ОГЛАВЛЕНИЕ

1. Введение в базы данных

1.1. Основные понятия и определения

Стержневые идеи современных информационных технологий базируются на концепции *баз данных.*

Согласно этой концепции, основой информационных технологий являются *данные,* которые должны быть организованы в базы данных в целях адекватного отображения изменяющегося реального мира и удовлетворения информационных потребностей пользователей.

Одним из важнейших понятий в теории баз данных является понятие *информации.* Под *информацией* понимаются любые сведения о каком-либо событии, процессе, объекте.

Данные — это информация, представленная в определенном виде, позволяющем автоматизировать ее сбор, хранение и дальнейшую обработку человеком или информационным средством. Для компьютерных технологий данные — это информация в дискретном, фиксированном виде, удобная для хранения, обработки на ЭВМ, а также для передачи по каналам связи.

База данных (БД) — именованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области, или иначе БД — это совокупность взаимосвязанных данных при такой минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений в определенной предметной области. БД состоит из множества связанных файлов.

Система управления базами данных (СУБД) — совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Автоматизированная информационная система (АИС) — это система, реализующая автоматизированный сбор, обработку, манипулирование данными, функционирующая на основе ЭВМ и других технических средств и включающая соответствующее программное обеспечение (ПО) и персонал. В дальнейшем в этом качестве будет использоваться термин *информационная система* (ИС), который подразумевает понятие автоматизированная.

Каждая ИС в зависимости от ее назначения имеет дело с той или иной частью реального мира, которую принято называть *предметной областью* (ПрО) *системы.* Выявление ПрО — это необходимый начальный этап разработки любой ИС. Именно на этом этапе определяются информационные потребности всей совокупности пользователей будущей системы, которые, в свою очередь, предопределяют содержание ее базы данных.

Банк данных (БнД) является разновидностью ИС. БнД — это система специальным образом организованных данных: баз данных, программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных.

Под *задачами обработки данных* обычно понимается специальный класс решаемых на ЭВМ задач, связанных с видом, хранением, сортировкой, отбором по заданному условию и группировкой записей однородной структуры.

Отдельные программы или комплекс программ, реализующие автоматизацию решения прикладных задач обработки данных, называются *приложениями.* Приложения, созданные средствами СУБД, относят к *приложениям СУБД.* Приложения, созданные вне среды СУБД с помощью систем программирования, использующих средства доступа к БД, к примеру, Delphi или Visual Studio, называют *внешними приложениями.*

1.2. Современное состояние технологий баз данных

Кратко сформулируем основные современные принципы организации баз данных.

- Значительная часть современных СУБД способна работать на компьютерах различной архитектуры под управлением разных операционных систем.
- Подавляющее большинство современных СУБД обеспечивают поддержку полной реляционной модели данных, обеспечивая целостность категорий и целостность на уровне ссылок.
- Современные СУБД для определения данных и манипуляции ими опираются на принятые стандарты в области языков, а при обмене данными между различными СУБД базируются на существующих технологиях по обмену информацией.
- Многие существующие СУБД относятся к так называемым сетевым СУБД, которые предназначены для поддержки многопользовательского режима работы с базой данных и поддержки возможности децентрализованного хранения данных.
- Такие СУБД имеют развитые средства администрирования баз данных и средства защиты хранимой в них информации.
- Подобные СУБД имеют средства подключения клиентских приложений.
- Современные СУБД характеризуются опытами применения концепции фундаментальной идеи объектно-ориентированного подхода, способствующей повышению уровня абстракции баз данных, являющейся перспективным этапом на пути развития технологий баз данных.

Информационные системы, созданные средствами технологии баз данных, иногда принято называть *банками данных (БнД)*.

БнД включает в себя:

- технические средства;
- одну или несколько БД;
- СУБД;
- словарь или каталог данных;
- администратора;
- вычислительную систему;
- обслуживающий персонал.

Схематично это выглядит так, как показано на рис. 1.1.

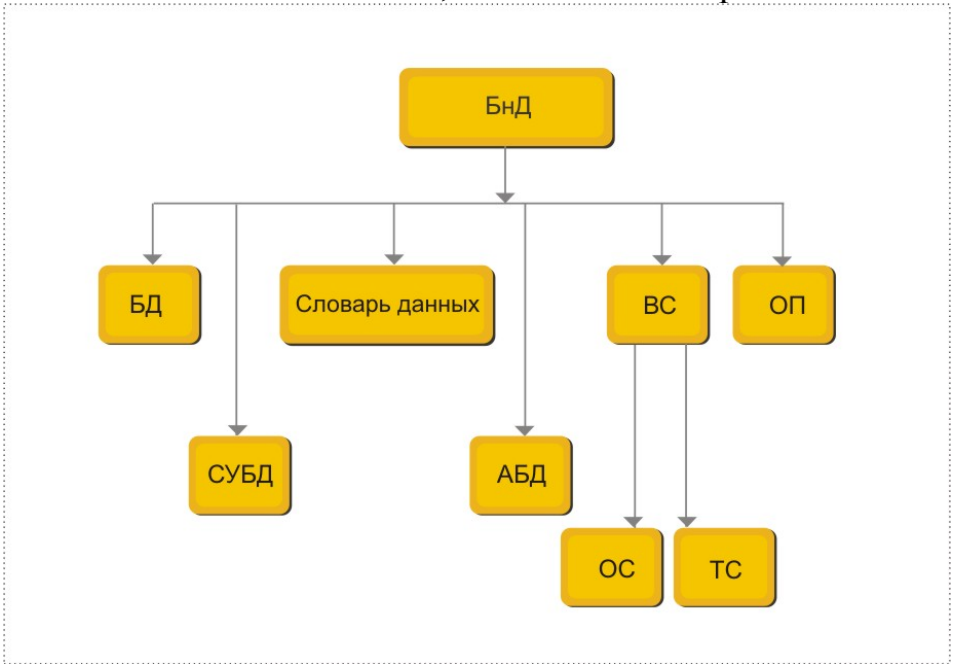


Рис. 1.1. Банк данных

Дадим краткие определения новым составляющим этой схемы.

Словарь или каталог данных служит для централизованного накопления и описания ресурса данных. Он содержит описание ПрО, сведения о структуре БД, о связях между элементами БД. Словарь данных можно рассматривать как часть самой базы данных.

Администратор БД (АБД) — человек или группа лиц, которые принимают решения. Основные функции АБД:

- участие в разработке БД;
- контроль правильности функционирования БД.

Вычислительная система (ВС) — включает программные (ПС) и аппаратные средства (ТС).

Обслуживающий персонал (ОП) — это лица, прямыми обязанностями которых является создание и поддержание корректного функционирования банка данных. Они ответственны за работу БнД и прикладного программного обеспечения. К обслуживающему персоналу

относятся: разработчики и администраторы базы данных, аналитики, программисты.

1.3. Базы данных

БД, как правило, создается как общий ресурс всего предприятия, где данные являются *интегрированными и общими.* Под понятием *интегрированные* данные подразумевается возможность представить базу данных как объединение нескольких отдельных файлов данных. Под понятием *общие* данные подразумевается возможность использования отдельных областей данных в БД несколькими различными пользователями для разных целей.

В базе данных информация должна быть организована так, чтобы обеспечить минимальную долю ее избыточности. Частичная избыточность информации необходима, но она должна быть минимизирована, так как чрезмерная избыточность данных влечет за собой ряд негативных последствий. Главные из них:

- увеличение объема информации, а значит, потребность в дополнительных ресурсах для хранения и обработки дополнительных объемов данных;
- появление ошибок при вводе дублирующей информации, нарушающих целостность базы данных и создающих противоречивые данные.

