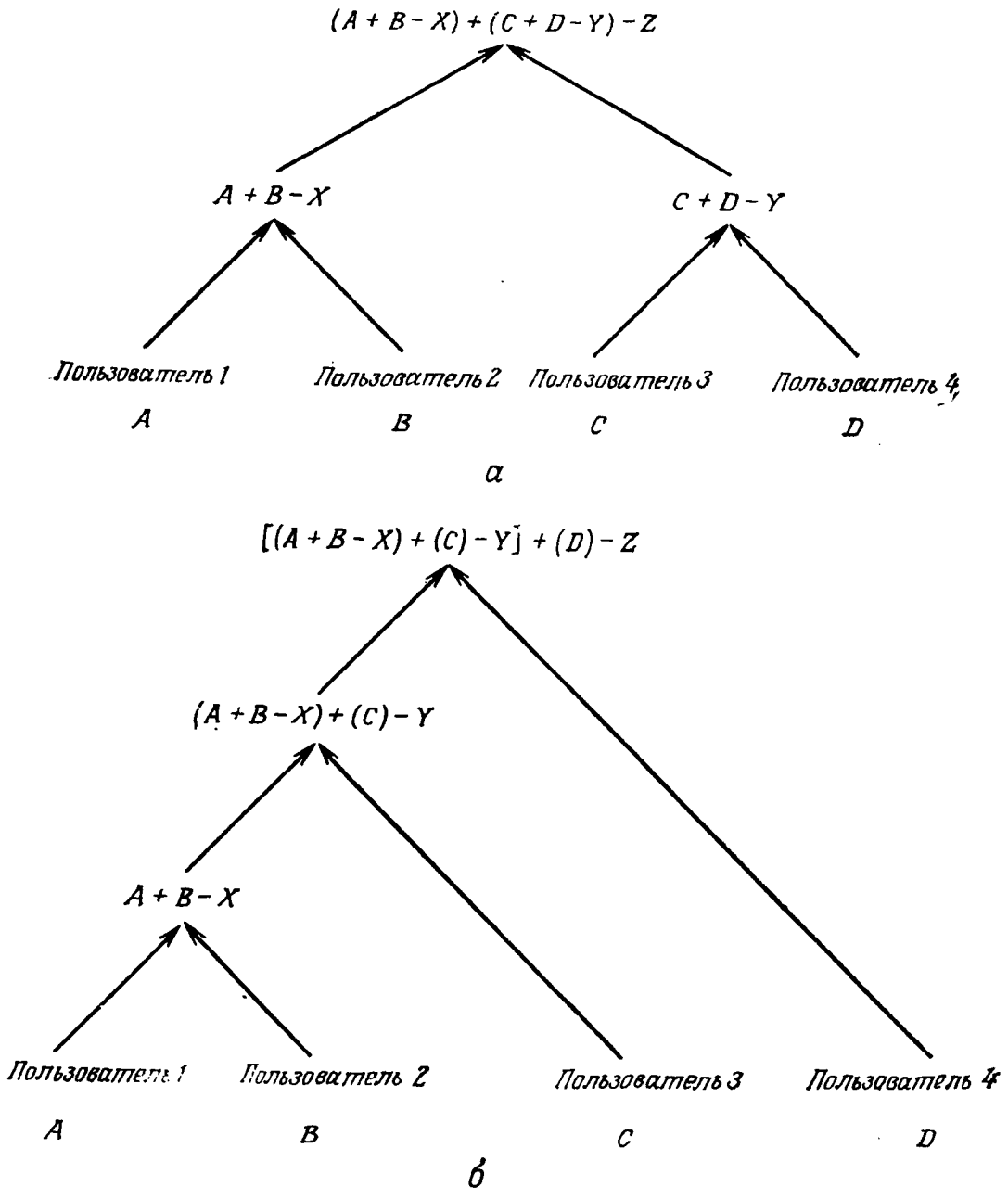


Рис. 5.17. Альтернативные структуры объединения.
а — симметричная, *б* — несимметричная.

Для упрощения процесса проектирования концептуальное проектирование рассматривалось как ряд представлений: обобщенное, информационное, прикладное и представление событий. Для упрощения процесса моделирования эти представления были снова разделены на ряд локальных представлений. Настало время для интеграции всех частных информационных представлений в единую глобальную информационную структуру.

Обычно процесс объединения развивается через проектные представления в следующем порядке. Начиная проектирование с информационного представления, следует объединить его с прикладным представлением, затем учесть представление событий и, наконец, слить с обобщенным представлением. Целесообразность такого порядка заключается в следующем. Начинать с информационного представления следует потому, что оно является наиболее богатым в отношении конструктивных элементов и дает наибольший вклад в конечную информационную структуру. Поэтому логично начинать именно с него. Прикладное представление сопоставляет реальности обработки данных с возможностями проектирования и, таким образом, обеспечивает реалистичность информационного представления. Далее рассматриваются времена наступления событий, процессов и выполнения приложений как средство обеспечения жизнеспособности проекта. Наконец, добавляются общие ограничения, завершая процесс проектирования.

Процесс объединения выполняется в несколько этапов:

Этап 1. Выбор представления

Вначале важно утвердить порядок следования проектных процедур в качестве последовательности объединения. Поскольку общий порядок был уже установлен, необходимо проверить его на соответствие целям и назначению проектируемой базы данных. Если, например, проектируется база данных для перерабатывающей отрасли, можно рассмотреть в качестве ключевых прикладное представление и представление событий и начать с них процесс объединения.

Этап 2. Упорядочение локальных представлений

Как только проектные представления выбраны и упорядочены, основное внимание в процессе объединения концентрируется на локальных представлениях внутри проектных представлений. В состав выбранного проектного представления входят несколько локальных представлений, следовательно, этот этап упорядочивает локальные представления в процессе объединения. Локальное представление упорядочивается в соответствии с важностью целей проектирования конкретной базы данных.

Этап 3. Объединение локальных представлений

Этот этап составляет основу процесса объединения. Для простоты и удобства мы используем бинарное объединение. Это означает, что в каждый момент времени объединяются только два представления пользователей, а n-арные объединения, насколько это возможно, исключаются. Порядок объединения определяется этапом 2, и процесс осуществляется следующим

образом. Вначале берутся два первых локальных представления в конкретном проектном представлении и объединяются на основе общих принципов объединения. Затем следующее по порядку локальное представление сливается с только что объединенными локальными представлениями. Процесс повторяется до тех пор, пока последнее локальное представление не будет слито со всеми предыдущими.

После того как процесс объединения будет завершен для первого проектного представления, рассматривается следующее проектное представление, и процесс продолжается, пока не будут интегрированы все представления.

Этап 4. Разрешение противоречий

Противоречия в процессе объединения возникают по ряду причин, главным образом благодаря большому количеству людей, вовлеченных в процесс проектирования, и отсутствию семантического содержания в конструктивных элементах моделирования. Противоречия могут быть также вызваны неполнотой или ошибочностью спецификаций или же некорректностью требований. Большинство из этих противоречий разрешается на этапе объединения путем применения соответствующих правил. Если имеются неразрешенные противоречия, которые должны разрешаться путем принятия специальных проектных решений, это происходит на данном этапе. Когда такое проектное решение принято, необходимо возвратиться к тому месту процесса объединения, где рассматриваемый конструктивный элемент включался в проектирование. В этом месте необходимо рассмотреть последствия принятого проектного решения с точки зрения его влияния на развитие процесса объединения.

Глава 6. Синтез атрибутов: пример концептуального проектирования

Донна Л. С. Ранд

В гл. 4 и 5 описывается основная терминология и методы концептуального проектирования. В данной главе на конкретном примере концептуального проектирования продолжается обсуждение методологии анализа требований, описанной в гл. 3.

Основными шагами проектирования, согласно этой методологии, являются:

- Шаг 2-1. Выбор сущностей и атрибутов.
- Шаг 2-2. Идентификация связей данных.
- Шаг 2-3. Представление информационной структуры в графической форме.
- Шаг 2-4. Интерпретация информационной структуры с целью ее верификации.

В последующих разделах каждый шаг проектирования будет исследован более глубоко.

6.1. Основная модель

Целью данной фазы анализа является идентификация компонентов выбранной концептуальной модели данных. Используется модель «сущность-связь», что предполагает анализ следующих компонентов:

1. Сущности представляются элементами данных, которые используются в организации для идентификации объектов, создания других элементов данных или для ссылок на другие элементы данных. При анализе будут различаться два типа сущностей: *уникальные сущности*, представляемые элементом данных, который идентифицирует отдельный, отличный от других предмет или объект (такой, как номер страхового полиса или порядковый номер), и *неуникальные сущности*, которые образуются, если для идентификации набора элементарных данных используются две или более уникальные сущности.

Наличие связи может привести к образованию новой сущности, идентичность которой устанавливается с помощью двух или более ключей. Такие сущности изображаются прямоугольником с двойной нижней линией. Они идентифицируются с помощью двух или более других сущностей и не могут существовать самостоятельно, если не существуют определяющие их сущности. Например, на рис. 6.1 каждый экземпляр ТОВАРНОГО-ЗАПА-

СА должен соотносится с существующим ТОВАРОМ и существующим СКЛАДОМ. Если исключить ТОВАР или СКЛАД, ТОВАРНЫЙ-ЗАПАС, относящийся к данному ТОВАРУ или СКЛАДУ, перестает существовать.

2. Атрибуты представляются элементами данных, которые имеют смысл только в отношении с уникальными или неуникальными сущностями (например, АДРЕС не имеет самостоятельного смысла, пока он не определен сущностью КЛИЕНТ).

Рис. 6.1. Графическое обозначение сущности, образованной вследствие наличия связи между двумя другими сущностями.

3. Связи представляют зависимости между двумя и более сущностями. Во время анализа могут быть выявлены три типа связей:

а. Связь типа *сущность-сущность* описывает квалифицируемое или неквалифицируемое отношение принадлежности между двумя сущностями. Например, сущность ПОСТАВЩИК владеет (неявным образом) сущностью ПОСТАВЛЯЕМЫЙ-ТОВАР.

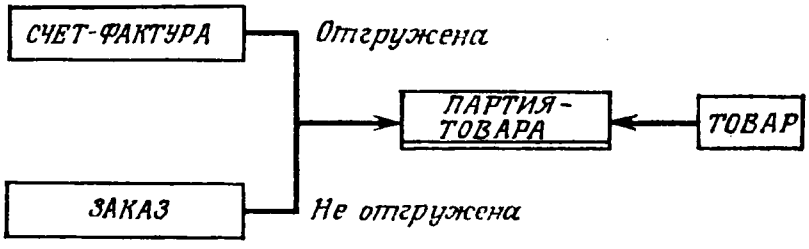


Рис. 6.2. Графическое изображение связи типа «или-или».

Эта связь означает, что сведения о поставляемом товаре не могут появиться в базе данных, если они не относятся к поставщику. В квалифицируемой связи принадлежность зависит от специфичных условий или статуса. Например, ЗАКАЗ владеет ПАРТИЕЙ-ТОВАРА до тех пор, пока ПАРТИЯ-ТОВАРА не отгружена. Как только ПАРТИЯ-ТОВАРА «отгружена», владельцем ПАРТИИ-ТОВАРА становится не ЗАКАЗ, а СЧЕТ-ФАКТУРА. Следовательно, связь между сущностью ПАРТИЯ-ТОВАРА и одной из двух сущностей ЗАКАЗ и СЧЕТ-ФАКТУРА квалифицируется статусом ПАРТИИ-ТОВАРА (рис. 6.2).

б. Связь типа *сущность-атрибут* описывает элементы данных, непосредственно принадлежащие одной или более сущностям. Как и в случае связи типа сущность-сущность, принадлежность

может быть квалифицируемой или неквалифицируемой. Если атрибут относится только к одной сущности, принадлежность является неквалифицируемой. В квалифицируемой связи атрибут всегда принадлежит одной главной сущности и квалифицируется одной или более второстепенными сущностями. Например, КОЛИЧЕСТВО-В-НАЛИЧИИ принадлежит сущности ВИД-ТОВАРА, но, кроме того, квалифицируется сущностью СКЛАД, поскольку тем самым уточняется принадлежность атрибута КОЛИЧЕСТВО-В-НАЛИЧИИ.

в. Связь типа *атрибут-атрибут* представляет неквалифицируемое отношение принадлежности между атрибутами, которые относятся к одной и той же сущности или связи типа сущность-сущность. Например, сущность НЕОПЛАЧЕННЫЕ-СЧЕТА владеет двумя атрибутами: ОБЩАЯ-СУММА-ЗАДОЛЖЕННОСТИ и ЗАДОЛЖЕННОСТЬ-СВЫШЕ-30-ДНЕЙ. Существование ОБЩЕЙ-СУММЫ-ЗАДОЛЖЕННОСТИ зависит от существования сущности НЕОПЛАЧЕННЫЕ-СЧЕТА. Существование же ЗАДОЛЖЕННОСТИ-СВЫШЕ-30-ДНЕЙ зависит от существования атрибута ОБЩАЯ-СУММА-ЗАДОЛЖЕННОСТИ. Это означает, что атрибут ЗАДОЛЖЕННОСТЬ-СВЫШЕ-30-ДНЕЙ не существует самостоятельно без существования ОБЩЕЙ-СУММЫ-ЗАДОЛЖЕННОСТИ.

6.2. Анализ информации с целью идентификации компонентов информационной концептуальной структуры

Методология анализа, предлагаемая в данном разделе, концентрирует внимание на преобразовании связей между задачами и данными в компоненты, описанные выше. Казалось бы, что, завершив проведение собеседований и проанализировав их, проектировщик на данной стадии проектирования может легко идентифицировать компоненты модели без дальнейшего анализа. Но хотя это и возможно для баз данных очень узкого направления, дальнейший анализ необходим по следующим причинам:

- Почти невозможно проанализировать все вероятные связи данных в случае, если число задач и элементов данных больше 30.
- Результаты анализа в значительной степени зависят от субъективного представления проектировщика о предметной области.
- Вполне вероятно, что неуникальные сущности и квалифицируемые связи не идентифицированы надлежащим образом, так как они обычно являются исключениями из правил и их трудно выявить даже с помощью формальных методов анализа.

Анализ делится на две основные фазы. Целью первой фазы (Шаг 2-1) является идентификация уникальных и неуникальных сущностей и атрибутов. Здесь же атрибуты будут подразделяться на те, которые относятся к уникальным сущностям, и те, которые относятся к неуникальным сущностям. Во второй фазе анализа (Шаг 2-2) результаты первой фазы будут использованы для идентификации связей между сущностями, сущностями и атрибутами и между атрибутами.

6.2.1 Идентификация сущностей и атрибутов (Шаг 2-1)

Первая фаза анализа основана на ряде гипотез, которые были разработаны в результате анализа использования элементов данных в задачах. Эти гипотезы выглядят следующим образом:

1. Элемент данных может являться *уникальной сущностью,* если:

а. Он используется в большом числе задач.

б. Он используется с большим числом других элементов данных.

в. Он довольно редко используется совместно с другими элементами данных по сравнению с общим числом его использования во всех задачах (в дальнейшем будем обозначать это как *коэффициент использования*).

2. Набор из двух и более элементов данных может являться *неуникальной сущностью,* если:

а. Каждый элемент данных из набора был вначале объявлен (или был близок к этому) как уникальная сущность.

б. Совместное использование каждого из элементов данных в составе набора встречается чаще по сравнению с отдельным использованием каждого из этих элементов данных в других задачах.

в. В тех задачах, в которых появляется этот набор элементов данных, чаще всего он используется совместно с другими элементами данных.

3. Элемент данных может являться *атрибутом, относящимся к уникальной сущности,* если:

а. Он используется в относительно небольшом числе задач.

б. Он используется с относительно небольшим числом других элементов данных.

в. Его совместное использование с другими элементами данных по сравнению с общим числом его использования во всех задачах (коэффициент использования) является довольно частым.

4. Элемент данных может являться *атрибутом, относящимся к неуникальной сущности,* если:

а. Он используется в среднем (т. е. ни в большом, ни в малом) числе задач.

б. Он используется со средним числом других элементов данных.

в. Его совместное использование с другими элементами данных по сравнению с общим числом его использования во всех задачах (коэффициент использования) является средним.

Важное замечание: Возможно, что набор элементов данных не будет содержать конкретного наименования сущности, т. е. агрегатного наименования для набора атрибутов (сравни разд. 4.2), а будет определяться только именами атрибутов. В этом случае атрибут, который однозначно идентифицирует эту сущность (т. е. является ее ключом), будет идентифицироваться как сущность в соответствии с вышеприведенной классификацией. Когда это встречается, то говорят, что идентифицирующий атрибут представляет сущность. Тем не менее этой сущности должно быть присвоено фактическое наименование, чтобы она могла представлять агрегацию атрибутов, включающую как идентифицирующий атрибут, так и остальные неидентифицирующие атрибуты.

В первой части анализа (Шаг 2-1-1) проводится идентификация уникальных и неуникальных сущностей и классификация

	Использование в задачах	*Использование совместно с другими элементами данных*	*Коэффициент использования*
Частое			
Среднее			
Редкое			

Рис. 6.3. Матрица критериев классификации элементов данных.

атрибутов по принадлежности к уникальным и неуникальным сущностям. Для этого необходимо определить значения элементов матрицы, показанной на рис. 6.3. Эти значения будут использоваться в качестве критериев для классификации всех элементов данных.

Чтобы определить эти значения, необходимо по каждой функциональной задаче проанализировать данные, связанные с этой задачей. Процесс анализа включает пять шагов. На пер-

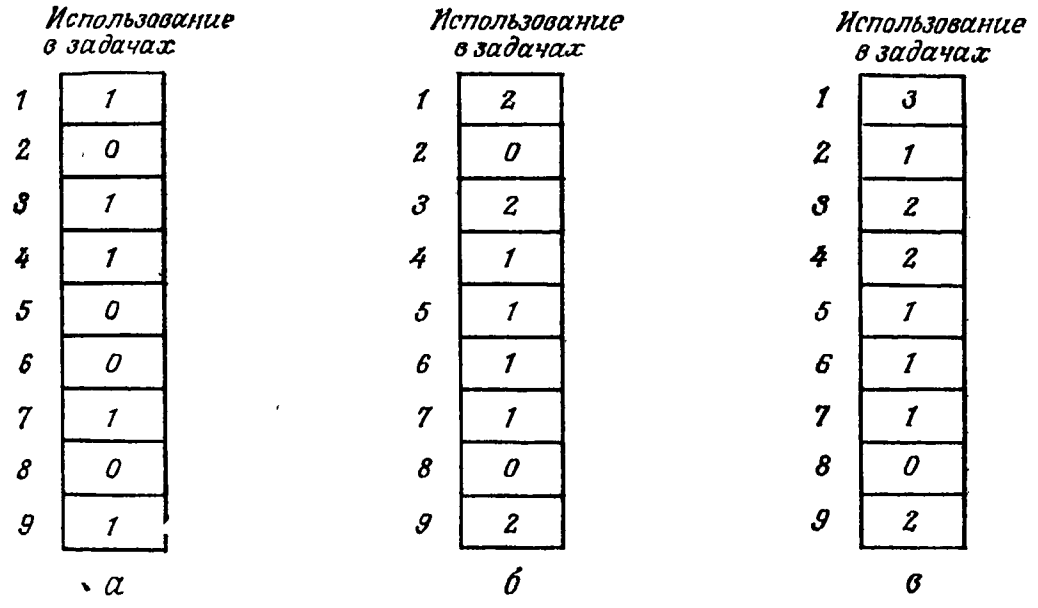


Рис. 6.4. Вектор использования задач.

а — значение вектора для первой задачи «Принять заказ»; *б* — значение вектора после добавления второй задачи «Проверить кредитоспособность»; *в* — значение вектора для всех трех задач.

Замечание: номер компонента обозначает номер элемента данных.

вом шаге этого процесса строится матрица, отображающая все связи между задачами и данными. Для наглядности используем пример.

Предположим, что в процессе анализа результатов собеседований были определены следующие связи между задачами и данными:

Задача *«Принять заказ»* использует элементы данных, 1, 3, 4, 7, 9.

• Задача *«Проверить кредитоспособность»* использует элементы данных 1, 3, 5, 6, 9.

• Задача *«Отклонить заказ»* использует элементы данных 1, 2, 4.

Процесс начинается с отображения в матрице связей данных по каждой задаче. Одновременно ведется подсчет числа задач, в которых используется каждый элемент данных. Для этого необходимо:

1. Определить матрицу (которая называется *матрицей связи*), число строк и столбцов которой равно числу элементов данных во всех задачах (т. е. девяти для трех задач).

2. Определить вектор использования задач, в котором число компонент равно числу элементов данных.

3. Для каждой связи между задачей и данными выполнить следующее:

а. Заполнить вектор использования задач. Для этого, последовательно просматривая список задач и данных, увеличивать на 1 значения тех компонент вектора, номера которых соответствуют номерам элементов данных, используемых в задаче. Например, задача «Принять заказ» использует элементы данных 1, 3, 4, 7 и 9. В соответствии с этим значения компонент 1, 3, 4, 7 и 9 вектора должны быть увеличены на 1 (рис. 6.4, *а*). На рис. 6.4, *б* показано значение вектора после просмотра второй задачи, а на рис. 6.4, *в* — после третьей.

б. Задать в матрице связей все взаимосвязи данных. Это выполняется путем регистрации связей каждого элемента данных со всеми другими элементами данных, используемыми в задаче. Например, в задаче «Принять заказ» используются элементы данных 1, 3, 4, 7 и 9. Выбрав первый по порядку элемент данных 1, поставим ему в соответствие строку 1 матрицы. Увеличим на 1 значения элементов матрицы, находящихся на пересечении строки 1 и столбцов 3, 4, 7 и 9, соответствующих остальным элементарным данным (рис. 6.5, *а*). Как только связи элемента данных 1 учтены, выбрать строку 3, соответствующую элементу данных 3, и увеличить на 1 значения элементов матрицы, находящихся на пересечении строки 3 и столбцов 1, 4, 7 и 9. Повторить этот процесс для всех данных, используемых в задаче 1. На рис. 6.5, *б* показана матрица связей после анализа задачи 1. На рис. 6.5, *в* показано ее значение после анализа и соответствующего отображения задачи 2, а на рис. 6.5, *г* — после анализа всех трех задач.

Результирующая матрица связи R может быть описана через векторы использования данных в каждой из трех задач u, v и w следующим выражением:

$$R_{ij} = (u_i u_j^T + v_i v_j^T + w_i w_j^T)\,\delta_{ij}, \qquad (6\text{-}1)$$

где компонента вектора u_i представляет использование элемента данных i в задаче u, T обозначает транспозицию, а δ_{ij} определяется выражением:

$$\delta_{ij} = \begin{cases} 0, & \text{если } i = j, \\ 1, & \text{если } i \neq j. \end{cases}$$

После отображения всех связей данных в матрице следующий шаг анализа (Шаг 2-1-2) заключается в построении вектора, каждая компонента которого соответствует элементу данных, а её значение — общему числу соотносящихся с ним остальных элементов данных. Для этого необходимо подсчитать число ненулевых элементов в каждой строке (т. е. для

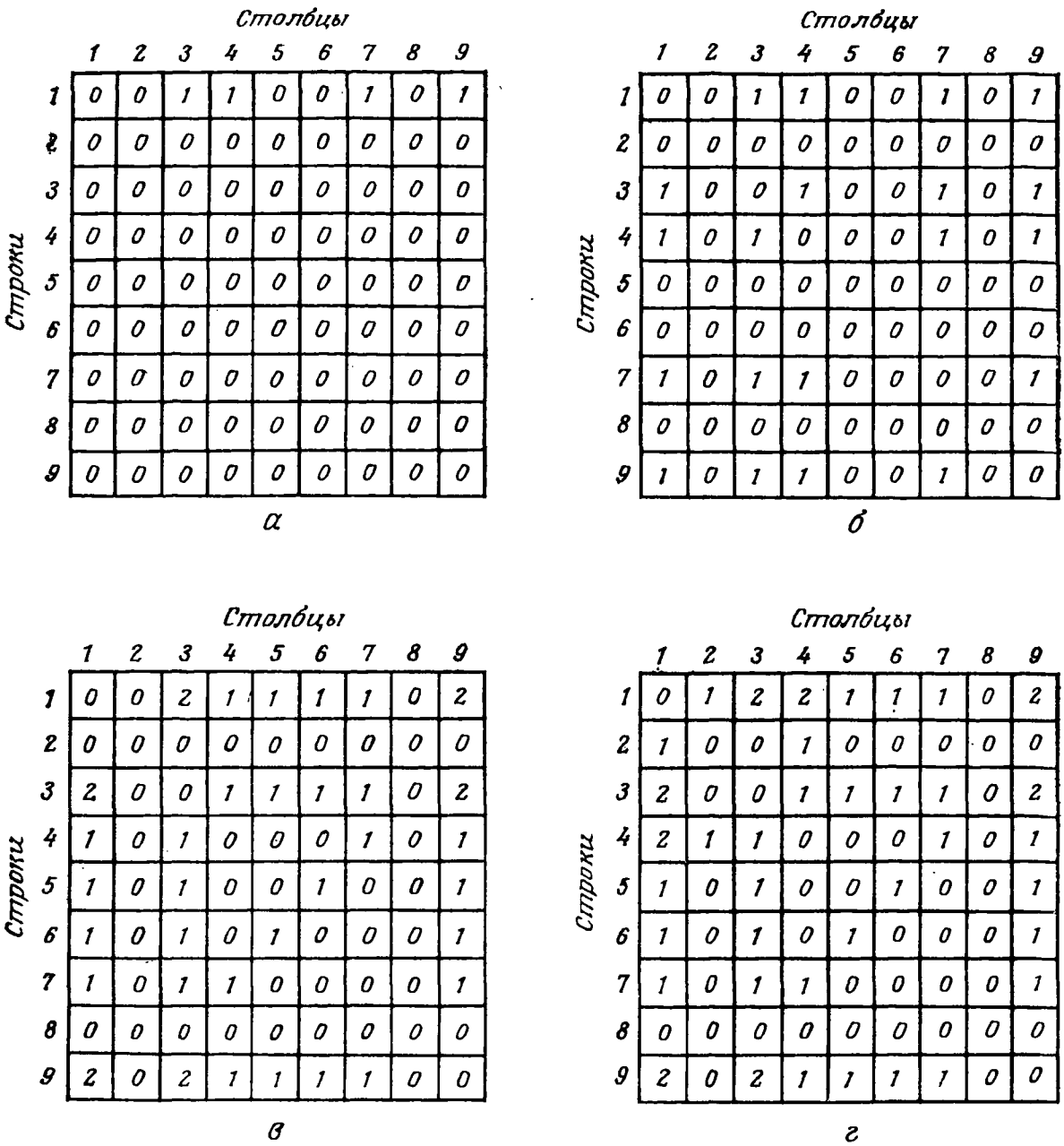


Рис. 6.5. Матрица связей.

а — отображены связи первого данного задачи «Принять заказ»; *б* — отображены все связи между данными задачи «Принять заказ»; *в* — отображены все связи между данными задач 1 и 2; *г* — отображены все связи между данными для трех задач.

Замечание: номера строк и столбцов соответствуют номерам элементов данных.

каждого элемента данных) матрицы. Результат этого подсчета отражается в виде *вектора полного числа связей* (рис. 6.6).

На третьем шаге анализа (Шаг 2-1-3) матрица связи и эти два вектора используются для построения третьего вектора, компоненты которого соответствуют коэффициенту использования каждого элемента данных. Для вычисления этого коэффициента необходимо:

1. Преобразовать матрицу связи, выразив в процентах отношение числа связей элемента данных (например, элемент данных 1 — строка 1 — соотносится с элементом данных 3 — столбец 3 — дважды, см. рис. 6.5, *г*) к общему числу его использования в задачах (например, элемент данных 1 — компонента 1 вектора использования задач, рис. 6.4, *в* — используется в трех задачах; в соответствии с этим коэффициент его использования с элементом данных 3 — столбец 3 матрицы — составляет 66 %, см. рис. 6.7).

2. Подсчитать в каждой строке матрицы число элементов, значение которых превышает значение некоторого граничного параметра *и*, обозначающего частое использование. (*Замечание:* Приемлемым начальным значением для *и* может быть, например, 70 %. Если число связей между задачами и данными невелико, значение параметра может быть увеличено, и наоборот, уменьшено при слишком большом числе связей.) Результат подсчета по каждой строке присваивается соответствующей компоненте *вектора учета использования* (рис. 6.8).

3. Значение каждой компоненты вектора учета использования (т. е. учета наиболее частого использования связей данных) поделить на соответствующие значения компонент вектора полного числа связей

	Полное число связей
1	7
2	2
3	6
4	5
5	4
6	4
7	4
8	0
9	6

Рис. 6.6. Вектор полного числа связей.

Замечание: номер компоненты соответствует номеру элемента данных.

Столбцы

Строки	1	2	3	4	5	6	7	8	9
1	0	33%	66%	66%	33%	33%	33%	0	66%
2	100%	0	0	100%	0	0	0	0	0
3	100%	0	0	50%	50%	50%	50%	0	100%
4	100%	50%	50%	0	0	0	50%	0	50%
5	100%	0	100%	0	0	100%	0	0	100%
6	100%	0	100%	0	100%	0	0	0	100%
7	100%	0	100%	100%	0	0	0	0	100%
8	0	0	0	0	0	0	0	0	0
9	100%	0	100%	50%	50%	50%	50%	0	0

Рис. 6.7. Матрица связей после преобразования числа связей между данными в процентное отношение к общему числу использования во всех задачах.

данных. Результаты, выраженные в процентном отношении, присвоить соответствующим компонентам *вектора коэффициента использования* (рис. 6.9).

На четвертом шаге анализа (Шаг 2-1-4) эти три вектора используются для построения гистограмм, позволяющих выбрать значения параметров, определяющие частое, среднее и редкое использование данных в задачах, а также совместное использование с другими элементами данных. Кроме того, гистограмма коэффициента использования позволяет определить значения граничных точек, определяющих высокое, среднее и низкое значения коэффициента использования.

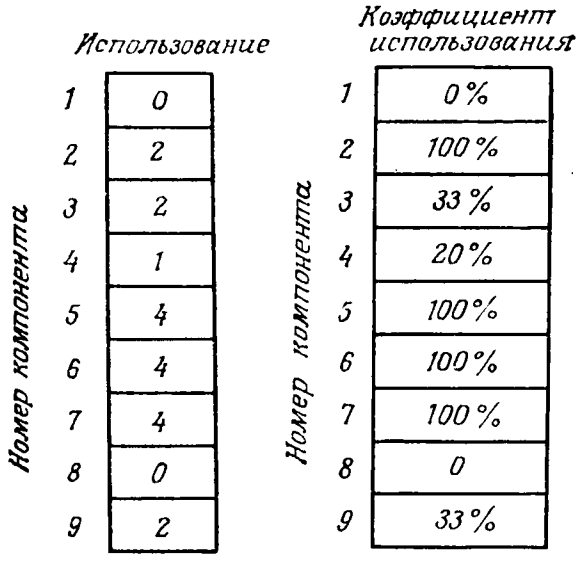


Рис. 6.8. Вектор учета использования. Задает число элементов строк матрицы (рис. 6.7), значение которых превышает значение параметра, определяющего частое использование.

Рис. 6.9. Вектор коэффициента использования.

На рис. 6.10 — 6.12 показаны гистограммы для каждого из трех векторов. Необходимо отметить, что выбор значения интервала, равного 1 (кроме гистограммы коэффициента использования), в нашем примере обусловлен небольшим числом связей между задачами и данными. При проектировании реальных баз данных значение интервала в гистограмме использования данных в задачах может быть равно 5, а для гистограммы использования с другими элементами данных — от 5 до 10. Построенные гистограммы затем используются для визуального анализа и выбора граничных значений ранее упоминавшихся параметров. Поскольку гистограммы, показанные на рис. 6.10—6.12, основаны на результатах анализа только трех задач, они не являются показательными для проектирования реальной базы данных. На рис. 6.13 показан тип кривой распределения, который обычно получается при анализе «использования в задачах» и «использования с другими элементами данных». Отметим, что разграничение кривой на интервалы с частым, средним и редким использованием производится в точках, где разница в частоте использования наиболее ощути-

Рис. 6.10. Гистограмма использования данных в задачах (см. рис. 6.4, *в*).

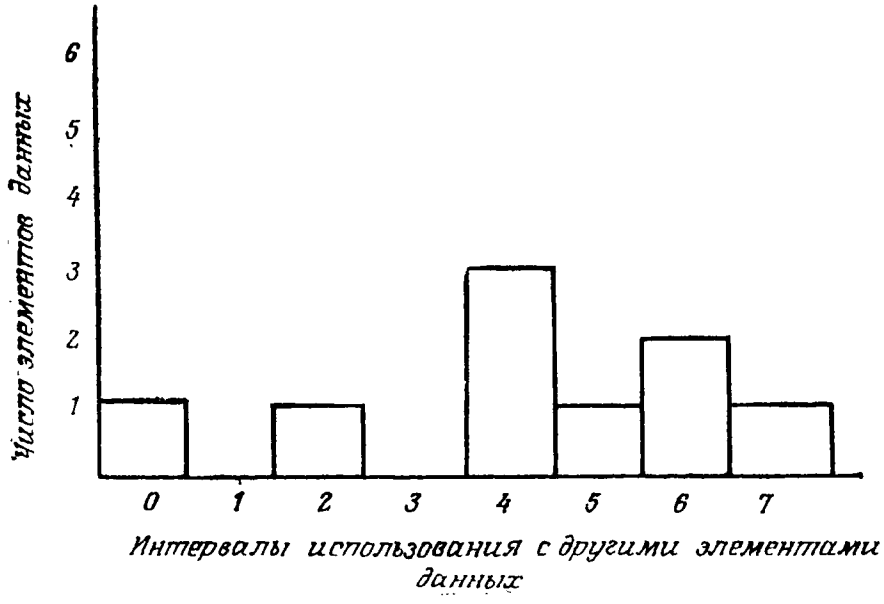


Рис. 6.11. Гистограмма использования с другими элементами данных (см. рис. 6.6).

ма. Используя способ разбиения на интервалы, показанный на рис. 6.13, можно определить для рис. 6.10 и 6.11 следующие граничные значения параметров:

- Использование элемента данных в задачах определяется как
  - — частое, если его значение $\geqslant 3$;
  - — среднее, если его значение >1 и <3;
  - — редкое, если его значение $\leqslant 1$.
- Совместное использование элемента данных с другими элементами данных определяется как
  - — частое, если его значение $\geqslant 7$;
  - — среднее, если его значение > 2 и < 7;
  - — редкое, если его значение $\leqslant 2$.

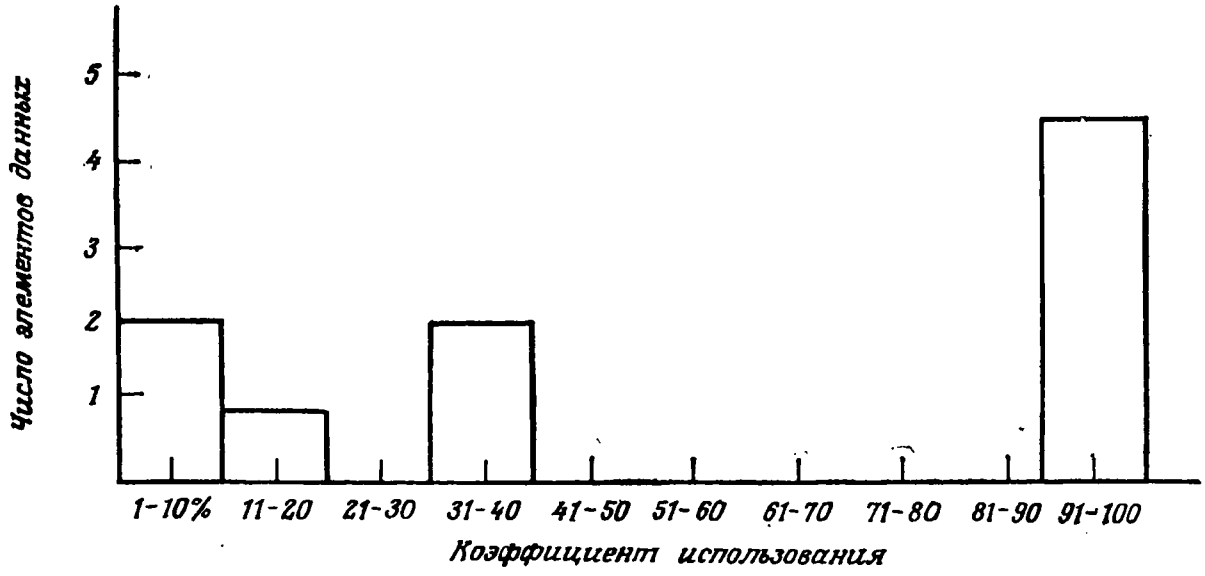


Рис. 6.12. Гистограмма коэффициента использования (см. рис. 6.9).

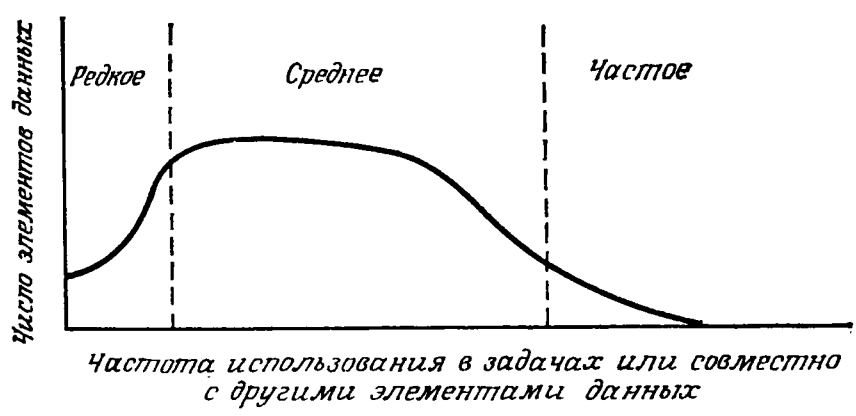


Рис. 6.13. Типичная кривая, получаемая при анализе использования данных в задачах или их использования с другими элементами данных.

Кривая, обычно получаемая при анализе коэффициента использования, показана на рис. 6.14. Поскольку цель анализа коэффициента использования — определить наиболее частое использование, на кривой обычно выделяется два отличных друг от друга участка. Используя способ деления на интервалы, показанный на рис. 6.14, можно определить граничные значения коэффициента использования (из нашего примера) так, как показано на рис. 6.15.

В зависимости от числа связей между задачами и данными и сферы применения базы данных целесообразно определить две или три граничные точки с целью проверки их влияния на распределение элементов данных по категориям сущностей и атрибутов. Если при этом в распределениях обнаруживаются значительные расхождения, необходимо вы-

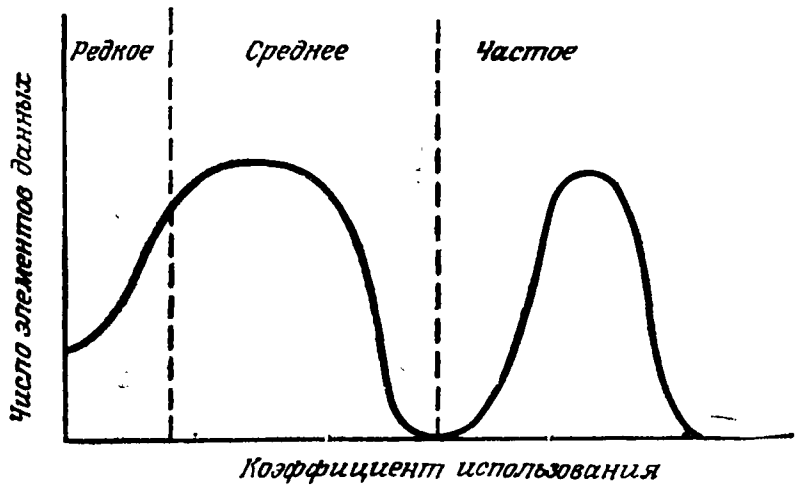


Рис. 6.14. Типичная кривая анализа коэффициента использования.

	Использование в задачах	Использование с другими данными	Коэффициент использования
Сущность	⩾3	⩾7	⩽11%
Атрибут, относящийся к неуникальной сущности	>1, <3	>2, <7	>11%, <41%
Атрибут, относящийся к уникальной сущности	⩽1	⩽2	⩾41%

Рис. 6.15. Заполненная матрица критериев классификации элементов данных.

полнить шаг 2-1, сгруппировав все задачи в разрезе подразделений организации. Такой раздельный анализ позволит проектировщику определить сущности для каждого подразделения и в соответствии с этим выбрать граничные значения параметров для всего проекта в целом. (*Замечание:* Вполне возможно, что в больших базах данных некоторые сущности могут быть определены только в результате анализа данных в разрезе подразделений организации, а не организации в целом. Поэтому должны быть выполнены оба вида анализа.)

Завершая четвертый шаг анализа, необходимо поместить три вышеназванных вектора в таблицу вместе с элементами данных и в соответствии со значениями компонент векторов и критериями частого, среднего и редкого использования данных

(рис. 6.15) определить принадлежность элемента данных к категории сущности или атрибута (табл. 6.1).

Таблица 6.1. **Классификация сущностей и атрибутов, относящихся к уникальным и неуникальным сущностям**

Элемент данных	Использование в задачах	Использование с другими задачами	Коэффициент использования	Классификация
1	3	7	1	Сущность
2	1	2	100	Атрибут уникальной сущности
3	2	6	33	Атрибут неуникальной сущности
4	2	5	20	То же
5	1	4	100	Неопределенное значение — возможно, атрибут уникальной сущности
6	1	4	100	То же
7	1	4	100	То же
8	0	0	0	Не используется
9	2	6	33	Атрибут неуникальной сущности

Пятый и последний шаг первой фазы анализа (Шаг 2-1-5) заключается в определении возможных комбинаций сущностей, составляющих неуникальную сущность. Для этого используется список сущностей и связей между задачами и данными. Обращаясь к примеру, представим, что во время четвертого шага анализа были определены пять уникальных сущностей, показанных в табл. 6.2, и что эти сущности используются в задачах так, как показано в табл. 6.3. Используя список задач (табл. 6.3) и сущностей (табл. 6.2), для определения возможных неуникальных сущностей необходимо выполнить следующую последовательность действий:

Таблица 6.2. **Пять сущностей, выделенных в результате анализа**

Номер элемента данных	Наименование	Номер элемента данных	Наименование
10	ТОВАР	22	ТОРГОВЫЙ-АГЕНТ
86	СКЛАД	68	ПОСТАВЩИК
61	КЛИЕНТ		

Таблица 6.3. **Соответствие между задачами и элементами данных** [1])

Номер задачи	Описание	Номера элементов данных
1	Проверить наличие запаса товара	**10**, 57, 69, **86**, 107, 89, **61**
2	Выписать заказ	**61**, 8, 9, **10**, 57, 6, 18, **22**
3	Получить товар	**86**, 69, **10**, 89
4	Назначить цену	**10**, 67, 106, **61**, 16
5	Заказать товар	**10**, 69, **86**, 89, 57, **68**

[1]) Выделены уникальные сущности.

1. Определить сущности, связанные с каждой задачей и составить список комбинаций сущностей:

Задача 1 10, 61, 86
Задача 2 10, 22, 61
Задача 3 10, 86
Задача 4 10, 61
Задача 5 10, 68, 86

2. Определить все возможные комбинации двух и более сущностей (табл. 6.4, столбец 1).

Таблица 6.4. **Анализ комбинаций сущностей**

Комбинация сущностей	Число совместных появлений	Частота появления, % (значение столбца 2, деленное на общее число задач)	Комбинация сущностей	Число совместных появлений	Частота появления, % (значение столбца 2, деленное на общее число задач)
10, 86	3	60	10, 22	1	20
10, 61	3	60	61, 10, 22	1	20
86, 61	1	20	10, 68	1	20
10, 86, 61	1	20	86, 68	1	20
61, 22	1	20	10, 86, 68	1	20

3. Определить количество появлений каждой комбинации (табл. 6.4, столбец 2).

4. Используя общее число задач (в данном случае 5), вычислить в процентах частоту появления каждой комбинации (табл. 6.4, столбец 3).

5. Выбрать для рассмотрения в качестве неуникальных сущностей те комбинации, которые появляются наиболее часто

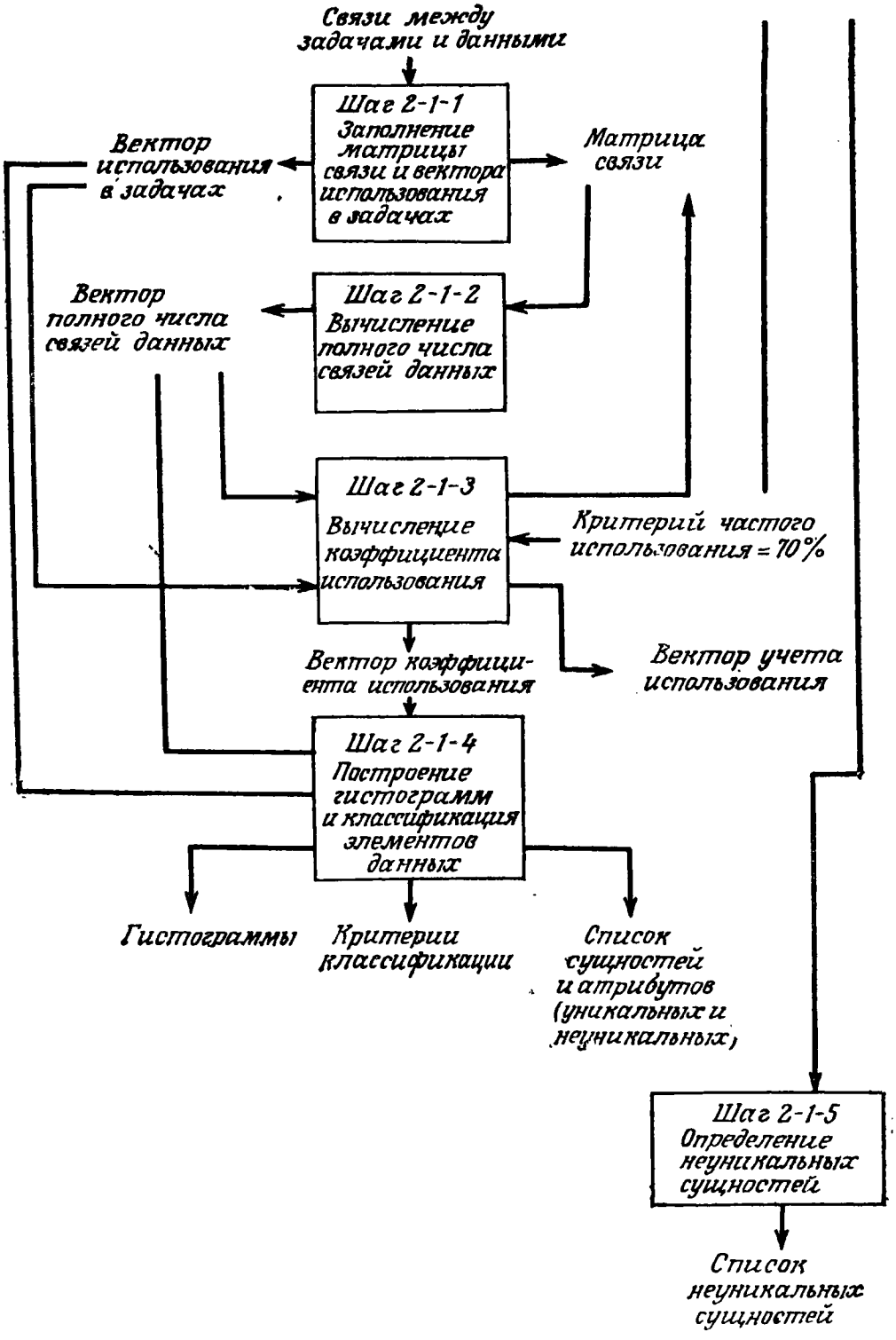


Рис. 6.16. Блок-схема первой фазы концептуального проектирвания (Шаг 2-1) — выбора сущностей и атрибутов.