БД содержит не только данные, всесторонне характеризующие деятельность самой организации, фирмы, процесса или другой предметной области, но и описания этих данных. Информацию о данных принято называть *"метаданными",* т. е. "данными о данных". В совокупности описания всех данных образуют *словарь данных.*

В БД должны храниться данные, логически связанные между собой. Для того чтобы данные можно было связать между собой, и связать так, чтобы эти связи соответствовали реально существующим в данной предметной области, последнюю подвергают детальному анализу, выделяя сущности или объекты. Сущность или объект — это то, о чем необходимо хранить информацию. Сущности имеют некоторые характеристики, называемые атрибутами, которые тоже необходимо сохранять в БД. Атрибуты по своей внутренней структуре могут быть простыми, а могут быть сложными. Простые атрибуты могут быть представлены простыми типами данных. Различного рода графические изображения, являющиеся атрибутами сущностей, — это пример сложного атрибута. Определив сущности и их атрибуты, необходимо перейти к выявлению связей, которые могут существовать между некоторыми сущностями. Связь — это то, что объединяет две или более сущностей. Связи между сущностями также являются частью данных, и они также должны храниться в базе данных.

Если все это: сущности, атрибуты сущностей и связи между сущностями определено, то схема базы данных может выглядеть примерно так, как представлено на рис. 1.2. На нем показан пример схемы базы данных, которую можно назвать ФАКУЛЬТЕТ.

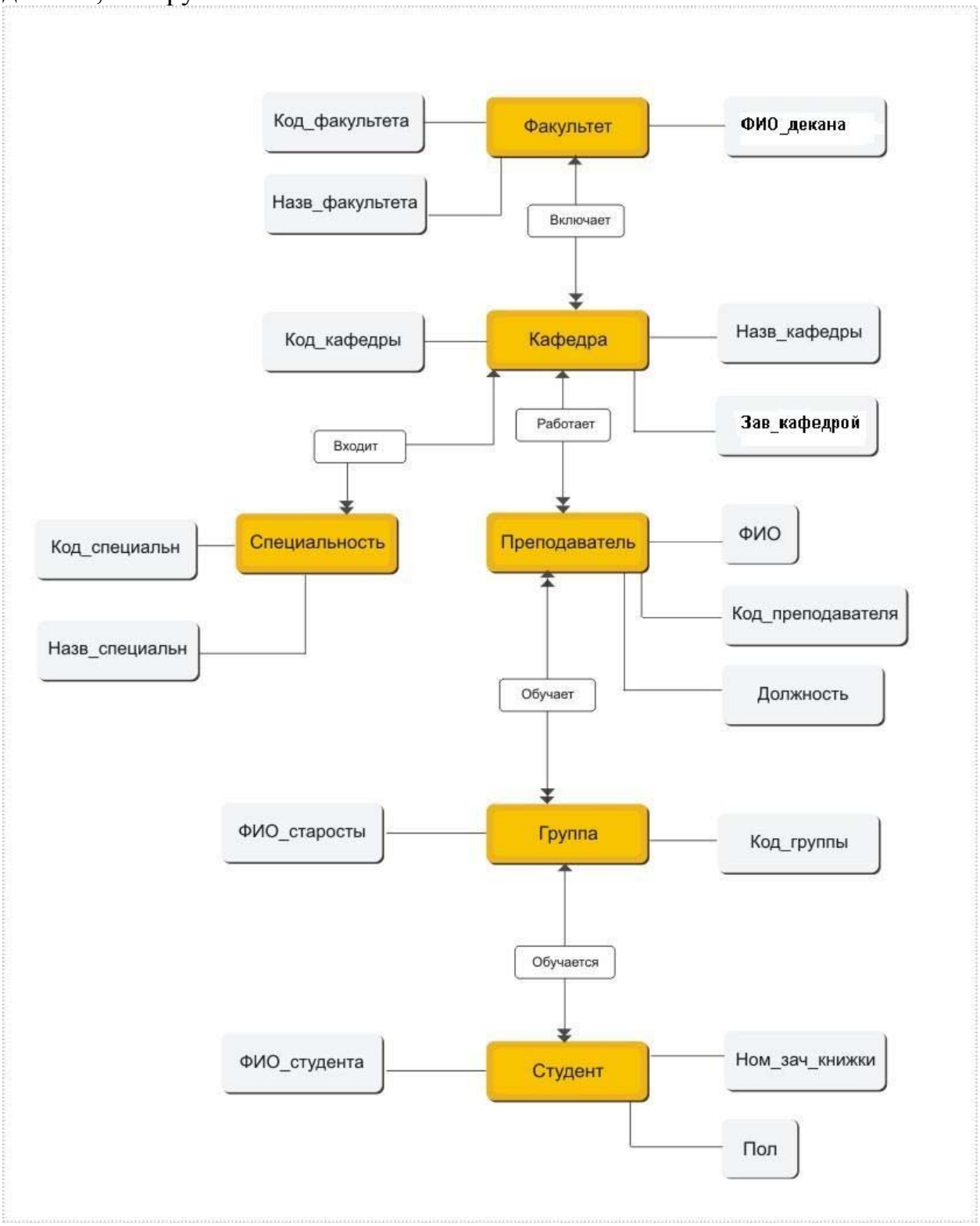


Рис. 1.2. Пример ER-диаграммы базы данных ФАКУЛЬТЕТ

Схема, которая называется ER-диаграммой (Entity-Relationship), состоит из следующих компонентов:

- шести сущностей, которые изображены прямоугольниками, каждый из которых имеет свои атрибуты, помещенные в овалы, а в нижеприведенном списке они перечислены в скобках рядом с именем сущностей:

ФАКУЛЬТЕТ: (Код_факультета, Назв_факультета, ФИО_декана);
КАФЕДРА: (Код_кафедры, Назв_кафедры, Зав_кафедрой);
СПЕЦИАЛЬНОСТЬ: (Код_специальности, Назв_специальности);
ПРЕПОДАВАТЕЛЬ: (Код_преподавателя, ФИО, Должность);
ГРУППА: (Код_группы, ФИО_старосты);
СТУДЕНТ: (Ном_зач_книжки, ФИО, Пол);

- пяти связей, которые обозначены стрелками и связывают те сущности, на которые они направлены:

связь **ВКЛЮЧАЕТ** показывает, что на факультете несколько кафедр;
связь **ВХОДИТ** изображает, что одна и та же кафедра готовит специалистов по нескольким специальностям;
связь **РАБОТАЕТ** определяет то, что на кафедре работает ряд преподавателей;
связь **ОБУЧАЕТ** с двойными стрелками в обоих направлениях поясняет тот факт, что один и тот же преподаватель преподает в разных группах, а одна и та же группа занимается с разными преподавателями;
связь **ОБУЧАЕТСЯ** определяет, что каждая группа включает в себя ряд студентов.

Из представленной диаграммы понятно, что данные обладают определенной структурой. Для выявления этой структуры база данных должна пройти процесс проектирования.

Проектируемая БД должна обладать определенными свойствами. Назовем основные свойства БД.

Целостность. В каждый момент времени существования БД сведения, содержащиеся в ней, должны быть непротиворечивы. Целостность БД достигается вследствие введения ограничений целостности, в частности, к ним относятся ограничения, связанные с нормализацией БД.

Восстанавливаемость. Данное свойство предполагает возможность восстановления БД после сбоя системы или отдельных видов порчи системы.

Безопасность. Безопасность БД предполагает защиту данных от преднамеренного и непреднамеренного доступа, модификации или разрушения. Применяется запрещение несанкционированного доступа, защита от копирования и криптографическая защита.

Эффективность. Свойство эффективности обычно понимается как:

- минимальное время реакции на запрос пользователя;

- минимальные потребности в памяти;
- сочетание этих параметров.

1.4. Системы управления базами данных

Итак, как уже не один раз упоминалось, СУБД — это программное обеспечение, с помощью которого пользователи могут определять, создавать и поддерживать базу данных, а также осуществлять к ней контролируемый доступ.

По степени универсальности различаются два класса СУБД — системы общего назначения и специализированные системы.

СУБД общего назначения не ориентированы на какую-либо конкретную предметную область или на информационные потребности конкретной группы пользователей. Каждая система такого рода реализуется как программный продукт, способный функционировать на некоторой модели ЭВМ в определенной операционной обстановке. СУБД общего назначения обладает средствами настройки на работу с конкретной БД в условиях конкретного применения.

В некоторых ситуациях СУБД общего назначения не позволяют добиться требуемых проектных и эксплуатационных характеристик (производительность, занимаемый объем памяти и прочее). Тем не менее создание *специализированных СУБД* весьма трудоемкий процесс и для того, чтобы его реализовать, нужны очень веские основания.

В процессе реализации своих функций СУБД постоянно взаимодействует с базой данных и с другими прикладными программными продуктами пользователя, предназначенными для работы с данной БД и называемыми приложениями.

Для того чтобы СУБД успешно справлялась со своими задачами, она должна обладать определенными возможностями.

Можно дать следующую обобщенную характеристику возможностям современных СУБД.

1. СУБД включает язык определения данных, с помощью которого можно определить базу данных, ее структуру, типы данных, а также средства задания ограничений для хранимой информации. В многопользовательском варианте СУБД этот язык позволяет формировать представления как некоторое подмножество базы данных, с поддержкой которых пользователь может создавать свой взгляд на хранимые данные, обеспечивать дополнительный уровень безопасности данных и многое другое.
2. СУБД позволяет вставлять, удалять, обновлять и извлекать информацию из базы данных посредством языка управления данными.
3. Большинство СУБД могут работать на компьютерах с разной архитектурой и под разными операционными системами, причем на

работу пользователя при доступе к данным практически тип платформы влияния не оказывает.

4. Многопользовательские СУБД имеют достаточно развитые средства администрирования БД.
5. СУБД предоставляет контролируемый доступ к базе данных с помощью:
    - системы обеспечения безопасности, предотвращающей несанкционированный доступ к информации базы данных;
    - системы поддержки целостности базы данных, обеспечивающей непротиворечивое состояние хранимых данных;
    - системы управления параллельной работой приложений, контролирующей процессы их совместного доступа к базе данных;
    - системы восстановления, позволяющей восстановить базу данных до предыдущего непротиворечивого состояния, нарушенного в результате аппаратного или программного обеспечения.

2. Архитектура СУБД

2.1. Трехуровневая архитектура базы данных

Одним из важнейших аспектов развития СУБД является идея отделения логической структуры БД и манипуляций данными, необходимыми пользователям, от физического представления, требуемого компьютерным оборудованием.

Одна и та же БД в зависимости от точки зрения может иметь различные уровни описания. По числу уровней описания данных, поддерживаемых СУБД, различают одно-, двух- и трехуровневые системы. В настоящее время чаще всего поддерживается трехуровневая архитектура описания БД (рис. 2.1), с тремя уровнями абстракции, на которых можно рассматривать базу данных.

Такая архитектура включает:

- внешний уровень, на котором пользователи воспринимают данные, где отдельные группы пользователей имеют свое представление (ПП) на базу данных;
- внутренний уровень, на котором СУБД и операционная система воспринимают данные;
- концептуальный уровень представления данных, предназначенный для отображения внешнего уровня на внутренний уровень, а также для обеспечения необходимой их независимости друг от друга; он связан с обобщенным представлением пользователей.

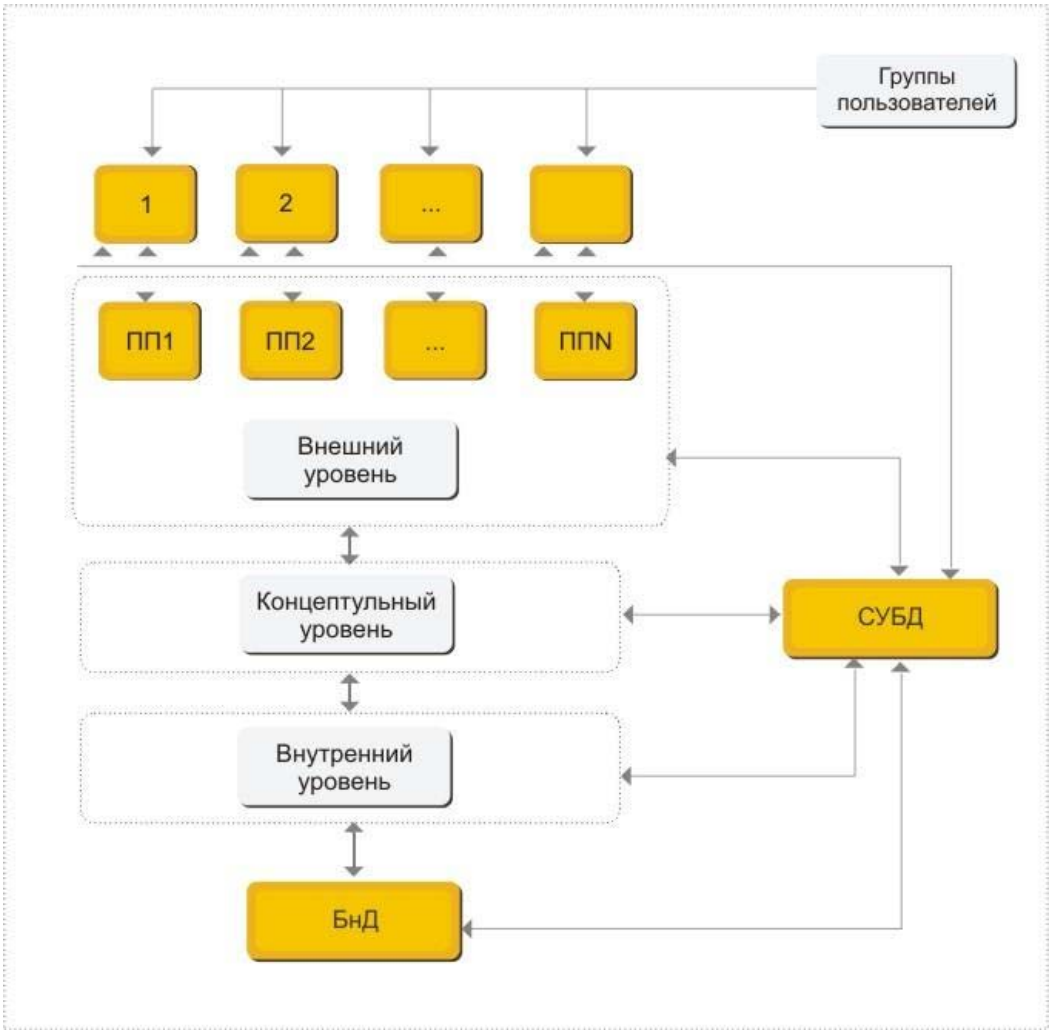


Рис. 2.1. Трехуровневая архитектура СУБД

Описание структуры данных на любом уровне называется *схемой.* Существует три различных типа схем базы данных, которые определяются в соответствии с уровнями абстракции трехуровневой архитектуры. На самом высоком уровне имеется несколько внешних схем или подсхем, которые соответствуют разным представлениям данных. На концептуальном уровне описание базы данных называют *концептуальной схемой,* а на самом низком уровне абстракции — *внутренней схемой.*

Основным назначением трехуровневой архитектуры является обеспечение независимости от данных. Суть этой независимости заключается в том, что изменения на нижних уровнях никак не влияют на верхние уровни. Различают два типа независимости от данных: логическую и физическую.

Логическая независимость от данных означает полную защищенность внешних схем от изменений, вносимых в концептуальную схему. Такие изменения концептуальной схемы, как добавление или удаление новых сущностей, атрибутов или связей, должны осуществляться

без необходимости внесения изменений в уже существующие внешние схемы для других групп пользователей.

Физическая независимость от данных означает защищенность концептуальной схемы от изменений, вносимых во внутреннюю схему. Такие изменения внутренней схемы, как использование различных файловых систем или структур хранения, разных устройств хранения, модификация индексов или хеширование, должны осуществляться без необходимости внесения изменений в концептуальную или внешнюю схемы.

Далее рассмотрим каждый из трех названных уровней.

Внешний уровень — это пользовательский уровень. Пользователем может быть программист, или конечный пользователь, или администратор базы данных. Представление базы данных с точки зрения пользователей называется *внешним представлением.* Каждая группа пользователей выделяет в моделируемой предметной области, общей для всей организации, те сущности, атрибуты и связи, которые ей интересны. Эти частичные или переопределенные описания БД для отдельных групп пользователей или ориентированные на отдельные аспекты предметной области называют *подсхемой.*

Концептуальный уровень является промежуточным уровнем в трехуровневой архитектуре и обеспечивает представление всей информации базы данных в абстрактной форме. Описание базы данных на этом уровне называется концептуальной схемой, которая является результатом концептуального проектирования.