(примем в качестве критерия величину 50 %). (*Замечание*: Не имеет смысла устанавливать жесткие правила для выбора критерия, поскольку это зависит от конкретных условий рассматриваемой задачи.) В данном примере наиболее часто появляются следующие комбинации:

- ТОВАР (10) и СКЛАД (86), появляющиеся совместно с частотой 60 % в задачах 1, 3 и 5.
- КЛИЕНТ (61) и ТОВАР (10), появляющиеся совместно с частотой 60 % в задачах 1, 2 и 4.

6. Для каждой из комбинаций (т. е. ТОВАР/СКЛАД и КЛИЕНТ/ТОВАР) рассмотреть элементы данных, с которыми они появляются совместно (табл. 6.5). Как видно из таблицы, имеются три элемента данных (57, 69, 89), которые сравнительно часто появляются вместе с ТОВАРОМ (10) и СКЛАДОМ (86). В то же время элементы данных, которые используются вместе с ТОВАРОМ (10) и КЛИЕНТОМ (61), почти не перекрываются между собой. Данный анализ показывает, что комбинацию ТОВАР/СКЛАД можно классифицировать как неуникальную сущность, в то время как ТОВАР/КЛИЕНТ — нельзя.

Таблица 6.5. **Элементы данных, используемые с комбинациями сущностей**

Комбинации сущностей	Задачи, в которых они появляются	Другие элементы данных	Комбинации сущностей	Задачи, в которых они появляются	Другие элементы данных
10, 86	1	57, 69, 89, 107	10, 61	1	57, 69, 89, 107
	3	69, 89		2	6, 8, 9, 18, 57
	5	57, 69, 89		4	16, 67, 106

Идентификацией неуникальных сущностей завершается первая фаза анализа информационной структуры. Блок-схема на рис. 6.16 содержит пять основных шагов этой фазы.

6.2.2. Идентификация связей (Шаг 2-2)

В разд. 6.1 были определены три типа связей: связи между сущностями; связи между сущностями и атрибутами; связи между атрибутами.

Целью второй фазы анализа (Шаг 2-2) является идентификация этих трех типов связей с тем, чтобы ранее определенные сущности и атрибуты могли бы быть выражены через графические обозначения модели «сущность-связь».

Идентификация связей между сущностями является первым шагом этого анализа (Шаг 2-2-1). Для этого необходимо список уникальных и неуникальных сущностей проанализировать совместно с набором правил, полученных в результате собеседований с руководством и выражающих политику организации, после чего составить таблицу, описывающую связи сущностей. Последовательность выполнения такого анализа показана в следующем примере:

1. Составить список всех уникальных и неуникальных сущностей, определенных на первой фазе анализа (табл. 6.6).

Таблица 6.6. **Список уникальных и неуникальных сущностей, определенных во время первой фазы анализа**

15. ЗАКАЗ	68. ПОСТАВЩИК
10. ТОВАР	54. ПОСТАВЛЯЕМЫЙ-ТОВАР
22. ТОРГОВЫЙ-АГЕНТ	110. ЗАКАЗ-НА-ПОСТАВКУ
86. СКЛАД	10. ТОВАР; 86. СКЛАД
61. КЛИЕНТ	15. ЗАКАЗ; 10. ТОВАР; 106. СЧЕТ-ФАКТУРА
106. СЧЕТ-ФАКТУРА	22. ТОРГОВЫЙ-АГЕНТ; 61. КЛИЕНТ

2. Просмотреть список правил, выражающих политику организации и полученных во время собеседований с руководством (табл. 6.7).

Таблица 6.7. **Примеры правил функционирования организации — выдержки из протокола собеседований с руководством**

1. Клиент всегда имеет дело с одним торговым агентом.
2. Каждый товар может появиться в составе любого товарного запаса, поэтому необходимо следить за количеством товара на каждом складе.
3. Заказы от клиента принимаются только после того, как официально проверена его кредитоспособность, открыт счет и назначен торговый агент.
4. Закупки производятся у многих поставщиков, поэтому имеются полные списки всех товаров, предлагаемых каждым поставщиком.
5. Часто один и тот же товар может быть закуплен у разных поставщиков, поэтому каждый вид товара имеет собственный номер, который используется при ссылке на поставляемые товары.
6. С целью улучшения обслуживания отдельная партия товара, соответствующая одному заказу, может комплектоваться из разных складов.
7. Как только партия товара, соответствующая заказу, отгружена, на нее выписывается счет-фактура.

3. Проанализировать каждое правило из табл. 6.7 с целью выявления отношения принадлежности между сущностями или выявления атрибутов, относящихся к неуникальной сущности.

Сущность-владелец связи	Сущность-член связи	Идентификация неуникальной сущности-члена связи	Условия членства и другие комментарии
(1) 22. ТОРГОВЫЙ-АГЕНТ	→ 61. КЛИЕНТ		
(2) 10. ТОВАР 86. СКЛАД		→ ТОВАРНЫЙ-ЗАПАС	
(3) 22. ТОРГОВЫЙ-АГЕНТ 61. КЛИЕНТ	→ 61. КЛИЕНТ → 15. ЗАКАЗ		
(4) 68. ПОСТАВЩИК	→ 54. ПОСТАВЛЯЕМЫЙ-ТОВАР		
(5) 10. ТОВАР	→ 54. ПОСТАВЛЯЕМЫЙ-ТОВАР		
(6) 86. СКЛАД		→ ПАРТИЯ-ТОВАРА (неясно, какая неуникальная сущность идентифицирует партию-товара)	Комплектуется из
15. ЗАКАЗ		→ ПАРТИЯ-ТОВАРА	
(7) 15. ЗАКАЗ 106. СЧЕТ-ФАКТУРА 10. ТОВАР		⊣ → ПАРТИЯ-ТОВАРА ⊣	Связь с сущностью ЗАКАЗ в случае, если „не отгружена". Связь с сущностью СЧЕТ-ФАКТУРА только после того, как „отгружена"
(8) 22. ТОРГОВЫЙ-АГЕНТ 61. КЛИЕНТ			Связь определена, (строки 1 и 3), но не образована группа атрибутов

Рис. 6.17. Определение связей сущностей.

Далее следует пример анализа правил, описанных в табл. 6.7. Результатом этого анализа является схема связей сущностей, показанная на рис. 6.17.

• Правило 1 касается двух сущностей: КЛИЕНТ и ТОРГОВЫЙ-АГЕНТ. Оно устанавливает, что каждый КЛИЕНТ соотносится только с одним ТОРГОВЫМ-АГЕНТОМ, подразумевая при этом, что ТОРГОВЫЙ-АГЕНТ «владеет» КЛИЕНТОМ (строка 1, рис. 6.17).

• Правило 2 касается двух сущностей: ТОВАР и СКЛАД. Оно также идентифицирует ТОВАРНЫЙ-ЗАПАС как атрибут, которым «владеет» неуникальная сущность. Это означает, что комбинация ТОВАР и СКЛАД «владеет» атрибутами, составляющими ТОВАРНЫЙ-ЗАПАС (строка 2).

• Правило 3 включает три сущности: КЛИЕНТ, ЗАКАЗ и ТОРГОВЫЙ-АГЕНТ, и определяет две связи: между ТОРГОВЫМ-АГЕНТОМ и КЛИЕНТОМ и между КЛИЕНТОМ и ЗАКАЗОМ (строка 3).

• Правило 4 непосредственно касается сущности ПОСТАВЩИК и косвенным образом — сущности ПОСТАВЛЯЕМЫЙ-ТОВАР. Это обозначает, хотя и неявно, наличие связи между сущностями ПОСТАВЩИК и ПОСТАВЛЯЕМЫЙ-ТОВАР (строка 4).

• Правило 5 устанавливает связь между сущностями ТОВАР и ПОСТАВЛЯЕМЫЙ-ТОВАР (строка 5).

• Правило 6 относится к двум сущностям: ЗАКАЗ и СКЛАД. Смысл этого правила состоит в том, что связь между названными сущностями осуществляется через ПАРТИЮ-ТОВАРА. Поскольку комбинация ЗАКАЗ/СКЛАД не является неуникальной сущностью, из правила явным образом не следует, как эта связь может быть определена. Однако необходимо отметить два момента, которые могут в дальнейшем прояснить ситуацию. Во-первых, то, что ПАРТИЯ-ТОВАРА может комплектоваться из разных складов, и, во-вторых, то, что ЗАКАЗЫ владеют ПАРТИЯМИ-ТОВАРА (строка 6).

• Правило 7 помогает уточнить нечеткое определение связи, предполагаемой правилом 6. Оно включает две сущности: ЗАКАЗ и СЧЕТ и нигде не определенную группу атрибутов ПАРТИЯ-ТОВАРА. Просматривая список неуникальных сущностей (табл. 6.6), находим комбинацию сущностей ЗАКАЗ, ТОВАР и СЧЕТ-ФАКТУРА. Можно с уверенностью предположить, что ПАРТИЯ-ТОВАРА определяется этими сущностями. Кроме того, связь ПАРТИИ-ТОВАРА с сущностью ЗАКАЗ определена при условии «не отгружена», а с сущностью СЧЕТ-ФАКТУРА — при условии «отгружена» (строка 7).

После того как набор правил, определяющих политику организации, проанализирован, необходимо просмотреть список неуникальных сущностей, чтобы проверить, все ли группы атрибутов определены (как, например, была определена группа атрибутов ТОВАРНЫЙ-ЗАПАС через неуникальную сущность ТОВАР/СКЛАД). Если группа атрибутов не определена, надо поместить неуникальные сущности в конец списка связей сущностей, отметив, что эта специфическая связь пока еще не определена (строка 8, рис. 6.17). Даже если связь сущностей

определена ранее во время анализа правил поведения, весьма вероятно, что одно определение будет противоречить другому. Это противоречие может быть в дальнейшем выявлено с помощью графических обозначений информационной структуры.

Кроме идентификации связей сущностей второй шаг этой фазы (Шаг 2-2-2) включает идентификацию связей между сущностями и атрибутами. На подготовительной стадии строятся три списка. Первый список должен содержать все элементы данных, классифицированные как атрибуты уникальных сущностей, а второй — атрибуты неуникальных сущностей

Таблица 6.8. **Список атрибутов**

Атрибуты, относящяеся к уникальным сущиостям	Атрнбуты, отиосящнеся к неуникальным сущностям
14. ОПИСАНИЕ-ТОВАРА	42. КОЛИЧЕСТВО-В-НАЛИЧИИ
16. НОМЕР-КЛИЕНТА	56. ВРЕМЯ-ПОПОЛНЕНИЯ-ЗАПАСА
18. АДРЕС-КЛИЕНТА	67. СТОИМОСТЬ-ЗАКАЗА
19. ПОЧТОВЫЙ-ИНДЕКС-КЛИЕНТА	69. ОТПУСКНАЯ-СТОИМОСТЬ
21. ТЕРРИТОРИАЛЬНЫЙ-КОД	70. ДАТА-ЗАКЛЮЧЕНИЯ-ДОГОВОРА
25. НОМЕР-ЗАКАЗА	71. СРОК-ДЕЙСТВИЯ-ДОГОВОРА
34. ОБЩАЯ-ЗАДОЛЖЕННОСТЬ	
35. ЗАДОЛЖЕННОСТЬ-СВЫШЕ-30-ДНЕЙ	
78. НОМЕР-ПОСТАВЩИКА	
87. ИДЕНТИФИКАТОР-СКЛАДА	

(табл. 6.8). В третьем списке каждый элемент данных должен быть сопоставлен с другими элементами данных и указана частота их совместного пользования. (*Замечание:* Этот список является более удобочитаемой формой матрицы, показанной на рис. 6.7. Если анализ проводится с помощью ЭВМ, этот список может быть сгенерирован автоматически — табл. 6.9.)

После того как эти три списка подготовлены, выполняются следующие процедуры:

1. Вначале необходимо определить все атрибуты, относящиеся к уникальным сущностям. Для этого:

а. Выбирается первый атрибут ОПИСАНИЕ-ТОВАРА (т. е. 14 в табл. 6.8) и просматривается список связей данных (табл. 6.9), чтобы определить, с какими сущностями этот атрибут используется. В данном примере такой единственной сущностью является ТОВАР. Поскольку ОПИСАНИЕ-ТОВАРА не относится ни к какой другой сущности, относим этот атрибут к сущности ТОВАР (табл. 6.10, а).

Таблица 6.9. **Пример списка связей данных**

Исходное данное	С каким данным используется	Частота использования,
14. ОПИСАНИЕ-ТОВАРА		
	20. ВЕС-ТОВАРА	90
	10. ТОВАР	100
	94. ГАБАРИТЫ-ТОВАРА	80
	152. ЦЕНА	65
18. АДРЕС-КЛИЕНТА		
	19. ПОЧТОВЫЙ-ИНДЕКС-КЛИЕНТА	100
	72. ДАТА-ОТГРУЗКИ-КЛИЕНТУ	42
	61. КЛИЕНТ	100
	44. ШТАТ	98
	106. СЧЕТ-ФАКТУРА	8
42. КОЛИЧЕСТВО-В-НАЛИЧИИ		
	67. СТОИМОСТЬ-ЗАКАЗА	30
	43. ЗАКАЗАННОЕ-КОЛИЧЕСТВО	61
	10. ТОВАР	100
	56. ВРЕМЯ-ПОПОЛНЕНИЯ-ЗАПАСА	46
	86. СКЛАД	98
	15. ЗАКАЗ	21
70. ДАТА-ЗАКЛЮЧЕНИЯ-ДОГОВОРА		
	71. СРОК-ДЕЙСТВИЯ-ДОГОВОРА	96
	22. ТОРГОВЫЙ-АГЕНТ	98
	117. УСЛОВИЯ-ДОГОВОРА	92
	61. КЛИЕНТ	100

б. Если из списка связей данных видно, что атрибут относится к двум и более сущностям, необходимо выполнить следующее:

1) Сравнить частоту совместного появления атрибута с каждой из сущностей и исключить из рассмотрения сущности, редко появляющиеся вместе с данным атрибутом, поскольку частота появления атрибута вместе с владеющей им сущностью обычно является очень высокой. Например, частота совместного использования атрибута АДРЕС-КЛИЕНТА с сущностью КЛИЕНТ составляет 100 %, а с сущностью СЧЕТ-ФАКТУРА — 8 %. Поэтому сущность СЧЕТ-ФАКТУРА исключается из рассмотрения, а АДРЕС-КЛИЕНТА назначается сущности КЛИЕНТ.

Таблица 6.10. **Назначение атрибутов**

Атрибуты, относящиеся к сущности 10. ТОВАР:
14. ОПИСАНИЕ-ТОВАРА
а — Начало списка атрибутов, которыми владеет сущность ТОВАР.

Атрибуты, относящиеся к сущности 10. ТОВАР:
14. ОПИСАНИЕ-ТОВАРА
Атрибуты, относящиеся к сущности 61. КЛИЕНТ
18. АДРЕС-КЛИЕНТА
19. ПОЧТОВЫЙ-ИНДЕКС-КЛИЕНТА
34. ОБЩАЯ-ЗАДОЛЖЕННОСТЬ
35. ЗАДОЛЖЕННОСТЬ-СВЫШЕ-30-ДНЕЙ
Атрибуты, относящиеся к сущности 22. ТОРГОВЫЙ-АГЕНТ:
21. ТЕРРИТОРИАЛЬНЫЙ-КОД
б — После назначения всех атрибутов, на которые ссылаются уникальные сущности.

2) Если после исключения из рассмотрения сущностей с низкой частотой совместного использования с рассматриваемым атрибутом вопрос, какая из сущностей владеет атрибутом, остается неясным — необходимо просмотреть все задачи, в которых появляется данный атрибут (в примере с АДРЕСОМ-КЛИЕНТА такой задачей может являться задача «Открыть счет клиента»). Просмотрев все связанные с этой задачей элементы данных, определить, какой из них является сущностью.

В большинстве случаев в каждой задаче имеется только одна уникальная сущность, которая владеет атрибутом. Если это так, необходимо назначить атрибут этой сущности. Если имеется более одной сущности и атрибут появляется с каждой из них с высокой частотой, необходимо попытаться ответить на следующий вопрос: «Какая из сущностей придает атрибуту смысловое значение?» В предыдущем примере исключение сущности КЛИЕНТ делает бессмысленным существование атрибута АДРЕС-КЛИЕНТА, в то время как при исключении сущности СЧЕТ-ФАКТУРА смысл атрибута не теряется. В соответствии с этим атрибут АДРЕС-КЛИЕНТА приписывается сущности КЛИЕНТ, но следует помнить, что это назначение может быть сомнительным и что следует возвратиться к этому вопросу после того, как будут назначены все атрибуты.

2. После того как все атрибуты, на которые ссылаются уникальные сущности, назначены (табл. 6.10, б), можно приступить к рассмотрению атрибутов, относящихся к неуникальным сущностям. Прежде чем начинать попытки определения неуникальных сущностей, к которым должны быть отнесены

атрибуты, необходимо просмотреть список связей сущностей (рис. 6.17) и составить новый список, содержащий все неуникальные сущности и имена относящихся к ним групп атрибутов (табл. 6.11). Затем необходимо выполнить следующее:

Таблица 6.11. **Список неуникальных сущностей и относящихся к ним атрибутов**

Сущности, составляющие неуникальную сущность	Атрибуты, относящиеся к неуникальным сущностям
10. ТОВАР 86. СКЛАД	ТОВАРНЫЙ-ЗАПАС
15. ЗАКАЗ 106. СЧЕТ-ФАКТУРА 10. ТОВАР	ПАРТИЯ-ТОВАРА
110. ЗАКАЗ-НА-ПОСТАВКУ 54. ПОСТАВЛЯЕМЫЙ-ТОВАР	?
22. ТОРГОВЫЙ-АГЕНТ 61. КЛИЕНТ	?

Взять первый атрибут из табл. 6.8 (42. КОЛИЧЕСТВО-В-НАЛИЧИИ) и просмотреть список связей данных (табл. 6.9), чтобы определить, с какими сущностями используется этот атрибут.

Просмотреть список неуникальных сущностей (табл. 6.11), чтобы проверить, является ли комбинация найденных на предыдущем шаге сущностей (10. ТОВАР, 86. СКЛАД и 15. ЗАКАЗ) неуникальной сущностью. Если нет (как в рассматриваемом случае), проверить частоту использования атрибута с каждой из сущностей и исключить сущности с низкой частотой использования (в рассматриваемом примере — ЗАКАЗ).

Повторить просмотр списка неуникальных сущностей. На этот раз найдена неуникальная сущность ТОВАР/СКЛАД, и атрибут КОЛИЧЕСТВО-В-НАЛИЧИИ назначается группе атрибутов ТОВАРНЫЙ-ЗАПАС, которая идентифицируется найденной неуникальной сущностью.

3. Большинство остальных атрибутов может быть определено таким же способом, что и КОЛИЧЕСТВО-В-НАЛИЧИИ. Если при назначении атрибута неуникальной сущности имя группы атрибутов не определено (как, например, для неуникальной сущности ТОРГОВЫЙ-АГЕНТ/КЛИЕНТ, табл. 6.11), необходимо попытаться определить семантику данной группы по смыслу имен атрибутов. Например, не вызывает сомнений, что атрибут ДАТА-ЗАКЛЮЧЕНИЯ-ДОГОВОРА должен быть

назначен неуникальной сущности ТОРГОВЫЙ-АГЕНТ/КЛИЕНТ. Поскольку этот атрибут, а также все те, с которыми он связан, ссылаются на «договор», это наименование может быть выбрано для идентификации группы атрибутов.

В табл. 6.12 показаны все атрибуты, назначенные уникальным или неуникальным сущностям.

Таблица 6.12. **Связи между сущностями и атрибутами**

Атрибуты, относящиеся к сущности 10: ТОВАР:
14. ОПИСАНИЕ-ТОВАРА
Атрибуты, относящиеся к сущности 22. ТОРГОВЫЙ-АГЕНТ:
21. ТЕРРИТОРИАЛЬНЫЙ-КОД
Атрибуты, относящиеся к сущности ПАРТИЯ-ТОВАРА:
(15. ЗАКАЗ, 106. СЧЕТ-ФАКТУРА, 10. ТОВАР)
67. СТОИМОСТЬ-ЗАКАЗА
79. ОТПУСКНАЯ-СТОИМОСТЬ
Атрибуты, относящиеся к сущности 61. КЛИЕНТ:
18. АДРЕС-КЛИЕНТА
34. ОБЩАЯ-ЗАДОЛЖЕННОСТЬ
35. ЗАДОЛЖЕННОСТЬ-СВЫШЕ-30-ДНЕЙ
19. ПОЧТОВЫЙ-ИНДЕКС-КЛИЕНТА
Атрибуты, относящиеся к сущности ТОВАРНЫЙ-ЗАПАС:
(10. ТОВАР, 86. СКЛАД)
42. КОЛИЧЕСТВО-В-НАЛИЧИИ
56. ВРЕМЯ-ПОПОЛНЕНИЯ-ЗАПАСА
Атрибуты, относящиеся к сущности ТОРГОВЫЙ-ДОГОВОР:
(22. ТОРГОВЫЙ-АГЕНТ, 61. КЛИЕНТ)
70. ДАТА-ЗАКЛЮЧЕНИЯ-ДОГОВОРА
71. СРОК-ДЕЙСТВИЯ-ДОГОВОРА

Последний шаг второй фазы анализа (Шаг 2-2-3) включает идентификацию связей между атрибутами. Несмотря на то что идентификация связей атрибутов является частью процесса логического проектирования, она играет второстепенную роль, так как эти связи не влияют на основную информационную структуру базы данных. Их важность возрастает при построении физической хранимой структуры. Поэтому процесс идентификации связей атрибутов не слишком формализован и в значительной степени основан на общих рассуждениях (здравом смысле).

Для идентификации и документирования связей между атрибутами используется следующая процедура. (*Замечание:* Рассматриваемые связи существуют только между теми атрибутами, которые принадлежат одной из уникальных или неуникальных сущностей. Следовательно, данная процедура должна быть выполнена для каждой из связей «сущность-атрибут», показанных в табл. 6.12.) В качестве примера рассмотрим атрибуты, относящиеся к сущности КЛИЕНТ.

1. Идентифицировать все задачи, связанные с четырьмя атрибутами сущности КЛИЕНТ (табл. 6.13).

Таблица 6.13. **Задачи, связанные с атрибутами сущности КЛИЕНТ**

Задачи, связанные с атрибутами АДРЕС-КЛИЕНТА и ПОЧТОВЫЙ-ИНДЕКС-КЛИЕНТА
Зарегистрировать нового клиента
18. АДРЕС-КЛИЕНТА
61. КЛИЕНТ
19. ПОЧТОВЫЙ-ИНДЕКС-КЛИЕНТА
Задачи, связанные с атрибутом ОБЩАЯ-ЗАДОЛЖЕННОСТЬ
Сделать проводку по счету клиента
61. КЛИЕНТ
34. ОБЩАЯ-ЗАДОЛЖЕННОСТЬ
Задачи, связанные с атрибутом ЗАДОЛЖЕННОСТЬ-СВЫШЕ-30-ДНЕЙ
Разнести по срокам задолженность клиента
61. КЛИЕНТ
34. ОБЩАЯ-ЗАДОЛЖЕННОСТЬ
35. ЗАДОЛЖЕННОСТЬ-СВЫШЕ-30-ДНЕЙ

2. Просмотреть каждую задачу с целью идентификации связей атрибутов (в табл. 6.14 показано, каким образом документируется каждая связь).

Таблица 6.14. **Форма документирования связей атрибутов**

НОМЕР-КЛИЕНТА	
Сильно связанные атрибуты	Атрибуты с зависимостями
18. АДРЕС-КЛИЕНТА	34. ОБЩАЯ-ЗАДОЛЖЕННОСТЬ
19. ПОЧТОВЫЙ-ИНДЕКС-КЛИЕНТА	35. ЗАДОЛЖЕННОСТЬ-СВЫШЕ-30-ДНЕЙ

• Первая задача «Зарегистрировать нового клиента» показывает, что существует сильная связь между атрибутами 18. АДРЕС-КЛИЕНТА и 19. ПОЧТОВЫЙ-ИНДЕКС-КЛИЕНТА.

• Вторая задача «Сделать проводку по счету клиента» не выявляет новых связей.

• Третья задача «Разнести по срокам задолженность клиента» показывает существование определенной связи и зависи-

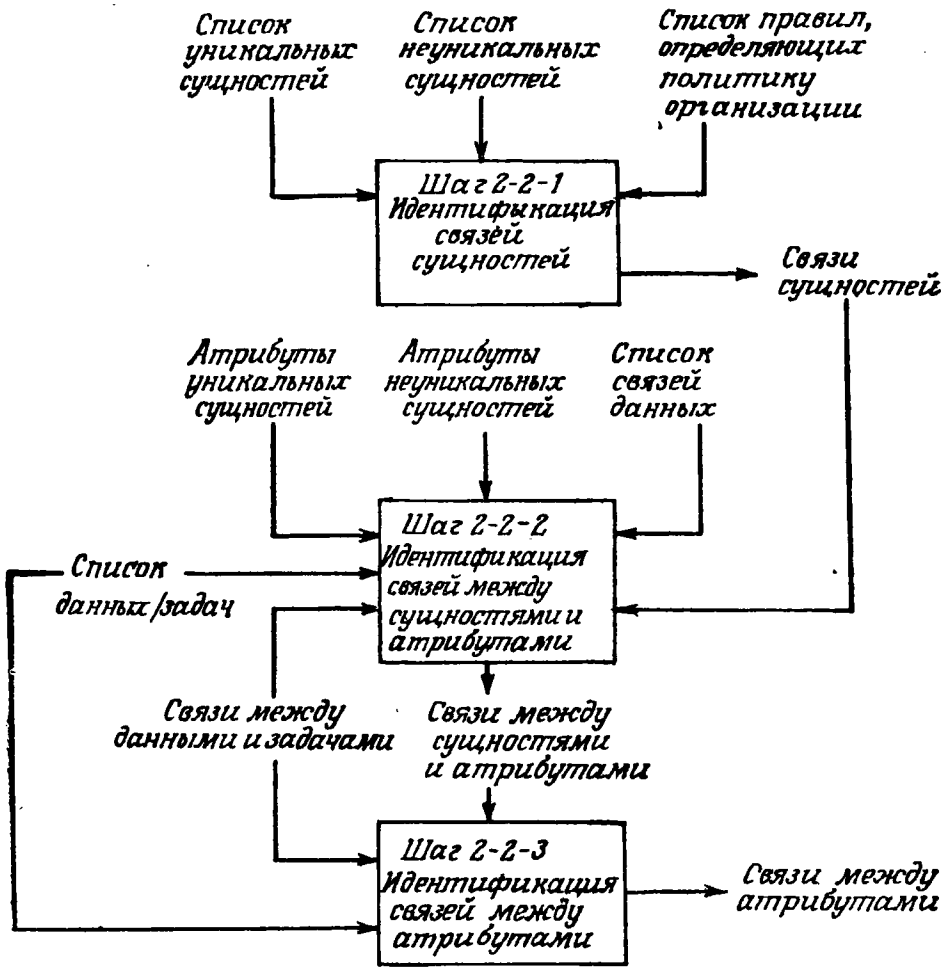


Рис. 6.18. Блок-схема второй фазы концептуального проектирования (Шаг. 2-2) — идентификации связей данных.

мости (в соответствии со здравым смыслом) между атрибутами ЗАДОЛЖЕННОСТЬ-СВЫШЕ-30-ДНЕЙ и ОБЩАЯ-ЗАДОЛЖЕННОСТЬ.

Этим завершается вторая фаза анализа. На рис. 6.18 показана блок-схема шагов, выполняемых во время этой фазы.

6.3. Изображение сущностей, атрибутов и связей в графических обозначениях информационной структуры типа «СУЩНОСТЬ-СВЯЗЬ» (Шаг 2-3)

Целью данной фазы проектирования информационной структуры является изображение идентифицированных на фазе анализа сущностей, атрибутов и связей в терминах графических изображений модели «сущность-связь». В процессе построения графической структуры данных можно выделить три основных шага:

• Представление уникальных и неуникальных сущностей (Шаг 2-3-1).

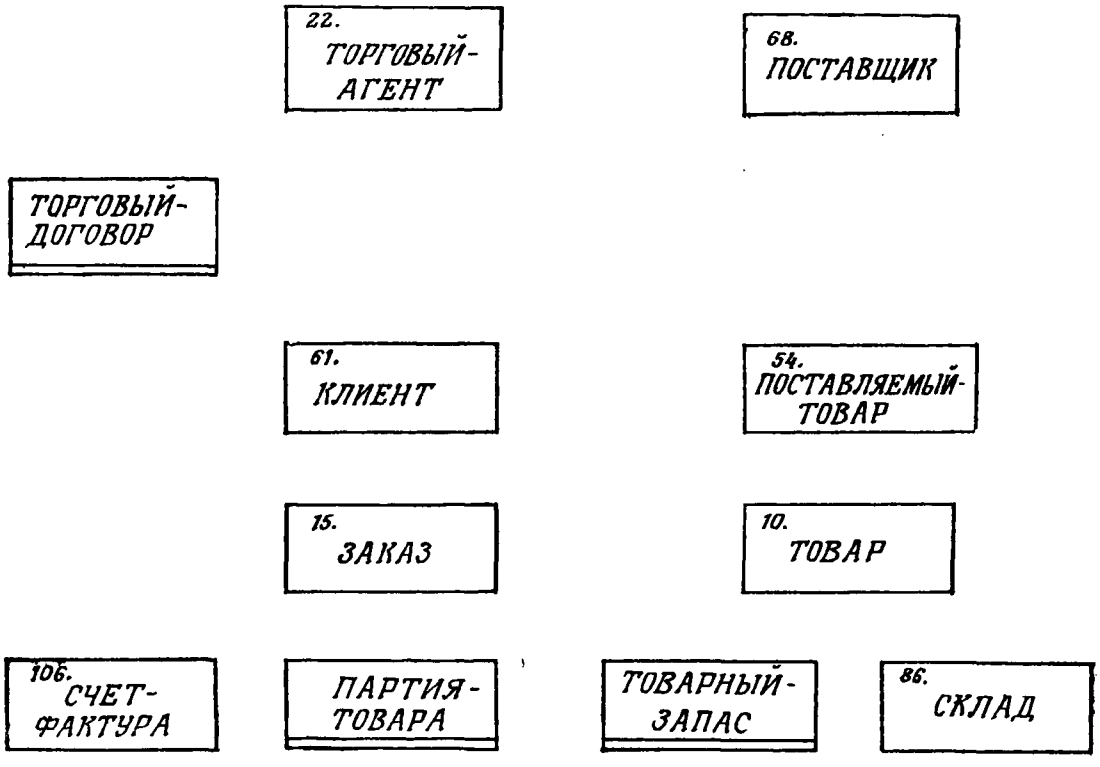


Рис. 6.19. Графическое представление уникальных и неуникальных сущностей.

• Представление связей между сущностями (Шаг 2-3-2).

• Представление связей между атрибутами и между сущностями и атрибутами (Шаг 2-3-3).

Как было указано в разд. 6.1, сущности изображаются прямоугольниками, причем уникальные сущности — с одной нижней чертой, неуникальные — с двумя. На рис. 6.19 показаны некоторые из сущностей, определенных в табл. 6.6.

После того как все сущности изображены графически, необходимо показать на диаграмме их связи друг с другом. Для этого используется построенная во время анализа схема связей (рис. 6.17):

• Строка 1, рис. 6.17, показывает, что сущность ТОРГОВЫЙ-АГЕНТ владеет сущностью КЛИЕНТ. Связь на диаграмме обозначается стрелкой от сущности ТОРГОВЫЙ-АГЕНТ к сущности КЛИЕНТ (рис. 6.20, *а*).

• Строка 2 идентифицирует неуникальную сущность ТОВАРНЫЙ-ЗАПАС и идентифицирующие ее сущности ТОВАР и СКЛАД. Эта связь должна быть обозначена одной стрелкой от сущности ТОВАР к сущности ТОВАРНЫЙ-ЗАПАС и другой стрелкой от сущности СКЛАД к сущности ТОВАРНЫЙ-ЗАПАС (рис. 6.20, *б*).

• Строка 3 идентифицирует две связи: одну между сущностями ТОРГОВЫЙ-АГЕНТ и КЛИЕНТ, другую между сущно-

стями КЛИЕНТ и ЗАКАЗ. Обе эти связи показаны на рис. 6.20, *в*.

• Строка 4 определяет связь между сущностями ПОСТАВЩИК и ПОСТАВЛЯЕМЫЙ-ТОВАР, показанную на рис. 6.20, *г*.

• Строка 5 определяет связь между сущностями ТОВАР и ПОСТАВЛЯЕМЫЙ-ТОВАР, показанную на рис. 6.20, *д*.

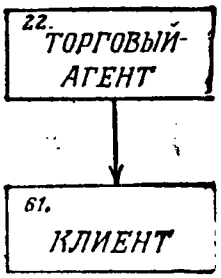
• Строка 6 определяет связь между сущностями СКЛАД и неуникальной сущностью ПАРТИЯ-ТОВАРА. Кроме того, строка 6 указывает, что эта связь должна быть снабжена комментарием «комплектуется из» и что она является условной. Определена также связь между сущностями ЗАКАЗ и ПАРТИЯ-ТОВАРА. Обе связи показаны на рис. 6.20, *е*.

• Строка 7 определяет неуникальную сущность ПАРТИЯ-ТОВАРА через две условные связи и одну обычную. Обычная связь определена между сущностями ТОВАР и ПАРТИЯ-ТОВАРА. Две оставшиеся связи зависят от статуса сущности ПАРТИЯ-ТОВАРА, который определяется как «отгружена»/«не отгружена». Поскольку ПАРТИЯ-ТОВАРА не может быть одновременно и «отгружена», и «не отгружена», связь сущности ПАРТИЯ-ТОВАРА должна принадлежать или сущности ЗАКАЗ, или СЧЕТ-ФАКТУРА, но не обоим одновременно. Эти связи показаны на рис. 6.20, *ж*.

К тому времени, когда были определены связи сущностей, было неясно, существует ли неуникальная сущность ТОРГОВЫЙ-АГЕНТ/КЛИЕНТ (строка 8, рис. 6.17), поскольку не была идентифицирована группа атрибутов, определяемая этой сущностью. Чтобы определить, должна ли эта связь быть включенной в модель структуры данных, необходимо просмотреть список связей между сущностями и атрибутами (табл. 6.12). Из него видно, что определена группа атрибутов ТОРГОВЫЙ-ДОГОВОР, которая подразумевает существование связи между сущностями ТОРГОВЫЙ-АГЕНТ и ТОРГОВЫЙ-ДОГОВОР. Эта связь показана на рис. 6.20, *з*.

После изображения с помощью соответствующих графических обозначений отдельных связей сущностей приступают к следующему шагу, который включает построение общей информационной структуры путем объединения отдельных связей. Полученная в результате структура показана на рис. 6.21.

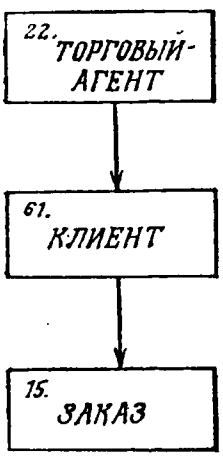
После того как построена основная информационная структура, приступают к изображению в графической форме связей между атрибутами и между сущностями и атрибутами. В отличие от связей между сущностями связи между атрибутами и между сущностями и атрибутами изображаются с помощью иерархических диаграмм. Для того, чтобы определить, как группируются между собой атрибуты (например, сильно связанные атрибуты АДРЕС-КЛИЕНТА и ПОЧТОВЫЙ-


22.
ТОРГОВЫЙ-
АГЕНТ
61.
КЛИЕНТ

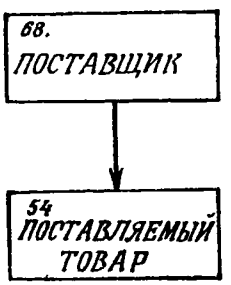
а


10.
ТОВАР
ТОВАРНЫЙ-
ЗАПАС
86.
СКЛАД

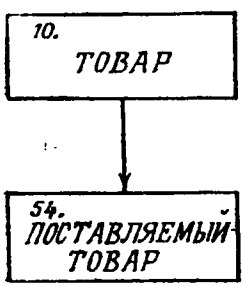
б


22.
ТОРГОВЫЙ-
АГЕНТ
61.
КЛИЕНТ
15.
ЗАКАЗ

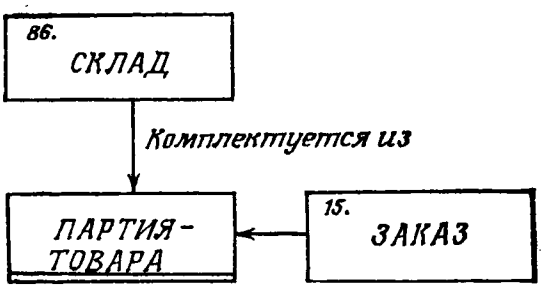
в


68.
ПОСТАВЩИК
54
ПОСТАВЛЯЕМЫЙ
ТОВАР

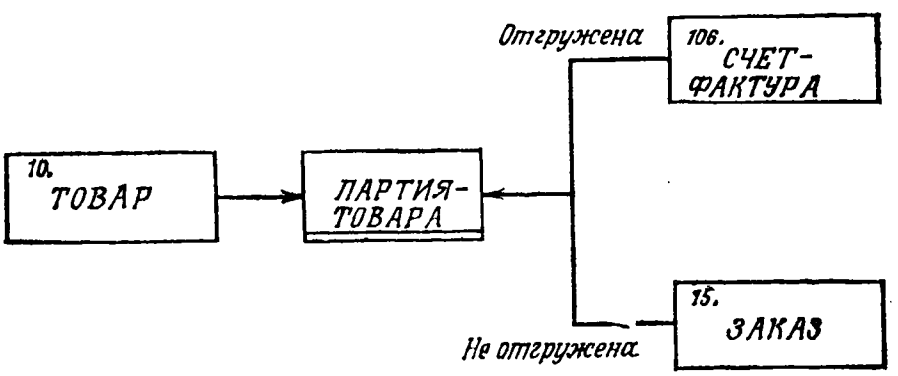
г


10.
ТОВАР
54.
ПОСТАВЛЯЕМЫЙ
ТОВАР

д


86.
СКЛАД
Комплектуется из
ПАРТИЯ-
ТОВАРА
15.
ЗАКАЗ

е


Отгружена
106.
СЧЕТ-
ФАКТУРА
10.
ТОВАР
ПАРТИЯ-
ТОВАРА
15.
ЗАКАЗ
Не отгружена

ж


22.
ТОРГОВЫЙ-
АГЕНТ
ТОРГОВЫЙ-
ДОГОВОР
61.
КЛИЕНТ

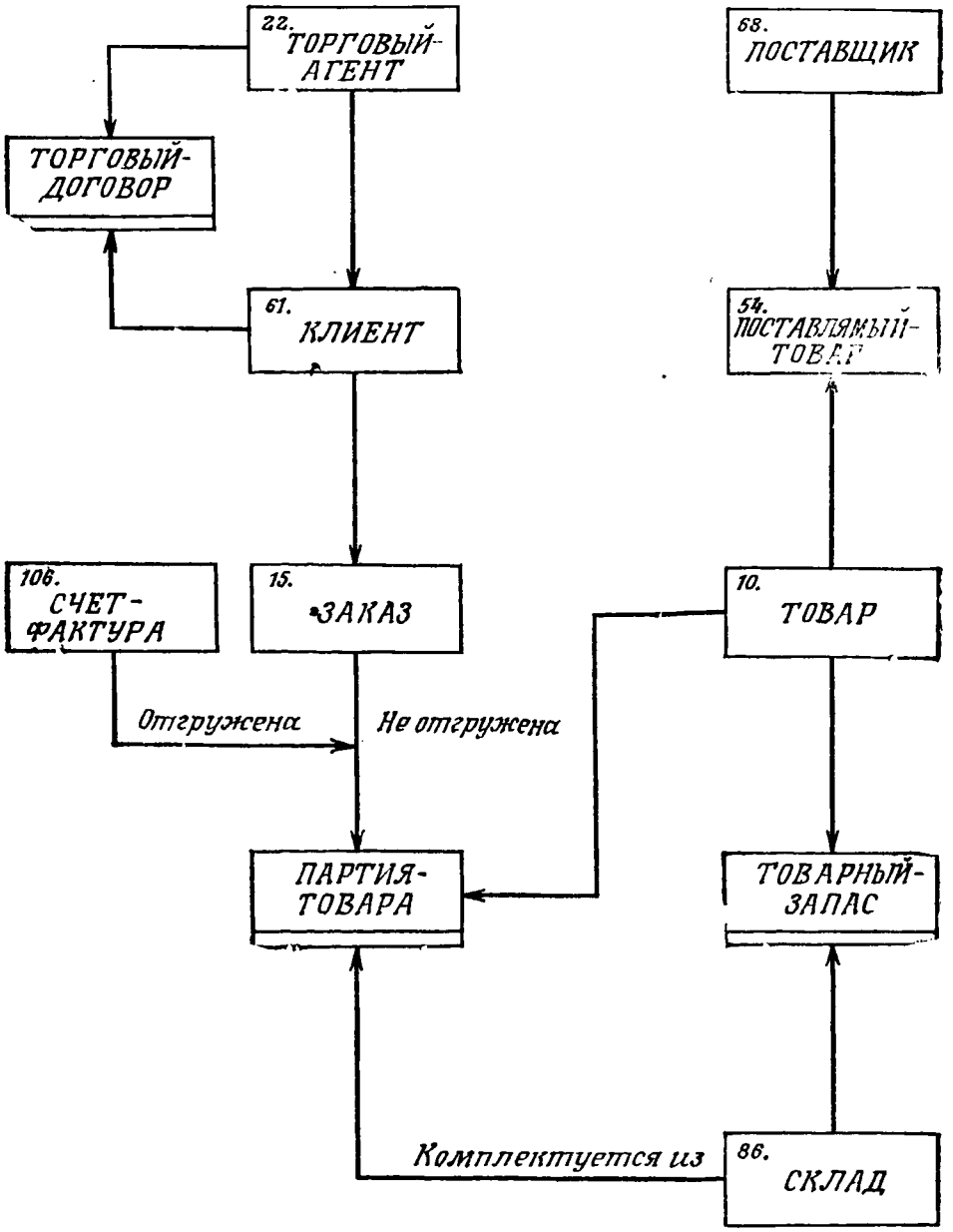


Рис. 6.21. Результирующая информационная структура, показывающая все связи сущностей.

ИНДЕКС-КЛИЕНТА на рис. 6.20, *з* должны быть сгруппированы под именем АДРЕС) или как они зависят один от другого (например, ЗАДОЛЖЕННОСТЬ-СВЫШЕ-30-ДНЕЙ зависит от

Рис. 6.20. Примеры связей данных для рассматриваемой базы данных.

а — связь между сущностями ТОРГОВЫЙ-АГЕНТ и КЛИЕНТ; *б* — связи между компонентами неуникальной сущности ТОВАРНЫЙ-ЗАПАС; *в* — связи между сущностями ТОРГОВЫЙ-АГЕНТ, КЛИЕНТ и ЗАКАЗ; *г* — связь между сущностями ПОСТАВЩИК и ПОСТАВЛЯЕМЫЙ-ТОВАР; *д* — связь между сущностями ТОВАР и ПОСТАВЛЯЕМЫЙ-ТОВАР; *е* — условные связи между сущностью СКЛАД и неуникальной сущностью ПАРТИЯ-ТОВАРА; *ж* — условная связь типа «или-или» между сущностями СЧЕТ-ФАКТУРА, ЗАКАЗ и ПАРТИЯ-ТОВАРА и обычная связь между сущностями ТОВАР и ПАРТИЯ-ТОВАРА; *з* — связи между сущностями ТОРГОВЫЙ-АГЕНТ и ТОРГОВЫЙ-ДОГОВОР и сущностями КЛИЕНТ и ТОРГОВЫЙ-ДОГОВОР.

атрибута ОБЩАЯ-ЗАДОЛЖЕННОСТЬ), необходимо использовать построенный на шаге анализа список связей атрибутов (табл. 6.14). Иерархическая диаграмма на рис. 6.22 изображает связи между сущностью КЛИЕНТ и всеми ее атрибутами. Такие диаграммы должны быть построены для всех уникальных и неуникальных сущностей.

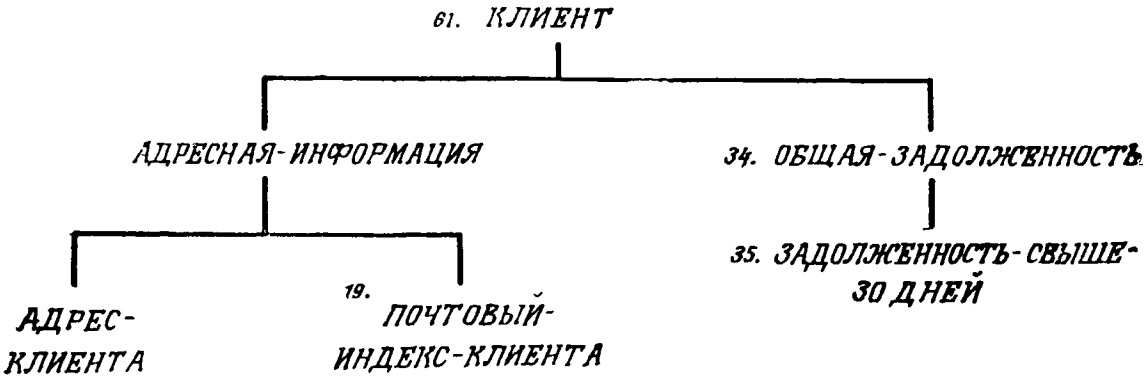


Рис. 6.22. Иерархическая диаграмма связей между атрибутами, сущностями и атрибутами.

Основная информационная структура, состоящая из уникальных и неуникальных сущностей и связей между ними, и иерархические диаграммы связей между атрибутами и между атрибутами и сущностями вместе составляют полную информационную структуру.

6.4. Интерпретация информационной структуры для ее верификации всеми пользователями (Шаг 2-4)

Процесс построения информационной структуры обязательно включает обработку и интерпретацию большого количества информации о том, как создаются и/или используются данные различными подразделениями организации. Причина такой структуризации методологии проектирования заключается в том, что чрезвычайно трудно идентифицировать и распознавать все связи и условия их существования. Даже при такой степени структуризации весьма вероятно, что некоторые связи будут упущены или неточно определены. Кроме того, информационная структура является единственным средством, с помощью которого можно в явном виде определить, как различные подразделения организации используют свои данные и управляют ими. Резонно предположить, что руководство организации пожелает удостовериться в правильности используемых данных и, возможно, предложит некоторые изменения. Поэтому необходимо предоставить руководству возможность для проверки правильности структуры связей данных и для уяснения того, как эти связи влияют на определенные аспекты деятельности

организации. Для этого каждая связь независимо от того, включена ли она в информационную структуру или нет, должна быть идентифицирована и выражена очень простыми и четкими предложениями в форме утверждений, которые могут быть рассмотрены и оценены руководством. После такого рассмотрения проект, если необходимо, может быть скорректирован с целью учета мнения руководства.

Для интерпретации информационной структуры необходимо выполнить следующие шаги (все примеры используют рис. 6.21):

Шаг 2-4-1 устанавливает зависимость каждой сущности. Например:

- Каждый КЛИЕНТ должен быть назначен ТОРГОВОМУ-АГЕНТУ, прежде чем будет открыт счет.
- Все ЗАКАЗЫ принимаются только от тех КЛИЕНТОВ, у которых открыт счет.
- Каждый ТОВАРНЫЙ-ЗАПАС должен находиться на определенном СКЛАДЕ и включать только те ТОВАРЫ, которые проходят по бухгалтерским отчетам компании.