Концептуальное проектирование базы данных включает анализ информационных потребностей пользователей и определение нужных им элементов данных. Таким образом, *концептуальная схема* — это единое логическое описание всех элементов данных и отношений между ними, логическая структура всей базы данных. Для каждой базы данных имеется только одна концептуальная схема.

Концептуальная схема должна содержать:

- сущности и их атрибуты;
- связи между сущностями;
- ограничения, накладываемые на данные;
- семантическую информацию о данных;
- обеспечение безопасности и поддержки целостности данных.

Внутренний уровень является третьим уровнем архитектуры БД. Внутреннее представление не связано с физическим уровнем, так как физический уровень хранения информации обладает значительной индивидуальностью для каждой системы.

На нижнем уровне находится внутренняя схема, которая является полным описанием внутренней модели данных. Для каждой базы данных существует только одна внутренняя схема.

Внутренняя схема описывает физическую реализацию базы данных и предназначена для достижения оптимальной производительности и обеспечения экономного использования дискового пространства. На внутреннем уровне хранится следующая информация:

- распределение дискового пространства для хранения данных и индексов;
- описание подробностей сохранения записей (с указанием реальных размеров сохраняемых элементов данных);
- сведения о размещении записей;
- сведения о сжатии данных и выбранных методах их шифрования.

СУБД отвечает за установление соответствия между всеми тремя типами схем разных уровней, а также за проверку их непротиворечивости.

Ниже внутреннего уровня находится *физический уровень,* который контролируется операционной системой, но под руководством СУБД. Физический уровень учитывает, каким образом данные будут представлены в машине. Он обеспечивает физический взгляд на базу данных: дисководы, физические адреса, индексы, указатели и т. д. За этот уровень отвечают проектировщики физической базы данных, которые работают только с известными операционной системе элементами. Область их интересов: указатели, реализация последовательного распределения, способы хранения полей внутренних записей на диске. Однако функции СУБД и операционной системы на физическом уровне не вполне четко разделены и могут варьироваться от системы к системе.

2.2. Функции СУБД

Управление данными во внешней памяти

Данная функция предоставляет пользователям возможности выполнения самых основных операций, которые осуществляются с данными, — это сохранение, извлечение и обновление информации. Она включает в себя обеспечение необходимых структур внешней памяти как для хранения данных, непосредственно входящих в БД, так и для служебных целей, например для ускорения доступа к данным.

Управление транзакциями

Транзакция — это последовательность операций над БД, рассматриваемых СУБД как единое целое. Транзакция представляет собой набор действий, выполняемых с целью доступа или изменения содержимого базы данных. Примерами простых транзакций может служить добавление, обновление или удаление в базе данных сведений о некоем объекте. Сложная же транзакция образуется в том случае, когда в базу данных требуется внести сразу несколько изменений. Инициализация транзакции может быть вызвана отдельным пользователем или прикладной программой.

Восстановление базы данных

Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Обычно рассматриваются два возможных вида аппаратных сбоев:

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

Поддержание надежности хранения данных в БД требует избыточности хранения данных, причем та часть данных, которая используется для восстановления, должна храниться особо надежно. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД.

Поддержка языков БД

Для работы с базами данных используются специальные языки, называемые языками баз данных.

В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных. Стандартным языком наиболее распространенных в настоящее время реляционных СУБД является язык SQL (Structured Query Language — язык структурированных запросов). Язык SQL позволяет определять схему реляционной БД и манипулировать данными.

Словарь данных

Одной из основополагающих идей рассмотренной выше трехуровневой архитектуры является наличие интегрированного системного каталога с данными о схемах, пользователях, приложениях и т. д. Системный каталог, который еще называют словарем данных, является, таким образом, хранилищем информации, описывающей данные в базе данных. Предполагается, что каталог доступен как пользователям, так и функциям СУБД. Обычно в словаре данных: содержится следующая информация:

- имена, типы и размеры элементов данных;
- имена связей;
- накладываемые на данные ограничения поддержки целостности;
- имена пользователей, которым предоставлено право доступа к данным;

- внешняя, концептуальная и внутренняя схемы и отображения между ними;
- статистические данные, например частота транзакций и счетчики обращений к объектам базы данных.

Управление параллельным доступом

Одна из основных целей создания и использования СУБД заключается в том, чтобы множество пользователей могло осуществлять параллельный доступ к совместно обрабатываемым данным. Параллельный доступ сравнительно просто организовать, если все пользователи выполняют только чтение данных, поскольку в этом случае они не могут помешать друг другу. Однако когда два или больше пользователей одновременно получают доступ к базе данных, конфликт с нежелательными последствиями легко может возникнуть, например, если хотя бы один из них попытается обновить данные.

СУБД должна гарантировать, что при одновременном доступе к базе данных многих пользователей подобных конфликтов не произойдет.

Управление буферами оперативной памяти

СУБД обычно работают с БД значительного размера. Понятно, что если при обращении к любому элементу данных будет производиться обмен с внешней памятью, то вся система будет работать со скоростью устройства внешней памяти. Практически единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. В развитых СУБД поддерживается собственный набор буферов оперативной памяти с собственной дисциплиной замены буферов.

Контроль доступа к данным

СУБД должна иметь механизм, гарантирующий возможность доступа к базе данных только санкционированных пользователей и защищающий ее от любого несанкционированного доступа.

В современных СУБД поддерживается один из двух широко распространенных подходов к вопросу обеспечения безопасности данных: избирательный подход или обязательный подход.

В большинстве современных систем предусматривается избирательный подход, при котором некий пользователь обладает различными правами при работе с разными объектами. Значительно реже применяется альтернативный, обязательный подход, где каждому объекту данных присваивается некоторый классификационный уровень, а каждый пользователь обладает некоторым уровнем допуска.

Поддержка целостности данных

Термин *целостность* используется для описания корректности и непротиворечивости хранимых в БД данных. Реализация поддержки целостности данных предполагает, что СУБД должна содержать сведения о тех правилах, которые нельзя нарушать при работе с данными, и обладать инструментами контроля за тем, чтобы данные и их изменения соответствовали заданным правилам.

2.3. Языки баз данных

В СУБД поддерживается несколько специализированных по своим функциям подъязыков. Их можно разбить на две категории:

- язык определения данных БД — ЯОД {DDL — Data Definition Language);
- язык манипулирования данными— ЯМД (DML — Data Manipulation , Language).

2.3.1. Язык определения данных

Язык определения данных — описательный язык, с помощью которого описывается предметная область: именуются объекты, определяются их свойства и связи между объектами. Он используется главным образом для определения логической структуры БД.

Схема базы данных, выраженная в терминах специального языка определения данных, состоит из набора определений. Язык ЯОД используется как для определения новой схемы, так и для модификации уже существующей.

Результатом компиляции ЯОД — операторов является набор таблиц, хранимый в системном каталоге, в котором содержатся метаданные — т. е. данные, которые включают определения записей, элементов данных, а также другие объекты, представляющие интерес для пользователей или необходимые для работы СУБД. Перед доступом к реальным данным СУБД обычно обращается к системному каталогу.

2.3.2. Языки манипулирования данными

Язык манипулирования данными содержит набор операторов манипулирования данными, т. е. операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные.

Множество операций над данными можно классифицировать следующим образом:.

1. операции селекции;
2. действия над данными:
    - включение — ввод экземпляра записи в БД с установкой его связей;

- удаление — исключение экземпляра записи из БД с установкой новых связей;
- модификация — изменение содержимого экземпляра записи и коррекция связей при необходимости.

Языки манипулирования данными делятся на два типа. Это разделение обусловлено коренным различием в подходах к работе с данными, а следовательно, различием в базовых конструкциях в работе с данными.

Первый тип — это процедурный ЯМД.

Второй тип — это декларативный (непроцедурный) ЯМД.

К процедурным языкам манипулирования данными относятся и языки, поддерживающие операции реляционной алгебры, которую основоположник теории реляционных баз данных Э. Ф. Кодд ввел для управления реляционной базой данных. Реляционная алгебра — это процедурный язык обработки реляционных таблиц, где в качестве операндов выступают таблицы в целом.

Декларативные языки предоставляют пользователю средства, позволяющие указать лишь то, какие данные требуются. Решение вопроса о том, как их следует извлекать, берет на себя процессор данного языка, работающий с целыми наборами записей.

Реляционные СУБД обычно включают поддержку непроцедурных языков манипулирования данными — чаще всего это бывает язык структурированных запросов SQL или язык запросов по образцу QBE.

В настоящее время нормой является поддержка декларативного языка SQL, в основе которого лежит реляционное исчисление, также введенное Э Коддом. Этот язык стал стандартом для языков реляционных баз данных, что позволяет использовать один и тот же синтаксис и структуру команд при переходе от одной СУБД к другой

Следует отметить, что язык SQL имеет сразу два компонента: язык DDL (ЯОД) для описания структуры базы данных, и язык DML (ЯМД) для выборки и обновления данных.

Другим широко используемым языком обработки данных является язык QBE, который заслужил репутацию одного из самых простых способов извлечения информации из базы данных. Особенно это ценно для пользователей, не являющихся профессионалами в этой области Язык предоставляет графические средства создания запросов на выборку данных с использованием шаблонов Ответ на запрос также представляет собой графическую информацию

Часть непроцедурного языка ЯМД, которая отвечает за извлечение данных, называется *языком запросов* Язык запросов можно определить как высокоуровневый узкоспециализированный язык, предназначенный для удовлетворения различных требований по выборке информации из базы данных.

2.4. Архитектура многопользовательских СУБД

2.4.1. Модели двухуровневой технологии "клиент — сервер"

Систему баз данных можно рассматривать как систему, где осуществлено распределение процесса выполнения по принципу взаимодействия двух программных процессов, один из которых в этой модели называется "клиентом", а другой, обслуживающий клиента, — *сервером* (машина, хранящая базы данных). Клиентский процесс запрашивает некоторые услуги, а серверный процесс обеспечивает их выполнение. При этом предполагается, что один серверный процесс может обслужить множество клиентских процессов (рис. 2.2).

Рис. 2.2. Структура системы БД с выделением клиентов и сервера

Сервер в простейшем случае — это собственно СУБД. Он поддерживает все основные функции СУБД и предоставляет полную поддержку на внешнем, концептуальном и внутреннем уровнях.

Клиенты — это различные приложения, которые выполняются над СУБД.

Обычно в приложении выделяются следующие группы функций:

- функции ввода и отображения данных;
- прикладные функции, определяющие основные алгоритмы решения задач приложения;
- функции обработки данных внутри приложения,
- функции управления информационными ресурсами;
- служебные функции, играющие роль связок между функциями первых четырех групп.

Если все пять компонентов приложения распределяются только между двумя процессами, которые выполняются на двух платформах: на

клиенте и на сервере, то такая модель называется *двухуровневой.* Она имеет несколько основных разновидностей. Рассмотрим их.

Файловый сервер

Модель файлового сервера называется моделью *удаленного управления данными.* Данная модель предполагает следующее распределение функций - на клиенте располагаются почти все части приложения: презентационная часть приложения, прикладные функции, а также функции управления информационными ресурсами. Файловый сервер содержит файлы, необходимые для работы приложений и самой СУБД и поддерживает доступ к файлам (рис. 2.3).

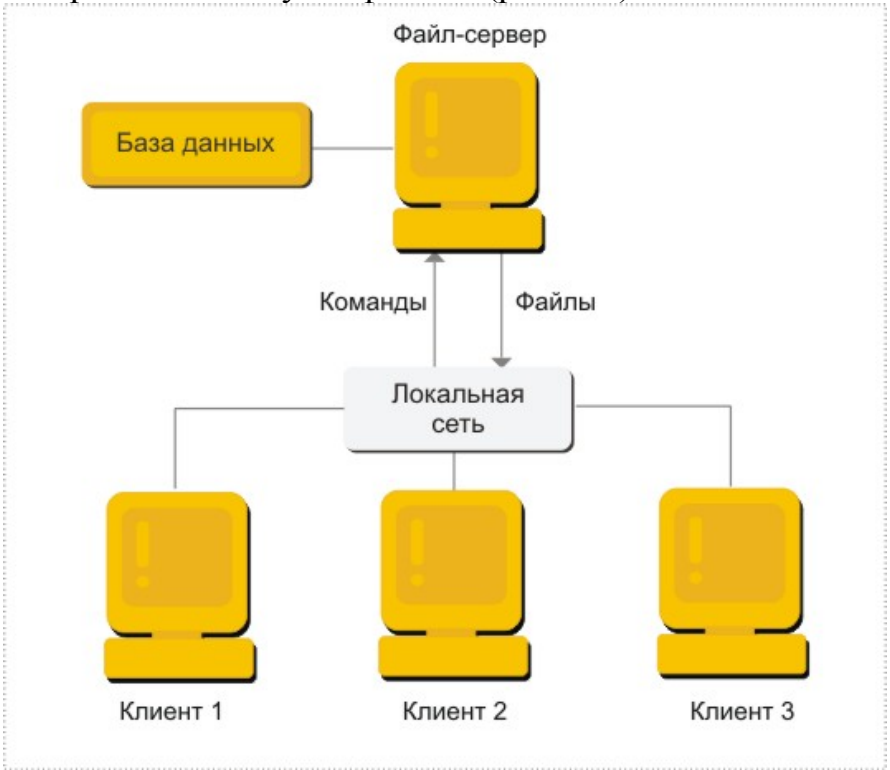


Рис. 2.3. Модель файлового сервера

Поскольку передача файлов представляет собой длительную процедуру, такой подход характеризуется значительным сетевым трафиком, что может привести к снижению производительности всей системы в целом.

Помимо этого недостатка использование файлового сервера несет еще и другие:

- на каждой рабочей станции должна находиться полная копия СУБД;
- управление параллельностью, восстановлением и целостностью усложняется, поскольку доступ к одним и тем же файлам могут осуществлять сразу несколько экземпляров СУБД;
- узкий спектр операций манипулирования данными, который определяется только файловыми командами;

- защита данных осуществляется только на уровне файловой системы.

Основное достоинство этой модели, заключается в том, что в ней уже осуществлено разделение монопольного приложения на два взаимодействующих процесса. При этом сервер может обслуживать множество клиентов, обращающихся к нему с запросами.

Модель удаленного доступа к данным

В модели удаленного доступа база данных также хранится на сервере. На сервере же находится и ядро СУБД. На клиенте располагаются части приложения, поддерживающие функции ввода и отображения данных и прикладные функции.

Клиент обращается к серверу с запросами на языке SQL. Структура модели удаленного доступа приведена на рис. 2.4.