Шаг 2-4-2 определяет направление каждой стрелки (следует отметить, что каждая стрелка обозначает связь «один-комногим»). Например:

- Каждый ТОРГОВЫЙ-АГЕНТ может обслуживать неограниченное число КЛИЕНТОВ.
- Каждый КЛИЕНТ может обслуживаться только одним ТОРГОВЫМ-АГЕНТОМ.
- Каждый КЛИЕНТ может иметь неограниченное число ЗАКАЗОВ.
- Каждый ЗАКАЗ поступает только от одного КЛИЕНТА.
- Каждый СКЛАД может иметь ТОВАРНЫЙ-ЗАПАС, который содержит все ТОВАРЫ или часть ТОВАРОВ, продаваемых компанией.
- Каждый ТОВАР может содержаться в ТОВАРНЫХ-ЗАПАСАХ всех или некоторых СКЛАДОВ.

Шаг 2-4-3 определяет, какие данные не могут существовать, если из базы данных исключается экземпляр сущности. Например:

- Если заменяется ТОРГОВЫЙ-АГЕНТ, все КЛИЕНТЫ должны быть немедленно назначены новому ТОРГОВОМУ-АГЕНТУ, а все ТОРГОВЫЕ-ДОГОВОРЫ заключены заново.
- Если СЧЕТ-КЛИЕНТА исключен или заморожен, все текущие ЗАКАЗЫ этого КЛИЕНТА снимаются, а ТОРГОВЫЙ-ДОГОВОР аннулируется.
- Если ТОВАР упразднен или исключен каким-либо другим способом, он должен быть исключен из всех ТОВАРНЫХ-ЗАПАСОВ всех СКЛАДОВ.

• Если СКЛАД продан или ликвидирован, никакой ТОВАРНЫЙ-ЗАПАС не может размещаться в указанном месте.

Шаг 2-4-4 определяет, от чего не зависит каждая сущность (следует заметить, что в данном случае не существует никаких связывающих стрелок). Например:

• ТОРГОВЫЙ-АГЕНТ не имеет непосредственных связей с ПОСТАВЩИКАМИ, СЧЕТ-ФАКТУРАМИ, ТОВАРАМИ или СКЛАДАМИ.

• КЛИЕНТЫ не имеют непосредственных связей со СЧЕТ-ФАКТУРАМИ, ПОСТАВЩИКАМИ, ТОВАРАМИ или СКЛАДАМИ.

• ТОВАРНЫЕ-ЗАПАСЫ не имеют непосредственных связей с ПОСТАВЩИКАМИ, СЧЕТ-ФАКТУРАМИ, ЗАКАЗАМИ, КЛИЕНТАМИ или ТОРГОВЫМИ-АГЕНТАМИ.

Эти же четыре шага должны выполняться при интерпретации иерархических диаграмм каждой сущности. Все утверждения, полученные при этом, должны быть добавлены к предыдущим спискам.

До того как преобразованная таким образом информационная структура будет представлена руководству, необходимо проверить влияние возможных будущих изменений (ранее выявленных в собеседованиях с руководством) и отметить области возникновения потенциальных проблем. Пусть в результате собеседований с руководством выявлены следующие возможные изменения:

• Возможность совместного использования несколькими КЛИЕНТАМИ одного ЗАКАЗА с целью экономии на перевозках.

• Прикрепление ТОРГОВЫХ-АГЕНТОВ к конкретному СКЛАДУ, которое предполагает, что ТОРГОВЫЙ-АГЕНТ не сможет предлагать ТОВАРЫ, находящиеся на других СКЛАДАХ.

• Разрешение КЛИЕНТАМ иметь открытые заказы на срок от 6 до 12 месяцев.

Каждое из этих утверждений о возможных изменениях должно быть проанализировано следующим образом:

1. Определить, не содержится ли возможное изменение в существующей информационной структуре.
2. Определить и представить в виде диаграммы изменения, которые могут иметь место при реализации новых требований.
3. Определить с помощью четких утверждений последствия этих изменений.

Далее описано, как выполняются эти шаги для каждого из трех приведенных выше возможных в будущем изменений.

Изменение 1. Более одного клиента на заказ

Первое потенциальное изменение означает, что несколько клиентов могут владеть заказом. Поскольку существующая информационная структура прямо указывает, что каждый ЗАКАЗ должен принадлежать только одному КЛИЕНТУ (т. е. стрелка на рис. 6.21 выходит из сущности КЛИЕНТ и входит в сущность ЗАКАЗ), в рамках данного проектного решения это изменение не может быть реализовано.

Чтобы ввести это изменение, необходимо разорвать связь между сущностями ЗАКАЗ и КЛИЕНТ и связать конкретные ЗАКАЗЫ и КЛИЕНТОВ с конкретными ПАРТИЯМИ-ТОВАРА (например, ЗАКАЗ номер 1 включает ПАРТИИ-ТОВАРА от первой до двадцатой, в то же время КЛИЕНТУ А принадлежат ПАРТИИ-ТОВАРА от первой до десятой, а КЛИЕНТУ Б — от одиннадцатой до двадцатой). Структура, учитывающая это изменение, показана на рис. 6.23.

Рис. 6.23. Информационная структура, отображающая первое возможное изменение.

Последствия этого изменения могут быть сформулированы следующим образом:

- Атрибуты, относящиеся к сущности ЗАКАЗ, описывают такие понятия, как дату отгрузки и адрес доставки. Поскольку ЗАКАЗ больше не относится к конкретному КЛИЕНТУ, эти атрибуты должны быть отнесены к каждой ПАРТИИ-ТОВАРА.
- Поскольку между сущностями ЗАКАЗ и КЛИЕНТ нет прямой связи, в системе может появиться недействительный ЗАКАЗ (например, такой, для которого не существует ни одного КЛИЕНТА).
- СЧЕТ-ФАКТУРЫ, которые ранее неявно соотносились с КЛИЕНТОМ через связь с ПАРТИЕЙ-ТОВАРА (которой владели ЗАКАЗ и КЛИЕНТ), теперь должны быть непосредственно связаны как с КЛИЕНТОМ, так и с ПАРТИЕЙ-ТОВАРА (т. е. от КЛИЕНТА должна быть направлена стрелка к СЧЕТ-ФАКТУРЕ).

Изменение 2. Прикрепление торговых агентов к складу

Это изменение не позволяет ТОРГОВОМУ-АГЕНТУ, обслуживающему конкретный склад, принимать заказ на товары, которых нет на данном складе. Поскольку существующая информационная структура не содержит прямой связи между сущностями ТОРГОВЫЙ-АГЕНТ и СКЛАД и, кроме того, позволяет каждую ПАРТИЮ-ТОВАРА «комплектовать из» лю-

бого СКЛАДА, это изменение трудно реализовать.

Для того чтобы ТОРГОВЫЙ-АГЕНТ продавал товары только из назначенного СКЛАДА, необходимо ввести несколько изменений в структуру данных. Во-первых, необходимо ввести связь между сущностями СКЛАД и ТОРГОВЫЙ-АГЕНТ. Добавление этой связи будет означать, что поскольку ТОРГОВЫЙ-АГЕНТ принадлежит конкретному СКЛАДУ, все его КЛИЕНТЫ и их ЗАКАЗЫ также будут принадлежать этому СКЛАДУ. Однако это не означает, что КЛИЕНТЫ смогут заказывать товары только из одного СКЛАДА (имеется в виду

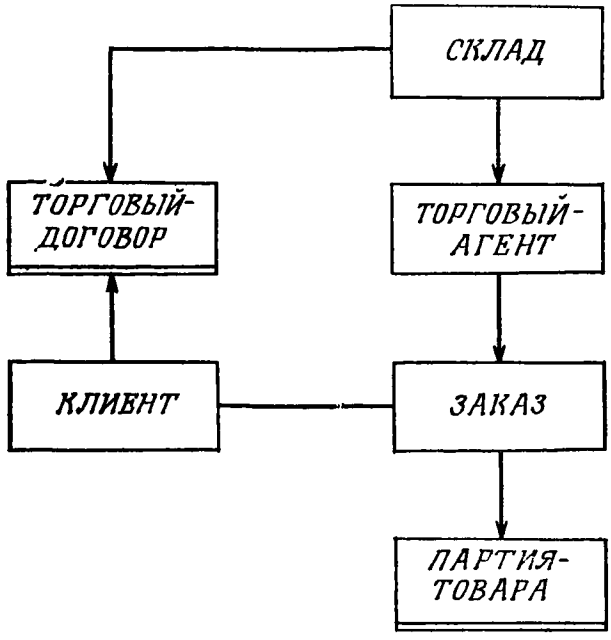


Рис. 6.24. Информационная структура, отображающая второе возможное изменение.

то, что КЛИЕНТ может вести дела с несколькими ТОРГОВЫМИ-АГЕНТАМИ, прикрепленными к разным СКЛАДАМ), поэтому связь между ТОРГОВЫМ-АГЕНТОМ и КЛИЕНТОМ должна быть ликвидирована, а ТОРГОВЫЙ-АГЕНТ должен быть теперь непосредственно связан с ЗАКАЗОМ. Кроме того, связь между ТОРГОВЫМ-АГЕНТОМ и ТОРГОВЫМ-ДОГОВОРОМ должна быть заменена связью между СКЛАДОМ и ТОРГОВЫМ-ДОГОВОРОМ. Это позволит КЛИЕНТУ иметь различные ТОРГОВЫЕ-ДОГОВОРЫ с каждым СКЛАДОМ. Последнее изменение заключается в удалении связи «комплектуется из» между СКЛАДОМ и ПАРТИЕЙ-ТОВАРА. Скорректированная структура данных показана на рис. 6.24.

Последствия второго изменения заключаются в следующем:

• КЛИЕНТ больше не связан с каким-либо одним ТОРГОВЫМ-АГЕНТОМ или СКЛАДОМ. Он может делать закупки из любого СКЛАДА, но при этом должен поддерживать связь с каждым складом через ТОРГОВЫХ-АГЕНТОВ, обслуживающих этот СКЛАД.

• КЛИЕНТ может заключать отдельные ТОРГОВЫЕ-ДОГОВОРЫ с любым СКЛАДОМ, предполагая при этом, что цены за одинаковые ЗАКАЗЫ будут назначаться в каждом случае по-разному.

• Каждый СКЛАД может иметь не менее одного ТОРГОВОГО-АГЕНТА, но каждый ТОРГОВЫЙ-АГЕНТ обслуживает только один СКЛАД.

• ТОРГОВЫЙ-АГЕНТ не несет ответственности ни за одного КЛИЕНТА, но несет ответственность за ЗАКАЗЫ, которые он принимает.

• Все ЗАКАЗЫ, принятые ТОРГОВЫМ-АГЕНТОМ, должны комплектоваться только из СКЛАДА, к которому он прикреплен.

Изменение 3. Включение открытых заказов

Третье изменение описывает промежуток времени, в течение которого клиент может иметь открытый ЗАКАЗ. Так как это изменение не вводит новых связей, а существующая структура позволяет выписывать СЧЕТ-ФАКТУРУ на ПАРТИЮ-ТОВАРА после того, как она «отгружена», можно легко реализовать включение сущности ОТКРЫТЫЙ-ЗАКАЗ в существующую структуру.

После того как информационная структура представлена в виде набора утверждений и определены последствия всех возможных изменений, необходимо подготовить отчет для руководства. Чтобы не перегружать отчет лишним материалом, рекомендуется представить в нем описание сущностей и отразить все спорные вопросы и последствия возможных изменений (табл. 6.15). После того как каждый руководитель просмотрит отсчет и сделает замечания (для которых необходимо оставить место в отчете), следует собрать и обработать эти замечания следующим образом:

1. Если большинство руководителей возражает против какого-либо утверждения и выдвигает взамен новое согласованное утверждение, необходимо отразить его в информационной структуре. Например, если все руководители несогласны с утверждением 7 в табл. 6.15 и предлагают новое утверждение, заключающееся в том, что «счет-фактура должен непосредственно относиться к одному клиенту», следует добавить стрелку от КЛИЕНТА к СЧЕТ-ФАКТУРЕ.

2. Если существует расхождение во мнениях, необходимо доложить о различных точках зрения и их последствиях высшему руководству для принятия решения. (*Замечание:* Не следует надеяться на немедленное решение проблемы, так как руководству не часто приходится иметь дело с такими вопро-

Таблица 6.15. **Пример отчета, представляемого руководству для утверждения**

ИНФОРМАЦИЯ О КЛИЕНТАХ

Изложение правил использования информации о клиентах в повседневной деятельности

1. Каждый клиент должен быть прикреплен к торговому агенту, прежде чем будет открыт счет.
   ЗАМЕЧАНИЯ: _____

2. Все заказы принимаются только от тех клиентов, на которых открыт счет.
   ЗАМЕЧАНИЯ: _____

3. Каждый клиент может обслуживаться только одним торговым агентом.
   ЗАМЕЧАНИЯ: _____

4. Каждый клиент может иметь неограниченное число заказов.
   ЗАМЕЧАНИЯ: _____

5. Если заменяется торговый агент, все его клиенты должны быть прикреплены к новому торговому агенту, а все торговые договоры заключены заиово.
   ЗАМЕЧАНИЯ: _____

6. Если клиент исключается или оказывается в пассиве, все его текущие заказы снимаются, а торговый договор аннулируется.
   ЗАМЕЧАНИЯ: _____

7. Клиент не имеет непосредственных связей со счет-фактурами, поставщиками, номерами товаров и складами.
   ЗАМЕЧАНИЯ: _____

8. В каждом заказе могут участвовать несколько клиентов.
   СОГЛАСНЫ _____
   НЕСОГЛАСНЫ _____

9. Товары клиенту продает склад, а не торговый агент. В соответствии с этим каждый клиент может заключить торговый договор с любым складом.
   СОГЛАСНЫ _____
   НЕСОГЛАСНЫ _____

сами.) После того как решение принято, отразить его в информационной структуре.

3. Решения относительно будущих изменений, связанных с перестройкой информационной структуры, должны приниматься на самом высшем уровне. Процедура принятия решения аналогична описанной выше.

После того как приняты все решения, а все утверждения и диаграмма согласованы между собой, построение информационной структуры, являющейся итогом концептуального проектирования, считается завершенным. Прежде чем передать результаты лицу, ответственному за построение логической структуры базы данных, полезно ознакомить группу руководителей, просматривавших отчет, с окончательной структурой в виде набора утверждений. С помощью этой процедуры каждый руководитель получает набор правил, в рамках которых должно функционировать его подразделение.

Часть III

Проектирование реализации

Глава 7. Концепции проектирования реализации

Из-за отсутствия установившейся терминологии точную границу между концептуальным и физическим проектированием баз данных провести достаточно трудно. Тем не менее принято считать, что на этапе концептуального проектирования данные рассматриваются без учета специфики используемой СУБД, а особенности физического хранения базы данных в памяти ЭВМ включаются в описание ее структуры на этапе физического проектирования. Этап между концептуальным и физическим проектированием, в результате выполнения которого получается СУБД-ориентированная схема базы данных, будем называть проектированием реализации. Изменения, которые вносятся в структуру базы данных на этом этапе, определяются стремлением удовлетворить требованиям конкретной СУБД и наиболее общим ограничениям, специфицированном в требованиях пользователей.

Основной задачей проектирования реализации является разработка СУБД-ориентированной схемы, которая удовлетворяет всему диапазону требований пользователей, начиная с требований целостности и непротиворечивости проектируемой базы данных и кончая показателями эффективности функционирования при ее расширении и усложнении. Так как одновременно с проектированием базы данных проводится работа по составлению прикладных программ, то между разработчиками этих подсистем должно быть налажено постоянное взаимодействие. При этом проводится анализ программных спецификаций высокого уровня и разрабатывается руководство по проектированию программ с учетом предложенной структуры базы данных.

В данной главе исследуются основные вопросы проектирования реализации, описывается процесс проектирования схем и подсхем, проводится детальный анализ эффективности проектирования схемы. В гл. 8 эти вопросы проиллюстрированы примером.

7.1. Содержание процесса проектирования реализации

На рис. 7.1 дана диаграмма, отражающая состав входных и выходных данных этапа проектирования реализации [206]:

Исходные данные

1. *СУБД-независимая схема.* Основной результат фазы концептуального проектирования; исходная схема, которая в дальнейшем будет преобразовываться на фазе проектирования реализации.

2. *Количественная оценка эксплуатационных характеристик.* Спецификации требований целостности, восстанавливаемости, безопасности, ограничений на времена отклика, а также прогноз роста объема и изменений структуры базы данных.

3. *Количественная оценка объема и частоты выполнения приложений.* Размер базы данных, который оценивается исходя из количества экземпляров данных и частоты выполнения приложений.

4. *Требования непротиворечивости.* Правила поддержания взаимной непротиворечивости элементов данных, правила устранения противоречивости данных, а также ограничения на дублирование и обновление данных.

5. *Программные спецификации высокого уровня.* Результаты анализа требований к программам.

6. *Характеристики СУБД.* Правила задания СУБД-ориентированных логических схем и подсхем, а также синтаксиса программ.

7. *Вычислительные средства.* Ограничения на конфигурацию и объем аппаратного и математического обеспечения.

Результаты

1. *СУБД-ориентированная схема.* Спецификация структуры базы данных, которая может быть реализована конкретной СУБД, не содержит (или использует по умолчанию) большинства физических параметров, определяющих группирование записей или размеры блоков. Однако она может включать некоторые параметры путей доступа, такие, как упорядоченность, указатели и механизмы поиска.

2. *Подсхемы.* Структура СУБД-ориентированной базы данных, совместимая с представлениями отдельных пользователей и ограничениями безопасности.

3. *Спецификации для физического проектирования.* Полностью документированные схемы и подсхемы с указанием объема, частоты выполнения приложений и характеристик аппаратного и программного обеспечения, необходимые для этапа физического проектирования.

Рис. 7.1. Исходные данные и результаты этапа проектирования реализации.

4. *Руководство по проектированию программ.* Советы прикладным программистам по выбору путей доступа, основанные на анализе характеристик предложенной структуры данных.

5. *Руководство для группы сопровождения базы данных.* Краткие требования, ограничения и данные об имеющихся в наличии технических средствах и математическом обеспечении

для администратора базы данных и обслуживающего персонала.

Проектирование реализации состоит в тщательном документировании в виде схемы всей имеющейся в распоряжении разработчика номенклатуры данных, требований и ограничений с тем, чтобы не возникло вопросов или недоразумений на последующих этапах проектирования.

7.2. Процесс проектирования реализации

В литературе предложен ряд методов проектирования реализации схемы, начиная от осуществляемых полностью вручную до автоматизированных способов [70, 126, 175, 353]. Для наглядности на рис. 7.2 приведены в общем виде основные шаги процесса проектирования реализации. Для того чтобы показать полную последовательность операций, процесс проектирования представлен в виде шагов алгоритма, однако он ни в коем случае не является полным и не содержит соответствующих критериев для оценки сходимости процесса проектирования в целом или его отдельных шагов. Следует отметить, что проектирование реализации может быть в большей степени, чем любая другая стадия разработки, требует взаимодействия с другими этапами проектирования базы данных и прикладного программного обеспечения.

Использованная здесь нумерация шагов соответствует обозначениям, принятым в общей схеме, описанной в гл. 2. Первые из них являются, вероятно, наиболее очевидными и самыми легкими для выполнения. Основой для выполнения описываемых ниже шагов проектирования схемы является концептуальная схема (информационная структура). Задача заключается в том, чтобы сформулировать СУБД-ориентированную схему базы данных, изоморфную относительно концептуальной схемы. Чтобы выполнить эту операцию, необходима дополнительная информация, а именно характеристики модели данных конкретной СУБД. В случае сетевой или реляционной модели данных, отображение концептуальной схемы в СУБД-ориентированную схему производится с использованием набора простых правил. Например, нужны правила для отображения связей типа $M : N$ с использованием связующих записей (в сетевых моделях данных), либо новых отношений (в реляционных моделях), а также правила для создания точек входа. Иерархическая модель данных и ее модификации требуют дополнительных правил: например, для отображения связей порожденного типа записей с несколькими исходными и для обработки связей типа $M : N$.

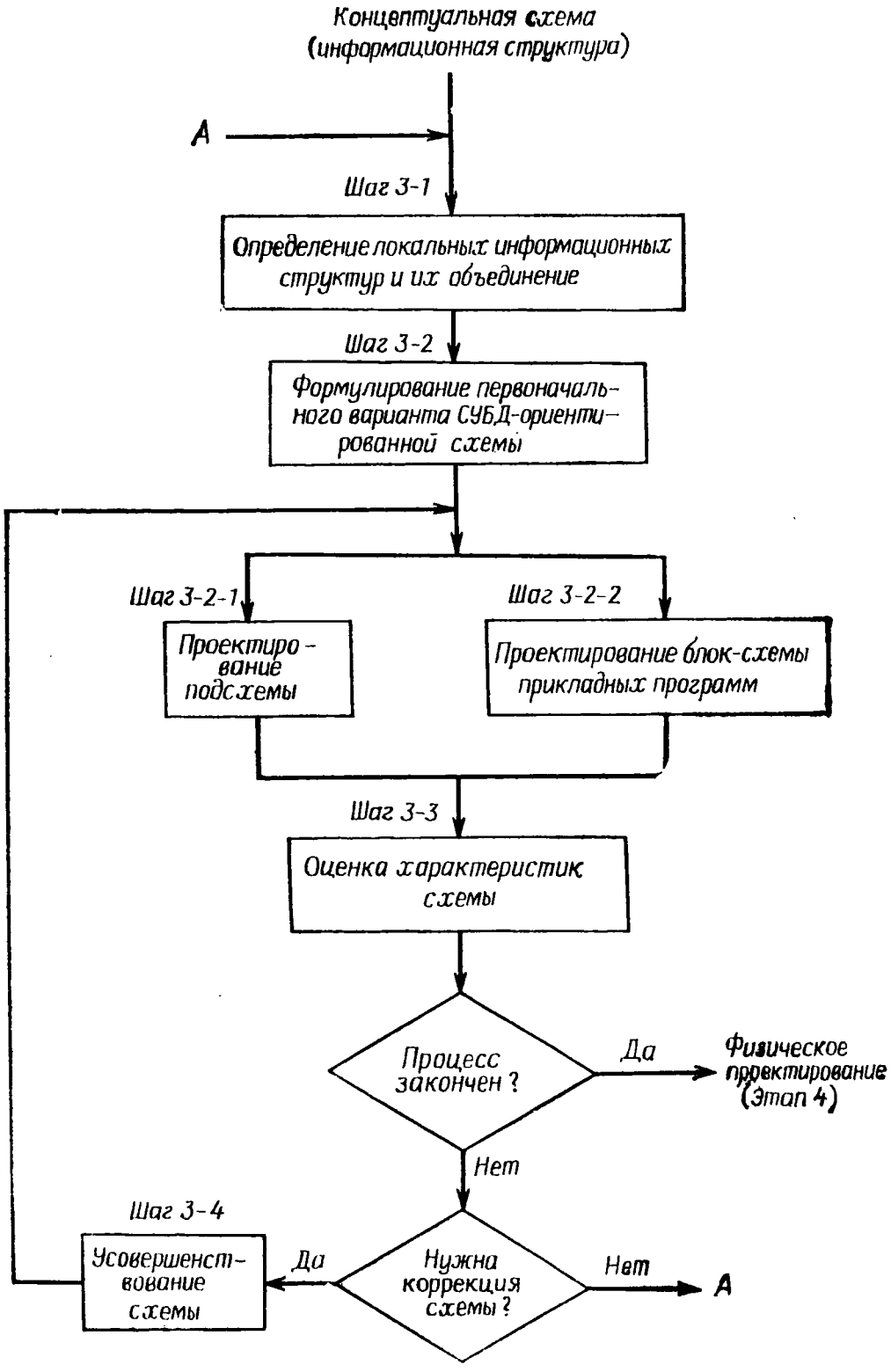


Рис. 7.2. Процесс проектирования реализации.

Выделение локальных информационных структур и их объединение (Шаг 3-1)

На этом шаге выполняется анализ требований обработки с целью выявления семантики данных, идентификации источников их выборки или обновления, а также конкретных данных, подлежащих выборке или обновлению. Если представляется возможным, то для каждого приложения, такого, как формирование отчета, специальный запрос или программа обновления, следует выбрать соответствующее подмножество исходной информационной структуры. Локальные информационные структуры и исходная структура, разработанная на этапе 2, имеют одинаковый формат. После того как будет проанализировано каждое приложение, исходная структура, полученная на этапе 2, может быть объединена с выделенными локальными структурами в новую, пересмотренную информационную структуру. В некоторых случаях введение локальных структур приводит к расширению первоначальной структуры, так как может возникнуть необходимость определения новых сущностей и связей. Таким образом, пересмотренная структура может существенно отличаться от исходной.

После того как первый вариант СУБД-ориентированной схемы сформулирован, процесс проектирования распадается на две параллельные ветви (рис. 7.2).

Используя полученную на этом шаге проектирования в результате объединения локальных информационных структур интегрированную информационную структуру, связи между данными и процессами их обработки, а также характеристики поддерживаемых СУБД типов записей, можно задать исходные типы записей. В простейшем случае сущностям будут соответствовать типы записей, а атрибутам — типы элементов записей. В более сложных случаях сущности могут быть расщеплены или объединены для образования новых типов записей. На данном этапе впервые учитываются специфические для СУБД правила и ограничения определения базы данных, однако рассмотрение большинства характеристик схемы, аналогичных предложенным в Языке описания хранения данных [84], откладывается до этапа физического проектирования.

Проектирование подсхемы (Шаг 3-2-1)

Эта ветвь процесса проектирования аналогична интеграции частных представлений, выполняемой на этапе 2 разработки информационной структуры. Целью этого шага является проектирование интерфейса схемы с прикладными программами, т. е. подсхем. Подсхема представляет собой подмножество схемы, используемое для обеспечения эффективного доступа кон-

кретного пользователя к данным (при этом вопросы защиты данных не рассматриваются). Правила проектирования и оценки схем распространяются и на подсхемы.

Проектирование прикладных программ (Шаг 3-2-2)

Другая ветвь процесса проектирования отражает аспекты разработки прикладных программ. Здесь не ставится цель полностью разработать прикладные программы, так как это было бы трудной задачей. Скорее, на этом шаге разрабатывается схема или структура программы. Каждая транзакция рассматривается достаточно подробно для того, чтобы определить функции доступа к данным и интерфейс с логикой. На этом шаге получаются данные для дальнейшей оценки схемы.

Оценка схемы (Шаг 3-3)

На данном шаге проектирования проводится количественная оценка логической структуры на основе эффективности функционирования базы данных. Определим понятие *«объем обработки»* как сочетание двух параметров: частота обработки и объем данных. *Частота обработки* — частота, с которой нужно проводить обработку отдельного приложения. *Объем данных* — количество хранимых в данный момент в базе данных экземпляров каждого типа записи. Рост базы данных проявляется через возрастание частоты обработки (иногда называемой частотой выполнения приложений) и увеличение объема данных.

На этапе проектирования реализации в качестве количественных оценок эффективности используются оценки количества обращений к логической записи, общего количества байтов, передаваемых при обработке приложения, и полного объема базы (в байтах). С их помощью можно попытаться оценить усредненные значения таких характеристик физической базы данных, как временные затраты и требуемый объем физической памяти. Эта задача более успешно решается для структур баз данных навигационного типа (т. е. сетевых и иерархических), чем для инвертированных структур в случае, если индексы не описаны как дополнительные типы записей. На этом шаге при тех же предположениях относительно индексов можно анализировать реляционные структуры. Следовательно, в зависимости от целей разработчика оценка схемы на данном шаге может быть выполнена либо в терминах проектирования реализации, либо в терминах физического проектирования.

Как и на предыдущих этапах, здесь достаточно трудно установить факт успешного завершения процесса проектирования. Однако в данном случае могут оказаться эффективными такие простые критерии, как соответствие проектного решения концептуальной схеме и обеспечение для подсхемы минималь-

ного объема передачи данных. Если полученное проектное решение этим критериям не удовлетворяет, выполняется усовершенствование схемы (шаг 3-4). В противном случае процесс проектирования реализации считается успешно законченным и можно приступать к физическому проектированию.

Усовершенствование схемы (Шаг 3-4)

Целью этого шага является адаптация схемы к различным способам представления данных. В этом случае мы используем такие средства СУБД, как индексация или функция хеширования. Естественно, что содержание и состав данных изменению не подлежат. Если же схема не может быть улучшена без модификации состава данных, то проектирование схемы откладывается и осуществляется возврат к анализу исходных требований. Результаты усовершенствования схемы проверяются по критериям, описанным на шаге 3-3 (рис. 7.2).

7.3. Эффективность логической структуры базы данных

Чтобы оценить среднее количество экземпляров записей каждого типа, выбираемых из базы данных при обработке конкретного приложения, выполняется расчет количества обращений к логическим записям (LRA). С учетом частот выполнения приложений эти величины указывают приложение, требующее наибольшего количества обращений к базе для ввода-вывода данных. Более того, отмечая приложения, являющиеся доминирующими по частоте выполнения или по значению LRA, отнесенных к единице времени, можно определить, где при проектировании схемы наиболее целесообразно вводить усовершенствования.

Если определить значение LRA для каждого приложения и типа записи, то легко рассчитать общий объем потока данных между прикладными программами и системой управления базой данных, т. е. объем передачи. Объем передачи одного типа записи в одном приложении определяется ее размером и значением LRA этого типа записи в данном приложении. Просуммировав указанный объем передачи по всем типам записей приложения, можно вычислить объем передачи в байтах для данного приложения. В конечном итоге для получения общего объема передачи эти величины должны быть просуммированы с весами, пропорциональными частотам обработки соответствующих приложений.

Оценку величины объема передачи следует использовать в качестве дополнительного параметра при принятии решений по усовершенствованию информационной структуры, так как простая минимизация только LRA часто приводит к слишком

большим по размеру типам записей. Кроме того, использование только LRA в качестве целевой функции не позволяет на физическом уровне успешно разрешить противоречие между временами доступа и объемом памяти (например, увеличение количества индексов может привести к сокращению времени доступа); помимо этого, рассматриваемый подход отдает предпочтение системам навигационного типа, таким, как CODASYL или IMS, в отличие от систем с инвертированной индексацией, таких, как ADABAS или System 2000. Однако эти ограничения можно преодолеть и расширить понятие навигационного доступа, включив туда наряду с данными пользователя индексы в виде индексных типов записей.

7.3.1. Вариант оценки доступа к логическим записям

Расчет характеристик эффективности логических структур баз данных может быть проведен следующим образом [321]:

• *Количество обращений к логическим записям* (LRA)

Количество обращений ко всем типам записи приложения i дается следующим выражением:

$$\mathrm{LRA}_i = \sum_{j=1}^{N} \mathrm{LRA}_{ij}, \tag{7-1}$$

где N — число типов записей в базе данных, LRA_{ij} — количество обращений к логическим записям типа j в приложении i.

Количество обращений к базе данных в единицу времени равно

$$\mathrm{LRA} = \sum_{i=1}^{M} \sum_{j=1}^{N} \mathrm{LRA}_{ij} \times F_i, \tag{7-2}$$

где M — общее количество приложений для данной базы данных; F_i — частота (число) выполнения i-го приложения в единицу времени.

• *Объем передачи данных*

Объем передачи данных (в байтах) для приложения i задается выражением

$$\mathrm{TRVOL}_i = \sum_{j=1}^{N} \mathrm{LRA}_{ij} \times \mathrm{RECSIZE}_j, \tag{7-3}$$

где $\mathrm{RECSIZE}_j$ — средний размер логической записи j в байтах.

Объем передачи данных в единицу времени (в байтах) равен

$$\mathrm{TRVOL} = \sum_{i=1}^{M} \sum_{j=1}^{N} \mathrm{LRA}_{ij} \times \mathrm{RECSIZE}_j \times \mathrm{F}_i. \tag{7-4}$$

•*Объем памяти*

Объем памяти (в байтах) равен

$$\text{DSTOR} = \sum_{j=1}^{N} \text{RECSIZE}_j \times \text{NREC}_j, \qquad (7\text{-}5)$$

где NREC_j — количество экземпляров j-го типа записей в базе данных.

Объем памяти для указателей (в байтах) равен

$$\text{PTRSTOR} = \sum_{j=1}^{N} \text{NREC}_j \times \text{PS} \times \text{NPTR}_j, \qquad (7\text{-}6)$$

где PS — длина указателя (в байтах) и NPTR_j — среднее количество указателей, хранимых совместно с записью типа j.

После того как определена возможная схема и собраны сведения об объемах обработки, все вышеперечисленные параметры, за исключением параметра LRA_{ij}, становятся известными. LRA_{ij} оценивается для каждого из приложений базы данных отдельно. Приложения могут быть достаточно сложными и содержать целый ряд взаимообусловленных между собой обращений к базе данных. В таких случаях полезно выделить отдельные компоненты приложения, указав для них соответствующие им исходные состояния в базе данных, зависящие от реализованных в СУБД правил поддержания «текущего состояния», а также один из перечисленных ниже типов выборки следующей записи:

• Найти уникальный экземпляр записи типа j.

• Найти все экземпляры записи типа j (связанные с исходной записью типа k).

• Найти некоторое подмножество экземпляров записей типа j, удовлетворяющих определенному булевому критерию.

Для этих типов (или классов) выборки величина LRA обычно равна соответственно 1, NRECR_j или $\text{NRECR}_j/2$, где NRECR_j — среднее количество экземпляров записей типа j в определенной связи (обычно порожденных записей для соответствующей исходной) либо мощность отношения.

Опытные проектировщики могут пожелать проанализировать функцию распределения экземпляров записей вместо учета средних оценок количества экземпляров порожденных записей для одной исходной; однако проще, а часто и более полезно проанализировать наилучший и наихудший случаи. Например, если в иерархической базе данных две исходные записи типа B (рис. 7.3) имеют соответственно 10 и 50 экземпляров порожденных записей C, то равная 30 средняя оценка количества записей не является показательной для конкретного слу-

чая, но может служить разумной оценкой для большого количества записей типа *B*. Если разброс значений числа экземпляров записей типа *C*, связанных с одной записью типа *B*, достаточно велик, то способ оценки средней величины должен рассматриваться как наихудший случай. Способ вычисления средней величины LRA иллюстрируется ниже.

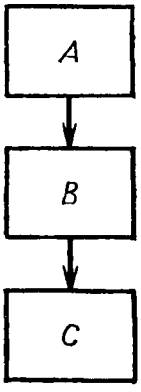


Рис. 7.3. Простейшая иерархическая структура из трех типов записей.

Рассмотрим приведенные на рис. 7.4 три варианта схем, удовлетворяющие определенным требованиям и ограничениям базы данных. Оценим эти схемы, используя введенные ранее

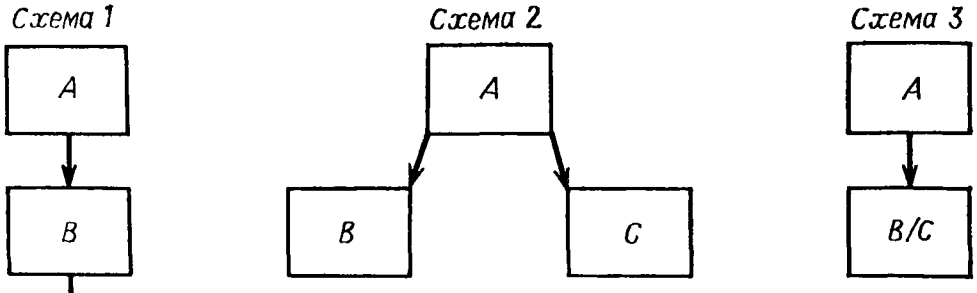


Схема 1: $NREC_A = 100$, $NREC_B = 200$ *для A*, $NREC_C = 30$ *для B*.
Схема 2: $NREC_A = 100$, $NREC_B = 200$ *для A*, $NREC_C = 1000$ *для A*.
Схема 3: $NREC_A = 100$, $NREC_{B/C} = 1000$ *для A* (*5 C для B, B-избыточно*)
$RECSIZE_A = 100$ *байт*, $RECSIZE_B = 200$ *байт*, $RECSIZE_C = 50$ *байт*,
$RECSIZE_{B/C} = 250$ *байт*.
Принятые указатели: первый, исходный, первый порожденный.

Рис. 7.4. Возможные варианты схем и объем данных.

основные характеристики эффективности. Пусть заданы следующие приложения, причем каждое из них выполняется один раз: 1. Найти произвольный экземпляр записи типа *A*. 2. Найти произвольный экземпляр записи типа *B*. 3. Найти произвольный экземпляр записи типа *C*. 4. Найти все экземпляры записи типа *A*. 5. Найти все экземпляры записи типа *B*, связанные с *заданной* записью типа *A*. 6. Найти все экземпляры

записи типа *C*, связанные с *заданной* записью типа *B*. 7. Найти все экземпляры записи типа *A*, связанные с *заданной* записью типа *C*. Для простоты предположим, что приложения с 1 по 4 используют в качестве точки входа запись типа *A* и допускается только последовательный доступ. В данном случае для рассматриваемых схем не имеет значения ни способ их получения, ни их физический смысл.

Таблица 7.1. **Расчетные значения LRA для вариантов схем на рис. 7.4**

Приложения	Схема 1	Схема 2	Схема 3
1	50 *A*	50 *A*	50 *A*
2	50 *A* + 100 *B*	50 *A* + 100 *B*	50 *A* + 500 *B/C*
3	50 *A* + 100 *B* + 3 *C*	50 *A* + 500 *C*	50 *A* + 500 *B/C*
4	100 *A*	100 *A*	100 *A*
5	200 *B*	200 *B*	1000 *B/C*
6	5 *C*	1 *A* + 1000 *C*	4 *B/C*
7	1 *B* + 1 *A*	1 *A*	1 *A*
Всего	251 *A* + 401 *B* + + 8 *C* = 660	252 *A* + 300 *B* + + 1500 *C* = 2052	251 *A* + 2004 *B/C* = = 2255

В табл. 7.1 приведены значения LRA для каждого из приложений. Длина пути поиска произвольной записи сопровождается просмотром половины общего количества экземпляров записей списка. Для последовательного списка поиск всех записей требует времени в два раза больше, чем поиск одной произвольной записи. Уменьшение времени поиска произвольной записи достигается использованием методов индексного доступа или хеширования, которые также могут быть проанализированы с помощью расчета LRA. Один из методов заключается в представлении индекса в схеме новым типом записи и определении LRA для каждого поддерживающего доступ индекса. Объем передачи можно рассчитать исходя из характеристик каждого индексного входа. При моделировании метода хеширования можно положить LRA = 1 для каждого произвольного обращения. Возможные случаи синонимии и переполнения здесь не учитываются, так как такие физические параметры, как размер и процент заполнения блока, на данном этапе проектирования еще не рассматриваются. Таким образом, анализ эффективности произвольного доступа с использованием оценок LRA имеет ряд ограничений.

В соответствии с табл. 7.1 минимум LRA обеспечивает схема 1. Однако существуют и другие критерии оценки качества схемы. В табл. 7.2 и 7.3 приведены результаты расчета объема передачи и объема памяти для рассматриваемых вариантов схем. Из таблиц следует, что, хотя схеме 1 соответствует минимальный объем передачи, она стоит на втором месте по требуемому объему памяти; таким образом, задача состоит в сравнении скорости доступа и объема памяти. Выбор «лучшей» схемы зависит от используемой проектировщиком функции стоимости доступа и объема памяти. В нашем случае схема 1 явно является наиболее эффективной, и при отсутствии строгого ограничения на объем памяти (10,0 Мбайт), предпочтение следует отдать схеме 1.

Таблица 7.2. **Объем передачи в Кбайтах для вариантов схем на рис. 7.4**

Приложения	Схема 1, К	Схема 2, К	Схема 3, К
1	5	5	5
2	25	25	130
3	25,15	30	130
4	10	10	10
5	40	40	250
6	0,25	50,1	1,25
7	0,3	0,1	0,1
Всего	105,7	160,2	526,35

Таблица 7.3. **Объем памяти данных для вариантов схем на рис. 7.4** [1])

	Схема 1	Схема 2	Схема 3
Общее количество записей	$100A + 20\,000B + 100\,000C$	$100A + 20\,000B + 100\,000C$	$100A + 100000B/C$
Память данных, Мбайт	9,01	9,01	25,01
Память указателей, Мбайт	1,04	0,96	0,80
Общий объем, Мбайт	10,05	9,97	25,81

[1]) Указатель занимает 4 байт (исходная запись содержит один указатель, порожденная — 2 указателя).

Из таблиц видно, что обработка некоторых приложений является значительно более дорогой по сравнению с остальными. Для таких приложений наряду с показателями эффективности, указанными выше, необходимо также рассмотреть объем или частоту их обработки. Учет других факторов, таких, как приоритет приложений или ограничения на время отклика, говорит о том, что выбор доминирующих приложений не может основываться только на учете LRA. Это тот случай, когда надлежащее установление системных целей играет важную роль в процессе проектирования реализации.

7.3.2 Взаимосвязь между логическими и физическими показателями эффективности

Взаимосвязь между обращениями к логическим записям и физическим вводам-выводам определяется многими факторами, включая размер блока и физическое группирование записей. Сформулируем здесь некоторые общие правила. Предположим, что физический ввод-вывод состоит в выполнении ряда произвольных обращений к физическим блокам (PBA_r) со средним временем доступа и передачи каждого произвольного блока TRBA и ряда последовательных обращений к физическим блокам PBA_s со средним временем доступа и передачи каждого последовательного блока TSBA. Тогда задача состоит в переходе от значений LRA к функциям PBA_s и PBA_r с последующим преобразованием количества обращений к блокам в общее время обслуживания ввода-вывода, TIO_s и TIO_r. Например:

Для чтения физически последовательного файла требуется

$$PBA_s = \left\lceil \frac{LRA}{BF} \right\rceil \text{ последовательных обращений к блокам,} \qquad (7\text{-}7)$$

где BF — коэффициент блокирования, а $\lceil\ \rceil$ обозначает наименьшее целое число, превосходящее N.

$$TIO_s = PBA_s \times TSBA \text{ с.} \qquad (7\text{-}8)$$

Для прямого доступа к записи потребуется

$$PBA_r = LRA = 1 \text{ произвольных обращений к блокам} \qquad (7\text{-}9)$$

и соответственно

$$TIO_r = PBA_r \times TRBA = TRBA \text{ с.} \qquad (7\text{-}10)$$

Следует отметить, что подобный расчет времени ввода-вывода не учитывает буферизации и возможности того, что блок, к которому производится обращение, уже находится в основной памяти. Следовательно, приведенное выше выражение для TIO

соответствует худшему случаю. Отметим также, что величины TSBA и TRBA зависят от типа используемого оборудования: допускает ли оно доступ в режиме разделения времени или только в монопольном режиме. Обычно при работе в режиме разделения времени предполагается, что каждое обращение к блоку, произвольное или последовательное, осуществляется в случайном порядке из-за влияния других пользователей. Работа в монопольном режиме предполагает, что при произвольном доступе поиск ограничен пределами экстента хранимых данных независимо от того, находятся ли они на одном цилиндре, или на нескольких устройствах; с другой стороны, эффективность последовательного доступа можно повысить путем организации чередования и других способов синхронизации обращений к запоминающему устройству. Например, обычно среднее время обслуживания для НМД, работающего в монопольном режиме, равно

$$\mathrm{TSBA} = \mathrm{ROT}/2 + \mathrm{BKS}/\mathrm{TR}, \tag{7-11}$$

$$\mathrm{TSBA} = \mathrm{SEEK}(\mathrm{NCYL}) + \mathrm{ROT}/2 + \mathrm{BKS}/\mathrm{TR}, \tag{7-12}$$

где ROT — время полного оборота диска, BKS — размер блока в байтах, TR — скорость передачи в байтах в секунду и SEEK (NCYL) — среднее время установки головок в экстенте базы данных, занимающем NCYL смежных цилиндров. Среднее время обслуживания при работе в режиме разделения времени обычно равно

$$\mathrm{TSBA} = \mathrm{TRBA} = \mathrm{SEEK}(\mathrm{CPD}) + \frac{\mathrm{ROT}}{2} + \frac{\mathrm{BKS}}{\mathrm{TR}}, \tag{7-13}$$

где SEEK(CPD) — среднее время установки головок в экстенте, занимающем подряд CPD цилиндров.

Естественно, что эти выражения справедливы не при любых обстоятельствах, но они соответствуют наиболее типичным случаям. Например, требуется определенная коррекция, если для поиска конкретного значения идентификатора записи используется аппаратно реализованное сканирующее устройство. В этом случае TSBA = BKS/TR, так как каждый блок просматривается очень быстро со скоростью прохождения информации под считывающими головками. Здесь могут возникнуть дополнительные задержки, если при считывании случайно приходится пересекать границу цилиндра.

Очень важно попытаться установить аналитическую зависимость между характеристиками эффективности логической и физической баз данных. При этом следует быть очень осторожными, так как эти зависимости могут оказаться достаточно

сложными и поэтому формулировку выражений надо выполнять очень тщательно с учетом характеристик используемых технических и программных средств.

Другим аспектом, на который следует обратить внимание, является использование LRA в качестве единственного показателя эффективности. На рис. 7.5 представлены два простых случая реализации приложения вида «Найти все записи типа *A*». В случае 1 обработка приложения требует 100 LRA, 100 PBA, и 100 TRBA. Так как объем передачи одинаков для обоих случаев (если не учитывать свободное пространство в

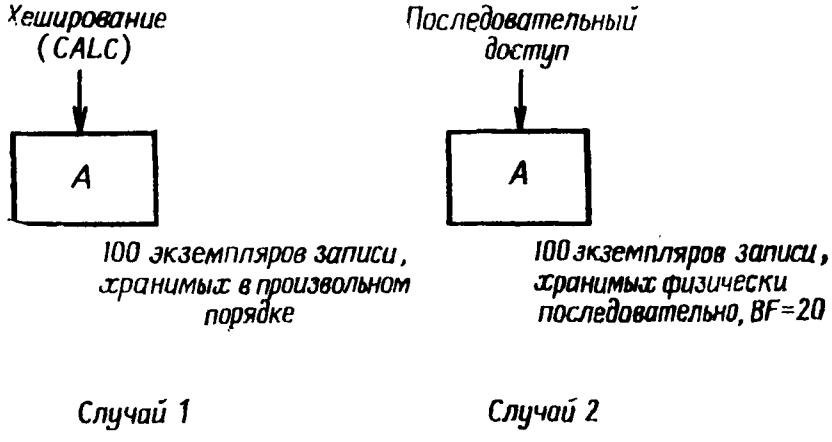


Рис. 7.5. Два варианта схем для реализации приложения вида «Найти все записи типа *A*».

блоке), то различия между рассматриваемыми вариантами состоят в количестве и типах доступов к блоку. Обычно произвольный доступ к блоку требует больше времени, чем последовательный.

Поскольку подсчет величины LRA не позволяет определить количество обращений к блокам, то возможности этого метода для точного предсказания абсолютного значения физической эффективности ограничены. Однако такой подход полезен при сравнительной оценке схем, использующих одинаковые методы доступа.

Как уже отмечалось, различные варианты подхода, основанные на расчете LRA, используются при проектировании иерархических, а также и сетевых структур баз данных [46, 73, 247]. Например, в [73] оцениваются общие затраты, связанные с поиском, коррекцией и генерацией отчета, включая операции реструктурирования данных и сортировки методом слияния. В настоящее время такой подход ограничен иерархическими базами данных. Серьезный недостаток всех описанных методов состоит в отсутствии формального обоснования адекватности принятых решений практическим характеристикам,

полученным путем тестирования реализованной схемы. Это характерно почти для всех методов, связанных с аналитическим моделированием структур баз данных.