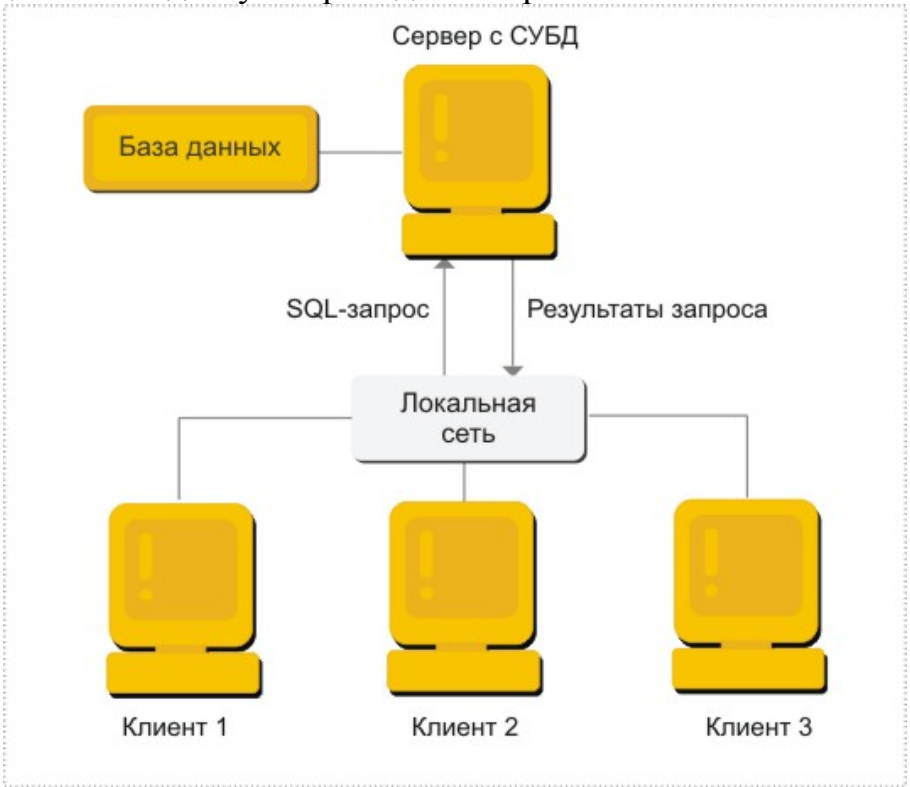


Рис. 2.4. Модель удаленного доступа

Сервер принимает и обрабатывает запросы со стороны клиентов, проверяет полномочия пользователей, гарантирует соблюдение ограничений целостности, выполняет обновление данных, выполняет запросы и возвращает результаты клиенту, поддерживает системный каталог, обеспечивает параллельный доступ к базе данных и ее восстановление. К тому же резко уменьшается загрузка сети, так как по ней от клиентов *к* серверу передаются не файловые команды, а запросы на SQL, и их объем существенно меньше. В ответ на запросы клиент получает только данные, соответствующие запросу, а не блоки файлов, как в модели файлового сервера.

Тем не менее, данная технология обладает и рядом недостатков:

- запросы на языке SQL при интенсивной работе клиентских приложений могут существенно загрузить сеть;
- презентационные и прикладные функции приложения должны быть повторены для каждого клиентского приложения;
- сервер в этой модели играет пассивную роль, поэтому функции управления информационными ресурсами должны выполняться на клиенте.

Модель сервера баз данных

Технологию "клиент — сервер" поддерживают большинство современных СУБД: Informix, Ingres, Sybase, Oracle, MS SQL Server. В основу данной модели добавлен механизм хранимых процедур и механизм триггеров.

Механизм *хранимых процедур* позволяет создавать подпрограммы, работающие на сервере и управляющие его процессами.

Таким образом, размещение на сервере хранимых процедур означает, что прикладные функции приложения разделены между клиентом и сервером. Трафик обмена информацией между клиентом и сервером резко уменьшается.

Централизованный контроль целостности базы данных в модели сервера баз данных выполняется с использованием механизма триггеров. Триггеры также являются частью БД.

Триггер — это особый тип хранимой процедуры, реагирующий на возникновение определенного события в БД. Он активизируется при попытке изменения данных — при операциях добавления, обновления и удаления. Триггеры определяются для конкретных таблиц БД.

Внедрение триггеров незначительно влияет на производительность сервера и часто используется для усиления приложений, выполняющих многокаскадные операции в БД.

В данной модели (рис. 2.5) сервер является активным, потому что не только клиент, но и сам сервер, используя механизм триггеров, может быть инициатором обработки данных в БД. Поскольку функции клиента облегчены переносом части прикладных функций на сервер, он в этом случае называется "тонким".

При всех положительных качествах данной модели у нее все же есть один недостаток — очень большая загрузка сервера.

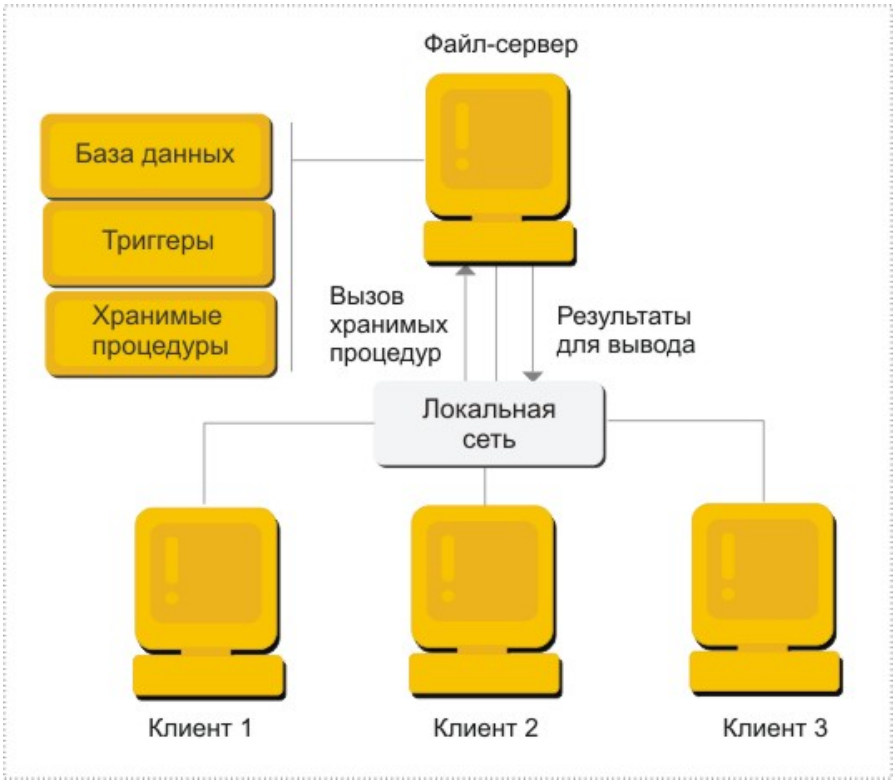


Рис. 2.5. Модель сервера БД

2.4.2. Сервер приложений. Трехуровневая модель

Эта модель является расширением двухуровневой модели и в ней вводится дополнительный промежуточный уровень между клиентом и сервером. Архитектура трехуровневой модели приведена на рис. 2.6.

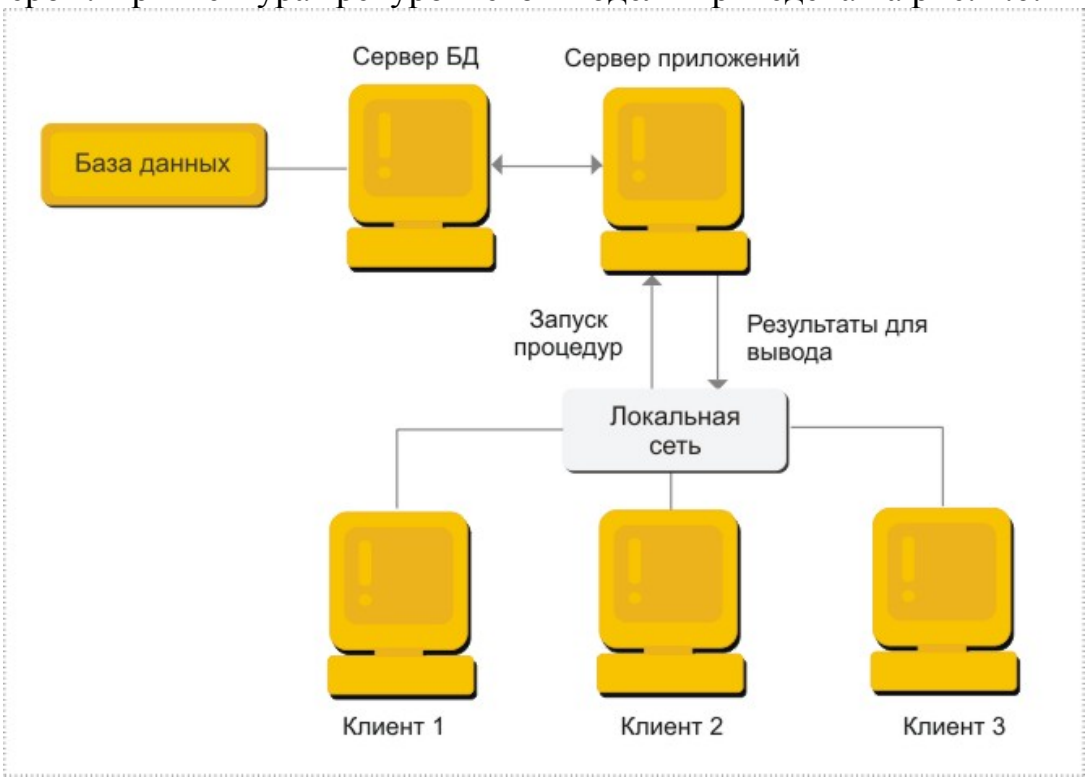


Рис. 2.6. Архитектура трехуровневой модели

Такая архитектура предполагает, что на *клиенте* располагаются: функции ввода и отображения данных, включая графический пользовательский интерфейс, локальные редакторы, коммуникационные функции, которые обеспечивают доступ клиенту в локальную или глобальную сеть.

Серверы баз данных в этой модели занимаются исключительно функциями управления информационными ресурсами БД: обеспечивают функции создания и ведения БД, поддерживают целостность БД, осуществляют функции создания резервных копий БД и восстановления БД после сбоев, управления выполнением транзакций и так далее.

Промежуточному уровню, который может содержать один или несколько серверов приложений, выделяются общие не загружаемые функции для клиентов: наиболее общие прикладные функции клиента, функции, поддерживающие сетевую доменную операционную среду, каталоги с данными, функции, обеспечивающие обмен сообщениями и поддержку запросов.

Преимущества трехуровневой модели наиболее заметны в тех случаях, когда клиенты выполняют сложные аналитические расчеты над базой данных.

3. Концепции проектирования БД

3.1. Жизненный цикл БД

Как и любой программный продукт, база данных обладает собственным жизненным циклом (ЖЦБД). Главной составляющей в жизненном цикле **БД** является создание единой базы данных и программ, необходимых для ее работы.

ЖЦБД включает в себя следующие основные этапы (рис. 3.1):

1. планирование разработки базы данных;
2. определение требований к системе;
3. сбор и анализ требований пользователей;
4. проектирование базы данных:
   - концептуальное проектирование базы данных;
   - логическое проектирование базы данных;
   - физическое проектирование базы данных;
5. разработка приложений:
   - проектирование транзакций;
   - проектирование пользовательского интерфейса;
6. реализация;
7. загрузка данных;
8. тестирование;
9. эксплуатация и сопровождение:

- анализ функционирования и поддержка исходного варианта БД;
- адаптация, модернизация и поддержка переработанных вариантов.

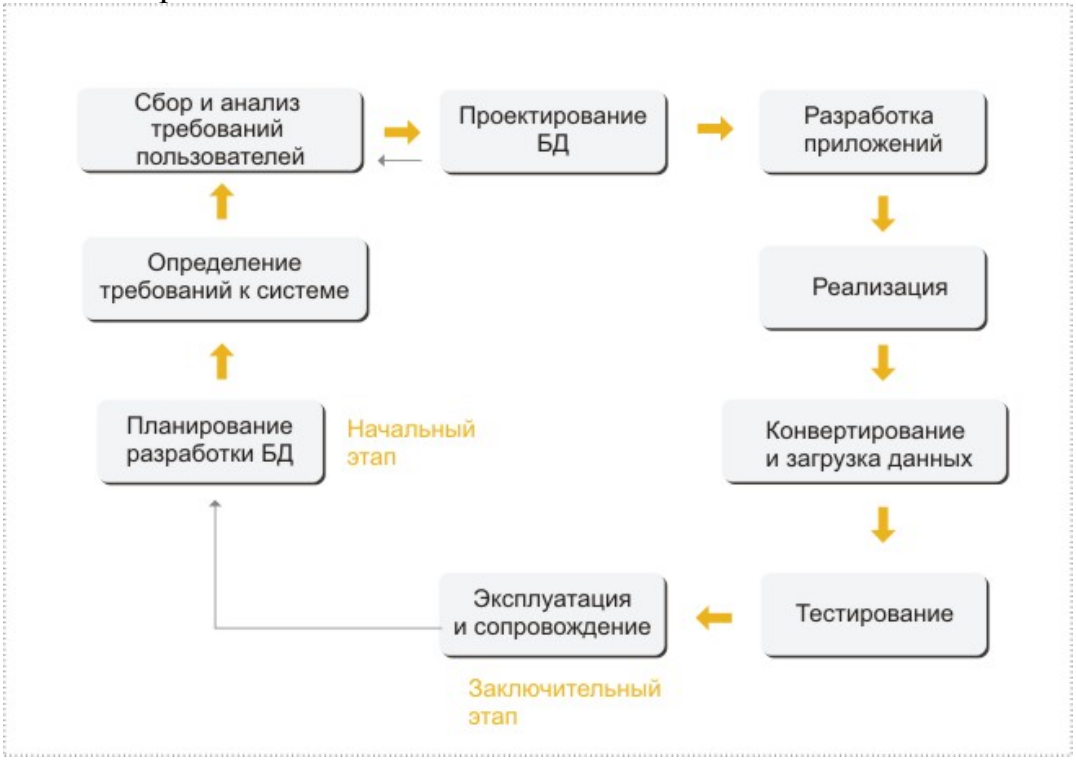


Рис. 3.1. Жизненный цикл БД

3.1.1. Планирование разработки базы данных

Содержание данного этапа — разработка стратегического плана, в процессе которой осуществляется предварительное планирование конкретной системы управления базами данных.

Планирование разработки базы данных состоит в определении трех основных компонентов: объема работ, ресурсов и стоимости проекта.

Важной частью разработки стратегического плана является проверка осуществимости проекта, состоящая из нескольких частей.

Первая часть — проверка технологической осуществимости. Она состоит в выяснении вопроса, существует ли оборудование и программное обеспечение, удовлетворяющее информационным потребностям фирмы.

Вторая часть — проверка операционной осуществимости — выяснение наличия экспертов и персонала, необходимых для работы БД.

Третья часть — проверка экономической целесообразности осуществления проекта. При исследовании этой проблемы весьма важно дать оценку ряду факторов, в том числе и таким:

- целесообразность совместного использования данных разными отделами;
- величина риска, связанного с реализацией системы базы данных;

- ожидаемая выгода от внедрения подлежащих созданию приложений;
- время окупаемости внедренной БД;
- влияние системы управления БД на реализацию долговременных планов организации.

3.1.2. Определение требований к системе

На данном этапе необходимо определить диапазон действия приложения базы данных, состав его пользователей и области применения.

Определение требований включает выбор целей БД, выяснение информационных потребностей различных отделов и руководителей фирмы и требований к оборудованию и программному обеспечению.

3.1.3. Сбор и анализ требований пользователей

На данном этапе необходимо создать для себя модель движения важных материальных объектов и уяснить процесс документооборота. По каждому документу необходимо установить периодичность использования, определить данные, необходимые для выполнения выделенных функций (анализируя существующую и планируемую документацию, выясняют, как получается каждый элемент данных, кем получается, где в дальнейшем используется, кем контролируется.

Собранная информация о каждой важной области применения приложения и пользовательской группе должна включать следующие компоненты: исходную и генерируемую документацию, подробные сведения о выполняемых транзакциях, а также список требований с указанием их приоритетов.

Формализация собранной на этом этапе информации может быть повышена с помощью методов составления спецификаций требований, к числу которых относятся, например, технология структурного анализа и проектирования, диаграммы потоков данных и графики "вход — процесс — выход".

3.1.4. Проектирование базы данных

Полный цикл разработки базы данных включает концептуальное, логическое и физическое ее проектирование.

Концептуальное проектирование базы данных

Первая фаза процесса проектирования базы данных заключается в создании для анализируемой части предприятия *концептуальной модели данных.*

Проектирование сложных баз данных с большим количеством атрибутов осуществляется использованием, так называемого, нисходящего подхода.