7.4. Две методики проектирования реализации

На практике всегда существуют две основные методики проектирования СУБД-ориентированной схемы исходя из возможностей СУБД-независимой структуры базы данных: анализ и синтез. Анализ схемы обычно проще, так как вариант СУБД-ориентированной схемы уже разработан (обычно вручную) и остается оценить его эффективность для заданной рабочей нагрузки; чаще всего оценка выполняется в единицах длины пути доступа и требуемого объема памяти. Подход, описанный в разд. 7.3, связан с расчетом LRA и является типичным для этого случая.

Проектировщик, выполняющий синтез схемы, тоже должен проводить оценку принятых решений, однако наряду с указанной перед ним стоит и дополнительная задача: спроектировать реализуемую и по возможности «оптимальную» в смысле принятого критерия эффективности логическую структуру данных. При этом используются различные подходы, но они еще не обобщены и не проанализированы с позиций целесообразности применения. В данном разделе приведены некоторые из наиболее известных разрабатываемых в настоящее время подходов.

Методы синтеза схемы можно разделить на четыре категории: прагматический (ручной) подход, алгоритмы кластеризации атрибутов данных для формирования записей, оптимизация связей и метод доказательства теорем [126]. Прагматический подход подразумевает *синтез схемы базы данных,* основанный на использовании связей между сущностями [69, 70, 72]. В этом случае типы сущностей определяют типы записей, а связи вида $1:N$ определяют типы наборов, содержащие соответствующие им типы записей, причем один тип записи является владельцем типа набора. Таким образом, элементы данных группируются по типам записей в соответствии с СУБД-независимой диаграммой сущностей, которая была выведена ранее.

Иногда более эффективным может оказаться разбиение атрибутов сущностей на некоторые группы типов записей, или наоборот, объединение атрибутов нескольких объектов в один тип записи. Некоторые исследователи предложили матричные методы группирования атрибутов в типы записей. Обычно создается матрица $N \times N$, где N — количество элементов в информационной структуре, а элемент матрицы (i, j) соответствует

частоте, с которой атрибуты i и j будут обрабатываться вместе. Большие частоты совместного использования указывают пары атрибутов, которые следовало бы поместить в один тип записи. Например, метод, описанный в [24], применяет для объединения атрибутов, которые чаще всего используются вместе, «алгоритм ближайших центроид». В этом случае атрибут может быть отнесен к нескольким кластерам и, таким образом, это приводит к избыточности в хранении данных. Метод Хаффера [155] использует для кластеризации *алгоритм энергии связи* (BEA), основанный на использовании мер парного подобия атрибутов, заданных в виде матрицы. Группируются вместе атрибуты, соответствующие подматрицам матрицы подобия, имеющим бо́льшие значения элементов (i, j).

В работе [159] рассмотрен метод автоматизированного проектирования логической структуры баз данных. Этот метод основан на объединении частных информационных структур, называемых локальными представлениями, отражающих специфику отдельных приложений. Эти локальные представления задаются в виде списка элементов данных, бинарных ассоциаций между элементами данных и описаний ассоциаций. Некоторые элементы данных определяются как ключи, остальные — как атрибуты. Метод предполагает объединение элементов данных в группы в соответствии с их ассоциацией таким образом, что каждый кластер содержит один ключ. Ассоциации между ключами становятся ассоциациями между соответствующими кластерами. Если этот метод используется для проектирования модели данных CODASYL, то кластер реализуется одним типом записи, а реализация межгрупповых ассоциаций осуществляется в соответствии с их размерностью. Ассоциация типа $1{:}n$ становится типом набора, а ассоциация типа $m{:}n$ представляется типом записи-связи и двумя типами набора, каждый из которых содержит тип записи-связи в качестве члена типа набора. Владельцами типов набора являются каждый из двух типов записей, соответствующих двум кластерам. Этот метод реализован в программном обеспечении системы IMS как Средство Проектирования Базы Данных IBM.

Одним из первых примеров применения методов оптимизации к решению задачи автоматизации проектирования логической структуры баз данных был подход, предложенный в [229]. В этом методе в качестве исходных данных берется список элементов данных и описания взаимосвязей между ними. Результатом является информационная структура, описанная как схема CODASYL. Каждая связь может быть реализована одним из 12 способов. Для нахождения комбинации связей, которая обеспечивает минимум средней стоимости доступа при ограничении на объем памяти, используется метод целочис-

ленного программирования. Расширение этой модели применительно к системе IMS наряду с системами CODASYL выполнено в [172]. В работе [25] этот подход был распространен на случай использования памяти со страничной организацией.

Модель, названная DESIGNER (ПРОЕКТИРОВЩИК), использует методы искусственного интеллекта для проектирования структуры логической базы данных CODASYL [131, 132]. В ПРОЕКТИРОВЩИКЕ используется метод доказательства теорем для того, чтобы синтезировать логическую схему CODASYL, исходя из запросов, обрабатываемых предполагаемой системой баз данных. Запросы преобразуются системой в набор утверждений о структуре взаимосвязей элементов данных, участвующих в запросе. Новые утверждения выводятся из старых как теоремы. Эти утверждения определяют допустимые ограничения на логическую структуру и используются для генерации логической структуры, способной представить всю требуемую в запросах информацию.

Глава 8. Пример проектирования схемы

Рассматриваемый пример иллюстрирует основные проблемы, с которыми сталкивается разработчик схемы, пытающийся максимизировать производительность интегрированной базы данных [321]. Предлагаемое решение представляет собой пересмотренный вариант подхода Бубенко [43, 44].

8.1. Спецификации требований (Этап 1)

Крупная промышленная компания намерена разработать автоматизированную систему управления коллективом рабочих и служащих на ряде своих заводов. Компания объединяет 50 заводов, расположенных в 40 штатах, и примерно 100 000 служащих. Каждый завод разделен на цехи, а они в свою очередь на производственные участки. Всего в данной компании насчитывается 100 цехов и 500 производственных участков. Каждый цех снабжен оперативным регистрирующим устройством, которое фиксирует время прихода и ухода служащего. Рабочее задание служащего связано с одним из 20 видов работ. Любой вид работы может выполняться на любом заводе компании. В течение рабочего дня служащий может выполнить несколько рабочих заданий, каждое из которых состоит из различных видов работ и может быть выполнено на любом производственном участке. Каждый производственный участок оборудован регистратором производства, работающим в режиме «on-line». С помощью регистратора служащий сообщает о выполнении очередного рабочего задания. В компании представлено 5 профессиональных союзов, каждый служащий является членом только одного из них. Хотя численность компании остается стабильной, в течение года около 20 % общего объема служащих увольняется и заменяется новыми сотрудниками.

Требования обработки

Для удовлетворения информационных потребностей компании предполагаемая автоматизированная система должна обрабатывать следующие входные транзакции:

T1. Дата найма служащего, входной номер служащего, имя, адрес, дата рождения, наименование завода, профсоюз, срок найма.

T2. Дата прекращения работы служащего, имя служащего, номер служащего, дата увольнения. Записи о служащем удаляются и создается отдельный файл о бывших служащих.

T3. Момент выполнения рабочего задания, номер служащего, номер производственного участка, вид работы, текущая дата, число затраченных часов.

В результате проведения предварительного анализа информационных потребностей фирмы для успешного управления коллективом рабочих и служащих была выявлена необходимость в следующих отчетах:

R1. Поквартальный отчет каждому профсоюзу, содержащий: название компании, количество членов данного профсоюза, работавших в данном квартале, общее количество рабочих часов, общая сумма заработной платы; кроме того, для каждого профсоюза выдается список его членов с указанием имени, адреса, номера страхового полиса (SSN), количества проработанных дней и среднемесячного заработка за этот квартал.

R2. Ежегодный отчет для администрации штатов о служащих, проживающих в данном штате, с указанием имени служащего, SSN, общей суммы заработной платы.

R3. Ежемесячный отчет для администрации штата, в котором расположен завод: наименование компании, наименование заводов, расположенных в штате, количество занятых служащих, количество уволенных служащих и количество служащих на последний день месяца.

R4. Ежемесячный отчет правлению компании по каждому заводу: наименование завода, общее количество проработанных служащими часов, общий объем заработной платы, а также рабочие часы и описание работ.

R5. Ежедневный отчет руководству завода: имя служащего, SSN, время простоя (в час) на каждый вид работы и общее количество рабочих часов на данном производственном участке (для всех служащих по типам работ).

R6. Ежемесячные заводские платежные ведомости с указанием для каждого служащего: имени, адреса, SSN, заработной платы.

R7. Справки в отдел кадров в оперативном режиме о каждом служащем: имя, количество дней, проработанных в этом году, заработная плата за текущий год, количество проработанных часов по отчетам (на момент ввода запроса).

С учетом требований, сформулированных выше, цель проектирования базы данных состоит в следующем: обеспечить быстрый доступ к данным с помощью оперативных устройств ввода данных (регистраторов производства); обеспечить своевременное представление отчетов различным уровням управления с

ежедневной, ежемесячной, а иногда и с большей периодичностью; обеспечить обработку оперативных запросов об отдельных служащих; обеспечить наряду с обработкой текущих приложений обработку произвольных приложений, причем время выполнения некоторых из них измеряется несколькими секундами. Ограничения на показатели эффективности работы автоматизированной системы в явном виде не заданы, поэтому к ним специальных требований не предъявляется.

За счет подробной формулировки поставленной задачи фаза анализа в описанном примере значительно упрощена. Все вводимые допущения базируются на информации, содержащейся в описании задачи, а отсутствие реальных пользователей устраняет необходимость пересмотра спецификаций.

8.2. Концептуальное проектирование (Этап 2)

Первое, что делают на этом этапе — это анализируют общие информационные требования и формулируют первоначальную информационную структуру (т. е. концептуальную схему). Такой подход обеспечит требуемую гибкость дальнейшего проектирования; полученная концептуальная схема послужит отправной точкой итерационной процедуры проектирования логической структуры базы данных. И возможно, что при наличии нескольких различных представлений пользователей на данном этапе потребуется произвести их объединение.

Исходная информационная структура, представленная на рис. 8.1, является результатом анализа естественных связей внутри промышленной фирмы, описанных в спецификациях требований. Понятие связи типа 1:*M* иллюстрируется потенциальным существованием различных цехов в пределах одного завода. Связь определяется для общего случая, хотя определенный завод может иметь только один цех. Отношение типа 1:1 определено между сущностями ЧАСЫ и ЦЕХ, для того чтобы показать уникальность отсчета текущего времени для каждого цеха. Отношение типа *M*:*N* между сущностями СЛУЖАЩИЙ и ПРОИЗВ-УЧАСТОК отражает тот факт, что служащий в течение рабочего дня или более длительного периода времени может работать на разных производственных участках.

Первоначальная информационная структура базируется на многочисленных предположениях относительно данных, и поэтому она должна быть тщательно проверена. Например, в нашем случае считается, что каждый экземпляр сущности ПРОИЗВ-УЧАСТОК связан со многими экземплярами сущности ВИД-РАБОТЫ, поскольку известно, что такое утверждение справедливо для многих заводов. Однако явного указания к данному утверждению в постановке задачи не содержится. Тем не менее эти и другие более простые предположения в неявном

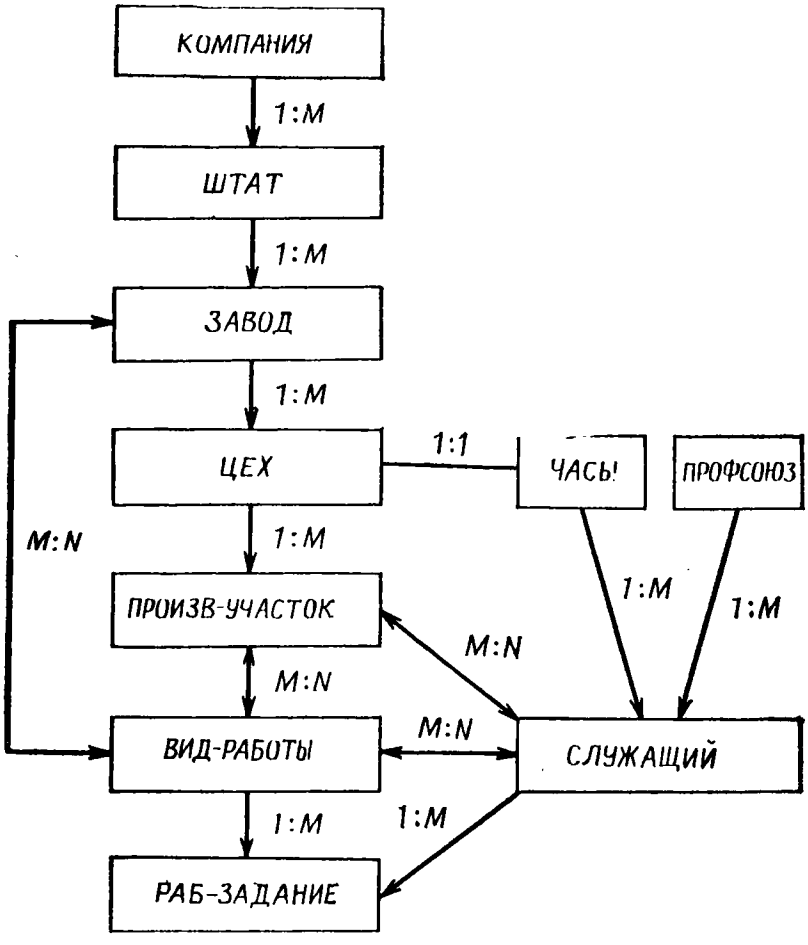


Рис. 8.1. Исходная информационная структура.

виде в информационной структуре присутствуют. Отметим также, что в нашем примере рассматриваемый этап проектирования значительно упрощен, так как в спецификациях требований присутствует только единственное концептуальное представление. И поэтому проводить объединение представлений не требуется. Однако на Этапе 3 такая необходимость возникнет.

Объем данных на Этапе 1 и Этапе 2 не учитывается, но его легко оценить на уровне сущностей исходя из общих информационных требований. Это промежуточное использование объема данных может служить подтверждением значимости предложенных связей.

8.3. Проектирование реализации (Этап 3)

Шаг 3-1. Выделение и объединение локальных информационных структур

Этот шаг начинается с анализа содержания каждого требования обработки данных. Затем для каждого приложения может быть составлена соответствующая локальная информа-

ционная структура (сущности, связи и их атрибуты), которая удовлетворяет всем требованиям к данным соответствующего приложения (т. е. отчет или результат транзакции обновления). Первоначальная информационная структура может быть использована как основа для формирования локальных структур, но обычно требования обработки требуют ее расширения. Если целью является только краткосрочная эффективность, то первоначальная информационная структура после этого этапа больше не понадобится. Локальные информационные структуры объединяются в одну глобальную структуру, из которой исключено большинство избыточных сущностей и их атрибутов, но которая содержит требуемые для обработки данных пути доступа.

Требования обработки данных промышленной компании определяют семь отчетов и три транзакции обновления, которые должны быть реализованы с использованием базы данных. В дополнение к этому списку рано или поздно должна быть определена по крайней мере еще одна предполагаемая транзакция: для каждого служащего изменять дату и сумму любого повышения заработной платы (возможно, поквартально). Другие обновления, такие, как изменение адреса служащего, здесь не рассматриваются, хотя на практике они должны быть учтены. Локальные информационные структуры для всех приложений представлены на рис. 8.2. Атрибуты сущностей перечислены справа от соответствующего квадрата. В данной упрощенной методике не проводится никакого различия между атрибутами сущностей и элементами данных, которые составляют логические записи. Однако пока не предполагается, что сущностям соответствуют отдельные типы логических записей. Иногда новые сущности должны быть определены или выделены из других, уже существующих сущностей, или наоборот, получены в результате их объединения.

Для удобства анализа содержания обрабатываемых данных целесообразно составить, как это показано на рис. 8.3, *матрицу обработки*, в которой каждая транзакция или отчет представляются в терминах необходимых элементов данных (атрибутов). Эта матрица используется с целью компактного представления сведений о требованиях обработки. Причем в ней не указано, являются ли эти данные результатом обработки, или они хранятся в базе данных в явном виде. В ней просто собраны все данные, требуемые в приложениях.

Объединение локальных информационных структур в единую глобальную информационную структуру проводится путем пошагового объединения локальных структур друг с другом или с первоначальной информационной структурой. Для примера рассмотрим транзакции Т2 и Т3 на рис. 8.2, которые содержат

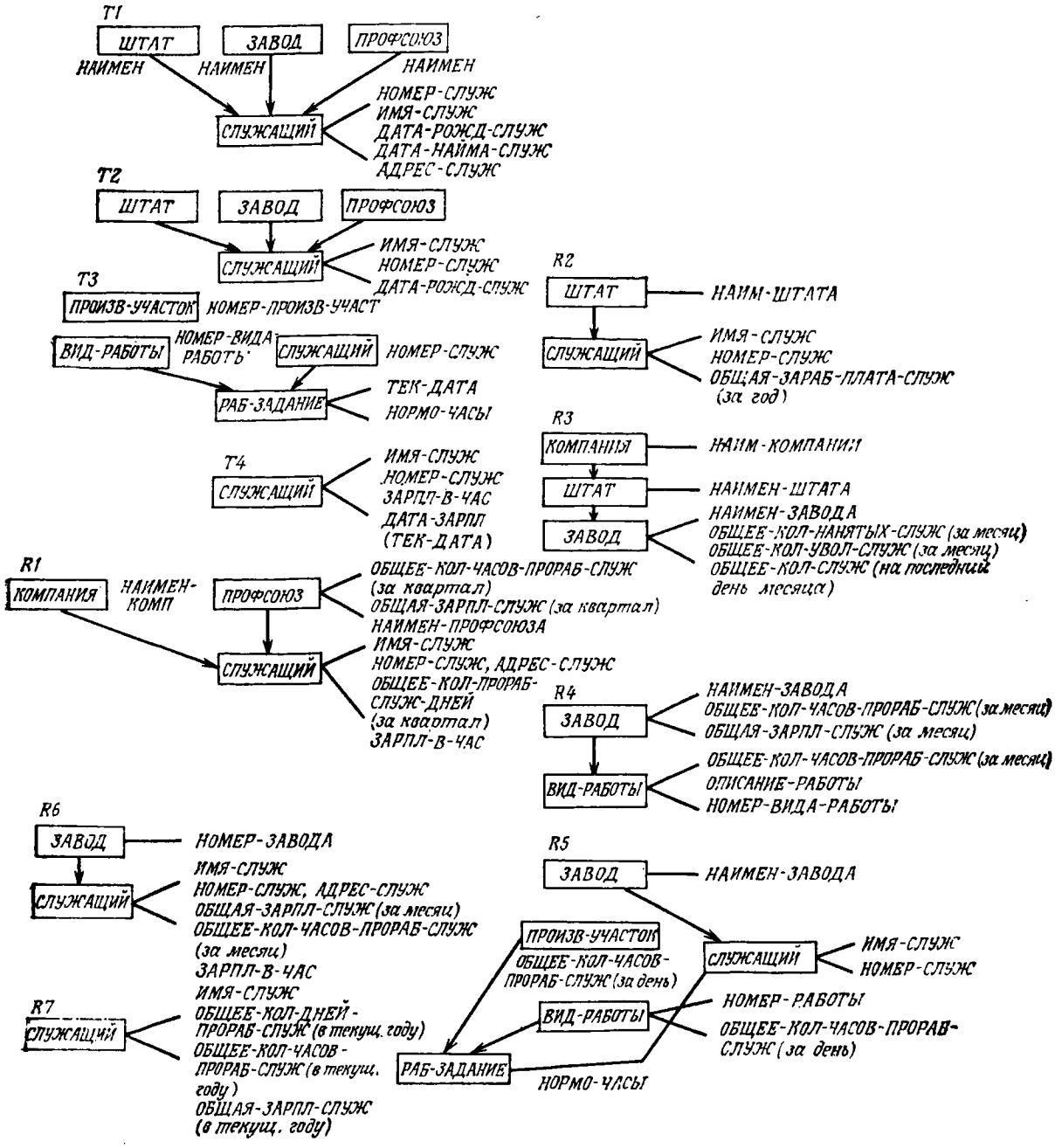


Рис. 8.2. Локальные информационные структуры, полученные на основе анализа содержания обрабатываемых данных.

общую сущность СЛУЖАЩИЙ. Объединение этих двух приложений представляет собой простое слияние сущностей и связей (рис. 8.4). В случаях когда сущности или связи сущностей в приложениях не совпадают, эта операция будет более сложной. В локальной структуре отчета R5, которая будет объединяться со структурой, представленной на рис. 8.4, сущность ПРОИЗВ-УЧАСТОК имеет непосредственную связь с сущностью РАБ-ЗАДАНИЕ. Их объединение добавит эту связь в

Элементы данных	Транзакции				Отчеты / запросы						
	T1	T2	T3	T4	R1	R2	R3	R4	R5	R6	R7
1. НАИМ-КОМПАНИИ					X	X	X				
2. ТЕК-ДАТА			X	X							
3. ТЕК-ВРЕМЯ			БУДУЩЕЕ?								
4. ИМЯ-УВОЛЕННОГО	БУДУЩЕЕ?										
5. АДРЕС-СЛУЖ	X				X					X	
6. ДАТА-РОЖД-СЛУЖ	X										
7. ДАТА-УВОЛЬН		X									
8. ДАТА-НАЙМА	X										
9. ИМЯ-СЛУЖ	X	X		X	X	X			X	X	X
10. НОМЕР-СЛУЖ	X	X	X	X	X	X			X	X	
11. ОПИСАН-РАБОТЫ								X			
12. ВИД-РАБОТЫ			X					X	X		
13. НАИМ-ЗАВОДА	X						X	X	X	X	
14. ЗАРПЛ-В-ЧАС				X	X					X	
15. ДАТА-ЗАРПЛ				X							
16. НАИМЕН-ШТАТА						X	X				
17. НОРМО-ЧАСЫ			X						X		
18. ОБЩ-КОЛ-СЛУЖ							ШТАТ/МЕСЯЦ				
19. ОБЩЕЕ-КОЛ-ДНЕЙ-ПРОРАБ-СЛУЖ					СЛУЖ/КВАРТАЛ						СЛУЖ/ГОД
20. ОБЩ-КОЛ-УВОЛ-СЛ							ШТАТ/МЕСЯЦ				
21. ОБЩАЯ-ЗАРПЛ-СЛУЖ					ПРОФСОЮЗ/КВАРТАЛ	СЛУЖ/ГОД		ЗАВОД/МЕСЯЦ		СЛУЖ/МЕСЯЦ	СЛУЖ/ГОД
22. ОБЩ-КОЛ-НАН-СЛ							ШТАТ/МЕСЯЦ				
23. ОБЩЕЕ-КОЛ-ЧАСОВ-ПРОРАБ-СЛУЖ					ПРОФСОЮЗ/КВАРТАЛ			ЗАВОД; РАБОТА/МЕСЯЦ	СЛУЖ; РАБОТА/ДЕНЬ	СЛУЖ/МЕСЯЦ	СЛУЖ/ГОД
24. НАИМ-ПРОФСОЮЗА	X				X						
25. НОМЕР-ПРОИЗ-УЧ			X						X		

Агрегат данных: на один объект в единицу времени

Рис. 8.3. Матрица обработки.

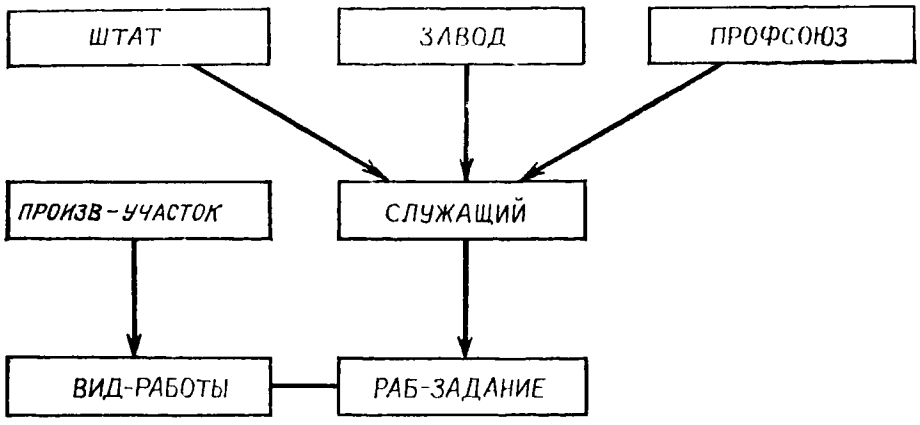


Рис. 8.4. Объединение транзакций Т2 и Т3.

получаемую структуру. Однако, для того чтобы определить, являются ли непосредственная или косвенная связь необходимыми, потребуется дальнейший анализ. Вопрос может быть решен с учетом того, что, во-первых, косвенная связь через ВИД-РАБОТЫ приводит к избыточности записей ВИД-РАБОТЫ (т. е. 20 записей на каждый экземпляр ПРОИЗВ-УЧАСТОК, или всего 20×500 записей). И во-вторых, так как в транзакции Т3 можно использовать прямую связь, то связь между ПРОИЗВ-УЧАСТОК и ВИД-РАБОТЫ в конце концов будет уничтожена. Тем не менее окончательное решение должно быть отложено до завершения анализа всех приложений.

Пересмотренная информационная структура, полученная в результате пошаговой процедуры, показана на рис. 8.5. Отметим, что сущности КОМПАНИЯ, ЦЕХ, ЧАСЫ опущены, так как в текущих требованиях обработки они непосредственно не используются. Удаление сущности КОМПАНИЯ является оправданным, потому что имеется только одно ее значение (т. е. одна компания) и ее название может легко поддерживаться прикладными программами. (Обычно название компании изменяется редко.) Сущность ЧАСЫ удаляется, так как изменение времени не существенно для базы данных. Такой атрибут, как ТЕКУЩЕЕ-ВРЕМЯ, обозначает время события и адекватно заменяет ЧАСЫ в этой базе данных. Кроме того, в текущих отчетах не используются сведения о времени начала и окончания работы служащих, поэтому их можно объединить на начальной стадии принятия решения в отдельный файл или базу данных. Сущность ЦЕХ также не используется в текущих отчетах, но ее удаление может оказаться ошибочным, если в перспективе администрация решит, что необходимы отчеты на уровне цехов. Следовательно, структура, изображенная на рис. 8.5, рассматривается как структура, предназначенная только для краткосрочного использования.

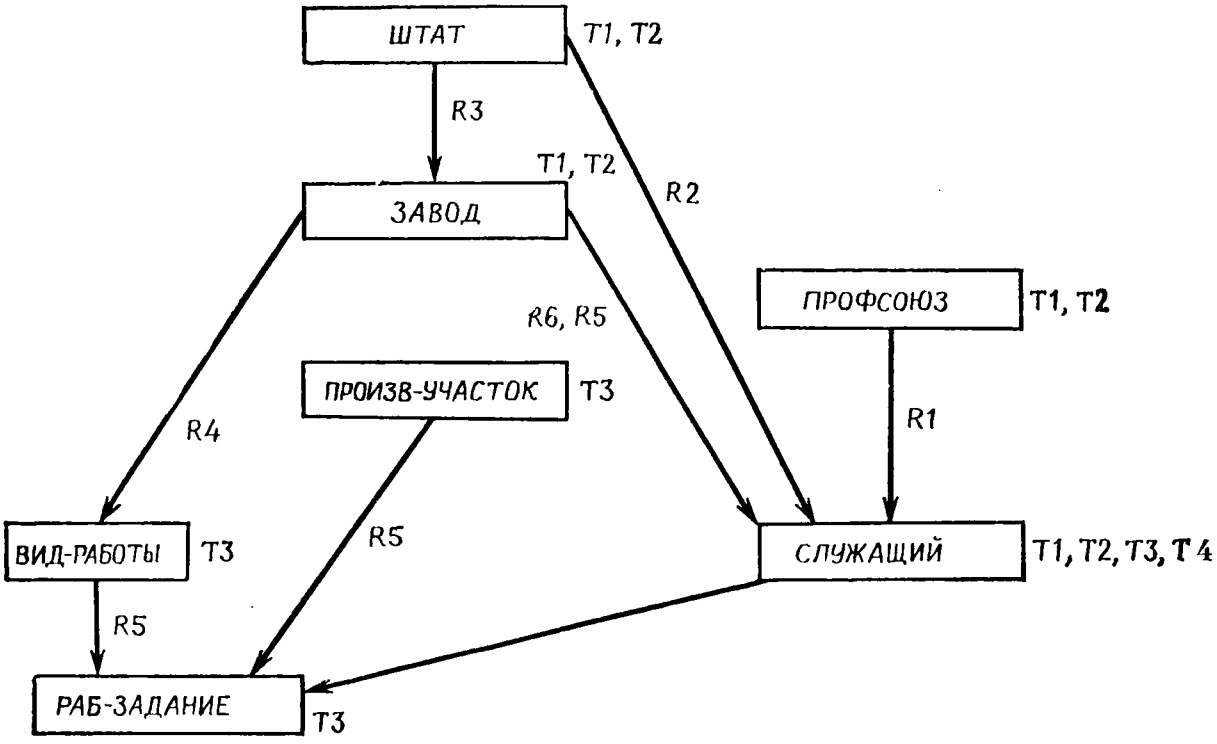


Рис. 8.5. Глобальная информационная структура.

Ниже перечислены дополнительные предположения, при которых существует полученная структура:

1. НОМЕР-СЛУЖАЩЕГО = НОМЕР СТРАХОВОГО ПОЛИСА.
2. Сведения о выполнении рабочего задания не накапливаются, так как итоги подводятся в конце каждого рабочего дня.
3. Отчет R4 «Общий объем заработной платы» отражает заработную плату каждого служащего за месяц.
4. Служащий проживает в том же штате, где он (или она) работает (достаточно сильное упрощение).
5. Информацию о предыдущих окладах отдельных служащих база данных не содержит.

На данном шаге разработки должна быть тщательно проверена допустимость этих и всех других предположений относительно целей проектирования, информационных требований и содержания обрабатываемых данных.

В локальных информационных структурах одни атрибуты сущностей легко определяются исходя из естественных свойств существующих объектов. Другие атрибуты являются производными явно существующих данных и могут использоваться для хранения значений данных, агрегированных, в частности, по различным периодам времени. Например, такие атрибуты, как

ОБЩЕЕ-РАБ-ВРЕМЯ-СЛУЖ, неоднократно используются в ежедневных, ежемесячных, промежуточных и годовых отчетах. Поскольку неэффективно допускать неоправданно большое накопление данных (при этом значительно ухудшаются такие характеристики, как затраты памяти и время доступа), то часто агрегирование данных проводится по мере их использования в отчете, для которого эти данные были предназначены. Так, поступающие ежедневно сведения о времени, затраченном служащим на выполнение задания, добавляются к ежемесячно накапливаемым итоговым значениям для их использования в ежемесячных отчетах и т. д. На рис. 8.2 показаны атрибуты, представляющие возможные агрегированные значения, необходимые для выдачи различных отчетов. Решение о том, хранить ли эти величины в явном виде или получать их с помощью агрегирования, откладывается до рассмотрения объемов обработки данных на Шаге 3-4.

Шаг 3-2. Формулирование СУБД-ориентированной логической структуры базы данных

На этом шаге в рассмотрение включаются спецификации используемой СУБД, а информационная структура преобразуется в СУБД-ориентированную логическую структуру (схему) базы данных. Особое внимание следует уделить ограничениям на содержимое записей и тому, как могут быть определены точки входа.

Преобразованная информационная структура, изображенная на рис. 8.5, является сетевой структурой, представляющей в наиболее обобщенной форме связанные сущности. Преобразование этой диаграммы, например, в сетевую схему CODASYL осуществляется очень просто, если в ней отсутствуют явные отношения типа $M : N$. С этого момента предполагается, что каждой сущности соответствует отдельный тип записи. Таким образом, получается возможный вариант СУБД-ориентированной схемы, который в дальнейшем модифицируется с учетом системных требований. Очень часто на различных этапах проектирования производится расщепление и/или объединение сущностей в типы записи так, чтобы получить максимальную эффективность либо удовлетворить ограничениям безопасности, целостности и т. п. Для преобразования структуры, изображенной на рис. 8.5, в схеме CODASYL не требуется проводить никаких изменений, за исключением указания точек входа. Отметим, что при переходе к схеме CODASYL связи определяют возможные пути доступа. В общем случае связи не обязательно реализуются как пути доступа, но на практике это соответствие обычно выполняется.

ЗАПИСЬ ШТАТ		2 СИМВ	
	ЭЛЕМЕНТ НАИМЕН-ШТАТА	2 СИМВ	КЛЮЧ
ЗАПИСЬ ЗАВОД		58 СИМВ	
	ЭЛЕМЕНТ НАИМЕН-ЗАВОДА	20 СИМВ	КЛЮЧ
	ЭЛЕМЕНТ ОБЩЕЕ-КОЛ-НАНЯТЫХ-СЛУЖ-В-МЕС	5 СИМВ	
	ЭЛЕМЕНТ ОБЩЕЕ-КОЛ-УВОЛ-СЛУЖ-В-МЕС	5 СИМВ	
	ЭЛЕМЕНТ ОБЩЕЕ-КОЛ-СЛУЖ	6 СИМВ	
	ЭЛЕМЕНТ ОБЩЕЕ-КОЛ-ЧАСОВ-ПРОРАБ-СЛУЖ-В-МЕС	10 СИМВ	
	ЭЛЕМЕНТ ОБЩАЯ-ЗАРАБ-ПЛАТА-СЛУЖ-В-МЕС	12 СИМВ	
ЗАПИСЬ ПРОФСОЮЗ		20 СИМВ	
	ЭЛЕМЕНТ НАИМЕН-ПРОФСОЮЗА	20 СИМВ	КЛЮЧ
ЗАПИСЬ СЛУЖАЩИЙ		96 СИМВ	
	ЭЛЕМЕНТ ИМЯ-СЛУЖАЩЕГО	20 СИМВ	КЛЮЧ
	ЭЛЕМЕНТ НОМЕР-СЛУЖ	10 СИМВ	КЛЮЧ
	ЭЛЕМЕНТ АДРЕС-СЛУЖ	24 СИМВ	
	ЭЛЕМЕНТ ДАТА-РОЖДЕН-СЛУЖ	6 СИМВ	
	ЭЛЕМЕНТ ДАТА-НАЙМА-СЛУЖ	6 СИМВ	
	ЭЛЕМЕНТ ОБЩЕЕ-КОЛ-ЧАСОВ-ПРОРАБ-СЛУЖ-В-ДЕНЬ	4 СИМВ	
	ЭЛЕМЕНТ ОБЩЕЕ-КОЛ-ЧАСОВ-ПРОРАБ-СЛУЖ-В-МЕС	4 СИМВ	
	ЭЛЕМЕНТ ОБЩЕЕ-КОЛ-ЧАСОВ-ПРОРАБ-СЛУЖ-В-ГОД	4 СИМВ	
	ЭЛЕМЕНТ ОБЩЕЕ-КОЛ-ДНЕЙ-ПРОРАБ-СЛУЖ-В-КВАРТАЛ	4 СИМВ	
	ЭЛЕМЕНТ ОБЩЕЕ-КОЛ-ДНЕЙ-ПРОРАБ-СЛУЖ-В-ГОД	4 СИМВ	
	ЭЛЕМЕНТ ОБЩАЯ-ЗАРАБ-ПЛАТА-СЛУЖ-В-ГОД	10 СИМВ	
ЗАПИСЬ ПРОИЗВ-УЧАСТОК		3 СИМВ	
	ЭЛЕМЕНТ НОМЕР-ПРОИЗВ-УЧАСТ	3 СИМВ	КЛЮЧ
ЗАПИСЬ ВИД РАБОТЫ		26 СИМВ	
	ЭЛЕМЕНТ НОМЕР-ВИДА-РАБОТЫ	2 СИМВ	
	ЭЛЕМЕНТ ОПИСАНИЕ-РАБОТЫ	20 СИМВ	КЛЮЧ
	ЭЛЕМЕНТ ОБЩЕЕ-КОЛ-ЧАСОВ-ПРОРАБ-СЛУЖ-В-МЕС	4 СИМВ	
ЗАПИСЬ РАБ-ЗАДАНИЕ		10 СИМВ	
	ЭЛЕМЕНТ НОРМО-ЧАСЫ	4 СИМВ	
	ЭЛЕМЕНТ ТЕК-ДАТА	6 СИМВ	

Рис. 8.6. Определение простых логических записей.

Требования обработки в рассматриваемом примере выявляют в качестве возможных точек входа семь типов записей. Типы записей и их содержание приведены на рис. 8.6. Специально выделены ключевые элементы данных. Их выбор зависит от уникальности идентифицирующих элементов данных, описывающих сущности (первичные ключи), и их использования в процессе доступа к данным в отдельных приложениях [266].

Превращение сетевой информационной структуры в несетевую СУБД-ориентированную структуру требует дополнительного анализа. Например, схема на рис. 8.5 является иерархической во всем, за исключением того, что СЛУЖАЩИЙ и РАБ-ЗАДАНИЕ связаны с несколькими типами записей, что недопустимо в иерархических базах данных. Однако в системе IMS фирмы IBM, например, одна из связей между исходной и порожденными типами записей может быть определена как «логическая связь» с «логическими» указателями между записями одной или двух «физических» баз данных. Отношения в реляционных базах данных могут быть также представлены типами записей, обозначенными на рис. 8.5. Явно указанные взаимосвязи между типами записей поддерживаются путем дублирования данных в существующих отношениях (типах записей) либо путем создания новых отношений, содержащих составные ключи двух связанных типов записей [113, 218]. Приемлемым альтернативным подходом для реляционных баз данных было бы начать проектирование на этом шаге с определения базы данных в нормальной форме и оптимизации количества LRA или объема передачи с учетом ограничений желаемого уровня нормализации [28, 85].

Шаг 3-3. Оценка схемы логической структуры базы данных по объему обработки

На этом шаге подробно анализируются требования обработки данных с целью определения объемов данных и частоты обработки. Для каждого приложения следует рассчитать такие показатели производительности, как количество обращений к логическим записям, объем передачи (количество переданных байтов) и требуемый объем памяти. Затем можно выделить критические приложения, имеющие строгий приоритет или особые требования, и доминирующие приложения, характеризующиеся наибольшим количеством записей или байтов, выбираемых из базы данных за данный промежуток времени. Общее число записей, выбранных за заданный промежуток времени, зависит от количества выбранных при выполнении приложения записей, помноженного на частоту его выполнения. Прослеживание путей доступа в объединенной информационной структу-

Таблица 8.1. **Количество обращений к логическим записям (LRA) для различных приложений (скорректированная информационная структура (рис. 8.5))** [1]

Приложение	Частота	LRA	Кол-во LRA в приложении	Кол-во LRA за день	Объем передачи данных, К
T1	100 за день	1 СЛУЖАЩИЙ + 1 ЗАВОД + 1 ПРОФСОЮЗ + 1 ШТАТ	4	400	17,6
T2	100 за день	1 СЛУЖАЩИЙ + 1 ЗАВОД + 1 ПРОФСОЮЗ + 1 ШТАТ	4	400	17,6
T3	200К за день	1 СЛУЖАЩИЙ + 1 РАБ-ЗАДАНИЕ + 1 ПРОИЗВ-УЧАСТОК + 1 ВИД-РАБОТЫ	4	800К	27000
T4	120К за год	1 СЛУЖАЩИЙ	1	500[2]	48
R1	Поквартально	5 ПРОФСОЮЗ + 100К СЛУЖАЩИЙ	100К	1,7К	160
R2	Ежегодно	40 ШТАТ + 100К СЛУЖАЩИЙ	100К	0,4К	40
R3	Ежемесячно	40 ШТАТ + 50 ЗАВОД	90	4,5	0,1
R4	Ежемесячно	50 ЗАВОД + 50 × 20 ВИД-РАБОТЫ	1050	53	1,4
R5	Ежедневно	50 ЗАВОД + 100К СЛУЖАЩИЙ + 200К РАБ-ЗАДАНИЕ + 200К ВИД-РАБОТЫ +200К ПРОИЗВ-УЧАСТОК	700К	700К	17600
R6	Ежемесячно	50 ЗАВОД + 100К СЛУЖАЩИЙ	100К	5К	480
R7	1000 за день	1 СЛУЖАЩИЙ	1К	1К	96

[1] Количество $LRA = 1{,}51 \times 10^6$ за день; объем передачи $= 45{,}5 \times 10^6$ байт за день; область памяти данных $= 11{,}63 \times 10^6$ байт; область памяти указателей (минимум) $= 4{,}01 \times 10^6$ байт; общий объем памяти $= 15{,}64 \times 10^6$ байт.

[2] Из расчета 240 рабочих дней в году.

ре (рис. 8.5) для каждого приложения приводит к появлению суммарных величин, представленных в табл. 8.1. Каждое приложение характеризуется частотой и количеством обращений к логическим записям LRA. Чтобы выделить наиболее активные составляющие, LRA разбивается на отдельные компоненты, соответствующие типам записей. Кроме того, это позволяет легко рассчитать объем передачи, взяв общую сумму для записей в соответствующих группах (без учета длин указателей) и умножив результат на частоту обработки приложения.

Основное допущение, сделанное относительно LRA, состоит в том, что приложения типа «Найти единственную запись» будут выполняться с помощью метода доступа, использующего индексный метод поиска или алгоритм хеширования (CALC), детали которых должны быть проанализированы на этапе физического проектирования. Кроме того, приложения T1 и T2, реализуемые в режиме пакетной обработки, пока не рассматриваются. Первое предположение дает фиксированную оценку нижней границы LRA и упрощает анализ. Например, запись или выборка записи с помощью хеширования в идеальном случае требуют только одного LRA; возможность появления синонимов и переполнения не рассматриваются вплоть до физического проектирования. Однако, если не сделать предположения относительно прямого доступа, поиск некоторой целевой записи может стать более длительным. Для расчета значений средней продолжительности поиска должен использоваться метод, изложенный в разд. 7.3.1. Отметим, что транзакции T1, T2 и T3, приведенные в табл. 8.1, содержат операции включения и удаления. Эти операции обычно реализуются так: сначала выбирается экземпляр исходной записи, связанный с включаемым или удаляемым экземпляром порожденной записи, и, возможно, модифицируются указатели исходной записи, затем отыскиваются предшествующие и/или последующие реализации порожденной записи и модифицируются их указатели. Конечно, имеется много способов реализации связи «исходный — порожденный» и их окончательную оценку надо отложить до этапа физического проектирования (этап 4). На этом уровне анализа предполагается, что исходная запись имеет непосредственные ссылки на порожденную.

Шаг 3-4. Усовершенствование схемы с целью повышения эффективности

Из табл. 8.1 видно, что если исходить из оценок значений количества LRA за день, то доминирующими являются приложения T3 и R5. Другие отчеты не столь важны для минимизации количества LRA или объема передачи. Однако такие факторы, как приоритет или ограничения на время обработки, могут вызвать реорганизацию доминирующего списка. Например, в эту категорию легко мог бы попасть запрос R7 из-за необходимости обработки его в оперативном режиме.

Доминирующие приложения — транзакция T3 и отчет R5 — могут быть выполнены более эффективно путем перемещения данных ПРОИЗВ-УЧАСТОК в записи РАБ-ЗАДАНИЕ. Это изменение схемы вполне возможно потому, что запись ПРОИЗВ-УЧАСТОК настолько мала, что можно добиться очевидного сокращения объема памяти для указателей путем ее

объединения с записью РАБ-ЗАДАНИЕ. Сокращается также объем передачи и значение LRA, поскольку к записям ПРОИЗВ-УЧАСТОК теперь не будет отдельных обращений. Однако потребуется немного больший объем памяти под данные, так как номер производственного участка (НОМЕР-ПРОИЗВ-УЧАСТ) является теперь избыточным в записях РАБ-ЗАДАНИЕ.

Таблица 8.2. **Количество обращений к логическим записям для различных приложений (окончательный вариант структуры логической базы данных (рис. 8.7))** [1])

Приложения	Частота	LRA	Количество LRA в приложении	Количество LRA за день (направление изменений)	Объем передачи за день, К
T1	100 за день	1 ЗАВОД + 1 СЛУЖАЩИЙ + 1 ДАННЫЕ-О-СЛУЖАЩЕМ	3	300 (↓)	16,6
T2	100 за день	1 ЗАВОД + 1 СЛУЖАЩИЙ + 1 ДАННЫЕ-О-СЛУЖАЩЕМ	3	300 (↓)	16,6
T3	200К за день	1 СЛУЖАЩИЙ + 1 РАБ-ЗАДАНИЕ + 1 ВИД-РАБОТЫ	3	600 К (↓)	22600
T4	120К за год	1 СЛУЖАЩИЙ	1	500	37
R1	Ежеквартально	100К СЛУЖАЩИЙ + 100К ДАННЫЕ-О-СЛУЖ	200К	3,4К (↑)	176,7
R2	Ежегодно	50 ЗАВОД + 100К СЛУЖАЩИЙ	100К	0,4К	30,8
R3	Ежемесячно	50 ЗАВОД	50	2,5 (↓)	0,1
R4	Ежемесячно	50 ЗАВОД + 50 × 20 ВИД-РАБОТЫ	1050	53	1,4
R5	Ежедневно	50 ЗАВОД + 100К СЛУЖАЩИЙ + 200К РАБ-ЗАДАНИЕ + 200К ВИД-РАБОТЫ	500К	500К (↓)	15400
R6	Ежемесячно	50 ЗАВОД + 100К СЛУЖАЩИЙ + 100К ДАННЫЕ-О-СЛУЖАЩЕМ	200К	10К (↑)	530
R7	1000 за день	1 СЛУЖАЩИЙ	1К	1К	74

[1]) Количество *LRA* = $1{,}11 \times 10^6$ за день; объем передачи—$38{,}9 \times 10^6$ байт за день; область памяти данных = $12{,}43 \times 10^6$ байт; область памяти указателей (минимум) = $3{,}21 \times 10^6$ байт; общий объем памяти = $15{,}64 \times 10^6$ байт.

Из сравнения табл. 8.1 и табл. 8.2 видно, как сокращается количество LRA для данного изменения схемы. Стрелки в пя-

той колонке показывают увеличение или уменьшение количества LRA по отношению к информационной структуре, приведенной на рис. 8.5. Объем передач легко получить, используя значения LRA из табл. 8.2 и размеры записей, указанные в таблице рис. 8.6.

Еще одно изменение структуры, которое приводит к сокращению объема передач в приложениях T3 и R5, заключается в разбиении записи СЛУЖАЩИЙ на два типа записей, один из которых содержит активные данные (СЛУЖАЩИЙ), а другой — довольно неактивные данные (ДАННЫЕ-О-СЛУЖАЩЕМ). Неактивные данные будут включать адрес служащего, дату рождения и дату найма на работу. Здесь необходим компромисс между увеличением количества LRA для отчетов R1 и R6, а также размера области памяти под указатели и не столь значительным сокращением затрат в доминирующих приложениях (см. табл. 8.2).

Существуют и менее значительные усовершенствования схемы. Например, перемещение данных ШТАТ в записи ЗАВОД сократит требуемый объем памяти для указателей, а также уменьшит количество LRA для отчета R3 на 40 единиц, при этом потребуется немного больше памяти для данных, так как указание 10 из 40 штатов в записях ЗАВОД будет избыточным; однако эти потери с избытком компенсируются уменьшением области памяти для указателей, необходимых для хранения отдельных типов записей в виде наборов данных CODASYL.

Размер области указателей, количество LRA и объем передачи в транзакциях T1 и T2 также можно сократить перемещением данных ПРОФСОЮЗ в записи СЛУЖАЩИЙ, но это приведет к значительному увеличению затрат памяти из-за необходимости хранения избыточных данных о названии профессионального союза. 20 символов имени союза, повторяющиеся в 100K экземплярах записей СЛУЖАЩИЙ, дают в результате 2 Мбайт избыточной памяти. Этот объем можно уменьшить до 200 кбайт, если сократить с помощью сжатия данных название профессионального союза до двух символов. Такой же подход может быть использован для сжатия названия завода, на которое также отведено 20 символов, однако это даст не такой значительный эффект, как в случае с записями СЛУЖАЩИЙ.

Окончательная схема (типа CODASYL), полученная в результате пересмотра проектных решений путем расщепления и объединения типов записей и сжатия данных, показана на рис. 8.7. Мы уменьшили сложность структуры до пяти типов записей, пяти типов наборов и двух сингулярных типов наборов для обеспечения входа в базу данных. Для сравнения с результатами табл. 8.1 новые оценки производительности приведены в нижней части табл. 8.2. Было получено 20—30 %-ное сокращение

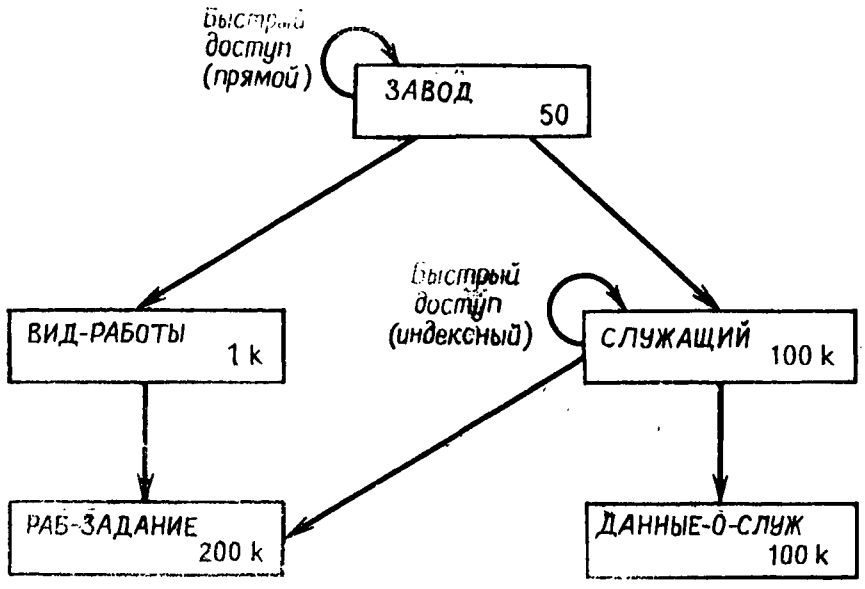


Рис. 8.7. Окончательный вариант логической структуры базы данных.

количества LRA и объема передачи при увеличении объема памяти для хранения данных только на 7 %. Если к объему памяти для хранения данных каждой из схем на рис. 8.5 (табл. 8.1) и рис. 8.7 (табл. 8.2) прибавить соответствующий минимальный объем памяти для хранения указателей (один указатель типа «следующий» для каждого экземпляра записи и один порожденный указатель для каждого экземпляра исходной записи), то общий объем памяти для обеих схем будет одинаковым.

Следует отметить, что принятое в схеме на рис. 8.6 первоначальное определение агрегированных типов элементов остается неизменным и в конечной схеме. В качестве примера использования предложенного подхода рассмотрим элемент ОБЩЕЕ-КОЛИЧ-ЧАСОВ-ПРОРАБ-СЛУЖ-В-МЕСЯЦ в записи ЗАВОД, который требуется для отчета R4. В данной реализации удобно, чтобы для приложения R5 каким-либо образом обновлялись записи ЗАВОД. Альтернативное решение заключается в исключении элемента ОБЩЕЕ-КОЛИЧЕСТВО-ПРОРАБ-СЛУЖ-ЧАСОВ-В-МЕСЯЦ из записей ЗАВОД и вычислении его каждый раз, когда выдается R4 (т. е. ежемесячно), суммируя значения ОБЩЕЕ-КОЛИЧ-ПРОРАБ-СЛУЖ-ЧАСОВ-В-МЕСЯЦ из каждого экземпляра записи СЛУЖАЩИЙ. Это потребует 100К дополнительных обращений к записи СЛУЖАЩИЙ при каждом получении отчета R4. Это не слишком серьезная проблема, поскольку R4 является лишь ежемесячным отчетом, но первая альтернатива является гораздо более эффективной.

8.4. Обзор основных результатов проектирования

В приведенном примере рассмотрены многие важные результаты разработки приложений, с которыми сталкивается разработчик базы данных. Они включают следующее:

Шаг 3-1

- Построение информационной структуры для отдельных обрабатываемых приложений.
- Определение предполагаемых транзакций.
- Верификация предположений о том, какие данные могут быть агрегированы и что необходимо сохранить в базе данных после агрегации.
- Согласование различных аспектов обработки; объединение данных и устранение избыточных связей.
- Принятие решения о том, можно ли исключить некоторые или все неиспользуемые в заданных приложениях объекты.
- Идентификация возможных первичных ключей.

Шаг 3-2

- Преобразование усовершенствованной информационной структуры в СУБД-ориентированную схему.
- Использование контролируемой избыточности для того, чтобы удовлетворить ограничениям СУБД.
- Определение первоначального содержимого записей.
- Определение первоначальных точек входа на основании требований обработки.

Шаг 3-3

- Выбор критериев оценки эффективности логической структуры или схемы базы данных.

Шаг 3-4

- Анализ специфики совместной обработки приложений с целью оптимизации общего времени обработки; определение приложений, эффективность обработки которых целесообразно повышать в первую очередь; сведение к минимуму количества повторных обращений к уже выбранной записи.
- Определение сущностей, объединение которых повышает эффективность обработки.
- Определение сущностей, которые целесообразно разделить, когда элементы внутри записей имеют различную активность.
- Определение элементов схемы, для которых целесообразно применение сжатия данных для сокращения объема памяти.

• Определение элементов схемы, где можно сократить путь доступа путем лучшего выбора точек входа, указателей, упорядочения записей (на основании минимизации количества LRA для данного шага).

Другие аспекты проектирования, встречающиеся на этапах формулирования требований физического проектирования, обсуждаются в [206, 271]. Существует ряд проблем, которые встают перед проектировщиком на логическом уровне проектирования, обычно на Шаге 3-4; некоторые из них можно классифицировать как чисто логические, а другие же окончательно решаются только на этапе физического проектирования. Здесь таких разграничений делаться не будет, но хотелось отметить, что некоторые решения, касающиеся физической организации данных, могут быть приняты уже на логическом уровне проектирования (по крайней мере на Этапе 3), а затем проверены и скорректированы на этапе физического проектирования. К таким вопросам могут быть отнесены следующие:

Нормализация записи. Объединение в один тип записи нескольких сущностей может привести к появлению ненормализованных записей, и, следовательно, при проведении коррекции могут возникнуть аномалии, аналогичные тем, которые имеют место в отношениях первой, второй и даже третьей нормальной формы [85, 97]. Разработчик должен найти компромисс между эффективностью системы в текущий момент и потенциальным снижением эффективности, которое может возникнуть в результате проведения будущих коррекций данных.

Декомпозиция базы данных. После объединения представлений многих пользователей в интегрированную структуру базы данных, учитывающую требование концептуальной модели и специфику обработки приложений, может оказаться целесообразным сформировать несколько баз данных (т. е. выбрать наиболее эффективную конфигурацию баз данных или файлов). Это часто вызывается большим диапазоном уровней изменчивости данных и частот поиска или наличием значительной доли «прошлых» данных. Декомпозиция базы данных необходима, во-первых, для более эффективного использования ресурсов ЭВМ (время и объем памяти), а во-вторых, для обеспечения требований индивидуальных пользователей (обычно требований безопасности данных). В рассмотренном выше примере следует изучить возможность использования нескольких баз данных, если потребуется обеспечить: во-первых, ежедневную обработку сведений о начале и конце работы служащего и, во-вторых, сохранение ежедневных НОРМО-ЧАСОВ после того, как в конце каждого дня проведено суммирование этих данных для отчета R5, и, в-третьих, сохранение предыстории заработной платы по каждому служащему.

Проектирование программ. Спецификации прикладных программ должны быть получены как результат этапа проектирования реализации. Данное требование должно выполняться безоговорочно, так как на Этапе 3 уже был завершен подробный анализ каждого приложения и возможных взаимосвязей между ними, проводимый с целью оценки количества обращений к логическим записям. Например, в то время когда осуществляется получение отчета R5, можно проводить ежедневную коррекцию ОБЩЕЕ-КОЛИЧ-ЧАСОВ-ПРОРАБ-СЛУЖ-В-МЕСЯЦ в записи ЗАВОД; аналогичная операция требуется для своевременного составления отчета R5. В противном случае придется разрабатывать отдельное приложение. Здесь был бы очень полезен совет прикладным программистам по выбору эффективных путей доступа.

Пакетная обработка. В нашем примере транзакции T1 и T2 являются типичными операциями коррекции, которые надо выполнять в пакетном режиме. Ежедневно требуется приблизительно 200 коррекций, но скорректированные данные не потребуются немедленно, если новый служащий не начнет работать в день его найма. В противном случае транзакция коррекции T1 должна быть проведена прежде, чем служащий выполнит какое-либо рабочее задание (т. е. T3). Реальная эффективность применения пакетной обработки не может быть определена до тех пор, пока на этапе физического проектирования не будет рассчитано время ввода/вывода и время работы центрального процессора.

Выбор аппаратных и программных средств. Обычно проектировщики баз данных работают в условиях ограничений, налагаемых средствами аппаратного и программного обеспечения (СУБД/ОС). Иногда процесс выбора аппаратных и программных средств является частью проблемы проектирования системы базы данных. Проектирование базы данных на Этапах 1, 2 и Шаге 3-1 выполняется независимо от СУБД. После Шага 3-1 должны быть оценены возможные СУБД для использования их на Шагах 3-2, 3-3, 3-4 и Этапе 4 в качестве СУБД. Исключение особенно неэффективных СУБД (исходя из задач проектирования) осуществляется обычно в конце Этапа 3. В это же время рассматриваются и другие вопросы, связанные с потерей эффективности. Окончательное решение может быть принято не только с учетом требований эффективности, но и последствий для организации в случае ее снижения [82]. Выбор аппаратного и системного программного обеспечения следует осуществлять так, чтобы обеспечить их совместимость с выбранной СУБД, учитывая при этом наличие в системе обрабатываемых приложений, не использующих СУБД.

Часть IV

Физическое проектирование

Глава 9. Принципы проектирования физической базы данных. Основные концепции

9.1. Введение в физическое проектирование

Третьим и самым нижним уровнем представления базы данных является физический уровень. Физическая организация данных оказывает основное влияние на эксплуатационные характеристики проектируемой базы, так как именно на этом уровне осуществляется ее привязка к физической памяти. Эксплуатационные характеристики в первую очередь измеряются стоимостью выполнения ежедневных планируемых работ по загрузке базы, а также затратами на проведение незапланированных коррекций и в гораздо меньшей степени зависят от долгосрочных информационных требований, которые важны на ранних стадиях проектирования. На этапе физического проектирования улучшение эксплуатационных характеристик достаточно легко измерить (например, из-за улучшения физической организации возможно сокращение эксплуатационных затрат в $2 \div 3$ раза).

В этой главе будут рассматриваться вопросы проектирования физической структуры широкомасштабных (интегрированных) систем баз данных с различными типами записей. Однако можно показать, что методика проектирования баз данных с единственным типом записей является тем фундаментальным инструментом, который необходим для анализа сложных структур. Во многих случаях, особенно если учитывать методы доступа, физические структуры с различными типами записей можно рассматривать как объединение структур с единственным типом записей и соответствующих механизмов поиска.

Основная единица данных в физических структурах — хранимая запись. *Хранимая запись* представляет собой совокупность связанных элементов данных (атрибутов), которая соответствует одной или нескольким логическим записям; она содержит все необходимые указатели, длину записи и некоторые дополнительные данные, а также схему кодирования для символьного представления. Иными словами, хранимая запись расширяет понятие логической записи вплоть до учета действительных форматов памяти. *Файл* — это множество аналогично построенных хранимых записей одного типа. Хранимые в фай-

ле записи могут иметь одинаковую или разную длину. *Физическая база данных* представляет собой совокупность совместно хранимых взаимосвязанных данных, состоящую из одного или нескольких типов хранимых записей. Файлы могут рассматриваться как вырожденная форма базы данных, содержащей хранимые записи только одного типа. В этой и последующих главах термин «файл» используется для описания физической базы данных с одним типом записей. *Организация файла*, или *структура файла*, — это представление хранимых записей, составляющих файл, отображающее связи внутри записей (формат записи), логическое и физическое упорядочение, возможные пути доступа, такие, как индексы, и распределение хранимых записей по физическим устройствам. Аналогично используется термин *организация физической базы данных*, или *структура физической базы данных*, чтобы представить формат хранимой записи, логическое и физическое упорядочение, возможные пути доступа, а также распределение базы данных с различными типами данных по устройствам. Термин *структура памяти* иногда может совпадать по значению с понятиями структуры файла или структуры физической базы данных.

Эти фундаментальные определения позволяют перейти к вопросу: «Что такое проектирование файла?» В общем случае *проектирование физической базы данных* (или физическое проектирование) — это процесс создания эффективной реализуемой структуры физической базы данных по заданной логической структуре, полученной исходя из требований пользователя к информации. Физическое проектирование часто предполагает удовлетворение определенным эксплуатационным ограничениям, таким, как требования к объему памяти и к распределению времени отклика. *Проектирование файла* — это процесс создания реализуемой структуры файла, удовлетворяющей различным эксплуатационным требованиям, специфицированным коллективом пользователей. Понятие структуры физической базы данных включает: формат хранимой записи, структура путей доступа и размещение записей на физических устройствах. Эти и некоторые другие понятия будут использованы ниже при описании этапов процесса физического проектирования.

9.1.1. Процесс физического проектирования

Как и ранее при рассмотрении начальных фаз процесса проектирования баз данных, фаза физического проектирования также может быть разбита на четкие шаги, основываясь на группах логически связанных проектных решений. Однако из-за очень сильной взаимосвязи между этими группами проект-

ных решений соответствующая упорядоченность шагов определяется с учетом конкретных обстоятельств. Практический опыт показал, что для данной проблемы проектирования нельзя указать определенно ни исходной точки, ни порядка шагов. С другой стороны, физическое проектирование может рассматриваться как итеративный процесс получения начального решения и последующего его улучшения. Каждый из предлагаемых шагов необходимо выполнить несколько раз, причем каждый последующий анализ должен выполняться быстрее, так как процедура решения уже известна, а количество зафиксированных параметров возрастает с каждой итерацией. Чаще всего достаточно двух или трех попыток, чтобы достичь сходимости решения. Относительная важность каждого шага в смысле его влияния на производительность системы становится очевидной в результате опыта и тщательного документирования всего процесса проектирования.

Из следующих пяти шагов проектирования первые три касаются принятия основных решений по физическому проектированию базы данных. Остальные два шага включают учет ограничений и проектирование программ. В последующих главах первые 3 этапа будут рассмотрены достаточно подробно.

Шаг 4-1. Проектирование формата хранимой записи

Учитывая, что структура логической записи была выбрана ранее, этот процесс состоит в форматировании хранимой записи с учетом характеристик типов элементов данных, распределения их значений и специфики их использования в различных приложениях. На этом шаге принимаются решения относительно избыточности данных и их сжатия на основании анализа явно хранимых значений данных.

Можно выделить определенные элементы данных, обращение к которым осуществляется гораздо чаще, чем к другим, но каждый раз, когда необходимо выбрать определенную часть данных, приходится обращаться как к хранимой записи в целом, так и ко всем записям в физическом блоке. *Секционирование записей* заключается в распределении элементов данных по отдельным физическим устройствам одного или различных типов или по отдельным экстентам одного и того же устройства так, чтобы для заданного множества приложений минимизировать общие затраты на осуществление доступа к информации. Считается, что если элементы данных, относящиеся к одной и той же сущности, логически связаны, то и физически, когда это необходимо, они должны отыскиваться вместе. Определим понятие *экстент* как последовательную область физической памяти на определенном устройстве.

Шаг 4-2. Кластеризация хранимых данных

Одним из наиболее важных вопросов, который должен быть решен при физическом проектировании, состоит в том, как физически распределить хранимые записи по экстентам. *Кластеризацией записи* называется такое объединение записей различного типа в физические группы, которое позволяет, как можно эффективнее использовать преимущество последовательного размещения данных. Для того чтобы новое размещение записей не привело к увеличению времени обработки, при кластеризации необходимо учитывать структуру путей доступа.

Выбор размера физического блока зависит от кластеризации записи и ее секционирования. В какой-то мере на него влияют размер хранимой записи и характеристики физических запоминающих устройств. Кроме того, обычно предполагается, что большие блоки обрабатываются последовательно, а малые по размерам блоки — в произвольном порядке. Таким образом, хотя размер блока тесно связан с кластеризацией, он также сильно зависит от пути доступа, типа приложения и характеристик аппаратного обеспечения. Следовательно, на отдельных итерациях процесса проектирования размер блока может в значительной степени изменяться.

Шаг 4-3. Проектирование метода доступа

Метод доступа обеспечивает возможность запоминания и выборки данных, хранимых на физических устройствах (обычно вторичной памяти). Можно выделить два важных компонента метода доступа: структура памяти и механизм поиска. Структура памяти определяет ограничения на возможные пути доступа с использованием индексов и хранимых записей, а механизмы поиска определяют пути доступа, которые следует выбрать для конкретного приложения. Здесь не рассматриваются вопросы проектирования внутренней структуры и секционирование записи; с другой стороны, на этом шаге важно осуществить проектирование индексов и связей между записями.

Выше мы определили, что хранимая запись состоит из набора атрибутов. *Атрибут* может представлять собой первичный ключ, вторичный ключ или неключевой элемент. *Первичный ключ* однозначно определяет запись, т. е. в базе данных не может быть другого ключа с таким же значением. *Вторичный ключ* — это атрибут, используемый в качестве индекса записей, но он не идентифицирует уникально эти записи (т. е. одно и то же значение ключа может появляться более чем в одном экземпляре записи). *Неключевым атрибутом* является атри-

бут, который не используется как первичный или вторичный ключ для индексации или другого механизма поиска записей.

Проектирование метода доступа состоит в выборе структуры первичного и вторичного путей доступа. Первичные пути доступа используются при первоначальной загрузке и размещении записей, что обычно влечет за собой поиск по первичным ключам. Подобным образом первоначально проектируются индивидуальные файлы, чтобы доминирующие приложения обрабатывались с наибольшей эффективностью. По той же причине физическая база данных может потребовать использования нескольких первичных путей доступа. Вторичные пути доступа включают в себя межфайловые связи и альтернативные точки входа для доступа к хранимым записям с помощью индексов вторичных ключей. Использование вторичных ключей может значительно сократить время доступа, но при этом возрастают накладные расходы на память и поддержание индексов. Эти вопросы детально рассматриваются в гл. 12—16.

Шаг 4-4. Вопросы целостности и безопасности данных

Как и на этапе проектирования реализации, на данном шаге должны быть проанализированы в комплексе требования целостности, безопасности и эффективности функционирования базы данных.

Шаг 4-5. Проектирование программ

Физическая независимость данных, если она достигнута, устраняет необходимость модификации прикладных программ. Для организации доступа к базе данных должны использоваться стандартные средства СУБД, а оптимизацию запросов или транзакций обновления следует выполнять на уровне системного программного обеспечения. Поэтому проектирование прикладных программ следует заканчивать на этапе логического проектирования базы данных. Например, программу, которая базируется на навигационном методе доступа, как в системах IMS и CODASYL, придется значительно модифицировать, если точки входа для доступа к данным были определены впервые на этапе физического проектирования базы данных.

При проектировании приходится принимать ряд проектных решений (многие из которых являются системно-зависимыми), например выбор размера буфера, степени избыточности хранимых записей и дифференциальных файлов. Эти вопросы оказываются в равной степени важны и трудны для анализа как при проектировании структуры физической базы данных, так и при проектировании файлов.

9.1.2. Компоненты этапа физического проектирования

Компоненты физического проектирования в основном те же, что и при проектировании файла. Однако многие решения, которые принимаются при проектировании файлов, значительно проще, чем при создании базы данных с различными типами записей. На рис. 9.1 представлены исходные данные и результаты этапа физического проектирования. Логическая структура базы данных, полученная на этапе проектирования реализации, определяет основу, на которой строит свою работу проектировщик физической базы данных. Логическая структура базы должна оставаться неизменной в течение процесса физического проектирования. Корректировать логическую структуру на этом этапе приходится только в случае изменения требований обработки данных или если в процессе физического проектирования обнаружится низкая эффективность принятых ранее решений. С другой стороны, физическое проектирование можно рассматривать как дальнейшее совершенствование этой структуры. В общем случае на этапе физического проектирования окончательно выбирается структура путей доступа и распределение записей, а также рассматриваются новые параметры. Так же как и при проектировании реализации, учитываются характеристики, описывающие объем данных, частоту обработки приложений и последовательность операций в прикладных программах. Кроме того, вводятся новые параметры, характеризующие методы доступа СУБД и операционной системы (ОС), емкость физических запоминающих устройств и времен-

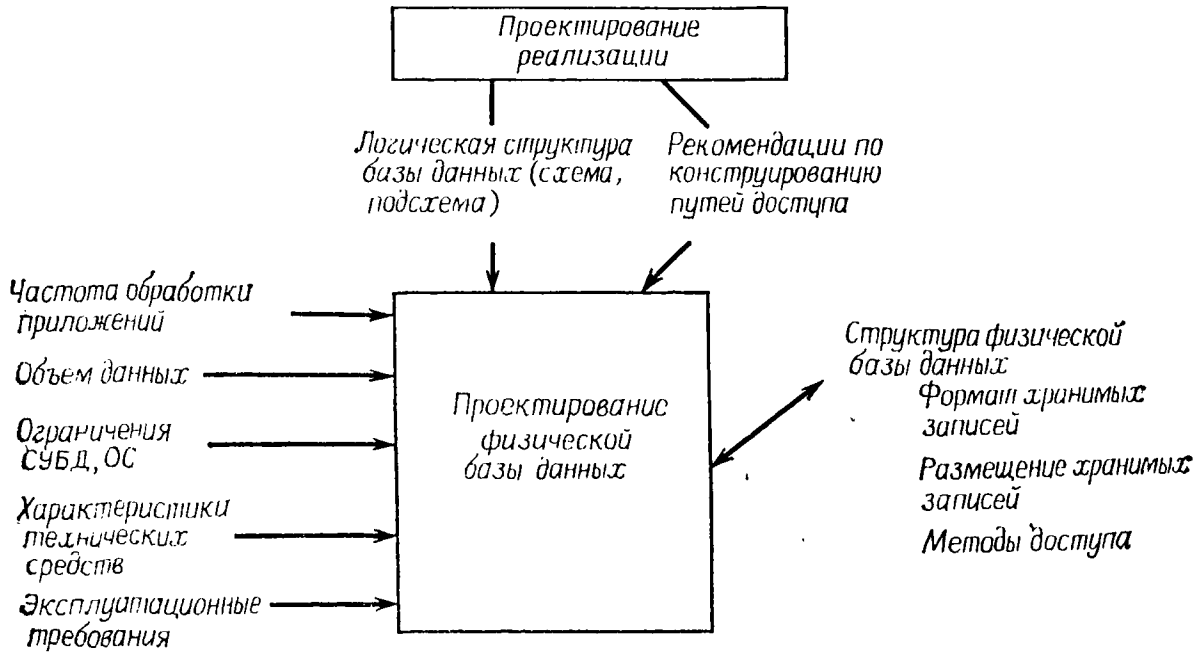


Рис. 9.1. Исходные данные и результаты физического проектирования.

ные характеристики, а также все эксплуатационные требования. Эксплуатационные требования обычно содержат ограничения целостности, безопасности, ограничения на время отклика в установившемся режиме работы и по возможности с учетом предполагаемого ее развития. Детализация этой группы параметров будет осуществляться по мере необходимости в следующих главах.

Очевидными компонентами результирующей физической структуры базы данных являются: формат хранимой записи, спецификации размещения хранимой записи и методы доступа. Эти спецификации должны удовлетворять всем эксплуатационным требованиям и ограничениям, налагаемым техническими и программными средствами системы. В процессе проектирования сначала необходимо получить хотя бы одно решение, удовлетворяющее всем ограничениям, а затем перейти к вопросу повышения производительности системы. Благодаря возможности изменять параметры физического проектирования обычно существует достаточно широкая область допустимых модификаций. Оставшаяся часть этой главы посвящена основным концепциям повышения производительности, в последующих главах этот вопрос будет рассматриваться в дополнение к описанию способов отыскания возможных физических структур с учетом ограничений среды.

9.1.3. Характеристики производительности

Выбор критерия оценки производительности очень важен при физическом проектировании. От этого зависит не только выбор конкретного решения, но и методы, которые при этом будут использоваться. Предлагаемая здесь нисходящая методология позволяет описать как общие затраты при проектировании базы данных, так и их составляющие, которыми разработчик имеет возможность управлять в пределах заданных ограничений в области допустимых модификаций. Мы не предлагаем какой-либо один доминирующий или наиболее желательный критерий; наоборот, целесообразно использовать группу критериев, которые отражают различные точки зрения на понятие производительности. Может потребоваться проведение специального сравнительного анализа индивидуальных оценок (иногда противоречивых), и только человек, которому поручено принимать окончательное решение, может учесть такие качественные факторы, как общая политика в отношении приобретения дополнительных емкостей памяти или дополнительных расходов за счет возможного увеличения времени отклика и т. п. Подход со многими критериями обеспечивает проектировщику достаточную гибкость как на этапе выработки

первоначального решения, так и на дальнейших стадиях проектирования.

Предположим, что производительность системы баз данных оценивается в терминах затрат. В разное время затраты могут быть заданы в единицах времени, объема памяти или, возможно, денежных величинах. Возвращаясь к обсуждению жизненного цикла системы баз данных, можно выразить общие затраты следующими характеристиками:

- затраты на планирование;
- затраты на проектирование: программ и баз данных;
- затраты на реализацию и тестирование программ и баз данных;
- эксплуатационные затраты: обслуживание пользователей и вычислительные ресурсы;
- затраты сопровождения: устранение программных ошибок, затраты на обеспечение целостности данных.

Оценка затрат на планирование, проектирование, реализацию и сопровождение общесистемного программного обеспечения хорошо известна и не будет здесь рассматриваться. Основная проблема, которую должен рассматривать проектировщик физической базы данных, состоит в том, как минимизировать настоящие и будущие эксплуатационные затраты на вычислительные ресурсы и удовлетворение потребностей пользователей (таких, как своевременность представления информации, достоверность данных).

Рассмотрим подробнее затраты на эксплуатацию. Предполагая, что перепроектирование схемы базы данных и прикладных программ рассматривается как часть этапов (пере)проектирования и сопровождения в жизненном цикле системы и, следовательно, анализируется на этапе концептуального проектирования или на этапе проектирования реализации, то эксплуатационные затраты, специфические для этапа физического проектирования, могут быть разбиты на следующие группы: время отклика; затраты на обновление; затраты на генерацию отчета; частота и затраты на реорганизацию базы данных; затраты на основную память; затраты на внешнюю память. Каждая из этих составляющих важна для проектировщика, но их приоритеты могут изменяться в зависимости от конкретных условий работы. Проанализируем указанные компоненты, используя в качестве основных единиц измерения затрат байты, секунды и т. д.

Время отклика

Время отклика определяется как интервал времени между инициализацией запроса и началом отображения результата запроса. Время отклика складывается из времени работы цен-

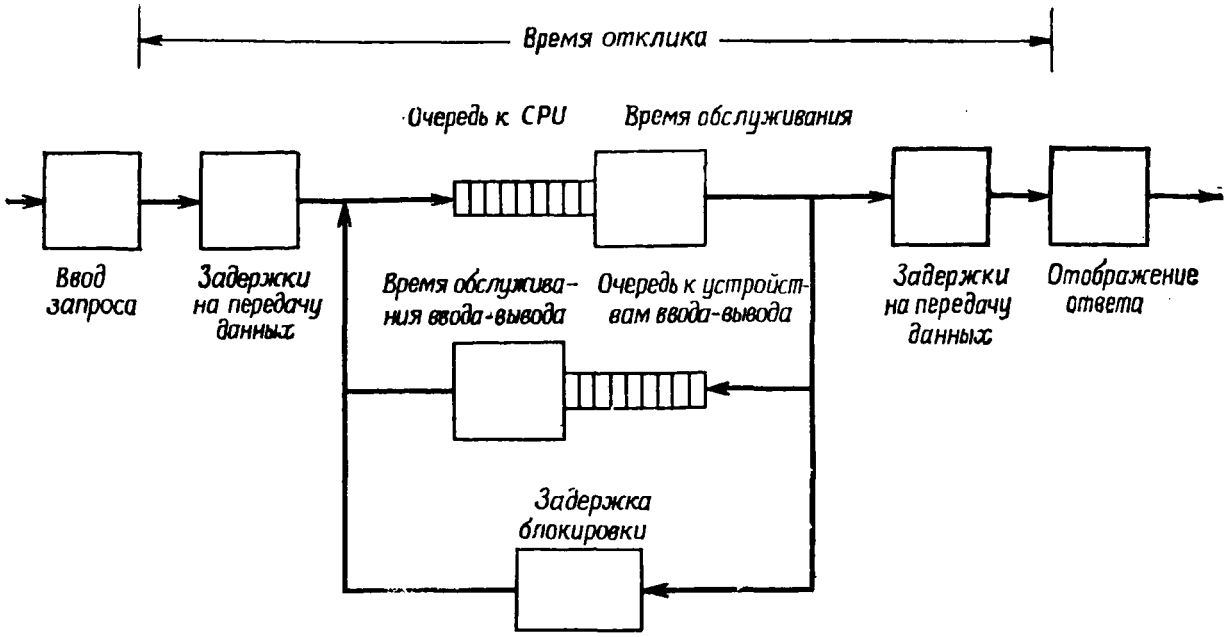


Рис. 9.2. Составляющие времени отклика на запрос.

трального процессора (CPU) по обслуживанию запроса, времени ожидания в очереди на обслуживание CPU, времени ожидания ввода-вывода, времени обслуживания ввода-вывода, задержки блокировки и задержек передачи данных (рис. 9.2). Время отклика не включает время работы пользователя по набору запроса и время вывода ответа. Это упрощает анализ и позволяет не рассматривать разнообразные методы ввода и отображения данных. (Этот вопрос не имеет отношения к проблеме проектирования базы данных.)

Время ожидания в очереди к CPU и в очереди ввода-вывода сильно зависит от конкретного набора заданий, находящихся в вычислительной системе. Если система не ориентирована специально на работу с базами данных, то эти задержки полностью выходят из-под контроля проектировщика базы данных. Если вычислительная система специально спроектирована для работы с базами данных, то можно уменьшить эффективное время обслуживания за счет организации параллелизма в работе CPU и устройств ввода-вывода или за счет специального проектирования процессов обработки запросов пользователей. Обычно время ожидания в очереди с учетом заданных ресурсов и других факторов является функцией времени обслуживания. В централизованной системе баз данных задержки на передачу данных не зависят от структуры базы данных и не могут быть сокращены за счет лучшего проектирования. Однако это не справедливо для систем с распределенными ба-

зами данных, где существуют различные варианты пересылки промежуточных данных, а также конечных результатов через сеть.

Задержки блокировки, обусловленные необходимостью обеспечения целостности базы данных, зависят от принятых на ранних стадиях проектирования решений относительно программного обеспечения СУБД. Организация блокировки на высоком уровне, например на уровне базы данных, приводит к большим внутренним очередям при выборке данных, но при этом требуется меньше времени CPU и создаются более короткие очереди к физическим ресурсам. Блокировка на уровне элементов данных приводит к противоположной ситуации: короткие внутренние очереди, но большее время CPU и потенциально более длинные очереди к физическим ресурсам. В настоящее время наблюдается тенденция к осуществлению блокировки на низком уровне, так как при этом сокращается общее время отклика (т. е. время ожидания во внутренней очереди становится основной составляющей полного времени отклика). Если блокировка осуществляется на уровне элементов данных, проектировщик базы не может существенно влиять на задержку блокировки. При более высоком уровне блокировки можно так распределить записи или базы данных, чтобы обеспечить возможность их параллельной обработки.

Время обслуживания CPU состоит из нескольких элементов, связанных с инициализацией и завершением вызовов ввода-вывода, работой подпрограмм СУБД и ОС, обеспечивающих выборку и обновление данных, и обработкой записей, выполняемой прикладными программами пользователей. Проектировщик базы данных не имеет возможности управлять временем работы какой-либо подпрограммы после ее вызова для выполнения. Однако имеется некоторая возможность управлять количеством обращений к системным подпрограммам путем эффективной кластеризации данных в записи, блоки и экстенты. Надлежащее проектирование прикладных программ наряду с проектированием структуры базы данных способствует уменьшению времени обслуживания CPU.

Время ввода-вывода может быть сокращено за счет правильного выбора путей доступа, а также рационального размещения и сжатия данных. Как только будет достигнуто сокращение времени ввода-вывода, то станет ощутимым сокращение времени обслуживания CPU и, следовательно, потенциальное сокращение как задержек в очередях на ввод-вывод, так и в очередях на обслуживание центральным процессором. В результате при оценке эффективности работы СУБД время ввода-вывода станет основной составляющей времени отклика. Это время меньше зависит от характеристик СУБД и ОС, чем

время обслуживания CPU, и поэтому его можно рассчитать аналитически.

Однако использование времени ввода-вывода, как основной меры производительности работы всей системы, не всегда оправданно. Сокращение времени ввода-вывода, несомненно, приводит к уменьшению времени отклика на запрос. Но этот выигрыш может оказаться незначительным и не оправдывающим затраты на реорганизацию базы данных. Всегда существует большая вероятность того, что устройства ввода-вывода не являются «узким местом», а снижение производительности системы определяется чем-то другим. В таких случаях персонал системных программистов должен выполнить детальное измерение производительности и настройку системы. Для практических целей желательно собрать информацию о возможно большем числе составляющих времени отклика с тем, чтобы при проектировании базы данных можно было внести соответствующие коррективы. В случае отсутствия этих данных время ввода-вывода является наиболее управляемой проектировщиком базы данных мерой производительности и всегда должно учитываться в процессе проектирования.

Затраты на обновление

Затраты на обновление измеряются временем, прошедшим с момента инициализации программы обновления до ее завершения. Так как эта программа может быть частью большой прикладной программы, то время ее выполнения трудно измерить без использования специального программного обеспечения.

Это время складывается из тех же составляющих, что и время отклика, но оно в большей степени зависит от используемых ресурсов. Выполнение коррекции может потребовать дополнительной обработки с целью модификации индексов и таких действий, как перезапись с контролем блоков или файлов. Однако эти дополнительные операции могут быть полностью описаны с помощью основных характеристик, используемых для определения времени отклика.

Затраты на генерацию отчета

Генерация отчета — это специальный вид поискового запроса, который требует вывода большого количества структурированных данных. Время генерации отчета (от момента окончания ввода данных до момента начала отображения ответа) складывается из тех же составляющих, что и время отклика на запрос и время обновления. Однако взаимодействие CPU и ввода-вывода здесь более сложно. Генерация отчета может

быть разбита на четыре основных этапа: выборка данных, реструктурирование данных, сортировка данных и отображение результатов. Более подробный пример приводится в разд. 12.4.

Частота и затраты на реорганизацию

Некоторые методы доступа (например, ISAM), эффективные для статических баз данных, плохо работают в условиях их динамического расширения. В процессе проектирования, с тем чтобы впоследствии не потребовалась частая реорганизация базы данных, необходимо учитывать возможность роста объема данных и увеличение частоты использования базы. Поэтому является важным не только выбор коэффициента загрузки (относительного объема начального заполнения каждого блока хранимыми записями), но и выбор метода доступа. Затраты на реорганизацию аналогичны затратам на обработку запросов и внесение изменений, так как реорганизация представляет собой длинную последовательность таких операций, как выборка и перезапись. Задачей проектировщика является рассмотрение затрат на реорганизацию в зависимости от частоты ее выполнения по сравнению с альтернативным выбором методов доступа и затрат на реорганизацию в случае их использования. Методы доступа с динамической реорганизацией, такие, как VSAM [166, 167], минимизируют затраты, обусловленные периодическим выполнением реорганизации, путем некоторого увеличения системных накладных расходов, связанных с расщеплением блоков и модификацией указателей, выполняемых в текущие периоды работы с базой. Накладные расходы, по-видимому, будут оправданы, если можно избежать длинных цепочек переполнения и уменьшить частоту реорганизации. Для подтверждения этого положения могут быть использованы приведенные ниже методы оценки.

Объем основной памяти

В системах баз данных в основной памяти хранятся программы и данные. В области программ должны быть размещены приложения пользователей, подпрограммы СУБД и ОС. В составе ОС есть специальные средства автоматического размещения программ для различных типов организации памяти (виртуальные структуры, страничная организация и т. д.). В области данных должны быть размещены рабочие области прикладных программ, рабочие области СУБД и ОС, а также буферы для индексов и данных. Разработчик базы данных может управлять только распределением буферов. В случае последовательной обработки данных обычно достаточно двух буферов на процесс. Если применяется последовательная обработка данных, то производительность повышается при увеличе-

нии области памяти, отведенной под буферы. Однако даже при произвольной обработке данных выделение слишком больших объемов буферов для работы с базой данных может снизить производительность работы остальных частей вычислительной системы. Основная память часто является наиболее критичным ресурсом в мультипрограммных системах, поэтому распределение этого ресурса между всеми активными приложениями должно быть выполнено очень тщательно.

Объем внешней памяти

Во внешней памяти хранятся сблокированные данные и индексы. Память в блоке дополнительно расходуется на флаги, счетчики, указатели и резервную область для данных, включаемых в базу после ее загрузки. Проектировщик может влиять на объем внешней памяти, выбирая размеры индексов, коэффициенты загрузки, типы указателей, способ представления и степень избыточности данных. В больших системах стоимость внешней памяти становится менее дорогой в сравнении со стоимостью работы CPU и ввода-вывода и, таким образом, не имеет большого значения, как стоимостный фактор. В малых системах (и в некоторых больших) объем внешней памяти очень ограничен и может быть решающей проблемой при проектировании базы данных. Такое изменение оценки значимости стоимости хранения информации во внешней памяти для различных систем в ближайшее время будет сохраняться и останется одним из важных факторов при проектировании. И даже в тех случаях, когда фактором стоимости хранения информации во внешней памяти можно пренебречь, этого не следует делать без предварительной оценки последствий.

В заключение отметим, что существует много критериев оценки производительности систем баз данных, которые рассматриваются в каждом конкретном случае индивидуально.

- Практически же проектировщик баз данных контролирует только два параметра: *время ввода-вывода*; *объем внешней памяти.*
- В ограниченных пределах при проектировании можно управлять: *задержками блокировки*; *временем CPU*, *объемом основной памяти.*
- Проектировщик практически лишен возможности влиять на: *время ожидания в очереди к CPU и на ввод-вывод*; *задержки передачи данных в централизованных базах данных.*

С помощью этих трех групп параметров разработчик может получить значения времени отклика, времени обновления, времени генерации отчета и времени, затрачиваемого на реорганизацию. Денежное выражение затрат можно получить исходя из стоимости используемых аппаратных ресурсов. Дополнитель-

но нужно учесть в денежном выражении потери, которые несет пользователь из-за задержки во времени отклика и отказов в линиях связи. В дальнейшем будут подробно рассмотрены две наиболее важные меры производительности: время ввода-вывода и объем внешней памяти.

9.2. Расчет производительности

Теперь перейдем к расчету времени ввода-вывода — важной характеристике производительности базы данных. В работе не ставится цель совершенствования моделей методов доступа, размещения записей и т. п. и предлагается рассматривать в качестве базовой модель последовательного и произвольного доступа. В дальнейшем после обсуждения основных концепций рассматриваются более сложные модели. Рассмотрение начинается с идеальной модели, позволяющей рассчитывать время ввода-вывода без учета характеристик физического устройства. Это служит отправной точкой для рассмотрения модели накопителя на магнитном диске с подвижными головками, которая позволяет учитывать в расчетах различные факторы, влияющие на величину времени ввода-вывода.

Будет показано, что производительность физических баз данных может оцениваться количеством обращений к физическим блокам и размером блока в байтах. Методика, позволяющая перейти от этих характеристик к оценкам времени ввода вывода, будет рассмотрена в следующем разделе.

Заметим, что производительность будет оцениваться в терминах *средних значений*. Хотя предпочтительнее было бы получить распределение оценок производительности, включающее среднее значение и дисперсию, но теоретические результаты для этого отсутствуют. Распределение существует для параметров базы данных (количество порожденных записей на одну исходную и т. п.), требований обработки (число конъюнкций в запросе), вычислительной системы (текущее соотношение числа задач обработки базы данных и фоновых задач). Однако из-за сложности взаимосвязей между пользовательскими приложениями, базами данных, СУБД, ОС и техническими средствами полный анализ возможен только средствами имитационного моделирования [161], которые достаточно дороги и являются неизбежно системно-зависимыми. Методы теории массового обслуживания применяются для анализа моделей ОС [186, 189], но специальных приложений для систем баз данных еще нет. Таким образом, мы будем использовать только средние значения. Следует подчеркнуть, что в данном разделе в качестве оценки производительности рассматривается только время ввода-вывода.

Аналитический подход приемлем, когда, исходя из практических соображений, можно указать определенные границы производительности для лучшего и худшего случаев. В следующих разделах рассматриваются различные режимы работы пользователей и использования внешних устройств и их влияние на время ввода-вывода при обработке физических блоков. Такой подход обеспечивает возможность практической оценки производительности.

9.2.1. Время ввода-вывода для базовой модели

На этапе проектирования реализации эффективность оценивалась количеством обращений к логическим записям (LRA). Соответственно при физическом проектировании эффективность характеризуется количеством обращений к физическим блокам (PBA). В зависимости от способа доступа различают последовательный доступ (sba), быстрый последовательный (fba) и произвольный доступ к блокам (rba). Доступ к блокам является последовательным, если порядок обращений к блокам соответствует порядку их хранения. При быстром последовательном доступе команда поиска поступает непосредственно в блок управления памятью и поиск идентификатора производится аппаратными средствами во время прохождения данных под считывающими головками. При этом не осуществляется никакой передачи данных из внешней памяти в основную. При обычном последовательном доступе каждый последовательный блок передается из внешней памяти в основную, где анализируется его содержание. При произвольном доступе порядок выборки блоков совершенно не зависит от порядка их размещения в памяти. Это подразумевает, что к хранимым в физической последовательности блокам может осуществляться произвольный доступ, а выборка блоков в логической последовательности может потребовать произвольного доступа, если порядок их первоначальной загрузки был произвольным. Для того чтобы выразить количество обращений к физическим блокам через количество обращений к логическим записям, требуется выполнить преобразование. Другое преобразование требуется для определения среднего времени ввода-вывода в зависимости от количества обращений к физическим блокам.

Рассмотрим последовательную базу данных (или файл), содержащую NR записей одинакового размера и формата. При последовательном просмотре всей базы данных имеют место следующие отношения:

$\mathrm{LRA} = \mathrm{NR}$ количество обращения к логическим записям, (9-1)

$\mathrm{PBA}_s = \mathrm{PBA}_f = \left\lceil \frac{\mathrm{LRA}}{\mathrm{BF}} \right\rceil$ при последовательном доступе к блокам, (9-2)

где BF — коэффициент блокирования. Если теперь рассмотрим произвольный доступ к файлу, состоящему из NR записей, причем к каждой записи будет производиться только одно обращение, то мы будем иметь

$$LRA = NR, \tag{9-3}$$

$PBA_r = LRA$ при произвольном доступе к блокам. (9-4)

Чтобы рассчитать время ввода-вывода (TIO) для обоих случаев, определим величину TSBA как среднее время доступа и передачи блока данных из внешней памяти в основную при последовательном доступе (т. е. время ввода-вывода одного блока). Обозначим через TFBA среднее время, в течение которого последовательный файл проходит под считывающей головкой (как при передаче его в основную память). И наконец, TRBA — время доступа и передачи произвольного блока из внешней памяти в основную. Тогда среднее время ввода-вывода для поиска во всей базе данных выразится:

$TIO_s = PBA_s \times TSBA$ для последовательного доступа к блоку, (9-5)

$TIO_f = PBA_f \times TFBA$ для быстрого последовательного доступа, (9-6)

$TIO_r = PBA_r \times TRBA$ для произвольного доступа. (9-7)

Пока мы не учитываем существование буферов. Для последовательных обращений идеальная модель предполагает использование однократной буферизации. Преимущество многократной буферизации для последовательных баз данных состоит в том, что можно получить совмещение процессов ввода (I), вычисления (C) и вывода (O) [149, 318]. До сих пор рассматривался лишь только процесс чтения (выборки) и предполагалось, что при этом не происходит обработка данных центральным процессором и запись базы данных. Если рассматривать весь цикл «чтение-вычисление-запись», то многократная буферизация при соответствующем выборе количества буферов в зависимости от количества используемых процессоров и устройств ввода-вывода, соотношения между требуемым временем обслуживания CPU и временем ввода-вывода может значительно повысить эффективность функционирования [283, 318, 324].

Буферизация также широко используется при произвольном доступе. Можно провести аналогию с системами со страничной организацией памяти, в которых увеличение количества буферов для произвольно организованных баз данных, как и уве-

личение количества выделенных для процесса страниц, повышает вероятность того, что следующий запрашиваемый блок (или страница) уже находится в основной памяти. Учитывая независимость между последовательными запросами в произвольно организованной базе данных, уравнение (9-7) для среднего времени ввода-вывода преобразуется в

$$\begin{aligned} TIO_r &= (1 - PMS) \times PBA_r \times TRBA = \\ &= (1 - NBUF/NBLK) \times PBA_r \times TRBA \end{aligned} \qquad (9\text{-}8)$$

для произвольного доступа ко всем записям, где PMS — вероятность того, что следующий запрашиваемый произвольный блок уже находится в основной памяти, NBUF — количество буферов, выделенных для данного приложения, и NBLK — количество блоков в базе.

В общем случае, когда процесс обработки данных представляет собой многошаговый процесс, в котором используется как произвольный, так и последовательный доступ к данным, то

$$\begin{aligned} TIO = &\sum_S PBA_s \times TSBA + \sum_F PBA_f \times TFBA + \\ &+ \sum_R (1 - NBUF/NBLK) \times PBA_r \times TRBA, \end{aligned} \qquad (9\text{-}9)$$

где S, F, R — множество шагов, использующих последовательный, быстрый последовательный и произвольный доступ соответственно.

Когда осуществляется прямой доступ к базе данных, то $PBA_s = PBA_f = 0$ и $PBA_r = 1$. В базе данных с последовательной организацией при использовании аппаратных средств для поиска произвольной записи $PBA_s = PBA_r = 0$ и $PBA_f \simeq \simeq (1 + \lceil LRA/BF \rceil)/2$. Данное уравнение не совсем строго, так как последний блок базы данных может быть не полным. Хотя определение точного значения величины PBA_f представляет интерес с математической точки зрения [347], это является достаточно сложным, причем точность результата повышается незначительно.

9.2.2. Время ввода-вывода для дисковой памяти

Приведенные в предыдущем разделе уравнения для расчета времени ввода-вывода не зависят от типа устройства и метода доступа. Различные методы доступа к блокам описываются различными выражениями, а различные типы технических устройств потребуют различных выражений для определения TSBA, TFBA и TRBA. Например, хорошо известное уравнение

для НМД с подвижными головками с одной кареткой будет иметь следующий вид:

$$TSBA = ROT/2 + BKS/TR, \quad (9\text{-}10)$$

$$TFBA = BKS/TR + MINSK/BLKPC, \quad (9\text{-}11)$$

$$TRBA = SEEK(NCYL) + ROT/2 + BKS/TR, \quad (9\text{-}12)$$

где ROT — время полного оборота в мс; BKS — размер блока в байтах; TR — скорость передачи в Кбайтах/с; MINSK — минимальное время перемещения считывающих головок на один цилиндр; BLKPC — количество блоков, размещенных последовательно на одном цилиндре; SEEK(NCYL) — среднее время установки головок, необходимое для поиска произвольного блока в заданном экстенте базы данных, зависящее от параметра; NCYL — количество смежных цилиндров, необходимых для хранения базы данных.

Время установки головок зависит от размера и расположения экстента базы данных. Это ограничение будет рассматриваться в следующем разделе. При считывании блока может возникнуть необходимость перевода головки на следующий цилиндр; выражение (9-10) предполагает, что связанные с этим дополнительные задержки пренебрежимо малы и могут быть компенсированы за счет параллельной работы CPU. Строгое выражение для этого случая приведено в [347]. Выражение (9-11) учитывает пересечение границ цилиндра каждым BLKPC-м последовательно считываемым блоком, так как задержки установки считывающих головок на следующий цилиндр обычно составляют значительную часть общего времени считывания. Если быстрый последовательный поиск охватывает большое количество дорожек и цилиндров, то более точно рассчитать время доступа можно по следующей формуле:

$$TFBA = ROT/BLKPT + MINSK/TPC, \quad (9\text{-}13)$$

где BLKPT — количество блоков, последовательно размещенных на одной дорожке; TPC — количество используемых дорожек в цилиндре данного устройства.

Уравнение (9-12) не учитывает возможности пересечения блоками границ цилиндров, так как предполагается, что при произвольном доступе к блокам все обращения производятся к разным цилиндрам. Это предположение базируется на допущении, что блоки хранимых записей размещены произвольно и вероятность того, что два последовательных обращения произойдут к одному и тому же цилиндру, пренебрежимо мала. Вероятность последнего события равна 1/CPD, где CPD — количество используемых цилиндров на одном диске.

Время передачи данных в канале (CHT) выражается как ROT/2 + BKS/TR или BKS/TR в зависимости от того, учитывается или нет при анализе работы канала время ожидания.

• **Пример 9-1.** Чему равно время работы диска и канала, необходимое для последовательного и произвольного поиска данных в странице объемом 4096 байт на накопителе на магнитном диске типа IBM 3350, если известно, что: SEEK(CPD) = 25 мс, ROT = 16,7 мс, TR = 1198 кбайт/с, BKS = 4096 байт.

Время работы канала не включает время ожидания.

TSBA = 16,7/2 + 4096/1198 = 11,8 мс,

TRBA = 25 + 16,7/2 + 4096/1198 = 36,8 мс,

TFBA = CHT = 4096/1198 = 3,4 мс. □

9.2.3. Монопольное и совместное использование вычислительных средств

Эффективность функционирования систем баз данных зависит от типа используемых вычислительных средств. Обычно при анализе структур баз данных рассматривается монопольный режим их использования, так как для этого случая получаются более достоверные оценки и упрощается проектирование таких структур. Многопользовательский режим гораздо сложнее из-за большого числа стохастических по природе и трудных для анализа взаимодействий. Отдельные попытки анализа были предприняты с использованием имитационных моделей [161, 203, 235]. В этом разделе мы ограничимся оценкой среднего времени ввода-вывода как функции используемых вычислительных средств.

С позиций пользователя, в предположении, что в качестве внешней памяти используется стандартный НМД с подвижными головками, режим использования вычислительных средств может быть разбит на три группы:

1. Однопользовательский режим с выделенным устройством (SUDD);
2. Многопользовательский режим с выделенным устройством (MUDD);
3. Многопользовательский режим с разделенным устройством (MUSD).

Однопользовательский режим предполагает, что каждое приложение использует монопольно всю систему. Многопользовательский режим работы подразумевает применение мультипрограммирования и разделение времени, причем целесообразно формировать смесь задач из запросов к базе данных и обычных приложений. В случае выделенного устройства база данных располагается на множестве смежных блоков, называемых

экстентами. На многопользовательском разделенном устройстве база данных не обязательно располагается на смежных блоках, но каждое устройство содержит несколько баз данных, принадлежащих различным пользователям. Тогда позиция головки считывания может рассматриваться как случайная величина. В наихудшем случае каждое обращение к физическому блоку можно рассматривать как случайный процесс, не зависящий от предыдущего запроса. При одно- и многопользовательском режиме работы с выделенным устройством положение головок не изменяется после перехода к поиску следующего блока. Однопользовательский режим с разделенным устройством отдельно рассматривать не имеет смысла, так как в этом случае пользователь имеет монопольный доступ к своим базам данных независимо от того, в каком режиме используется НМД.

Последовательный доступ к базе данных

Учитывая возможные режимы использования вычислительных средств, можно модифицировать выражение (9-10) для вычисления среднего времени последовательного доступа к блоку. Из основного предположения, что за время обработки записи в CPU (плюс другие задержки) угол поворота диска изменится на случайную величину, имеем

$$\mathrm{TSBA\,(SUDD)} = \mathrm{ROT}/2 + \mathrm{BKS/TR}, \tag{9-14}$$

$$\mathrm{TSBA\,(MUDD)} = \mathrm{ROT}/2 + \mathrm{BKS/TR}, \tag{9-15}$$

$$\mathrm{TSBA\,(MUSD)} = \mathrm{SEEK\,(CPD)} + \mathrm{ROT}/2 + \mathrm{BKS/TR}. \tag{9-16}$$

Единственное отличие от исходного выражения, наблюдаемое в случае MUSD, обусловлено произвольным местоположением на диске считывающих головок при обращениях к блокам. Таким образом, при обращении к каждому блоку требуется их установка в заданное положение. В выделенных устройствах этот эффект не имеет места.

Если предположить, что время работы CPU и другие задержки меньше времени оборота диска, то может возникнуть ситуация, при которой существует постоянная синхронизация между запросами на чтение. Тогда величина TSBA (SUDD) будет находиться в диапазоне

$$\mathrm{BKS/TR} + \mathrm{MINSK/BLKPC} \leqslant \mathrm{TSBA\,(SUDD)} < \mathrm{ROT} + \mathrm{BKS/TR}. \tag{9-17}$$

Нижняя граница представляет собой случай, когда время CPU (плюс все другие задержки) меньше, чем время обработки межблочного промежутка. Верхняя граница описывает худший

случай, когда указанное время на долю секунды больше, чем допускает межблочный промежуток, и его не хватает для поиска следующего блока. Тогда для осуществления доступа к следующему блоку требуется дополнительный полный оборот диска. Неравенство (9-17) справедливо для любого размера блока BKS.

Вычисление среднего времени ввода-вывода для каждого из трех режимов легко осуществляется с помощью выражения (9-5). Для последовательной базы данных время поиска и обработка запроса (TRESP) рассчитывается по общей для всех трех режимов формуле

$$\text{TRESP} = \text{TIO}_s + \text{PBA}_s \times \text{TCPUB} + \text{WAIT} + \text{COMMDELAY} =$$
$$= \text{PBA}_s \times (\text{TSBA} + \text{TCPUB}) + \text{WAIT} + \text{COMMDELAY}, \quad (9\text{-}18)$$

где TCPUB — среднее время, затрачиваемое CPU на обработку одного блока, WAIT — среднее суммарное время ожидания в очередях к ресурсам и COMMDELAY — среднее суммарное время передачи данных. При монопольном режиме WAIT = 0. Предполагается, что задержки блокировки включаются в WAIT.

Быстрый последовательный доступ к базам данных

В отличие от обычного последовательного доступа время быстрого последовательного доступа не зависит от режима использования вычислительных средств

$$\text{TFBA (SUDD)} = \text{BKS/TR} + \text{MINSK/BLKPC}, \quad (9\text{-}19)$$
$$\text{TFBA (MUDD)} = \text{BKS/TR} + \text{MINSK/BLKPC}, \quad (9\text{-}20)$$
$$\text{TFBA (MUSD)} = \text{BKS/TR} + \text{MINSK/BLKPC}. \quad (9\text{-}21)$$

В этом случае запрос обрабатывается аппаратными средствами, поэтому при использовании выделенных устройств время доступа имеет такие же характеристики, как и при последовательном доступе, когда вся внутренняя обработка осуществляется за время прохождения межблочного промежутка [т. е. соответствует нижней границе в уравнении (9-17)]. Использование разделенных устройств не приводит к дополнительным изменениям, так как путем формулирования непрерывной цепочки команд в устройстве управления НМД можно избежать прерывания обмена другим процессом (т. е. другой канальной программой) до завершения всех многочисленных обращений к блоку.

В настоящей работе предполагается, что обычно используется последовательный доступ, но для конкретных вычислительных систем может оказаться целесообразным перейти к быстрому последовательному доступу.

Произвольный доступ к базам данных

При произвольном доступе к базам данных нельзя говорить о синхронном режиме доступа к блокам, так как в этом случае время установки головок составляет большую часть общего времени ввода-вывода. Среднее время обработки произвольных блоков (для трех основных режимов использования вычислительных средств) равно

$$\mathrm{TRBA(SUDD)} = \mathrm{SEEK(NCYL)} + \mathrm{ROT/2} + \mathrm{BKS/TR}, \quad (9\text{-}22)$$

$$\mathrm{TRBA(MUDD)} = \mathrm{SEEK(NCYL)} + \mathrm{ROT/2} + \mathrm{BKS/TR}, \quad (9\text{-}23)$$

$$\mathrm{TRBA(MUSD)} = \mathrm{SEEK(CPD)} + \mathrm{ROT/2} + \mathrm{BKS/TR}. \quad (9\text{-}24)$$

Единственное отличие при произвольном доступе имеет место при использовании выделенных устройств (время SEEK(NCYL)), когда поиск осуществляется в определенном экстенте базы данных, т. е. имеет место локальный поиск. В зависимости от размера базы определяется диапазон изменения времени SEEK:

$$0 \leqslant \mathrm{SEEK(NCYL)} \leqslant \mathrm{SEEK(CPD)}. \quad (9\text{-}25)$$

Нижняя граница соответствует случаю, когда база данных размещается на одном цилиндре, а верхняя граница характеризует базу данных, которая занимает все цилиндры устройства. Среднее время установки головок в пределах экстента объемом NCYL цилиндров будем рассчитывать аналогично среднему времени SEEK(CPD) для всего диска. Допустим, что произвольно организованная база данных, занимающая NCYL цилиндров, равномерно распределена по всем цилиндрам. Это предположение не строго, так как последний цилиндр может быть заполнен лишь частично. При $\mathrm{NCYL} \geqslant 5$ ошибка незначительна. Для $\mathrm{NCYL} < 5$ вероятность пересечения головками записи-чтения заданного количества цилиндров может быть легко вычислена. Среднее значение количества пересекаемых цилиндров равно

$$\mathrm{SDIST} = \sum_{x=0}^{\mathrm{NCYL}-1} x \times P\,(\mathrm{D} = x). \quad (9\text{-}26)$$

Вероятность пересечения головками записи-чтения цилиндров равна

$$P\,(\mathrm{D} = x) = \frac{2\,(\mathrm{NCYL} - x)}{\mathrm{NCYL}^2}. \quad (9\text{-}27)$$

Выражение (9-27) может быть получено при учете каждого из NCYL возможных текущих положений считывающих головок и возможных расстояний в обоих направлениях, которые могут

быть доступны из данного положения в предположении, что следующий запрос потребует обращения к любому цилиндру с вероятностью 1/NCYL. Подставляя (9-27) в (9-26), получим [216]

$$\text{SDIST} = \sum_{x=0}^{\text{NCYL}-1} \frac{2x \times (\text{NCYL} - x)}{\text{NCYL}^2} = \frac{\text{NCYL}^2 - 1}{3\,(\text{NCYL})}. \qquad (9\text{-}28)$$

Чтобы перейти к среднему времени установки головки, пред-

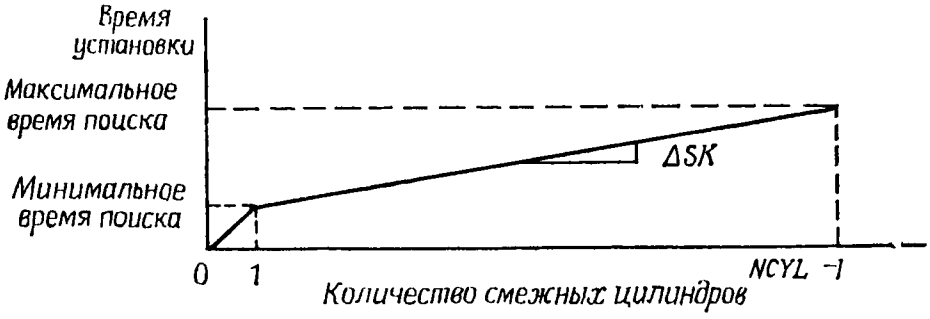


Рис. 9.3. Линейная апроксимация времени установки для НМД с подвижными головками.

положим, что эта величина изменяется по линейному закону, как показано на рис. 9.3. Тогда

$$\text{SEEK}(\text{NCYL}) = \begin{cases} 0 & \text{при NCYL} = 1, \\ \text{MINSK} + \Delta\text{SK}\left[\dfrac{\text{NCYL}^2 - 1}{3\,(\text{NCYL})} - 1\right] & \text{в противном случае,} \end{cases} \qquad (9\text{-}29)$$

где MINSK — минимальное время установки считывающих головок на один цилиндр, а ΔSK — наклон линейной функции времени установки, или

$$\Delta\text{SK} = \frac{\text{MAXSK} - \text{MINSK}}{\text{NCYL} - 2},$$

где MAXSK — максимальное время установки на NCYL — 1 цилиндр при NCYL > 2. Для 1 < NCYL < 5 точный расчет можно провести по формуле

$$\text{SEEK}(\text{NCYL}) = \sum_{x=0}^{\text{NCYL}-1} P(\text{D} = x) \times \text{SEEK}(x). \qquad (9\text{-}30)$$

Общее среднее время ввода-вывода и обработки запроса для всей базы данных рассчитывается так же, как и для баз дан-

ных с последовательным доступом

$$TIO_r = (1 - NBUF/NBLK) \times PBA_r \times TRBA, \quad (9\text{-}8)$$

$$TRESP = TIO_r + PBA_r \times TCPUB + WAIT + COMMDELAY. \quad (9\text{-}31)$$

Выражения для расчета среднего времени доступа и времени передачи сведены в табл. 9.1.

Таблица 9.1. **Среднее время ввода-вывода для физического блока**

	Последовательный доступ	Быстрый последовательный доступ	Произвольный доступ
Однопользовательский режим с выделенным устройством (SUDD)	ROT/2 + BKS/TR (без синхронизации) Нижняя граница (с синхронизацией): BKS/TR+ +MINSK/ /BLKPC Верхняя граница ROT + BKS/TR	BKS/TR+MINSK/ /BLKPC	SEEK(NCYL) + + ROT/2 + + BKS/TR
Многопользовательский режим с выделенным устройством (MUDD)	ROT/2+BKS/TR	BKS/TR + +MINSK/BLKPC	SEEK (NCYL) + +ROT/2 + + BKS/TR
Многопользовательский режим с разделенным устройством (MUSD)	SEEK (CPD) + +ROT/2 + +BKS/TR	BKS/TR + +MINSK/ /BLKPS	SEEK (CPD) + +ROT/2 + +BKS/TR

9.3. Объем внешней памяти

Напомним, что для проектировщика базы данных, помимо времени ввода-вывода, вторым и последним критерием оценки производительности является объем внешней памяти. Важность этого показателя зависит от типа системы; чем больше загружена внешняя память, тем более критичным является этот ресурс. Взаимосвязь между требуемой памятью и ее стоимостью

нелинейна и ее трудно выразить аналитически, поэтому иногда целесообразно просто задать ограничение на объем используемой памяти. Вопрос о том, рассматривать ли эту оценку как параметр или как ограничение, должен решаться руководством вычислительного центра. Мы ограничим наш анализ только расчетом объема памяти, а учет затрат может быть сделан отдельно.

9.3.1. Файловая память

На концептуальном уровне были определены сущности и связи между ними безотносительно к каким-либо физическим параметрам. При проектировании реализации рассматриваются размер элементов и параметры каждой связи для того, чтобы определить общий размер данных. Отдельно от области данных вычисляется объем области указателей, рассматривая связи в качестве потенциальных путей доступа. На уровне реализации, особенно для систем, поддерживающих навигационный метод доступа к базе данных, рассматриваются такие указатели, как «следующий», «предыдущий», «исходный», «первый порожденный» и др. Вопросы индексации пока не рассматриваются, так как на этом уровне еще не ясны многие параметры, от которых зависит выбор указателей. Точно так же, как количество обращений к логическим записям используется для прогнозирования реальной производительности, так и вычисление количества байтов, занимаемых данными и указателями, обеспечивает получение оценки нижней границы требуемого объема физической памяти. Если оценка общего требуемого объема памяти на этапе физического проектирования уже превышает допустимое значение, то следует вернуться к этапу логического проектирования и скорректировать принятые решения.

Существует несколько единиц измерения объема памяти на физическом уровне. Если нужно получить оценку, независимую от конкретного устройства, то в качестве единицы измерения можно использовать байт. Однако объем в байтах надо вычислять, как произведение размера блока на число блоков. Если используются несколько типов отличных по размеру блоков, то объем памяти рассчитывается по следующей формуле:

$$\text{BLKSTOR} = \sum_{k=1}^{\text{NBKS}} \text{BKS}_k \times \text{NBLK}_k \text{ байт}, \qquad (9\text{-}32)$$

где NBKS — количество типов блоков различного размера, BKS_k — средний размер k-го типа блока (в байтах), NBLK_k — количество блоков типа k в базе данных.

В интегрированных базах данных допускается использование блоков различных размеров. В файловых системах обычно размер блоков выбирается одинаковым, но использование индексов и областей переполнения приводит к возникновению блоков различной длины. По крайней мере внутренние форматы в этих блоках отличаются от форматов блоков данных, поэтому вычисление требуемого объема памяти должно проводиться по-разному.

Оценку объема памяти в зависимости от типа устройства лучше проводить в естественных (дискретных) единицах, таких, как «дорожка» или «цилиндр». Расчет в байтах может только ввести в заблуждение точно так же, как было бы заблуждением подсчитывать количество записей вместо количества блоков и игнорировать дополнительные расходы при организации данных во внешней памяти. Следовательно, общий объем вторичной памяти представляет собой сумму объемов основной области данных, области переполнения и области индексов.

Сначала выведем выражения для расчета основной области памяти, требуемой для хранения данных, используя при этом каждую из описанных выше единиц измерения. Рассмотрим файл, содержащий NR записей фиксированной длины. Считается, что каждая хранимая запись содержит элементы данных, указатели и другие необходимые служебные поля. Размер хранимой записи равен SRS байтов (записи переменной длины будут рассмотрены в следующем разделе). Тогда общий объем памяти для хранения записей равен

$$\text{SRSTOR} = \text{NR} \times \text{SRS} \text{ байт.} \tag{9-33}$$

Здесь

$$\text{SRS} = \sum_{i \in I} \text{IS}_i + \text{NPTR} \times \text{PS} + \text{ROVHD}, \tag{9-34}$$

где IS_i — размер элемента i-го типа множества I типов элементов записей (в байтах), NPTR — количество указателей в данном этапе записей, PS — длина указателя (в байтах), ROVHD — размер служебных полей (в байтах). Если используется сжатие данных, то оно распространяется на поля IS_i, PS и ROVHD.

Общее количество блоков оценивается по формуле

$$\text{NBLK} = \left\lceil \frac{\text{NR}}{\left\lfloor \frac{\text{BKS}}{\text{SRS}} \right\rfloor} \right\rceil, \tag{9-35}$$

где BKS — размер блока в байтах, а $\left\lfloor \frac{\text{BKS}}{\text{SRS}} \right\rfloor$ — количество записей, хранимых в каждом блоке (т. е. коэффициент блокирования). В этом достаточно идеальном случае не рассматри-

ваются служебные поля и коэффициент загрузки. Определим служебную область BOVHD, как область памяти, требуемую для размещения всей информации, необходимой операционной системе для управления блоком. В ней обычно содержатся управляющее слово и счетчики количества записей для случая записей переменной длины. Коэффициент загрузки, LF, указывает максимальную часть блока (без учета служебной области), выделенную для размещения записей, в момент их первоначальной загрузки. На рис. 9.4 приведен пример блока размером 100 байт с коэффициентом загрузки 0,8, служебной

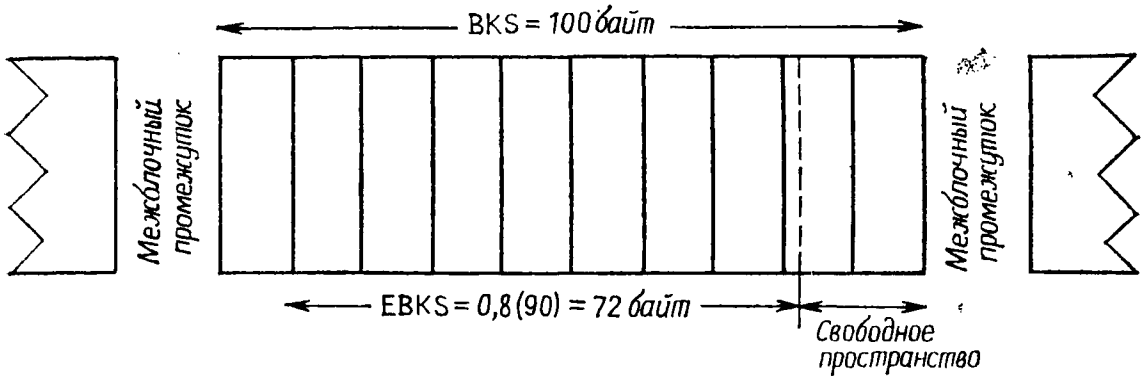


Рис. 9.4. Пример размещения информации в блоке при первоначальной загрузке файла.

областью размером 10 байт и размером хранимых записей 10 байт. Для этого случая коэффициент блокирования равен 9 и допускается первоначальная загрузка 7 записей на блок. Реальное использование блока составит 70 %.

Учитывая при расчете количества блоков, требуемых для хранения базы данных, размер служебной области и коэффициент блокирования, получим, что число хранимых записей в блоке (коэффициент полезного блокирования) равно

$$EBF = \left\lfloor \frac{(BKS - BOVHD) \times LF}{SRS} \right\rfloor, \tag{9-36}$$

где

$$EBKS = (BKS - BOVHD) \times LF \tag{9-37}$$

— есть полезный размер блока, а количество блоков равно

$$NBLK = \left\lceil \frac{NR}{EBF} \right\rceil = \left\lceil \frac{NR}{\left\lfloor \frac{(BKS - BOVHD) \times LF}{SRS} \right\rfloor} \right\rceil. \tag{9-38}$$

Следовательно, общее количество байтов вычисляется по формуле:

$$BLKSTOR = NBLK \times BKS \text{ байт.} \tag{9-39}$$

Общее количество дорожек может быть оценено как

$$NTRK = \left\lceil \frac{NBLK}{\left\lfloor \frac{BPT}{BKS} \right\rfloor} \right\rceil, \qquad (9\text{-}40)$$

где BPT — максимальный размер дорожки в байтах, а $\lfloor BPT/BKS \rfloor$ — допустимое количество блоков, размещаемых на дорожке

$$TRKSTOR = NTRK \times BPT. \qquad (9\text{-}41)$$

Практически для большинства НМД можно воспользоваться другим способом расчета допустимого количества блоков на дорожке с учетом размеров служебных областей дорожек и блоков. В табл. 9.2 приведена связь между размером блока и

Таблица 9.2. **Характеристики емкости трека для НМД IBM 3350 (KS = 0)**

Колнчество физических блоков на дорожке	Максимальное количество байтов в блоке	Количество физических блоков на дорожке	Максимальное количество байтов в блоке
1	19 069	9	1 954
2	9 442	10	1 740
3	6 233	11	1 565
4	4 628	12	1 419
5	3 665	13	1 296
6	3 024	14	1 190
7	2 565	15	1 098
8	2 221	16	1 018

допустимым количеством блоков для системы IBM 3350 [168]. Уравнение имеет вид

Емкость дорожки (в блоках фиксированной длины) = $= 19254/(C + KS + BKS)$,

где $C = \begin{cases} 185, & \text{если } KS = 0, \\ 267, & \text{если } KS \neq 0, \end{cases}$

KS — длина или размер ключа (в байтах),

BKS — размер блока (в байтах).

Константа C учитывает собственный адрес и другие служебные поля блока, а константа 19 254 — емкость дорожки.

Общее количество цилиндров рассчитывается по формуле

$$NCYL = \left\lceil \frac{NTRK}{TPC} \right\rceil, \qquad (9\text{-}42)$$

где TPC — количество используемых дорожек на цилиндре данного устройства.

• **Пример 9-2.** Дан файл, состоящий из 100 000 записей, размером 50 байт каждая. Максимальный коэффициент блокирования равен 30, тип устройства — IBM 3350. Требуется оценить ошибку в определении требуемого общего объема памяти, возникающую в зависимости от того, учитываются или нет параметры организации данных на НМД. Предположим, что LF = 0,9 и BOVHD = 12 байт.

Объем памяти (без учета параметров организации данных на НМД): $SRS = 50 \times 10^6/0{,}9 = 55{,}5 \times 10^6$ байт.

Размер блока: $BKS = 30 \times 50 + 12 = 1512$ байт.

Коэффициент полезного блокирования: $EBF = [(1512 - 12) \times 0{,}9/50] = 27$.

Общее количество блоков: $NBLK = \lceil 10^6/27 \rceil = 37\,038$. Из табл. 9.2 для длины блока 1512 байт получим, что количество блоков на дорожке равно 11.

Общее количество дорожек: $NTRK = \lceil 37\,038/11 \rceil = 3368$.

Объем памяти (с учетом параметров организации данных на НМД): $TRKSTOR = 3368 \times 19\,069 = 64{,}2 \times 10^6$ байт.

Ошибка (%) $= [(64{,}2 - 55{,}5)/64{,}2] \times 100 = 13{,}5\ \%$. □

Блоки данных переполнения и блоки индексов могут быть описаны тем же самым набором параметров, что и основные блоки данных. Обычно записи индексов содержат значения ключевых элементов и указателей, кроме того, в индексы могут также включаться служебные поля и свободные области.

9.3.2. Хранение базы данных: различные типы записей

Объем внешней памяти в системах баз данных состоит из основной памяти для данных, памяти для данных переполнения и памяти для индексов. Это те же самые основные компоненты, что и в файловых системах, однако задача проектирования файлов обычно проще из-за фиксированных размеров записей и блоков. В интегрированных базах данных чаще всего приходится иметь дело с различными типами и размерами записей и с блоками неоднородного состава и различных параметров.

Если физические блоки состоят из однородных типов записей фиксированной длины, то задача сводится к вычислению необходимого количества блоков или дорожек для каждого типа записей и суммированию результатов. Если блоки содержат записи переменной длины независимо от того, однородные типы записей или нет, то задача значительно усложняется. Когда и блоки, и записи имеют переменную длину, то сначала вычисляется общая память для блоков одного размера, а за-

тем производится суммирование по всем типам блоков. Основная проблема сводится к определению объема экстента памяти, содержащего блоки фиксированного размера, состоящие из записей переменной длины. Размеры записей либо порядок их расположения могут изменяться в некотором диапазоне. Если порядок расположения записей в блоке задан, то анализ можно легко провести с помощью методов, изложенных в разд. 9.3.1. Случай произвольного расположения записей является более трудным. Если же неизвестно точное распределение длин записей, то анализ становится невозможным.

Предложенный пример иллюстрирует экстремальные варианты расположения записей в блоке. Предположим, что имеется по 500 хранимых записей длиной 200 и 900 байт. Размер блока — 1 000 байт. Пусть LF = 1 и BOVHD = 0. В лучшем случае, при котором все короткие записи расположены в блоке после размещения в нем всех длинных записей, размер базы составит

$$\left\lceil \frac{500}{\left\lfloor \frac{1000}{900} \right\rfloor} \right\rceil + \left\lceil \frac{500}{\left\lfloor \frac{1000}{200} \right\rfloor} \right\rceil = \left\lceil \frac{500}{1} \right\rceil + \left\lceil \frac{500}{5} \right\rceil = 600 \text{ блоков.}$$

В худшем случае, который соответствует чередованию длинных и коротких записей при их размещении в блоке, размер базы данных равен 1 000 блокам. При любом другом порядке расположения записей в блоке (включая произвольный) значение размера базы данных будет находиться в диапазоне между этими двумя экстремальными значениями. Это соотношение физических размеров (почти 2 : 1) будет существенно влиять на производительность, а также стоимость памяти. Рассмотрим возможные случаи распределения записей.

• **Случай 1.** Рассмотренный выше пример иллюстрирует простейший случай, когда известно распределение длин записей и задан порядок их расположения. Нахождение общего числа блоков, необходимых для хранения базы данных, предполагает определение всех возможных вариантов расположения записей в блоке, вычисление для каждого варианта количества блоков и получения взвешенной суммы. Задача с записями фиксированной длины является частным случаем изложенного подхода.

• **Случай 2.** Как и в случае 1, известно распределение длин записей, но не задан порядок их расположения (т. е. произвольный физический порядок расположения записей (или близкий к этому)). Допустим, что максимальный размер хранимой записи много меньше размера блока: SRS (макс) $\ll$ BKS. При этих условиях известно, что максимальный объем свободного пространства в каждом блоке после первоначальной загрузки меньше, чем размер самой большой хранимой записи,

причем предполагается, что размер служебной области и коэффициент загрузки уже были учтены заранее. Пусть полезный размер блока определяется как в (9-37), тогда если EBKS используется для оценки средней величины использованного в каждом блоке пространства, то максимальная ошибка будет SRS(макс). Для достаточно большого разброса значений размера записей лучшей оценкой средней величины использованного пространства будет EBKS—ESRS, где ESRS — усредненный размер хранимых записей. Эта оценка наиболее точна для нормального закона распределения размера записей, но ее также можно использовать для многих других видов распределений, существующих на практике.

• **Случай 3.** Этот случай аналогичен случаю 2, но предполагается, что величина SRS (макс) сравнима с размером BKS. Поэтому подход, изложенный выше, здесь неприменим. Строгого решения задачи в изложенной постановке нет, однако метод для случая записей переменной длины, предложенный Оберлендером [244], дает хорошее приближение. Он предполагает, что размер любой записи не зависит от размера смежных с ней записей. Предполагается, что это справедливо для всех блоков, не считаясь с тем фактом, что первая запись в j-м блоке может оказаться первой записью, которую не удалось разместить во время загрузки базы данных в $(j-1)$-м блоке. Таким образом, хотя указанная независимость не всегда имеет место, ошибка от использования этого допущения мала. Проиллюстрируем этот метод на примере.

Пусть по определению база данных состоит из 300 записей и средний размер записи 200 байт. Допустим для простоты, что BOVHD = 0 и LF = 1. Размер блока 300 байт. Распределение размера записей следующее:

Тип записи	SRS, *байт*	Число экземпляров записи
A	100	100
B	200	100
C	300	100

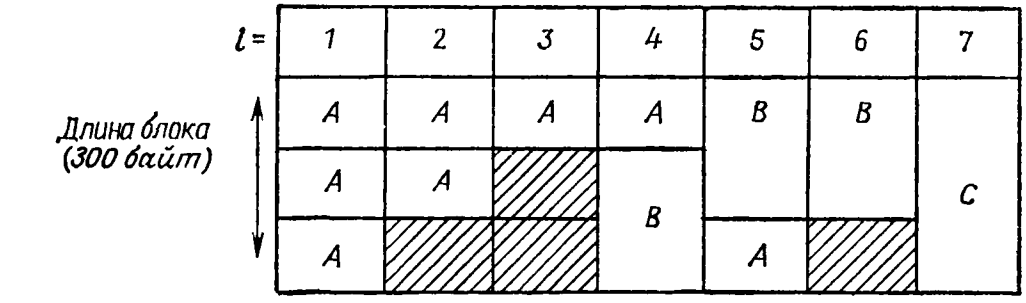
Для этого простейшего случая случайное распределение записей и типов записей в базе данных приводит в результате к семи возможным вариантам размещений записей в блоке, как показано на рис. 9.5, каждый из которых может возникнуть с вероятностью P_i, $i = 1, 2, \ldots, 7$. Из данных рис. 9.5 можно рассчитать среднюю величину используемого пространства па-

мяти для каждого блока и затем размер базы данных.

$$E\,[\text{используемое пространство}] = \sum_{i=1}^{T} P_i \times \text{USED}_i = 248 \text{ байт/блок},$$

$$\text{SRSTOR} = \sum_{j=1}^{3} \text{SRS}_j \times \text{NR}_j = 100 \times 100 + 200 \times 100 + 300 \times 100 = 60\,000 \text{ байт},$$

$$\text{NBLK} = \left\lceil \frac{60\,000 \text{ байт}}{248 \text{ байт/блок}} \right\rceil = 242 \text{ блока}.$$



Следующий тип записи в случайной последовательности	A, B или C	B или C	C	A, B или C	A, B или C	B или C	A, B или C
Вероятность варианта, P_i	$\frac{1}{27}$	$\frac{2}{27}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{2}{9}$	$\frac{1}{3}$
Используемое пространство в блоке, USED_i	300	200	100	300	300	200	300

Рис. 9.5. Случайное распределение типов записей в блоках.

С другой стороны (для неслучайного размещения записей):

1. *ABCABCA ...* последовательность: NBLK = 200 блоков (глобальный минимум).

2. *AA ... ABB ... BCC ... C* последовательность: NBLK = = 233 блока.

3. *ACACAC ... ACBBB ... B* последовательность: NBLK = = 300 блоков (глобальный максимум).

Моделирование, осуществляемое вручную, позволило разместить 300 случайно распределенных записей в 230 блоках. Таким образом, рассчитанная выше оценка для NBLK = 242 блокам дает погрешность в 5 %.

Сложность вычислений по этому методу быстро растет с увеличением размеров блоков и числа типов записей. При числе типов записей свыше 10 ручной расчет уже невозможен и требуется реализация алгоритма на ЭВМ. Сложность расчета возрастает полиномиально, практически он может быть выпол-

нен в большинстве случаев, хотя для баз данных, содержащих свыше 200 типов записей, это достаточно дорого. Если размер блока становится значительно больше размера хранимой записи, то более предпочтительным является использование способа, рассмотренного в случае 2.

• **Случай 4.** Распределение длин записей неизвестно, но задан средний размер записи. В этом случае проведение расчетов в предположении о фиксированной длине записи по ее среднему значению нежелательно, так как это может привести к существенно завышенным оценкам требований к памяти. Лучше предположить равномерное распределение размеров записей в интервале от 1 байт до $(2 \times SRS - 1)$ байт со средним значением SRS. В зависимости от размера блока могут быть применены методы, использованные в случае 2 или 3.

9.4. Заключение

Основными составляющими в проектировании физической структуры баз данных являются проектирование формата хранимой записи, кластеризация хранимых записей и выбор методов доступа. Налагаются также ограничения целостности и безопасности, как уже отмечалось при рассмотрении проектирования реализации. Выполняемый параллельно на этом этапе процесс проектирования программ должен быть независим от процесса проектирования базы для того, чтобы обеспечить максимальную физическую независимость данных.

Хотя существуют различные способы оценки производительности баз данных, наиболее удобными для проектирования являются оценки времени ввода-вывода и объема внешней памяти, требуемой для размещения записей переполнения и индексов методов доступа. Получение оценок объема базы данных с различными типами записей значительно более сложно, чем оценок размера файла (из-за неоднородности состава блоков).

Расчет времени ввода-вывода зависит от режима использования вычислительных средств (от однопользовательского с выделенными устройствами до многопользовательского с разделенными устройствами). Необходимо исследовать оба этих крайних случая, используя для расчета производительности метод средних значений. Переход от вычисления длины пути доступа к времени ввода-вывода осуществляется в 2 этапа: получение выражений для оценки количества обращений к физическим блокам для случая последовательного и произвольного доступа и вычисление времени обслуживания ввода-вывода при обращениях к физическому блоку. В данной главе изложен второй этап, в главах 12—16 будет рассмотрен первый, а оценка времени CPU будет дана в гл. 16.

Глава 10. Проектирование структуры записи

Важным этапом в процессе разработки базы данных является проектирование структуры и содержания физической (хранимой) записи. Основное содержание записи, тип элементов, их длина определяются при проектировании реализации. Проектирование структуры записи должно быть таким, чтобы при отображении логической записи в хранимую не произошло изменение ее содержания. Однако при выполнении физического проектирования возможны дополнительные модификации записи, для того чтобы достичь оценочных значений параметров производительности — это кодирование данных и разбиение записей.

Под кодированием данных понимается широкий класс методов кодирования значений элементов в нормальный или сокращенный формат. Сжатие данных производится с целью уменьшения затрат на их поиск и экономии объема памяти. Разбиение записи (часто называемое сегментацией) — это форма размещения записи, при которой сокращается логическая структура базы данных (схема), но элементы в пределах записи могут быть кластеризованы и перераспределены по отдельным физическим экстентам. Аналитические методы, которые использовались для определения способа группирования элементов в логических записях, с успехом могут быть применены для кластеризации элементов в хранимых записях. Отличие состоит в необходимости учета некоторых дополнительных независимых переменных, использующихся при физическом проектировании и в способах оценки производительности. Целью разбиения записей является сокращение затрат на поиск и обновление данных.

Процесс проектирования состоит в постепенном усовершенствовании структуры базы данных в пределах заданной логической схемы с использованием (или с частичной модернизацией) разработанных ранее методик. В этой главе проводится анализ и сравнение производительности различных методов кодирования, сжатия и разбиения записей.

10.1. Кодирование и сжатие элементов данных

Несмотря на большие успехи в технологии изготовления запоминающих устройств, задача сокращения объема используемой памяти и времени передачи данных будет продолжать

оставаться основной проблемой при рассмотрении вопросов проектирования физических баз данных. Эти составляющие общих системных затрат существенно зависят от способа представления данных в памяти. На рис. 10.1 показаны основные компоненты хранимой записи: элементы данных, указатели, служебные поля. Служебные данные, такие, как управляющие биты и индикатор длины записи, кодируются в соответствии с требованиями используемых аппаратных и программных

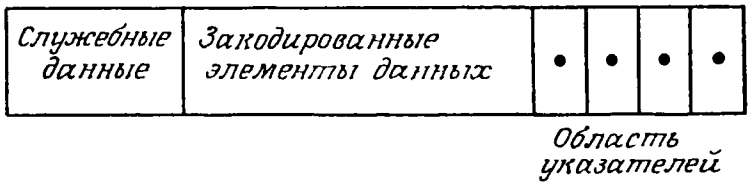


Рис. 10.1. Компоненты хранимой записи.

средств. Аналогично кодирование указателей выбирается таким, чтобы упростить быстрый поиск и сократить требуемый объем памяти. Например указатели, которые определяют абсолютные адреса, могут быть указаны как смещение относительно базы, если это значительно уменьшит размер указателей. Рассмотрим подробнее аспекты проектирования структуры записи.

10.1.1. Представление элементов данных

Было показано, что существует по крайней мере четыре простейших способа размещения элементов данных в хранимой записи [221, 245]: позиционный, с разделителями, индексный и с описателями. Эти способы приведены на рис. 10.2. Первые три метода требуют, чтобы в каждой записи порядок расположения элементов данных был одинаковым, четвертый метод допускает произвольный порядок расположения элементов. В первых трех случаях типы элементов данных явно не указываются и физически представляются только значения элементов. При использовании описателей в явном виде представляются как описатель (кодирующий тип элемента данных), так и значения элемента.

В позиционном методе для хранения значений элементов используются поля фиксированной длины; этот метод позволяет применить более эффективные алгоритмы поиска, но при этом память используется нерационально. Если данные имеют меньшую длину, чем размер соответствующего поля, то производится выравнивание (вправо или влево) и свободные места заполняются пробелами. Количество свободных мест может

быть сокращено путем использования разделителей либо индексного способа. В способе с разделителями пробелы заменяются специальным символом, который нигде в базе данных больше не используется. При индексном подходе для определения начала (и, неявно, конца) значений элементов используется массив указателей. Указатели могут представлять собой абсолютные адреса, относительные смещения или иногда массив длин элементов данных. Способ с описателями эффективен, когда запись содержит много элементов, значения которых

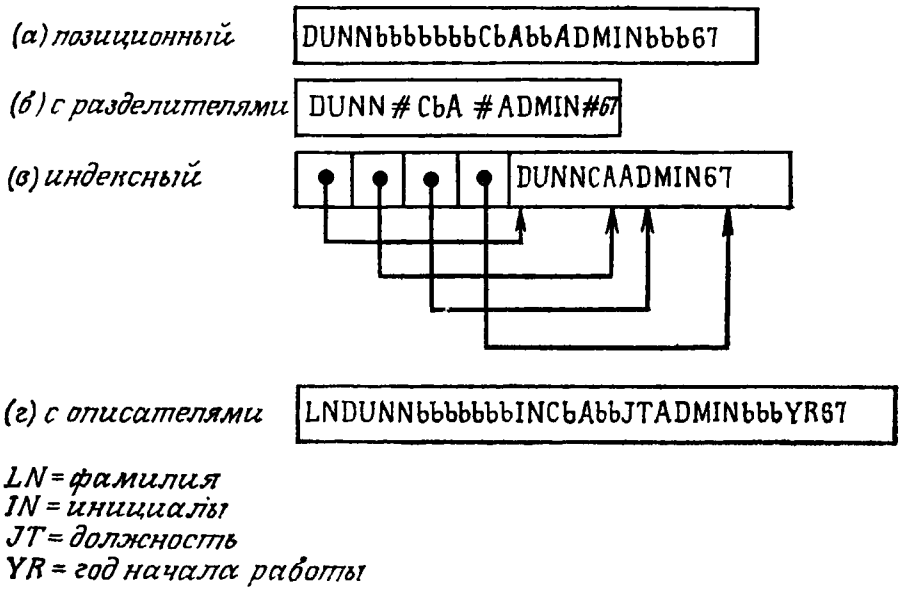


Рис. 10.2. Методы представления элементов данных в хранимых записях.

отсутствуют и необходимо явно хранить только элементы с известными значениями. Другие возможные способы представления записей описаны в [221].

Для заданного способа хранения элементов данных существуют различные варианты представления символов в виде последовательности двоичных битов. Наиболее часто используемые коды символов: ASCII (7 бит на символ; Американский национальный институт стандартов (ANSI)), EBCDIC (8 бит на символ; IBM); полевой код (6 бит на символ; U. S. Army); BCD (6 бит на символ; CDC) и код Бодо (5 бит плюс символ переключения) [217, 337]. Использование стандартных методов кодирования облегчает работы как по созданию системы, так и по ее конвертированию. В табл. 10.1 приведены коды ASCII. В следующей главе будут рассмотрены методы сжатия, которые позволяют получить из стандартных — специальные сжатые коды.

Таблица 10.1 **ASCII символьные коды** [1), 2)]

Позиции бита 4321	765 000	001	010	011	100	101	110	111
0000	NUL	DLE	blank	0	@	P	'	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	-	o	DEL

[1]) Младший разряд символа — справа.
[2]) ACK — подтверждение

BEL — звонок
blank — пробел
BS — возврат на шаг
CAN — аннулирование
CR — возврат каретки
DCx — управление устройством
DEL — вычеркивание/ожидание
DLE — авторегистр 1
EM — конец носителя
ENQ — кто там?
EOT — конец передачи
ESC — авторегистр 2
ETB — конец блока
ETX — конец текста
FF — перевод формата
FS — разделитель файла
GS — разделитель группы
HT — горизонтальная табуляция
LF — перевод строки
NAK — отрицание
NUL — пусто
RS — разделитель записи
SI — латинский регистр
SO — национальный регистр
SOH — начало заголовка
STX — начало текста
SUB — замена
SYN — синхронизация
US — разделитель элементов записи
VS — вертикальная табуляция

10.1.2. Методы сжатия

Существует много способов сокращения объема памяти для размещения хранимых записей. Уже упоминалось о кодировании указателей и служебных данных записи, а избыточность

типов элементов должна быть устранена на этапе концептуального проектирования и проектирования реализации (этапы 2 и 3). Методика кодирования элементов данных хорошо разработана и описана ранее [9, 268, 337]. Рассмотрим основные классы способов сжатия и их применение.

Основной оценкой эффективности является коэффициент сжатия, COMPR:

$$\text{COMPR} = \frac{\text{размер данных до сжатия}}{\text{размер данных после сжатия}}. \qquad (10\text{-}1)$$

Выбор единицы измерения размера данных зависит от конкретной задачи. Это может быть отдельный символ, если коэффициент сжатия фиксирован, либо файл в целом, если коэффициент сжатия является переменным для различных символов или последовательностей символов. При неравномерном сжатии COMPR представляет среднее значение на один символ. Для заданного коэффициента сжатия средняя величина сокращения размера данных рассчитывается по формуле:

$$\text{REDUC} = \left(1 - \frac{1}{\text{COMPR}}\right) \times 100\,\% = \frac{\text{COMPR} - 1}{\text{COMPR}} \times 100\,\%. \qquad (10\text{-}2)$$

Аббревиатуры

Аббревиатуры представляют собой более широкую категорию кодирования данных, чем сжатие, так как кодирование производится до загрузки базы данных или во время ее реорганизации. Несколько примеров аббревиатур приведены на

Штаты (2 байт)		*Города (2 байт)*		*Агенства (3 байт)*
Arizona	*AZ*	*Anaheim*	*C1*	*CIA*
California	*CA*	*Burbank*	*C2*	*DIA*
Michigan	*MI*	*Hollywood*	*C3*	*DOD*
New Jersey	*NJ*	*Los Angeles*	*C4*	*FBI*
New York	*NY*	*San Diego*	*C5*	*NSA*
Даты				
Июнь 15, 1968		*6. 15. 68*		*3256 (дни с 01.01.60)*
(13 байт)		*(6 байт)*		*(12 байт)*

Рис. 10.3. Коды аббревиатур ($2 \leqslant \text{COMPR} \leqslant 10$).

рис. 10.3. При этом получается очень значительное сокращение объема данных, однако при достаточно большом количестве значений элементов объем используемых в процессе декодирования таблиц становится недопустимо велик. Часто декодирование проводится вручную с помощью твердых копий таблиц. Также следует быть внимательным при выборе кодов аббревиатур для одного типа элементов (например, CANADA(CA)

и CALIFORNIA(CA) имеют одинаковые двухсимвольные аббревиатуры). В заключение отметим, что аббревиатуры могут быть заранее определены или являются статичными, позволяя тем самым выбирать для них уникальные коды.

Подавление нулей

Подавление нулей — это термин, обозначающий способы устранения избыточных нулей и пробелов. Хотя эти способы по сравнению с некоторыми другими обеспечивают меньший коэффициент сжатия, они эффективны, если в значениях данных преобладают нули и пробелы.

Исходные данные: *DUNNbbbbbbCbAbb450000655*

Сжатые данные: *DUNN#6CbAbb45@4b55*

COMPR = *24/18* = *1,33*

REDUC = *25%*

Рис. 10.4. Подавление нулей с помощью индикаторов длины.

Один из способов подавления нулей состоит в расширении способа использования разделителей применительно к хранению элементов данных. Это проиллюстрировано на рис. 10.4. Последовательность нулей и пробелов заменяется специальным

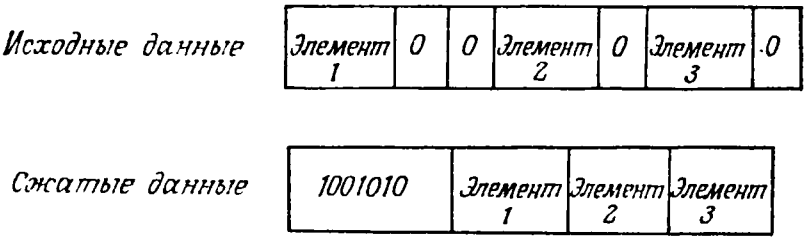


Рис. 10.5. Подавление нулей с использованием битовой карты (COMPR=1,75, REDUC=43 %).

символом, за которым следует число, указывающее длину последовательности. Очевидно, что последовательности длиной в один или два символа нецелесообразно преобразовывать таким способом и обычно они не кодируются.

Другой подход к подавлению нулей состоит в использовании битовых карт. Это вариант индексного метода представления данных. Значения фиксированных единиц данных, например слова или байта, можно заменить специальным битом: нулевое значение данного — битом 0, а ненулевое — битом 1. Эти биты хранятся в начале файла и их позиции точно соответствуют позициям символов, слов или других единиц в файле. Пример использования битовых карт приведен на рис. 10.5.

Подстановка по шаблону

Подстановка по шаблону — это класс методов, в которых часто повторяющиеся последовательности символов замещаются специальными более короткими кодами. На рис. 10.6 иллюстрируется использование таблиц подстановок, к которым производится обращение при просмотре символов данных. Подстановка производится при обнаружении соответствия между анализируемой последовательностью символов данных и одним из

Исходные данные	*Сжатые данные*
CED3690000BB52X0	#369?/52X0
CED3700000BB86X0	#370?/86X0
CED3710000BB12X3	#371?/12@

Таблица подстановок

CED	#
0000	?
BB	/
X3	@

Рис. 10.6. Подстановка шаблона ($1{,}6 \leqslant COMPR \leqslant 1{,}8$).

входов в таблицу. В общем случае таблицы подстановок могут храниться вместе с данными или кодирующими программами. Разработано достаточно много сложных методов подстановок одного символа вместо двух или более символов, которые подробно рассматриваются в [9, 268].

Статистическое кодирование

Статистическое кодирование — еще один класс методов, которые могут быть использованы самостоятельно или в комбинации с методами подстановки [9]. Статистическое кодирование полезно применять, когда известно распределение частоты появления символов; тогда для более часто встречающихся последовательностей символов можно использовать короткие обозначения, а для редко встречающихся — более длинные. В сочетании с методом подстановки короткие обозначения могут использоваться для некоторых часто встречающихся пар или других групп символов. Например, в азбуке Морзе короткие кодовые группы применяются для часто встречающихся букв.

При использовании для представления сообщений кодов переменной длины, состоящих из последовательности двоичных единиц и нулей, должно существовать правило, позволяющее определять, где находится последний символ кода и начинается другой код. Оно может быть выполнено, если код обладает *свойством префикса*, которое означает, что ни одна короткая кодовая комбинация не является началом другой, более длинной комбинации. Коды Хаффмана [160] обладают этим свойством и, кроме того, они обеспечивают минимальную избыточность, т. е. являются оптимальными в том смысле, что дан-

ные, представленные в этих кодах, не могут быть выражены меньшим числом битов.

Рис. 10.7 иллюстрирует способ формирования кода Хаффмана. Символы располагаются в порядке убывания частоты их появления и предлагается следующая последовательность формирования кода. Выбираются две группы с наименьшими частотами появления, и одной группе ставится в соответствие бит 0, а другой — 1. Эти значения будут представлять собой значения самого правого разряда в коде Хаффмана. В нашем случае самый правый разряд для A равен 1, а для $B = 0$ (но можно было принять и наоборот). В дальнейшем группы A и B рассматриваются как одно целое BA и ему приписывается

Символ	*Частота, %*	*Шаг 1*	*Шаг 2*	*Шаг 3*	*Код Хаффмана Шаг 4*
E	*60*				*0*
T	*20*			*0*	*10*
4	*10*		*0*	*10*	*110*
B	*6*	*0*	*10*	*110*	*1110*
A	*4*	*1*	*11*	*111*	*1111*

Рис. 10.7. Образование кода Хаффмана (COMPR=4,71, REDUC=79 %). Предполагаются 8-битовые символы [9].

определенное значение второго разряда. Затем этот процесс повторяется для списка E, T, 4, BA, где BA представляет собой группу A и B и имеет частоту появления 10 %. Снова выбираются две наименее часто встречающиеся группы (4 и BA) и символу 4 присваивается значение 0 во втором разряде кода Хаффмана, а символу BA — значение 1.

Полученный незавершенный код представлен Шагом 2 на рис. 10.7. Затем этот процесс повторяется на Шагах 3 и 4. При этом каждый раз формируется новый список, где объединяются те два элемента из предыдущего списка, которым уже присвоены величины, и кодируются два наименее часто встречающихся элемента в новом списке. В нашем примере запись данных в коде Хаффмана требует в среднем 1,7 бит на символ, в то время как использование записей фиксированной длины потребовало бы 3 бит.

В общем случае формирование кода Хаффмана для набора символов $c_1, \ldots, c_n$ с вероятностью появления $p_1, \ldots, p_n$ соответственно требует построения двоичного дерева, в котором каждый из n символов представлен как терминальная вершина, а нетерминальные вершины формируются следующим образом. Сначала из двух вершин с наименьшими вероятностями (допустим c_1 и c_2) формируется новая вершина $c_{1,2}$, являющаяся

исходной для вершин c_1 и c_2, которой приписывается значение вероятности, равное $p_1 + p_2$. Для сокращенного множества $n - 1$ вершин, которое состоит из символов $c_{1,2}$, c_3, ..., c_n с приписанными им значениями вероятности $p_1 + p_2$, p_3, ..., p_n соответственно, повторяется вышеописанная процедура до тех пор, пока сокращенное множество не будет состоять только из двух вершин.

Теперь рассмотрим двоичное дерево, состоящее из терминальных вершин и всех новых вершин, сформированных приведенным выше способом. Каждой ветви из пары ветвей, имеющей общий корень, приписываются значения 0 или 1. Полученный в результате для каждого символа код представляет

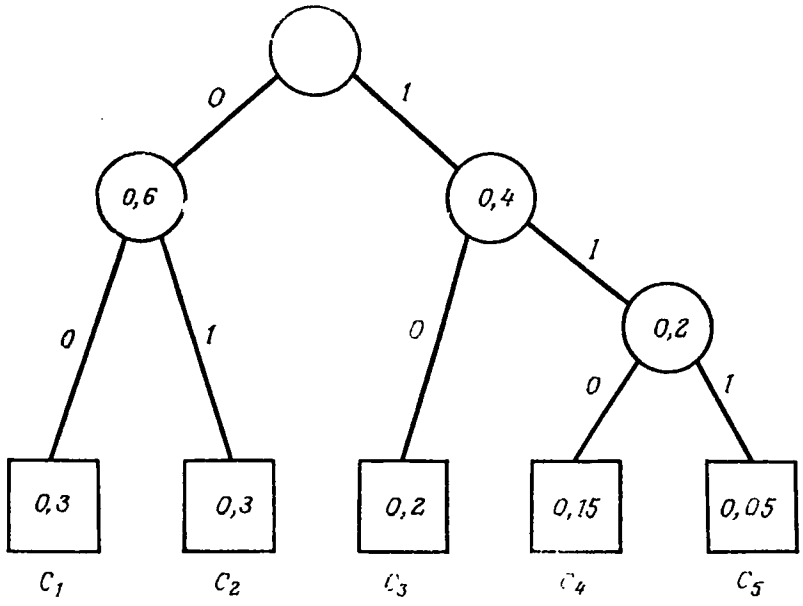


Рис. 10.8. Двоичное дерево для построения кода Хаффмана [9].

собой последовательность присвоенных значений, полученную путем прохождения по дереву от корня до каждой из терминальных вершин. Появление каждого нового поддерева элементов приводит к увеличению длины кода на единицу; предложенный метод обеспечивает минимальную среднюю длину кода.

Этот метод проиллюстрирован на рис. 10.8. Пусть заданы символы c_1, c_2, c_3, c_4 и c_5 с вероятностями появления 0,3, 0,3, 0,2, 0,15, и 0,05 соответственно. В дереве, построенном по изложенному методу, терминальные вершины обозначены квадратами, а внутренние — окружностями. Внутри каждой окружности указано значение соответствующей вероятности. Код Хаффмана для каждого символа следующий:

$c_1 = 00$, $c_4 = 110$, COMPR = 3,6 (2,2 бит на символ),
$c_2 = 01$, $c_5 = 111$, REDUC = 72 %,
$c_3 = 10$.

10.2. Разбиение записи

Вопрос разбиения хранимых записей достаточно глубоко изучен для различных вычислительных устройств [14, 78, 111, 145, 213]. Рассмотрим наиболее удачные методы и концепции, лежащие в их основе, с тем чтобы иметь возможность применять подходящие средства для решения типовых задач проектирования. Крайний случай представляет собой алгоритм оптимальной сегментации [213], в котором элементы в хранимой записи разделяются на первичные и вторичные [213]. Хотя этот метод интересен с теоретической точки зрения, следует отметить, что его практическая реализация требует применения специальных систем, которые могут обеспечить требуемое физическое разбиение. В работах [78, 145] рассмотрен более общий случай кластеризации элементов данных в интегрированных базах с различными типами записей.

10.2.1. Алгоритм сегментации записей

Алгоритм сегментации записей Марча и Северанса [213] основан на том общепризнанном факте, что 80 % обращений к базе приходится только на 20 % элементов хранимых данных (правило "80/20"). Следовательно, согласно теории, можно значительно увеличить эффективность обработки файла или базы данных за счет разбиения элементов данных на первичные и вторичные сегменты (называемого далее сегментацией) или экстенты в соответствии с активностью их использования в качестве критериев выбора или в отчетах.

Первичный сегмент должен хранить наиболее часто применяемые данные, и обычно он размещается на самых быстрых устройствах памяти. По грубым оценкам здесь должно размещаться примерно 20 % элементов данных. *Вторичный сегмент* должен хранить менее активные элементы данных; он может быть расположен на устройстве того же типа, что и первичный сегмент, но может также размещаться на более медленном устройстве. Каждый сегмент можно считать подфайлом. Все запросы к данным начинаются с обращения к первичному сегменту. Если данные не найдены, то второе обращение делается к вторичному сегменту. В основной памяти выделяется соответствующее место для размещения первичного и вторичного подфайлов так, чтобы минимизировать общие затраты на обработку выполняемых системой приложений. Производительность оценивается числом обращений к физическим устройствам для поиска и обновления данных и требуемым объемам памяти, причем все оценки даются в стоимостном выражении в относительных единицах (на одно обращение или один байт памяти).

Эта проблема была впервые проанализирована в работе [111] для задачи выборки отдельной записи из вторичного сегмента. При заполнении буферной области основной памяти записи первичного сегмента блокируются. Марч и Северанс моделируют наиболее общий случай, когда первичный и вторичный подфайлы сблокированы произвольно с целью наложения ограничения на объем буферной области основной памяти. Алгоритм Марча и Северанса на 20—30 % более эффективен, чем алгоритм Эйснера — Северанса, и предполагает сокращение стоимости системы на 65—80 %. Марч и Северанс сформулировали проблему сегментации записи как задачу нахождения потока в сети и разработали для ее решения эффективную процедуру, основанную на методе ветвей и границ. Время решения возрастает пропорционально $(m + n)^2$, где m — количество элементов данных в исходной записи, а n — количество приложений, в которых используются эти записи. Детали алгоритма описаны в работе Марча и Северанса [213].

Теперь давайте на интуитивном уровне рассмотрим основные принципы алгоритма Марча и Северанса. Информация об элементах данных, необходимая для сегментации записи, приведена в табл. 10.2. Каждая строка содержит типы элементов данных (d_i), а каждый столбец представляет приложение пользователя (u_j). Размер элемента обозначен как IS_i, а частота приложений — через F_j. Отдельный элемент матрицы отображает факт использования конкретного элемента данных в приложении, но ничего не говорит о способе его использования.

Таблица 10.2. **Характеристики использования элементов данных приложениями пользователей**

Элемент данных	Размер элемента, IS_i	Частота, F_j =	15	2	55	10
d_1	10		×		×	
d_2	12		×	×		
d_3	38					×
d_4	4				×	

В предложенной структуре хранимая запись будет состоять из двух сегментов: первичного и вторичного. Если обозначить множество элементов записей как $D = \{d_1, \ldots, d_i, \ldots, d_m\}$ и приложения пользователей как $U = \{u_1, \ldots, u_j, \ldots, u_n\}$, тогда вариант A сегментации записи определяется как подмножество $D_A \subseteq D$ элементов данных, расположенных в первич-

ном сегменте. Длина первичного сегмента выразится как

$$\mathrm{SRS}_1 = \sum_{d_i \in D_A} \mathrm{IS}_i, \tag{10-3}$$

а длина вторичного сегмента —

$$\mathrm{SRS}_2 = \sum_{d_i \in D - D_A} \mathrm{IS}_i, \tag{10-4}$$

где $\mathrm{SRS}_1 + \mathrm{SRS}_2 = \mathrm{SRS}$ — постоянная величина.

Варианту сегментации A соответствует множество пользователей $U_A \subseteq U$, использующих элементы данных, расположенные в первичном сегменте. Суммарную частоту использования приложений «удовлетворенных» пользователей, чьи данные находятся в первичном сегменте, за некоторый заданный период времени, можно записать как

$$F_s = \sum_{u_j \in U_A} F_j, \tag{10-5}$$

а суммарную частоту использования приложений «неудовлетворенных» пользователей (т. е. тех пользователей, чьи данные оказались во вторичном сегменте) запишем как

$$F_d = \sum_{u_j \in U - U_A} F_j, \tag{10-6}$$

где $F_s + F_d = F$ — постоянная величина.

Рассматриваемые здесь приложения касаются в основном только выборки данных, выборка первичного сегмента состоит из обращения и пересылки данных. Если в приложении требуется один или несколько элементов данных из вторичного сегмента, то нужен поиск как в первичном, так и во вторичном сегменте. Даже если все элементы данных находятся во вторичном сегменте, требуется обращение как к первичному, так и ко вторичному сегментам. Это ограничение необходимо для сохранения строгости математических формулировок. Можно утверждать, что влияние этого ограничения на возможные решения незначительно из-за того, что к элементам данных вторичного сегмента обращаются достаточно редко.

Задача сегментации записи может быть представлена как

$$\min \mathrm{TCOST} = \mathrm{NR}\left[\left(\frac{\mathrm{CA}_1}{\mathrm{BKS}_1} + \mathrm{CT}_1\right) \times \mathrm{SRS}_1 \times F + \right.$$
$$\left. + \left(\frac{\mathrm{CA}_2}{\mathrm{M} - \mathrm{BKS}_1} + \mathrm{CT}_2\right) \times \mathrm{SRS}_2 \times F_d + \mathrm{CS}_1 \times \mathrm{SRS}_1 + \mathrm{CS}_2 \times \mathrm{SRS}_2\right], \tag{10-7}$$

где TCOST — общие затраты на поиск и хранение данных за период времени T, NR — количество записей в файле,

Таблица 10.3. **Содержание базы данных персонального учета [213]**

Элемент данных, d_i			Запросы пользователей u_j									
			1	2	3	4	5	6	7	8	9	10
Номер п/п	Имя	IS_i	Частота использования F_j									
			1	1	10	5	10	10	50	500	300	300
1	Имя служащего	30	×	×	×	×		×	×	×		
2	Личный номер	6	×		×				×		×	×
3	Номер страхового полиса	9	×		×			×				
4	Заработная плата	6	×			×		×				
5	Статус служащего	1	×	×	×	×	×	×	×	×		
6	Код безопасности	1	×		×				×			
7	Код квалификации	6	×				×					
8	Код подразделения	2	×	×	×		×	×	×			
9	Код отдела	4	×	×	×		×	×	×	×		
10	Участие в проектах	8	×				×		×	×		
11	Затраченное время	12	×				×			×		×
12	Часы переработки	12	×				×	×		×		×
13	Смена	1	×				×	×				

14	Данные об отсутствии	10	×						×	×
15	Данные об удержаниях	15	×				×			
16	Подчиненные	2	×				×			
17	Адрес учреждения	5	×	×						
18	Телефон учреждения	4	×	×						
19	Дата поступления на работу	6	×							
20	Данные о постоянной работе	6	×							
21	Дата увольнения	6	×							
22	Изменения должности	30	×		×			×		
23	Изменение зарплаты	30	×		×					
24	Конфиденциальные данные	20	×		×					
25	Семейное положение	1	×		×					
26	Пол	1	×		×					
27	Дата рождения	6	×		×					
28	Образование	60	×							
29	Домашний телефон	10	×	×						
30	Домашний адрес	50	×	×						

M — размер буфера в основной памяти, он распределяется между первичным и вторичным подфайлами; BKS_1 — размер блока первичного подфайла ($BKS_2 = M - BKS_1$); CA_1 — стоимость одного обращения к первичному подфайлу; CT_1 — стоимость передачи одного байта данных из первичного подфайла; CS_1 — стоимость хранения одного байта данных в первичном подфайле в течение заданного периода времени.

Аналогично CA_2, CT_2 и CS_2 определяются для вторичного подфайла. SRS_1, SRS_2 и BKS_1 являются в уравнении (10-7) зависимыми переменными.

Проиллюстрируем применение алгоритма на следующем примере. Предположим, что индивидуальный файл пользователя определяется приведенными в табл. 10.3 типами элементов и характеристиками приложений. Остальные параметры системы даны в табл. 10.4. Результаты применения алгоритмов

Таблица 10.4. **Параметры системы и базы данных персонального учета [213]**

	Основное устройство		Вспомогательное устройство
Параметры системных затрат			
Затраты на доступ (долл./доступ) CA	$1{,}54\times10^{-3}$		$1{,}54\times10^{-3}$
Затраты на передачу (долл./символ) CT	$5{,}2\times10^{-8}$		$5{,}2\times10^{-8}$
Затраты на хранение (долл./символ/мес) CS	$6{,}0\times10^{-7}$		$6{,}0\times10^{-7}$
Параметры набора данных			
Длина записи (символ)		360	
Количество записей		100 000	
Ограничения на основную память: Требуемый объем буфера (символ)		13 000	

Марча — Северанса и Эйснера — Северанса для тестового примера приведены в табл. 10.5. Применяя эвристическое правило "80/20", мы можем проверить чувствительность общих затрат к изменениям параметров решения. Из 30 перечисленных элементов данных количество обращений к 6 наиболее часто используемым элементам (20 % старших) составили 74 % от общего количества обращений к файлу, а к 7 часто используемым элементам — 84 %. Эти величины были получены путем суммирования упорядоченных в порядке убывания суммарных частот использования элементов данных в приложениях пользователей.

Таблица 10.5. **Результаты теста базы данных персонального учета [213]**

Характеристики решения задачи сегментации данных

Относительная экономия для различных вариантов сегментации	
Несблокированный вторичный по сравнению с полностью сблокированным несегментированным	41,5%
Оптимальное блокирование по сравнению с полностью сблокированным несегментированным (Эйснер — Северанс)	55,3%
Оптимальное блокирование по сравнению с несблокированным вторичным (Марч—Северанс)	23,7%

Вариант сегментации	Распределение элементов данных по сегментам		Размер сегмента		Распределение буфера		
	Первичный	Вторичный	Первичный	Вторичный	Первичный	Вторичный	Цена в мес, долл.
Полностью сблокированный несегментированный	1, ..., 30		360	0	13 000	0	7305,76
Несблокированный вторичный сегмент (Эйснер — Северанс)	1, ..., 16, 22	17, ..., 21, 23, ..., 30	155	205	12 795	205	4278,22
Оптимальное блокирование (Марч — Северанс)	1, 2, 5, 9, ..., 12, 14	3, 4, 6, ..., 8, 13, 15, ..., 30	83	277	8 698	4302	3266,24

Суммарные затраты для варианта разделения (1, 2, 5, 9, 11, 12, 14), (3, 4, 6—8, 10, 13, 15—30) составляют 7282 долл., что близко к наихудшему случаю (приведенному в табл. 10.5); таким образом, правило "80/20" не является в этом случае хорошей эвристикой. Выбор только 6 наиболее часто используемых элементов не приводит к минимуму затрат. Однако видно, что на 8 старших по частоте использования элементов (27 %) приходится 94 % пользовательских обращений. Использование более восьми элементов ведет к значительному снижению эффективности: т. е. девяти наиболее часто используемым элементам соответствует 95 % обращений. Следовательно, для эффективного разбиения следует выбрать восемь самых старших элементов данных. Это подтверждается табл. 10.5. Решение этой задачи сводится к эвристической процедуре (если не удается предложить строгий алгоритм): для начального разбиения применить правило "80/20". Оценить это разбиение. Затем проверить чувствительность принятого решения к изменениям распределения элементов вблизи начального решения и остановиться, когда изменение частоты применений больше не является значительным. Выбрать самое лучшее разбиение из рассмотренных вариантов. Для этого величины SRS_1 и SRS_2 подставляют в выражение для BKS_1, берут производную и решают получившееся квадратное уравнение относительно единственной переменной BKS_1, определяя тем самым минимальное значение. Если правило "80/20" неприменимо, то требуется более тщательный анализ. Задачи, содержащие до 40 элементов данных и до 40 пользовательских приложений, решались довольно быстро с использованием алгоритма Марча — Северанса на ЭВМ CDC CYBER-74.

10.2.2. Алгоритм мощности связи

Более общим подходом, чем сегментация записи, является кластеризация элементов данных в интегрированных базах. Эта задача легко может быть переформулирована для кластеризации элементов данных в логических записях путем замены физических характеристик оценки производительности на количество обращений к логическим записям (LRA). Такой подход можно использовать для эффективной кластеризации записей в качестве альтернативы методам, обсуждаемым в гл. 11. *Алгоритм мощности связи* (BEA) — метод кластерного анализа, который был разработан для нахождения естественных группировок (кластеров), которые возникают в сложных массивах данных; он позволяет перестраивать строки и столбцы массива таким образом, чтобы сгруппировать вместе элементы больших массивов, облегчая тем самым их идентификацию [222]. В ра-

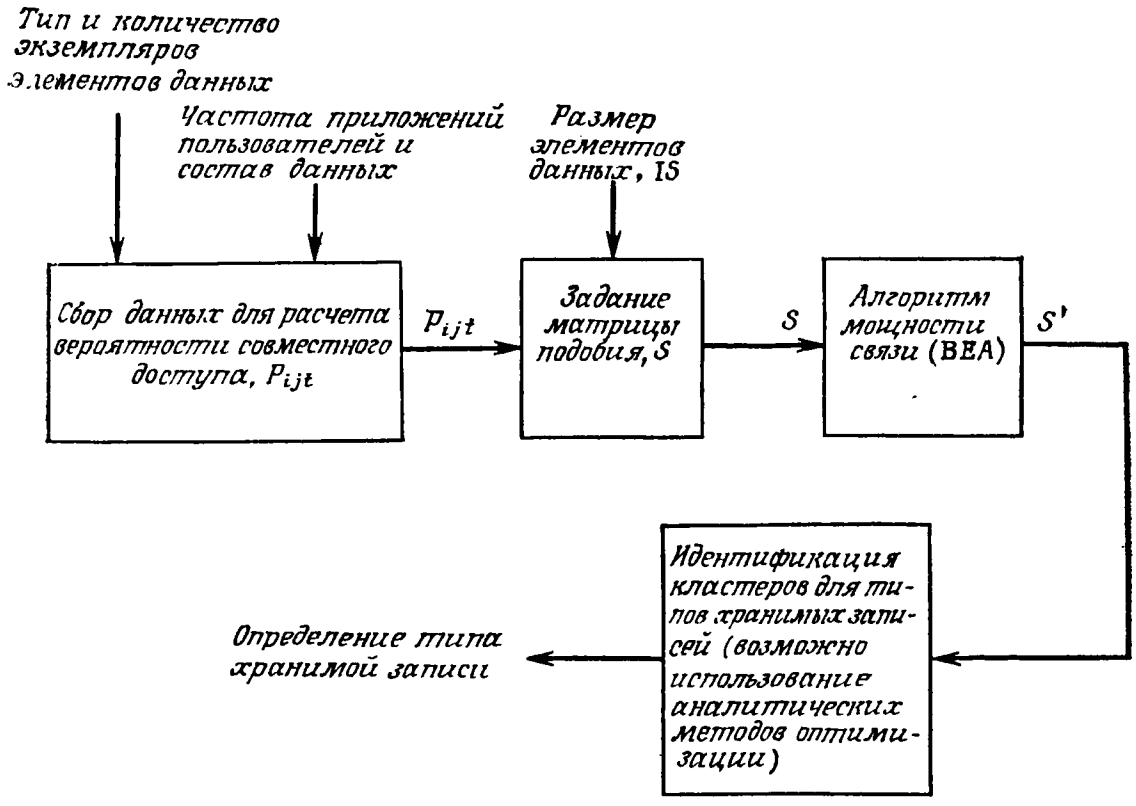


Рис. 10.9. Проектирование хранимой записи путем анализа кластеров.

ботах [152, 155, 78] описано применение данного алгоритма к решению задач разбиения записей.

Метод BEA основывается на том утверждении, что используемые совместно атрибуты данных следует хранить в одном и том же физическом подфайле. Если рассматривать совместное использование атрибутов в приложениях пользователей как форму некой связи, которая удерживает их вместе, то меру связности можно было бы вывести исходя из подобия атрибутов (элементов данных). BEA использует формальное понятие меры подобия пар атрибутов, основанное на их совместном использовании в приложениях. Значения мер подобия элементов данных образуют матрицу размерностью $m \times m$, где m — число типов элементов данных в базе данных.

Последовательность шагов, показанная на рис. 10.9, иллюстрирует место BEA и роль человека в процессе проектирования хранимых записей. Введение матрицы подобия S требует тщательного изучения приложений пользователей и методичного сбора данных, чтобы получить вероятности совместного использования каждой пары атрибутов i и j. Алгоритм преобразует матрицу S к виду S' с тем, чтобы максимизировать (или почти максимизировать) выбранную меру эффективности (т. е. подобие) для базы данных в целом. (BEA — это эвристический

алгоритм, обеспечивающий достижение квазиоптимального решения.) Проектировщик затем должен выделить кластеры из преобразованной матрицы подобия S', учитывая возможные ограничения на размеры хранимых записей, уровень безопасности данных и связи между ключевыми и неключевыми элементами. Следует отметить, что BEA рассматривается как метод квазиоптимизации, а весь процесс проектирования состава записи может варьироваться проектировщиком. Кроме того, на качество решения влияет точность оценок подобия атрибутов.

Сбор данных и определение параметров

Основным параметром, требуемым для кластеризации атрибутов, является P_{iu} — вероятность обращения к атрибуту i в приложении u. Эта вероятность вычисляется по формуле

$$P_{iu} = \frac{\mathrm{NVAL}_{iu}}{\mathrm{NIV}_i}, \tag{10-8}$$

где NVAL_{iu} — количество требуемых приложением u экземпляров элемента типа i, а NIV_i — количество экземпляров элемента типа i в базе данных. Данные для вычисления NVAL_{iu} и NIV_i получаются на основе экспертных оценок пользователя, использования аппаратных и программных мониторов или путем анализа программ ввода-вывода. Предпочтительно использовать третий способ, потому что он более точно определяет источник и объем данных, используемых в программах в отличие от первых двух способов, в которых сказывается субъективность оценок пользователя и служебные издержки времени в работе мониторов [76, 78].

Вывод функции подобия

Если известны вероятности P_{iu} и P_{ju} обращения к атрибутам i и j в приложении и, то вероятность совместного обращения к двум атрибутам в том же самом приложении будет

$$P_{iju} = \begin{cases} 1, & \text{если } P_{iu} = P_{ju} \neq 0, \\ 0, & \text{если } P_{iu} = 0 \text{ или } P_{ju} = 0, \\ \bar{P}_u & \text{в противном случае,} \end{cases} \tag{10-9}$$

где P_{iu} задается выражением

$$P_{iu} = \begin{cases} 1 & \text{для атрибутов, входящих в критерий выбора,} \\ \dfrac{\mathrm{NVAL}_{iu}}{\mathrm{NIV}_i} & \text{для атрибутов, предназначенных для выдачи (отображения),} \\ 0 & \text{для неиспользуемых атрибутов} \end{cases} \tag{10-10}$$

и $\bar{P}_u$ — наименьшее из P_{iu} и P_{ju}. Доля полезной для приложения u информации, выбираемой из подфайла, содержащего атрибуты типа i и j, выразится

$$C_{iju} = \begin{cases} 0, & \text{если } P_{iu} = P_{ju} = 0, \\ \dfrac{\mathrm{IS}_i \times \mathrm{P}_{iu} + \mathrm{IS}_j \times P_{ju}}{(\mathrm{IS}_i + \mathrm{IS}_j) \max (P_{iu}, P_{ju})} & \text{в противном случае.} \end{cases} \quad (10\text{-}11)$$

Если $P_{iu} = P_{ju}$, то выражение для C_{iju} упрощается

$$C_{iju} = \begin{cases} 0, & \text{если } P_{iu} = 0, \\ \dfrac{\mathrm{IS}_i + P_{iju} \times \mathrm{IS}_j}{\mathrm{IS}_i + \mathrm{IS}_j} & \text{в противном случае,} \end{cases} \quad (10\text{-}12)$$

где C_{iju} — представляет собой неубывающую функцию от P_{ijt}.

В качестве меры подобия наиболее просто использовать величину P_{iju}; с другой стороны, применение в качестве меры C_{iju} позволяет учесть совместное влияние размера атрибута и вероятности обращения к нему. В большинстве случаев при выборе других показателей подобия обязательным является требование, чтобы они являлись неубывающими функциями вероятности совместных обращений. Давайте примем в качестве меры подобия величину P_{iju} и построим матрицу S, вычисляя для каждого приложения u взвешенную в соответствии с частотой использования F_u сумму вероятностей совместных доступов:

$$S_{ij} = \begin{cases} \sum\limits_{u=1}^{U} F_u P_{iju} & \text{для } i \neq j, \\ \sum\limits_{u=1}^{U} F_u & \text{для } i = j. \end{cases} \quad (10\text{-}13)$$

Идентификация кластеров

Согласно [222], BEA выполняет следующие функции: определяет и группирует по подобию величины S_{ij} и определяет вторичные взаимосвязи между кластеризованными группами для возможного в дальнейшем группирования. Время выполнения алгоритма обозначим O (NIT^3), где NIT — количество типов элементов (или атрибутов) в базе данных. В алгоритме используется в качестве оценки мощности соседних связей величина, равная

$$\mathrm{BOND} = (1/4) \sum_{i=1}^{\mathrm{NIT}} \sum_{j=1}^{\mathrm{NIT}} S_{ij} (S_{i,j-1} + S_{i,j+1} + S_{i+1,j} + S_{i-1,j}), \quad (10\text{-}14)$$

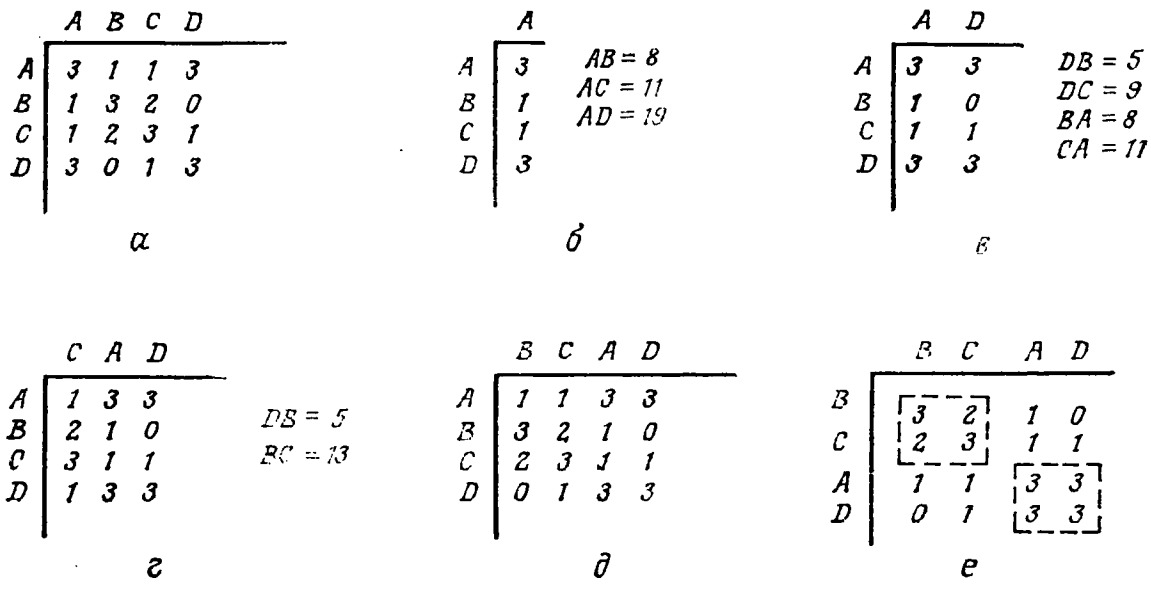


Рис. 10.10. Алгоритмы мощности связи базы данных из 4 атрибутов.

а — матрица *S*; *б* — исходный столбец; *в* — два столбца; *г* — три столбца; *д* — четыре столбца, законченная перестановка; *е* — законченная перестановка строк и столбцов.

где $S_{i,0} = S_{0,j} = S_{i,\,\mathrm{NIT}+1} = S_{\mathrm{NIT}+1,\,j} = 0$ для всех i и j. Алгоритм группирует подобные атрибуты, переставляя (обменивая местами) строки и столбцы матрицы S так, чтобы максимизировать величину BOND. Таким образом, если значение S_{ij} большое, рядом стоящие значения будут большими, и если значение S_{ij} мало, то оно будет окружено малыми величинами.

Уравнение (10-14) может быть упрощено, если учесть, что мера парного подобия симметрична (т. е. $S_{ij} = S_{ji}$), и приведено к виду:

$$\mathrm{BOND} = (1/2) \sum_{i=1}^{\mathrm{NIT}} \sum_{j=1}^{\mathrm{NIT}} S_{ij} (S_{i,\,j-1} + S_{i,\,j+1}). \qquad (10\text{-}15)$$

Рис. 10.10 иллюстрирует основные шаги алгоритма. На рис. 10.10, *а* матрица S задана для атрибутов A, B, C и D. На первом шаге выбирается произвольно один из атрибутов (столбцов), в нашем случае — атрибут A. Оставшиеся NIT — 1 атрибутов проверяются на смежность с A и вычисляется мощность связи пар $\sum_{i=1}^{\mathrm{NIT}} S_{iA} S_{ix}$ для $x = B, C, D$. Отбираются столбцы с самой большой мощностью связей [см. столбец D, рис. 10.10, *б*]. Затем анализируются оставшиеся NIT — 2 атрибутов, расположенных по обе стороны от выбранных столбцов, и снова выбирается расположение с максимальным значением мощности связи между парами (рис. 10.10, *в*). Перестановка

столбцов может выполняться в произвольном или в последовательном порядке, или путем направленного перебора, обеспечивающего пошаговую максимизацию на каждом этапе. Окончательный вариант перестановки столбцов показан на рис. 10.10, *д*. На окончательном этапе выполняется перестановка строк так, чтобы обеспечить полную симметрию матрицы (рис. 10.10, *е*). Полное преобразование матрицы из NIT столбцов потребует NIT(NIT — 1) вычислений. Это показано в табл. 10.6 для NIT = 4 и для расчета BOND по формуле (10-15). BEA значительно сокращает объем работ по нахождению эффективного размещения строк и столбцов в матрице подобия.

Таблица 10.6. **BOND-величины для всех неизбыточных 4-атрибутных размещений**

Парные связи	4-атрибутные	BOND-величины, уравнение (10-15)	
$AB=BA=8$	$ABCD=30$	$BACD=28$	
$AC=CA=11$	$ABDC=22$	$BADC=36$	
$AD=DA=19$	$ACBD=29$	$BCAD=43$	(оптимальное)
$BC=CB=13$	$ACDB=25$	$BDAC=35$	
$BD=DB=5$	$ADBC=37$	$CABD=24$	
$CD=DC=9$	$ADCB=41$	$CBAD=40$	

Теперь задача состоит в выделении квадратов (как показано на рис. 10.10, *е*) вдоль главной диагонали так, чтобы сгруппировать совместно используемые атрибуты. Числа в пределах квадрата должны резко отличаться в большую сторону по сравнению с числами, расположенными вне его. В данном примере объединяются вместе атрибуты *B* и *C*, а также *A* и *D* в два подфайла, предлагая, что в дальнейшем никакие ограничения не нарушат это разбиение. В данном примере выбран простейший вид области разбиения, хотя в общем случае проектировщик может задавать их форму произвольно. С другой стороны, эксперименты показали, что для сложных баз данных более подробный анализ приводит к незначительным изменениям в определении кластеров [78]. Квадраты могут быть образованы не обязательно строго вдоль диагонали. В этом случае они могут быть в дальнейшем связаны с другими группами, расположенными на главной диагонали, и могут быть объединены в составные квадраты. Элементы данных, расположенные вдоль главной диагонали, являются основными кандидатами для включения в подфайл. Возможно также значительное

наложение квадратов, и иногда целесообразно объединить перекрывающиеся квадраты, чтобы получить большие и более эффективные кластеры. Более подробный анализ может потребовать рассмотрения бóльшего числа вариантов, вручную или с использованием программы моделирования, а также методов оптимизации [78, 152].

10.2.3. Эвристическая модель разбиения

Другой эвристический подход к разбиению записей был предложен в работе [145]. Хотя цели проектирования остаются теми же самыми, что и в рассмотренном подходе (т. е. получение субоптимального разбиения записи, основанного на совместном использовании элементов данных в приложениях пользователей), процесс проектирования в значительной степени автоматизируется. Для этого разработан специальный пакет программ, который оценивает возможные варианты разбиения, исходя из оценки количества физических обращений, требуемых для выполнения приложений пользователей. Вычислительная среда представлена моделью многопользовательской реляционной системы баз данных с виртуальной (страничной) организацией памяти и соответствующим техническим и программным обеспечением. Была предпринята попытка смоделировать эффективную методику обработки транзакций в реляционных системах.

Эвристический процесс начинается с выделения какого-либо возможного варианта разбиения, а затем путем пошаговой минимизации количества обращений к страницам памяти осуществляется продвижение к оптимальной точке. Как и в методике BEA, на первом этапе рассматриваются парные группы, но уже на уровне блоков атрибутов. Все атрибуты объединены в блоки таким образом, чтобы блоки попарно не пересекались. Второй этап нацелен на улучшение параметров и продолжается до тех пор, пока никакие вариации не могут улучшить характеристик разбиения. Эти два этапа выполняются поочередно несколько раз, пока полный цикл оптимизации уже не дает никакого улучшения. Быстрая версия этого метода является более быстродействующей, чем BEA [т. е. $O(m^2)$], и позволяет получить почти оптимальное решение.

Глава 11. Кластеризация записей

Вариант загрузки хранимых записей в базу данных зависит от разработчика и характеристик конкретной СУБД. При этом, как уже отмечалось, в первую очередь учитывается логическая упорядоченность данных; физическая упорядоченность, или степень смежности данных, также является важным фактором при выборе варианта загрузки. Физическая упорядоченность влияет на процесс объединения хранимых записей в физические экстенты, который называется *кластеризацией записей*. Он отличается от сегментации записей, при которой элементы данных распределяются по блокам различной длины и по различным экстентам (см. гл. 10), хотя и в случае сегментации, и в случае кластеризации сами экстенты, для того чтобы уменьшить время ввода-вывода, могут быть распределены по различным физическим устройствам. В этой главе рассматривается кластеризация записей в иерархических и сетевых логических структурах баз данных.

Задача кластеризации состоит в определении того, каким образом для заданной модели данных необходимо хранить записи базы данных с тем, чтобы для типичных эксплуатационных нагрузок минимизировать время доступа. Обычно время доступа оценивается по времени ввода-вывода и времени обслуживания CPU; однако мы будем полагать, что для обычных процедур поиска, реализованных в современных системах, время процессора пренебрежимо мало (для оценки времени процессора см. гл. 16). Нам будет достаточно оценить характеристики обращений к физическим блокам и затем воспользоваться результатами гл. 9 для вычисления времени обслуживания ввода-вывода.

Правильной кластеризацией можно добиться значительного повышения эффективности функционирования системы баз данных, так как при этом наиболее часто используемые группы данных физически размещаются так, чтобы в максимальном числе случаев к ним был обеспечен последовательный доступ и до минимума были сокращены случаи произвольной выборки. Например, известно, что для системы регистрации студентов в одном из ведущих университетов было получено сокращение времени обслуживания ввода-вывода в 5 раз благодаря рациональному размещению данных. В общем случае оптимальную кластеризацию трудно осуществить для сложных

интегрированных баз данных, где данные, к которым осуществляется обращение с помощью различных прикладных программ, перекрываются, и не легко найти компромисс между последовательным и произвольным методами доступа. Однако последние исследования сделали возможным решение проблемы кластеризации в иерархических базах данных и по крайней мере применение эвристического подхода для сетевых структур.

11.1. Кластеризация в иерархических базах данных

Прежде всего рассмотрим вопросы кластеризации записей для наиболее распространенных на практике иерархических баз данных, таких, как IMS, а затем на их основе сформулируем более общие требования к размещению данных в сетевых базах. Простой и достаточно быстрый метод решения данной

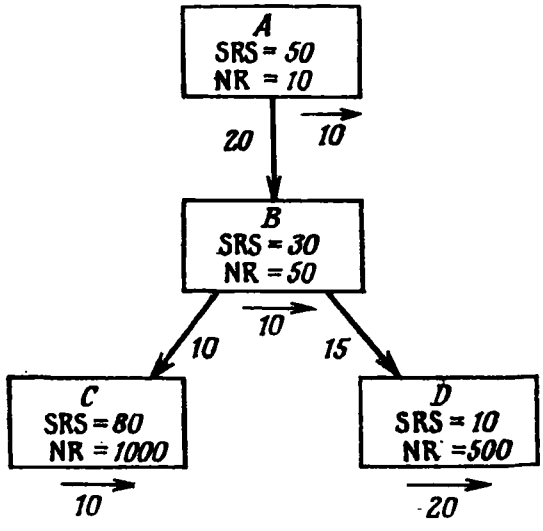


Рис. 11.1. Структура иерархической базы данных.

SRS — размер хранимой записи (в байтах); NR — число экземпляров записи; ← x показывает количество переходов между двумя логически смежными экземплярами записи, выполненных в данном направлении.

задачи был описан в литературе [270, 271]. Этот метод широко применяется, поэтому мы подробно проанализируем принцип работы и перспективы его использования.

Рассмотрим иерархическую структуру, изображенную на рис. 11.1. Каждый прямоугольник представляет собой тип записи, а стрелки показывают связи типа $1 : m$ между ними, образуя иерархию. Внутри каждого прямоугольника указаны размер хранимой записи (SRS) и общее число экземпляров данного типа (NR). Для простоты предположим, что число экземпляров порожденных записей распределено равномерно относительно исходных записей так, что, например, для каждой исходной записи *A* имеется в среднем пять экземпляров записей типа *B*. Распределения, более близкие к реально существующим, рассмотрим после того, как будет изучена простая модель [278]. Стрелки и цифры указывают направление и количество переходов в единицу времени между экземпляра-

ми соответствующих типов записей, осуществляемое при выполнении приложений базы данных.

Горизонтальные стрелки обозначают переход между экземплярами записей одного типа, а вертикальные стрелки обозначают переход от исходного типа записи к порожденному типу записи. Количество этих переходов может быть легко найдено, если знать либо последовательность операций в базе данных, либо статистические данные по доступу (за день, за час и т. п.). Пусть, например, $F(A, A)$ — частота перехода от одного экземпляра записи типа A к другому. Команда ПОЛУЧИТЬ ВСЕ B дает в результате $F(A, A) = 10$, $F(A, B) = 10$ и $F(B, B) = (5 - 1) \cdot 10 = 40$. Все другие частоты равны нулю. Величина $F(A, A)$ соответствует первоначальному доступу к корневой записи и всем последующим обращениям к экземплярам записей этого типа. Для каждого экземпляра записи A путь AB выбирается один раз. В конечном итоге для каждого экземпляра записи B, выбранного вслед за экземпляром записи типа A, последовательно выбирается четыре экземпляра записи типа B. Основное предположение, которое делается при этих вычислениях, состоит в том, что записи данных хранятся в иерархически упорядоченной последовательности A_1, $B_{1,1}$, $C_{1,1,1}$, ..., $C_{1,1,20}$, $D_{1,1,1}$ ..., $D_{1,1,10}$, $B_{1,2}$, $C_{1,2,1}$, ..., $C_{1,2,20}$, $D_{1,2,1}$, ..., $D_{1,2,10}$, $B_{1,3}$, ..., $B_{1,10}$, ..., A_2, ...; и указатели обеспечивают непосредственный переход от экземпляра исходного типа записи к первому экземпляру порожденного типа записи и от последнего экземпляра порожденного типа записи к следующему экземпляру исходного типа записи.

Задача кластеризации записей в иерархической базе данных состоит в определении оптимального размещения типов записей по смежным экстентам (например, по группам наборов данных в IMS или по областям в системах CODASYL), которое минимизирует число обращений к внешней памяти для ввода-вывода данных при заданном наборе приложений пользователей. В экстенте могут содержаться один или несколько типов записей, причем в нем должны содержаться все экземпляры соответствующего типа записи. Предполагается, что в пределах каждого экстента записи хранятся в иерархическом порядке так, что каждое возможное группирование записей в экстенте должно образовать в результате новую физическую последовательность записей. Например, если записи типа A и B объединены в группу, а C и D размещены отдельно, то в результате в этом кластере образуются три группы: 1. A_1, $B_{1,1}$, ..., $B_{1,10}$, A_2, ...; 2. $C_{1,1,1}$, $C_{1,1,2}$, ..., $C_{10,5,20}$; 3. $D_{1,1,1}$, $D_{1,1,2}$, ..., $D_{10,5,10}$. Когда $F(B, B)$ очень велико, как в команде ПОЛУЧИТЬ ВСЕ B, этот кластер из трех групп будет более эффективен, чем первоначальный кластер, содер-

жащий одну группу записей в отдельном экстенте. В исходном кластере экземпляры записи типа B разбросаны среди экземпляров записей C и D, а во втором случае экземпляры записи типа B сгруппированы и размещены в физически последовательном порядке. Исходная кластеризация могла бы быть более эффективной, если для получения отчета требовалось бы выбрать все данные в их обычной иерархической последовательности.

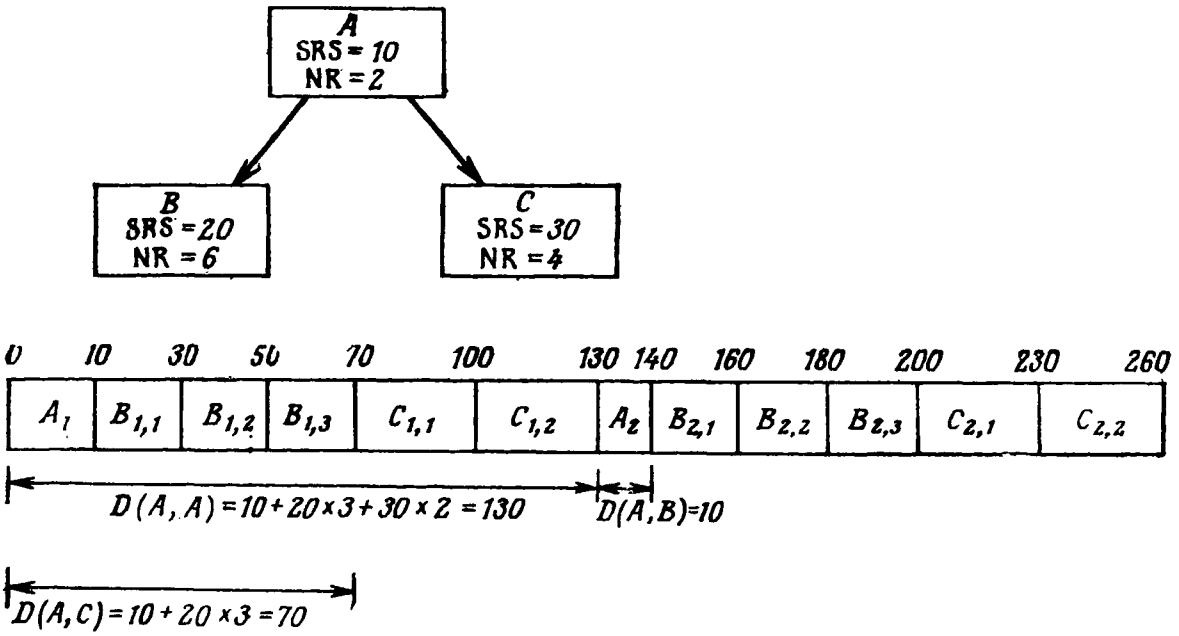


Рис. 11.2. Основные функции расстояния в иерархической базе данных.

Определение оптимальной кластеризации записей в физических экстентах включает, во-первых, нахождение оценки величины $D(I, J)$, характеризующей среднее расстояние между экземпляром записи типа I и первым экземпляром порожденной записи типа J, а также величины $D(I, I)$, характеризующей среднее *расстояние* между двумя соседними экземплярами записи типа I, при условии нахождения записей типа I и J в одной группе:

$$D(I, I) = \mathrm{SRS}(I) + \sum_{J \in S_1} D(J, J) \times \mathrm{ANC}(I, J), \qquad (11\text{-}1)$$

где

$$\mathrm{ANC}(I, J) = \mathrm{NR}(J)/\mathrm{NR}(I), \qquad (11\text{-}2)$$

$$D(I, J) = \mathrm{SRS}(I) + \sum_{K \in S_2} D(K, K) \times \mathrm{ANC}(I, K). \qquad (11\text{-}3)$$

В уравнении (11-1) величина $D(I,I)$ представляет собой длину записи типа I плюс сумму расстояний длин всех экземпляров подчиненных ей типов записей. S_1 — множество типов записей,

непосредственно порожденных записью типа I; суммирование производится по всем типам записей J, которые имеются в S_1; ANC(I, J) — среднее число порожденных записей типа J, приходящихся на один экземпляр записи типа J. В уравнении (11-3) $D(I, J)$ есть сумма длин экземпляра записи типа I и суммарного расстояния (длин экземпляров) подчиненных записей типа K, принадлежащих S_2. S_2 — множество записей типа K, непосредственно порожденных записью типа I и расположенных в иерархической структуре левее записи типа J, находящихся в одной с записями типов I и J группе. Функции расстояния для полностью упорядоченной иерархической базы данных показаны на рис. 11.2.

Среднее количество обращений к физическим блокам, связанных с переходом от I к J, можно записать

$$\mathrm{PBA}(I, J) = \begin{cases} \min[D(I, J)/\mathrm{EBKS}, 1], & \text{если записи типа } I, J \text{ принадлежат одной группе,} \\ 1, & \text{если записи принадлежат разным группам,} \end{cases} \quad (11\text{-}4)$$

где EBKS — полезный размер блока. Величина EBKS характеризует используемую часть блока и выражается через коэффициент блокирования EBF следующим образом:

$$\mathrm{EBKS} = \mathrm{EBF} \times \mathrm{SRS}. \quad (11\text{-}5)$$

Среднее число обращений к блоку при переходе от одного экземпляра записи к другому меньше 1, если расстояние между записями меньше, чем полезный размер блока. Если расстояние больше, чем размер блока, то потребуется самое большее один доступ к блоку. Школьник [270] решает эту задачу для страничной организации памяти с произвольным доступом (rba), когда иерархические указатели позволяют переходить непосредственно к следующему блоку без просмотра промежуточных. Для нестраничной организации памяти оценку расстояния между блоками желательно охарактеризовать с помощью некоторой модели, что потенциально более эффективно, чем просто выбрать произвольное значение. Для дальнейшего анализа мы будем предполагать страничную организацию, и, если читатель захочет ознакомиться, как связаны временные характеристики и расстояния, ему следует обратиться к гл. 9.

Общее количество обращений к блокам для определенной кластеризации записей в экстенте можно выразить

$$\mathrm{PBA}(\text{кластера}) = \sum_{I \in S} \left[F(I, I) \times \mathrm{PBA}(I, I) + \sum_{J \in S_1} F(I, J) \times \mathrm{PBA}(I, J) \right], \quad (11\text{-}6)$$

где S — множество всех типов записей I в кластере, а S_I — множество типов записей, являющихся непосредственно порожденными записью типа I. Общее количество обращений к блоку при переходах между экземплярами записей каждого типа можно оценить частотой этих переходов. Применив выражение (11-6) к иерархической базе данных, приведенной на рис. 11.1, в случае единственного кластера получим

$$\begin{aligned}\text{РВА (кластера)} = {} & F(A, A) \times \text{РВА}(A, A) + F(A, B) \times \\ & \times \text{РВА}(A, B) + F(B, B) \times \text{РВА}(B, B) + F(B, C) \times \text{РВА}(B, C) + \\ & + F(B, D) \times \text{РВА}(B, D) + F(C, C) \times \text{РВА}(C, C) + \\ & + F(D, D) \times \text{РВА}(D, D). \qquad (11\text{-}7)\end{aligned}$$

Для других баз данных мы можем получить соответствующие оценки аналогичным образом, однако, чтобы проиллюстрировать эффективность предложенного метода решения, полезно переписать выражение (11-6) в следующем виде:

$$\begin{aligned}&\text{РВА (кластера)} = C(R) = \\ &= F(R, R) \times \text{РВА}(R, R) + \sum_{J \in S_I} [F(R, J) \times \text{РВА}(R, J) + C(J)], \quad (11\text{-}8)\end{aligned}$$

где $C(R)$ — стоимость доступа к дереву, составляющему кластер с корнем R, измеренная количеством обращений к физическим блокам. S_I — множество типов записей, непосредственно порожденных записью типа R. $C(J)$ определяет ту часть общей стоимости $C(R)$, которая соответствует поддереву с корнем J. Отметим, что стоимость обращения к корню дерева соответствует стоимости обращения ко всему дереву. Заметим также, что уравнение (11-8) является рекуррентным и стоимость обращения к древовидной структуре представляет собой линейную функцию стоимостей обращения к составляющим эту структуру поддеревьям. Такое правило возможно только для иерархических (а не сетевых) структур, в которых члены каждой группы в кластере должны находиться друг с другом в отношении «исходный — порожденный». Несмотря на то, что другие способы группирования являются допустимыми, например для образования вторичных групп наборов данных IMS, они, как правило, не используются из-за неэффективности переходов. На рис. 11.3 показаны варианты группирования записей. Применяя выражение (11-8) к примеру, представленному на рис. 11.1, получим

$$\text{РВА (кластера)} = C(A), \qquad (11\text{-}9)$$

где

$$C(A) = F(A, A) \times \text{PBA}(A, A) + F(A, B) \times \text{PBA}(A, B) + C(B), \quad \text{(11-9а)}$$

$$C(B) = F(B, B) \times \text{PBA}(B, B) + F(B, C) \times \text{PBA}(B, C) + C(C) + F(B, D) \times \text{PBA}(B, D) + C(D), \quad \text{(11-9б)}$$

$$C(C) = F(C, C) \times \text{PBA}(C, C), \quad \text{(11-9в)}$$

$$C(D) = F(D, D) \times \text{PBA}(D, D). \quad \text{(11-9г)}$$

Используя выражение (11-9), а в общем случае — выражение (11-8), можно вычислить значение PBA (кластера), начав вы-

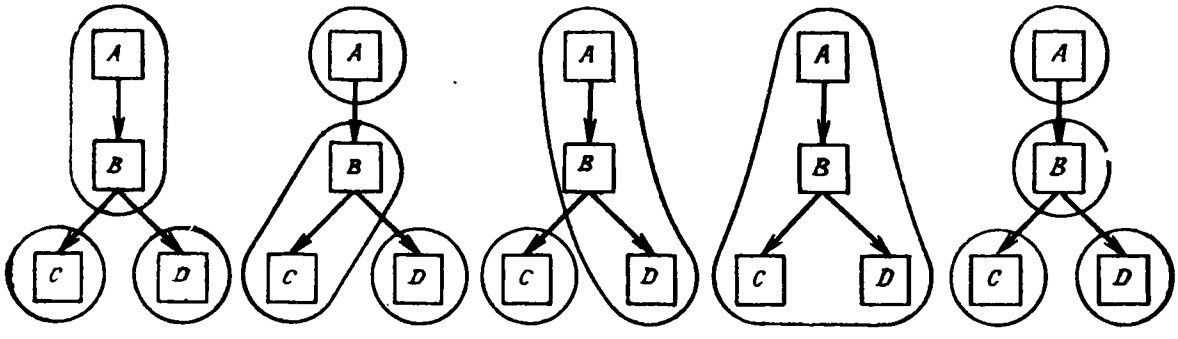


Рис. 11.3. Примеры группирования записей для формирования кластеров.

числения с терминальных вершин дерева, образующего кластер, и получая промежуточные результаты по мере передвижения снизу вверх.

Давайте обозначим кластер как двоичную функцию пар типов записей в базе данных. Например, определим A и B как первую возможную пару. Если A и B сгруппированы вместе, то паре AB присваивается значение 1, в противном случае — значение 0.

Теперь для рассматриваемой структуры (рис. 11.1) можно вычислить общую стоимость обращений к физическим блокам для всех анализируемых вариантов кластеров. Система обозначений пар записей может быть упрощена путем составления списка пар типов записей и использования двоичного представления: AB, BC, BD. Три пары и все их комбинации могут быть представлены трехразрядным двоичным числом. Например, число 100 соответствует $AB = 1$ (сгруппированы), $BC = 0$ (несгруппированы) и $BD = 0$ (несгруппированы). Взятый как целое кластер 100 представляет следующее группирование типов записей AB, C, D. Цель введения двоичных обозначений — облегчить представление групп записей в больших сложных базах данных, где было бы утомительным перечислять все

возможные группы записей по их буквенным обозначениям или именам; кроме того, двоичное исчисление облегчает реализацию вычислительных процедур.

Вычислим стоимость обращений для двух возможных кластеров для структуры, представленной на рис. 11.1, предполагая, что EBKS = 1000 байт.

Кластер ABCD (111):

$C(C) = F(C, C) \min [D(C, C)/\text{EBKS}, 1] = 10 \min (80/1000, 1) = 10 \times \times 0{,}08 = 0{,}8;$

$C(D) = F(D, D) \min [D(D, D)/\text{EBKS}, 1] = 20 \min (10/1000, 1) = = 20 \times 0{,}01 = 0{,}2;$

$C(B) = F(B, B) \times \text{PBA}(B, B) + F(B, C) \times \text{PBA}(B, C) + C(C) + + F(B, D) \times \text{PBA}(B, D) + C(D) = 10 \times \min (30 + 20 \times 80 + 10 \times \times 10/1000{,}1) + 15 \min (30/1000{,}1) + 0{,}8 + 15 \min (30 + 20 \times \times 80/1000{,}1) + 0{,}2 = 10 \times 1 + 15 \times 0{,}03 + 0{,}8 + 15 \times 1 + + 0{,}2 = 26{,}45;$

$C(A) = F(A, A) \times \text{PBA}(A, A) + F(A, B) \times \text{PBA}(A, B) + C(B) = = 10 \min (50 + 5 \times 30 + 5 \times 20 \times 80 + 5 \times 10 \times 10/1000{,}1) + + 20 \min (50/1000{,}1) + 26{,}45 = 10 + 20 \times 0{,}05 + 26{,}45 = 37{,}45;$

PBA (кластера $ABCD$) $= C(A) = 37{,}45$ rba.

Кластер AB, C, D (100):

$C(C) = 0{,}8; \quad C(D) = 0{,}2;$

$C(B) = 10 \min (30/1000{,}1) + 15 \times 1 + 0{,}8 + 15 \times 1 + 0{,}2 = 0{,}3 + + 15 + 0{,}8 + 15 + 0{,}2 = 31{,}3;$

$C(A) = 10 \min (50 + 5 \times 30/1000{,}1) + 20 \times 0{,}05 + 31{,}3 = 2 + 1 + + 31{,}3 = 34{,}3;$

PBA (кластер AB, C, D) $= C(A) = 34{,}3$ rba.

В табл. 11.1 приведены значения затрат для всех допустимых кластеров иерархической структуры, данной на рис. 11.1. Оптимальное значение затрат соответствует кластеру *ABD, C* (101) и дает 24 %-ный выигрыш в количестве обращений к блокам по сравнению со следующим лучшим кластером (100) и 50 %-ный выигрыш по сравнению с наихудшим случаем (000). Таким образом, показано, что для заданных частотных характеристик приложений и известной логической структуры базы

Таблица 11.1. **Среднее расстояние $D(A, A)$ и общие затраты $C(A)$ в rba для возможных вариантов кластеризации иерархической структуры (рис. 11.1)**

Кластеры	Двоичный код	$D(A, A)$	$C(A)$
A, B, C, D	000	50	51,80
A, BD, C	001	50	38,45
A, BC, D	010	50	46,95
A, BCD	011	50	46,95
AB, C, D	100	200	34,30
ABD, C	101	700	25,95
ABC, D	110	8200	37,45
ABCD	111	8700	37,45

данных методом простого перебора можно легко найти оптимальный вариант кластеризации. К сожалению, число допустимых кластеров в иерархической базе данных с n типами записей равно 2^{n-1}; так, например, при $n = 21$ существует более 10^6 возможных кластеров. Вручную можно проанализировать систему не более чем с шестью типами записей, а применение вычислительных средств позволит увеличить это число до 25—35 типов записей в зависимости от мощности ЭВМ.

В [270] предложена более быстрая процедура, время работы которой пропорционально n. Если принять во внимание, что дерево с корнем x может рассматриваться как поддерево некоторого большого дерева с корнем y, то можно разработать эвристическую процедуру, которая будет исключать из дальнейшего анализа заведомо неоптимальные кластеры поддерева с корнем x в составе дерева с корнем y. Можно показать, что кластер, для которого значения $D(x, x)$ и $C(x)$ равны или больше, чем аналогичные значения другого кластера в x, из дальнейшего рассмотрения для данного поддерева можно исключить. Так, в нашем примере можно избежать полного перебора всех восьми кластеров, если начать анализ с поддерева с корнем B. Результаты приведены в табл. 11.2.

Таблица 11.2. **Среднее расстояние и общие затраты для поддерева B (рис. 11.1)**

Кластеры	Двоичный код	$D(B, B)$	$C(B)$	Операция
B, C, D	00	30	31,30	Хранить
BD, C	01	130	17,95	Хранить
BC, D	10	1630	26,45	Уничтожить
BCD	11	1730	26,45	Уничтожить

Кластеры 10 и 11 исключаются из рассмотрения из-за того, что

$$D(B,\ B)_i \geqslant D(B,\ B)_j \text{ и } C(B)_i \geqslant C(B)_j \qquad (11\text{-}10)$$

для некоторого кластера j.

Возвращаясь к табл. 11.1 видно, что все кластеры, код которых оканчивается на 10 или 11, рассматривать в дальнейшем нет необходимости. Это означает, что для дерева с корнем A нужно проанализировать только кластеры 000, 001, 100 и 101. Для больших деревьев такой процесс исключения дает значительную экономию времени вычисления. Утверждается, что быстрый алгоритм кластеризации является достаточно гибким и позволяет анализировать более общие виды распределения количества экземпляров порожденных записей, приходящихся на один экземпляр исходной, нежели среднее значение. Кроме того, алгоритм допускает использование более общих функций затрат, учитывающих объем памяти и зависимость времени ввода-вывода от распределения устройств.

Коэффициент блокирования BF (или EBF) определяет полезный размер блока, EBKS, который непосредственно используется при вычислениях [см. уравнение (11-4)]. Таким образом, коэффициент блокирования оказывает значительное влияние на выбор варианта кластеризации записей. Например, очень маленький размер блока сводит на нет преимущества кластеризации записей, так что даже при страничной организации памяти не получается никакого улучшения от кластеризации по сравнению с простейшим вариантом группирования. Для случая когда каждый кластер может иметь различный размер блока, значительно возрастает вычислительная сложность алгоритма кластеризации записей.

11.2. Кластеризация в базах данных с сетевой структурой

Метод кластеризации записей, который мы анализировали для иерархических баз данных, с некоторыми ограничениями может быть применен и к сетевым структурам. Мы рассмотрим модель данных CODASYL [80], которая представляет широко используемый класс сетевых СУБД. Структура физической базы CODASYL описывается на Языке описания хранения данных (ЯОХД), который описан в CODASYL *Journal of Development* [84]. Существующие варианты СУБД, основанные на предложениях CODASYL, не обязательно обеспечивают все возможности, заложенные в ЯОХД. Однако принятое CODASYL разделение спецификаций структуры базы данных на схемы, описываемые на ЯОД, и схемы хранения (описываемые на ЯОХД) обеспечивает удобный и практичный способ

избежать двусмысленности в определении логической и физической структуры базы данных. В терминологии CODASYL на этапе проектирования реализации получается схема, описываемая на ЯОД, а после физического проектирования — схема хранения, описываемая на ЯОХД.

В соответствии со спецификациями ЯОХД CODASYL размещение записи определяется предложением PLACEMENT. Для каждого типа записей может быть выбран один из трех типов спецификации:

1. CALC. Значения одного или нескольких элементов данных используются для вычисления адреса хранимой записи.
2. CLUSTERED VIA SET s. Каждый новый экземпляр r_1 члена типа набора s размещается рядом с уже существующим экземпляром r_2 члена того же самого экземпляра типа набора s. r_1 размещается вблизи r_2 в соответствии с логической упорядоченностью экземпляров набора.
3. SEQUENTIAL. Экземпляры записей хранятся в физически последовательном порядке в соответствии с возрастанием или убыванием значений определенного элемента данных.

Эти варианты являются взаимоисключающими. Каждый возможный вариант размещения данной записи типа r специфицирует физическое размещение экземпляров записи типа r и соответствующий логический путь доступа к экземплярам записей этого типа: прямой логический путь доступа для CALC, последовательный логический путь доступа для SEQUENTIAL или логический путь доступа через тип набора для CLUSTERED. Выбирая соответствующий вариант размещения, разработчик базы данных задает предпочтительный путь логического доступа к экземплярам записи.

Если специфицировано предложение CLUSTERED VIA SET, то фразы OWNER и DISPLACEMENT позволяют разработчику определить взаиморасположение экземпляров записи-члена типа набора s и соответствующих экземпляров записи-владельца экземпляров данного типа набора. Разработчик может сделать так, что кластеризованные записи-члены были помещены либо рядом с записью-владельцем (NEAR OWNER), либо с точно определенным смещением от записи-владельца (DISPLACEMENT), либо независимо от расположения записи-владельца. Если запись типа r специфицирована как CLUSTERED VIA SET, но без указания NEAR OWNER, то каждый новый хранимый экземпляр r будет размещаться около других экземпляров записи r, которые относятся к тому же самому экземпляру типа набора s, но этот кластер связанных экземпляров записи r не будет расположен рядом с экземпляром записи-владельца. На рис. 11.4 иллюстрируется влияние фразы NEAR OWNER на размещение хранимой записи типа y,

специфицированной предложением CLUSTERED VIA SET, где через x обозначен тип записи-владельца типа набора s [244].

Кластеризованные записи могут быть распределены по областям, каждая из которых обычно соответствует файлу и состоит из смежных экстентов внешней памяти. Обычно единицей обмена между базой данных и буферами СУБД является страница. Все страницы области имеют одинаковый размер, хотя различные области могут иметь различный размер страниц. Предполагается, что экземпляры записи не могут пересекать границы страницы. Проектирование области состоит в выборе размера и количества страниц. Если к области осуществляется последовательное обращение, то желательно на этапе

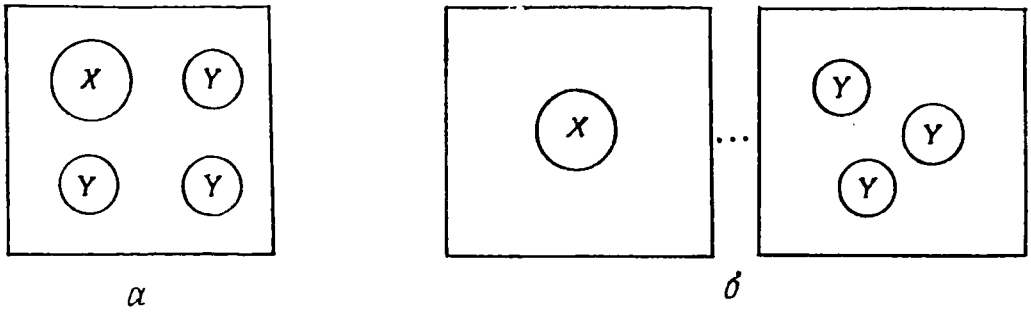


Рис. 11.4. Влияние фразы NEAR OWNER на физическое размещение.
а — NEAR OWNER, специфицировано; *б* — NEAR OWNER не специфицировано.

первоначальной загрузки в каждой странице резервировать память для последующих включений. В общем случае между областями и типами записей имеет место отношение типа $m : n$. Экземпляры записей типа A могут быть расположены в разных областях, но любой экземпляр записи хранится только в одной области. В данной области могут храниться экземпляры записей разных типов.

Задача выбора числа и содержания областей явно связана с обсуждаемой выше проблемой размещения записей. Для того чтобы СУБД могла выполнить правила размещения, определенные для различных типов записей, необходимо, чтобы кластеры типов записей, связанных посредством объявленных в спецификациях размещения CLUSTERED VIA SET NEAR OWNER типах набора, хранились в пределах той же самой области (или областей). Типы записей, которые не связываются подобной спецификацией, могут храниться в различных областях.

Рис. 11.5 иллюстрирует предлагаемое CODASYL размещение записей и определение области на примере кластеризации иерархической модели. Когда кластеризация специфицируется фразой NEAR OWNER (рис. 11.5, *а*), типы записей A и B хранятся в иерархической последовательности, образуя клас-

тер *AB*. Если фраза NEAR OWNER (рис. 11.5, *б*) не используется, записи типов *A* и *B* образуют отдельные кластеры *A* и *B*, хотя в обоих случаях кластеризация поддерживается в одной области. Ввиду того что проектные решения относительно кластеризации и областей хранения принимаются независимо, то становится возможным использование пошагового эвристического алгоритма для размещения записей:

• **Шаг 1.** Используя методику, предложенную в [270, 271], найти «вспомогательную» кластеризацию: сначала разложить сетевую структуру на иерархические составляющие (возможно,

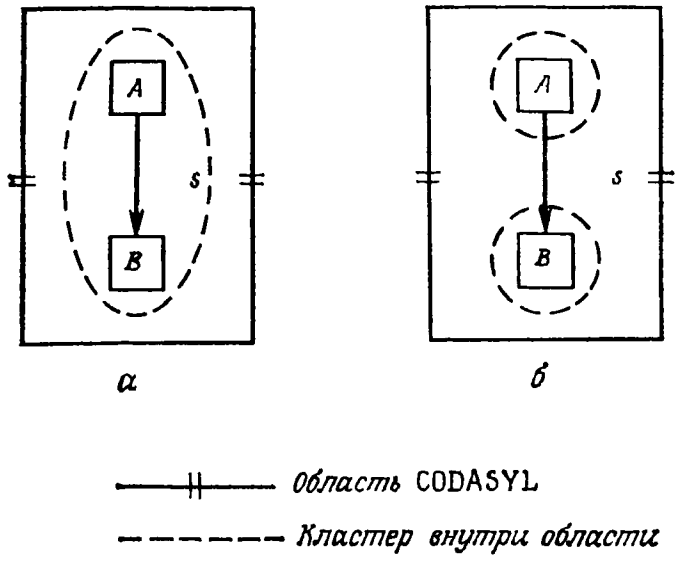


Рис. 11.5. Варианты кластерного размещения и области CODASYL.

а — с использованием фразы NEAR OWNER; *б* — без использования фразы NEAR OWNER.

перекрывающиеся). Затем найти оптимальную кластеризацию каждой иерархической составляющей.

• **Шаг 2.** По заданным оптимальным «вспомогательным» кластерам и с учетом их частичного совпадения оценить возможные варианты «основной» кластеризации в пределах областей хранения. Сократить пространство возможных решений путем рассмотрения доминирующих приложений пользователей и установления верхней границы размера физической области. Основная идея проектирования сводится к назначению отдельной области хранения каждому кластеру (на основе 1 : 1).

На рис. 11.6 приведен пример структуры сетевой базы данных с возможными вариантами кластеризации ее иерархических составляющих. Задача кластеризации сетевых баз данных сложнее, чем иерархических, так как области хранения могут содержать не только непосредственно связанные друг с другом типы записей, но любые комбинации типов записей базы данных, т. е. если существует *n* типов записей базы данных, то

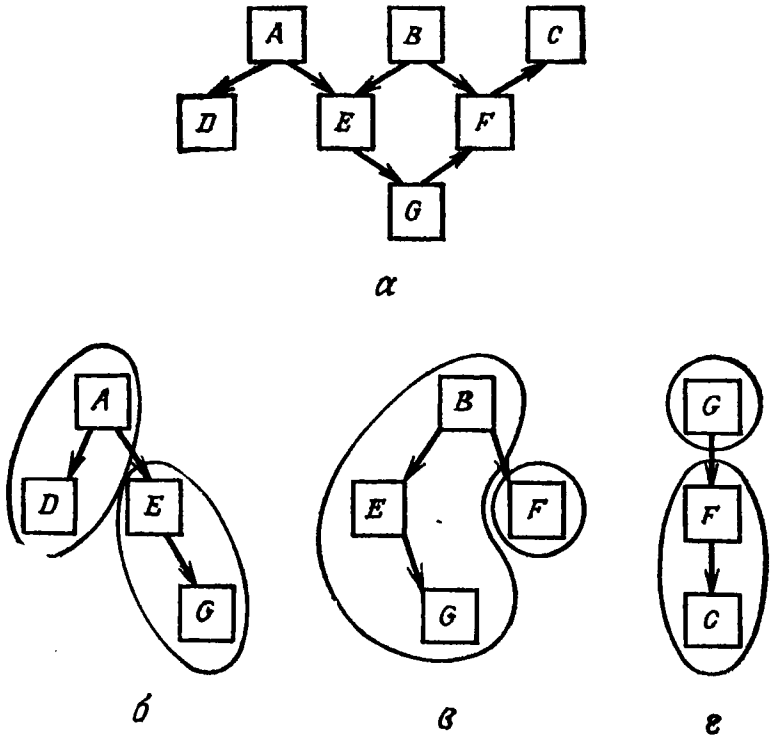


Рис. 11.6. Сетевая схема и иерархические компоненты с указанием возможных кластеров.

а — исходная сеть; *б* — иерархическая структура 1; *в* — иерархическая структура 2; *г* — иерархическая структура 3.

число возможных кластеров выражается быстрорастущей функцией Белла [5]:

$$B_0 = 1, \quad B_{n+1} = \sum_{k=l}^{n} \binom{n}{k} B_k. \qquad (11\text{-}11)$$

Примеры значений величины B_n приведены в таблице 11.3.

Теперь применим рассмотренный выше эвристический метод определения «вспомогательных» кластеров и областей на примере, представленном на рис. 11.6. Допустим, что три иерархические структуры имеют оптимальную кластеризацию, показанную на рис. 11-6 (*а*)—(*г*):

Структура 1: *AD, EG*
Структура 2: *BEG, F*
Структура 3: *G, FC*

} оптимальная «вспомогательная» кластеризация.

Существует перекрытие между *EG*, *BEG* и *G* и между *F* и *FC*. Кластер *AD* не перекрывается ни с каким-либо другим кластером и автоматически становится одним из кластеров сети.

Таблица 11.3. **Количество возможных кластеров в иерархических и сетевых базах данных с n-типами записей**

Количество	Иерархическая база данных	Сетевая база данных
1	1	1
2	2	2
3	4	5
4	8	15
5	16	52
6	32	203
7	64	877
8	128	4140
15	16 384	$1{,}38\times10^9$
⋮	⋮	⋮
n	2^{n-1}	$B_n=\sum_{k=0}^{n-1}\binom{n-1}{k}B_k \quad (B_0=1)$

Возможными вариантами кластеризации, которые будут оцениваться для сети в целом, являются

1. *AD, BEG, F, C*
2. *AD, BEG, FC*
3. *AD, B, EG, F, C*
5. *AD, B, EG, FC*
5. *AD, BE, G, F, C*
6. *AD, BE, G, FC*
7. *AD, B, E, G, F, C*
8. *AD, B, E, G, FC*

} кандидаты «основной», группы кластеризации.

Общее число вариантов равно 0 (n) для каждой иерархической структуры плюс восемь «кандидатов» кластеризации сети (т. е. 19 вариантов). Расчет всех возможных способов кластеризации сети потребует анализа 877 вариантов. Очевидно, что для выполнения этих вычислений потребуется использование ЭВМ. Допустим, кластеризация (*AD, BEG, FC*) является оптимальной, тогда возможны следующие варианты формирования областей хранения:

Область 1	**Область 2**	**Область 3**
1. *AD*	*BEG*	*FC*
2. *AD, BEG*	*FC*	—
3. *AD, FC*	*BEG*	—
4. *BEG, FC*	*AD*	—
5. *AD, BEG*	—	—

Ясно, что этот алгоритм является чисто эвристическим, и, если попытаться с его помощью найти глобальный оптимум

кластеризации, он не обязательно должен сходиться. Более общая (и гораздо более сложная) методика размещения записей и проектирования областей (в смысле CODASYL) была предложена в [244]; она предполагает использование эвристических алгоритмов, моделей теории графов, а также методов оптимизации. Другие методики подразумевают использование алгоритма энергии связи, рассмотренного в гл. 10 [77], и эвристического подхода, изложенного в [145]. Существует много общего между проблемой оптимизации кластеризации и задачей оптимального размещения записей, хотя их строгая адекватность не показана.

Приложение А

Упражнения по проектированию концептуальной схемы и схемы реализации

Ниже описаны в форме упражнений основные направления разработки концептуальной схемы и схемы реализации. Каждое из этих упражнений (или любое их подмножество) может быть использовано при решении соответствующим образом определенных проблем проектирования.

Целью этих упражнений является получение навыка в проектировании баз данных и прикладных программ, предназначенных для поддержки наборов данных и их обработки. Исследуются различные технические и организационные ограничения и предположения, влияющие на выбор и оценку проектных решений.

Упражнение А1. Концептуальное проектирование

Разработать СУБД-независимую структуру, отображающую информационное содержание базы данных. Информационная структура может быть определена в терминах диаграмм «сущность-атрибут-связь», нормализованных отношений или некоторой другой подобной концептуальной модели данных.

Если используются диаграммы «сущность-атрибут-связь», сделать набросок диаграммы сущностей и связей, основанной только на информационных требованиях. Если сущности выявить трудно, построить диаграмму всех элементов данных и их взаимосвязей.

Упражнение А2. Локальные информационные структуры

Используя первоначальную диаграмму в качестве проектного представления, построить набор локальных информационных структур (диаграмм сущностей), по одной на каждое из основных приложений обработки данных. Каждая локальная структура должна состоять из сущностей, изображаемых прямоугольниками, атрибутов (перечисленных рядом с прямоугольником, обозначающим сущность) и связей (обозначенных дугами

и стрелками), чтобы на диаграмме для любого приложения можно было построить некоторый путь доступа. Если выбрана реляционная модель, построить соответствующие связи функциональной зависимости.

Упражнение А3. Объединенные структуры

Объединить все локальные информационные структуры в единую глобальную структуру, из которой удалены избыточные сущности и атрибуты, но сохранены пути доступа для приложений обработки данных.

Необходимо отметить, что основной целью здесь является не достижение эффективности, а возможность полного удовлетворения информационных потребностей по обработке данных.

Упражнение А4. Обработка данных

Используя несколько сообщений и различных транзакций в качестве примеров, показать, как могли бы быть удовлетворены требования обработки данных с помощью конкретной информационной структуры.

Упражнение А5. Логическая структура базы данных

Целью данного упражнения является создание эффективной СУБД-зависимой логической структуры базы данных. Используя ранее построенную глобальную информационную структуру, необходимо последовательно совершенствовать ее, добавляя в любом порядке информацию об объеме данных, частоте обработки данных, а также ограничения СУБД. В качестве основы для выбора проектных решений принять минимальное число обращений к логическим записям, минимальный объем передаваемых данных и минимум памяти.

В результирующей диаграмме указать точки входа и определить упорядоченность записей. Составить таблицу обработки приложений и записей различных типов, указав число обращений к каждому типу записей.

Упражнение А6. Альтернативные модели данных

Построить СУБД-зависимые логические структуры баз данных для альтернативных моделей данных из классов иерархических, сетевых и реляционных систем. Для этого преобразовать построенную ранее структуру в такую, которая сравнима с конкретными системами, представляющими другие основные модели данных.

Упражнение А7. Проектирование прикладных программ

Сделать наброски блок-схем прикладных программ, предназначенных для удовлетворения требований обработки.

Упражнение А8. Дополнительные вопросы проектирования

Обсудить альтернативные решения для следующих проблем и/или ситуаций.

1. Какое влияние на базу данных и прикладные программы может оказать проведение поисковых транзакций в течение специально выделенного времени суток?
2. Какое влияние на базу данных и прикладные программы может оказать проведение транзакций обновления в течение специально выделенного времени суток?
3. Какое влияние на прикладные программы и базы данных может оказать наложение ограничений безопасности и целостности на систему?
4. Какие другие типы запросов на сообщения или транзакции могут оказать заметное влияние на спецификации базы данных и прикладных программ?

Задания для проектирования

Задание № 1. Система учета студентов

Необходимо спроектировать автоматизированную систему учета студентов для крупного университета. В университете имеется постоянный контингент из 35 000 студентов, 3000 факультетов, 100 отделений и 4000 курсов. Предполагается, что студенты, факультеты, отделения и курсы равномерно распределены по университету. Каждый студент изучает в среднем четыре курса за семестр. Система должна эффективно выдавать ответы и сообщения по запросам следующего характера (в скобках указана частота обращений для каждого приложения):

а. Составить список всех студентов, изучающих данный раздел конкретного курса (10 в день).

б. Составить список всех преподавателей, номеров аудиторий и курсов, которые они постоянно читают для данного отделения (20 в день).

в. Составить список всех курсов, читаемых для данного отделения (100 в день).

г. Определить, имеет ли студент соответствующую подготовку для изучения данного курса (5000 в день).

д. Включить студента в число (исключить из числа) слушателей раздела курса (20 000 в день).

е. Составить расписание зачетов для каждого студента (5000 в день).

Задание № 2. Система учета заказов

Компания D является посреднической фирмой, которая закупает изделия у поставщиков, складирует их и перепродает клиентам. Существует несколько типов каждого изделия. Конкретный тип изделия может быть предложен несколькими поставщиками по разной цене. Система учета заказов должна содержать информацию о поставщиках, изделиях, клиентах, заказах и т. д.

Виды обработки

Система учета заказов предназначена для выполнения различных приложений, каждое из которых должно сопровождаться отчетом. В число таких отчетов/запросов входят:

R1. *Отчет о клиентах.* Составить список, в котором для каждого клиента указать номера заказов и типы заказанных изделий.

R2. *Отчет о поставщиках.* Составить список, в котором для каждого поставщика указать наименования каталогов, наименования типов и номера изделий и их цену.

R3. *Справка о заказах клиентов.* По каждому номеру типа изделия составить список всех клиентов, имеющих открытый заказ на этот тип изделия.

R4. *Состояние запасов.* По каждому типу изделия, количество которого стало меньше уровня, необходимого для возобновления запаса, выдать сообщение, включающее наименование типа и номер изделия, поставщиков, оставшееся количество и размер заказа. (Составляет 10 % всего времени обработки.)

R5. *Справка о поставщиках.* По заданному номеру типа изделия составить список поставщиков, поставляющих данный тип изделия. В список включить имя, номер и адрес поставщика, цену изделия и ожидаемое время реализации заказа.

R6. *Справка о клиентах.* По заданному имени или номеру клиента выдать справочную информацию о клиенте.

Система должна также обрабатывать несколько типов транзакций, включая следующие:

T1. *Прием заказов.* В большинстве случаев заказы поступают по телефону от клиентов или торговых агентов. Заказ поступает в систему. После ввода заказа необходимо опреде-

лить наличие запаса по каждому заказанному типу изделий. После проверки текущего счета клиента выписывается накладная и счет-фактура. Перечисление типов изделий в счет-фактуре должно соответствовать порядку, в котором они перечислены в заказе, в то время как перечисление типов изделий в накладной должно определяться порядком выполнения заказа.

Т2. *Новый клиент.* Для каждого нового клиента ввести номер, имя, размер кредита, адрес и т. д.

Т3. *Новые изделия.* Для каждого нового типа изделия ввести информацию, описывающую тип изделия, его расположение в хранилищах, а также идентифицирующую его поставщиков. Изделие может быть добавлено или исключено.

Т4. *Новый поставщик.* Для каждого нового поставщика вводится общая описательная информация и обновляется база данных. Поставщики могут быть добавлены или исключены.

Т5. *Поступление товара.* Для каждого поступающего типа изделий определяется его расположение в хранилище и обновляется база данных.

Объем данных и обработки.

ПРИЛОЖЕНИЯ	ЧАСТОТА ОБРАБОТКИ
Т1. Прием заказов	1000 заказов в день
Т2. Новый клиент	50 в день
Т3. Типы изделий	Добавляется 1 раз в день Исключается 1 раз в день
Т4. Поставщики	Добавляется 1 раз в день Исключается 1 раз в день
Т5. Поступление товара	100 типов изделий в день
R1. Отчет о клиентах	1 в 100 дней
R2. Отчет о поставщиках	1 в 100 дней
R3. Справка о заказах клиентов	100 в день
R4. Состояние запасов	1 в день
R5. Справка о поставщиках	100 в день
R6. Справка о клиентах	1200 в день

ИНФОРМАЦИЯ ОБ ОБЪЕМЕ ДАННЫХ
(ТЕКУЩЕЕ СОСТОЯНИЕ)

40 поставщиков
50 000 клиентов
1000 типов изделий
5000 единиц хранимых изделий каждого типа (в среднем)

80 каталогов
200 изделий в каждом каталоге (в среднем, каждое изделие, поставляемое конкретным поставщиком, встречается в каталогах только один раз)
10 хранилищ на каждый тип изделия (в среднем)
2000 хранилищ (всего)
150 000 заказов
4 строки в заказе (в среднем)
2 адреса на каждого клиента (в среднем)
16 поставщиков каждого типа изделия (в среднем)
400 типов изделий на одного поставщика (в среднем)

Задание № 3. База данных космической федерации

Каждая из перечисленных ниже (в порядке возрастания сложности) задач представляет собой отдельное задание по проектированию баз данных. Все задачи базируются на основе одной общей гипотетической концепции, которую приближенно можно сформулировать как *«Космические путешествия»*.

Задача 3А

Исходные данные. Под управлением Космической Федерации находятся 100 галактических секторов, в которых патрулируют 25 космических кораблей, в среднем один корабль на четыре сектора.

Моделируемые ситуации

1. Как только какой-либо галактический сектор оказывается в преддверии некоторых трудностей (например, при возникновении угрозы межпланетной войны), для разрешения возникших проблем вызывается космический корабль, в ведении которого находится данный сектор (частота — 20 %).

2. Руководящему совету часто требуется знать, какой галактический сектор управляется конкретным космическим кораблем, в частности космическим кораблем «Энтерпрайз» (частота — 80 %).

Задача 3Б

Исходные данные. Космическая Федерация имеет 25 космических кораблей, базирующихся в известных участках космического пространства. Для снабжения космических кораблей, доставки на них топлива и снаряжения предназначены 100 грузовых кораблей. Поскольку каждый космический корабль является уникальной конструкцией, каждый грузовой корабль должен быть специально оборудован для обслуживания только одного из них и не может обслуживать никакой другой космический корабль. Для обеспечения непрерывного обслуживания

на каждый космический корабль выделено несколько грузовых кораблей.

Для обеспечения своевременной отправки грузовых кораблей и заблаговременного назначения экипажа Федерация хранит списки команд и координаты каждого грузового корабля.

Моделируемые ситуации

1. Как только космический корабль нуждается в обслуживании, необходимо найти ближайший грузовой корабль, который может провести обслуживание (частота — 90 %).

2. Когда грузовой корабль получает назначение для проведения обслуживания, Федерация извещает об этом тех членов экипажа на каждом из кораблей, которые имеют родственников на другом корабле. Для этого необходимо получить список членов экипажей, имеющих одинаковые фамилии и находящихся на том и другом кораблях (частота — 10 %).

Задача 3В

Исходные данные. Каждый грузовой корабль Космической Федерации состоит из пяти отсеков. Отсеки предназначены для конкретных ремонтных, навигационных и грузовых работ. В каждом отсеке проводится в среднем три различных вида работ.

На корабле пятьдесят членов экипажа, каждый из которых работает в среднем в двух отсеках. Федерация хранит персональные учетные сведения на каждого члена экипажа.

Моделируемые ситуации

1. Командиру корабля часто необходимы сведения об отдельных членах экипажа, работающих в конкретном отсеке (частота — 90 %).

2. Время от времени корабль оказывается не полностью укомплектованным членами экипажа. При этом возникает задача перераспределения обязанностей между остальными членами экипажа. Главное, что при этом принимается во внимание — это набор тех функциональных обязанностей, которые уже выполняются членом экипажа (частота — 10 %).

Задача 3Г

Исходные данные. Организация космического корабля *«Энтерпрайз»* очень проста. В состав корабля входит несколько подразделений, каждое из которых возглавляется руководителем. На каждое подразделение возложен ряд задач, выполняемых членами экипажа. Каждый член экипажа выполняет обычно более одной задачи, однако ни один из них не выполняет задачи двух или более подразделений.

Некоторые задачи требуют для выполнения нескольких членов экипажа. На каждого из членов экипажа (включая

руководителей подразделений) хранится персональная учетная информация. Каждому члену экипажа по каждой выполняемой им задаче присваивается класс, обозначающий качество выполнения данной работы данным членом экипажа.

Моделируемые ситуации

1. Найти учетные данные и имена членов экипажа, способных выполнить задачу T подразделения D, с классом не ниже R (частота — 50 %).

2. Найти имена членов экипажа, которые могут выполнить любую задачу в подмножестве задач подразделения (частота — 10 %).

3. Найти имена всех членов экипажа, работающих под руководством конкретного руководителя (частота — 40 %).

Задача 3Д

Исходные данные. Больничный лазарет космического корабля *«Энтерпрайз»* рассчитан на 10 врачей и 100 больных. Каждый врач ведет наблюдения в среднем за 400 членами экипажа. На каждого члена экипажа врач ведет историю заболеваний, а также регистрирует лекарства, противопоказанные данному члену экипажа. В среднем на каждого члена экипажа приходится по пять таких противопоказаний. Кроме того, врач регистрирует посещения планет членами экипажа (в среднем по 10 посещений на каждого члена экипажа). Для каждой планеты составляется список известных заболеваний, характерных для данной планеты, а также описание симптомов и методов лечения.

Моделируемые ситуации

1. Найти имена всех членов экипажа, находящихся под наблюдением конкретного врача (частота — 20 %).

2. Найти наименования планет, которые посетил конкретный член экипажа, а также составить список болезней, характерных для каждой из этих планет, симптомов и методов лечения (частота — 50 %).

3. Найти конкретный метод лечения для данной болезни и список членов экипажа, которым противопоказано данное лечение (частота — 30 %).

Задача 3Е

Исходные данные. Клингоны исследуют Вселенную. Они обнаружили уже более 500 солнечных систем, каждая из которых имеет в среднем по три пригодные для проживания планеты. На каждую пригодную для проживания планету Клингоны назначают губернатора. На каждого губернатора с целью его дальнейшего продвижения по службе ведется послужной список, в котором регистрируются поощрения.

В обязанности губернатора входит сбор данных о пригодных к эксплуатации природных ресурсах планеты и профессиях Клингонов, заселяющих ее. Население каждой планеты в среднем 1 млн. человек. Из-за малочисленности населения каждый житель планеты обязан владеть в среднем тремя специальностями. При этом принимается во внимание оценка классности по каждой специальности.

Моделируемые ситуации

1. Найти пригодные для проживания планеты в конкретной солнечной системе и составить список природных ресурсов этих планет, а также списки местных губернаторов (частота — 30 %).

2. Получить информацию о губернаторе конкретной планеты заданной солнечной системы (частота — 10 %).

3. Клингоны испытывают острую необходимость в конкретной специальности. С этой целью им необходимо знать имена и классность тех жителей империи Клингонов, которые владеют этой специальностью (частота — 60 %).

Задача 3Ж

Исходные данные. Космос нелегко дается космическим кораблям. Для того чтобы поддерживать их в рабочем состоянии, Федерация содержит 10 000 космических станций. Каждая из них расположена в таком месте, которое определяется оптимальным набором координат для обслуживания флотилии, и каждая называется по имени знаменитого капитана или храброго волонтера, погибшего в сражении.

Справочник по космическому кораблю насчитывает 1000 видов стандартных ремонтных работ. Из-за сложности и необходимости снаряжения специальным оборудованием каждая космическая станция сконструирована таким образом, что она может выполнить только 50 видов ремонтных работ. Таким образом, конкретный специфический вид ремонта могут выполнить только 500 из всех станций.

На каждой станции работает около 100 механиков. Каждый из них обучен 10 видам ремонтных работ, при этом различается уровень их классности по каждому виду работ.

Космические корабли Федерации появляются на различных станциях для проведения профилактического обслуживания, а также тогда, когда неисправности не могут быть устранены силами технического персонала космического корабля. Федерация регистрирует эти ремонтные работы. Каждой станцией было проведено около 1000 таких ремонтных работ.

Моделируемые ситуации

1. Космический корабль нуждается в профилактическом обслуживании конкретной системы управления. Командир ко-

рабля хочет найти ближайшую ремонтную станцию, которая может выполнить данный вид обслуживания (частота — 30 %).

2. Федерация периодически делает проверки правильности распределения ремонтных станций в космическом пространстве. Это выполняется путем просмотра списка космических кораблей, прошедших ремонтное обслуживание на конкретной станции (частота — 40 %).

3. *«Энтерпрайз»* нуждается в проведении особо сложного ремонта. Ведомство Федерации хочет найти для этой работы лучших механиков и станцию, где работают эти механики (частота — 30 %).

Оглавление

Часть II. Концептуальное проектирование

Часть III. Проектирование реализации

Часть IV. Физическое проектирование

Приложение А

УВАЖАЕМЫЙ ЧИТАТЕЛЬ!

Ваши замечания о содержании книги, ее оформлении, качестве перевода и другие просим присылать по адресу: 129820, Москва, И-110, ГСП, 1-й Рижский пер., 2, издательство «Мир».

Тоби Тиори, Джеймс Фрай

ПРОЕКТИРОВАНИЕ СТРУКТУР БАЗ ДАННЫХ, кн. 1

Научный редактор Т. Н. Шестакова
Младший научный редактор М. Ю. Григоренко
Художник В. П. Груздев
Художественный редактор Н. П. Иванов
Технический редактор Н. И. Манохина
Корректор С. А. Денисова

ИБ № 5009

Сдано в набор 25.06.84. Подписано к печати 26.12.84. Формат 60×90¹/₁₆. Объем 9 бум. л. Бумага типографская № 2. Гарнитура литературная. Печать высокая. Усл. печ. л. 18,00. Усл. кр.-отт. 18,00. Уч.-изд. л. 18,07. Изд. № 20/3330. Тираж 28 000 экз. Зак. 245. Цена 1 р. 60 к.

ИЗДАТЕЛЬСТВО «МИР»
129820, ГСП, Москва, И-110, 1-й Рижский пер., 2.

Ленинградская типография № 2 головное предприятие ордена Трудового Красного Знамени Ленинградского объединения «Техническая книга» им. Евгении Соколовой Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 198052, г. Ленинград, Л-52, Измайловский проспект, 29.