Этот подход начинается с разработки моделей данных, которые содержат несколько высокоуровневых сущностей и связей, затем работа продолжается в виде серии нисходящих уточнений низкоуровневых сущностей, связей и относящихся к ним атрибутов.

Нисходящий подход демонстрируется в концепции модели "сущность — связь" (Entity-Relationship model — ER-модель) — самой популярной технологии высокоуровневого моделирования данных, предложенной П. Ченом.

Модель "сущность — связь" относится к семантическим моделям. *Семантическое моделирование* данных, связанное со смысловым содержанием данных, независимо от их представления в ЭВМ.

В построении *общей концептуальной модели данных* выделяют ряд этапов.

- Выделение локальных представлений, соответствующих обычно относительно независимым данным. Каждое такое представление проектируется как подзадача.
- Формулирование сущностей, описывающих локальную предметную область проектируемой БД, и описание атрибутов, составляющих структуру каждой сущности.
- Выделение ключевых атрибутов.
- Спецификация связей между сущностями. Удаление избыточных связей.
- Анализ и добавление неключевых атрибутов.
- Объединение локальных представлений.

Созданная концептуальная модель данных предприятия является источником информации для фазы логического проектирования базы данных.

Логическое проектирование базы данных

Цель второй фазы проектирования базы данных состоит в создании логической модели данных для исследуемой части предприятия.

Логическая модель, отражающая особенности представления о функционировании предприятия одновременно многих типов пользователей, называется *глобальной логической моделью данных.*

Процесс проектирования БД должен опираться на определенную модель данных (реляционная, сетевая, иерархическая), которая определяется типом предполагаемой для реализации информационной системы СУБД.

Концептуальное и логическое проектирование — это итеративные процессы, которые включают в себя ряд уточнений, продолжающиеся до тех пор, пока не будет получен наиболее соответствующий структуре предприятия продукт.

Физическое проектирование базы данных

Целью проектирования на данном этапе является создание описания СУБД ориентированной модели БД.

Действия, выполняемые на этом этапе, слишком специфичны для различных моделей данных, поэтому их сложно обобщить. Остановимся на реляционной модели данных. В этом случае под физическим проектированием подразумевается:

- создание описания набора реляционных таблиц и ограничений для них на основе информации, представленной в глобальной логической модели данных;
- определение конкретных структур хранения данных и методов доступа к ним, обеспечивающих оптимальную производительность системы с базой данных;
- разработка средств зашиты создаваемой системы.

3.1.5. Разработка приложений

Параллельно с проектированием системы базы данных выполняется разработка приложений. Главные составляющие данного процесса — это проектирование транзакций и пользовательского интерфейса.

Проектирование транзакций

Транзакции представляют некоторые события реального мира. Транзакция может состоять из нескольких операций, однако с точки зрения пользователя эти операции представляют собой единое целое, переводящее базу данных из одного непротиворечивого состояния в другое. Реализация транзакций опирается на тот факт, что СУБД способна обеспечивать сохранность внесенных во время транзакции изменений в БД и непротиворечивость базы данных даже в случае возникновения сбоя.

Проектирование транзакций заключается в определении:

- данных, которые используются транзакцией;
- функциональных характеристик транзакции;
- выходных данных, формируемых транзакцией;
- степени важности и интенсивности использования транзакции.

Проектирование пользовательского интерфейса

Интерфейс должен быть удобным и обеспечивать все функциональные возможности, предусмотренные в спецификациях требований пользователей.

Специалисты рекомендуют при проектировании пользовательского интерфейса использовать следующие основные элементы и их характеристики:

- содержательное название;
- ясные и понятные инструкции;

- логически обоснованные группировки и последовательности полей;
- визуально привлекательный вид окна формы или поля отчета;
- легко узнаваемые названия полей;
- согласованную терминологию и сокращения;
- согласованное использование цветов;
- визуальное выделение пространства и границ полей ввода данных;
- удобные средства перемещения курсора;
- средства исправления отдельных ошибочных символов и целых полей;
- средства вывода сообщений об ошибках при вводе недопустимых значений;
- особое выделение необязательных для ввода полей;
- средства вывода пояснительных сообщений с описанием полей;
- средства вывода сообщения об окончании заполнения формы.

3.1.6. Реализация

На данном этапе осуществляется физическая реализация базы данных и разработанных приложений, позволяющих пользователю формулировать требуемые запросы к БД и манипулировать данными в БД.

База данных описывается на языке определения данных выбранной СУБД. В результате компиляции его команд и их выполнения создаются схемы и пустые файлы базы данных. На этом же этапе определяются и все специфические пользовательские представления.

Прикладные программы реализуются с помощью языков третьего или четвертого поколений. Кроме того, на этом этапе создаются другие компоненты проекта приложения — например, экраны меню, формы ввода данных и отчеты.

Реализация этого, а также и более ранних этапов проектирования БД может осуществляться с помощью инструментов автоматизированного проектирования и создания программ, которые принято называть CASE-инструментами (Computer-Aided Software Engineering).

3.1.7. Загрузка данных

На этом этапе созданные в соответствии со схемой базы данных пустые файлы, предназначенные для хранения информации, должны быть заполнены данными. Наполнение базы данных может протекать по-разному, в зависимости от того, создается ли база данных вновь или новая база данных предназначена для замены старой.

3.1.8. Тестирование

Для оценки законченности и корректности выполнения приложения базы данных может использоваться несколько различных стратегий тестирования:

- нисходящее тестирование;
- восходящее тестирование;
- тестирование потоков;
- интенсивное тестирование.

Нисходящее тестирование начинается на уровне подсистем с модулями, которые представлены заглушками, т. е. простыми компонентами, имеющими такой же интерфейс, как модуль, но без функционального кода. Каждый модуль низкого уровня представляется заглушкой. Постепенно все программные компоненты заменяются фактическим кодом и после каждой замены снова тестируются.

Восходящее тестирование выполняется в противоположном направлении по отношению к нисходящему. Оно начинается с тестирования модулей на самых низких уровнях иерархии системы, продолжается на более высоких уровнях и заканчивается на самом высоком уровне.

Тестирование потоков осуществляется при тестировании работающих в реальном масштабе времени систем, которые обычно состоят из большого количества взаимодействующих процессов, управляемых с помощью прерываний. Стратегия тестирования потоков направлена на слежение за отдельными процессами.

Стратегия *интенсивного тестирования* часто включает серию тестов с постепенно возрастающей нагрузкой и продолжается до тех пор, пока система не выйдет из строя.

3.1.9. Эксплуатация и сопровождение

Основные действия, связанные с этим этапом сводятся к наблюдению за созданной системой и поддержке ее нормального функционирования по окончании развертывания.

Поддержка БД предполагает разрешение проблем, возникающих в процессе эксплуатации БД и связанных как с ошибками реализации БД, так и с изменениями в самой предметной области, созданием дополнительных программных компонент или модернизацией самой БД.

3.2. Концептуальное проектирование

3.2.1. Фундаментальные понятия

Для нормального функционирования информационной системы необходимо, чтобы концептуальная модель адекватно отображала реалии той предметной области, для которой она разрабатывается. Методологии, позволяющие эффективно отображать существующую смысловую содержательность реальности в конструкции модели, относятся к так называемым *семантическим* методологиям.

Наиболее популярной семантической моделью стала уже упоминавшаяся раннее модель "сущность — связь" (ER-модель), предложенная П. Ченом в 1976 году, которая с тех пор неоднократно усовершенствовалась самим Ченом и многими другими специалистами.

Главными элементами семантической модели данных являются *сущности, их атрибуты* и *типы связей.* Сущности часто представляют в виде *существительных,* а типы связей — в виде *глаголов.*

Семантическая модель предметной области изображается в виде диаграммы с учетом принятых обозначений для ее элементов {рис. 3.2).

3.2.2. Сущности

Сущность — это то, о чем накапливается информация в информационной системе и что может быть однозначно идентифицировано.

Сущность - тип (в дальнейшем просто сущность) характеризуется независимым существованием и представляет множество объектов реального мира с одинаковыми свойствами. Отдельные объекты, которые входят в данный тип, называют экземплярами сущности.

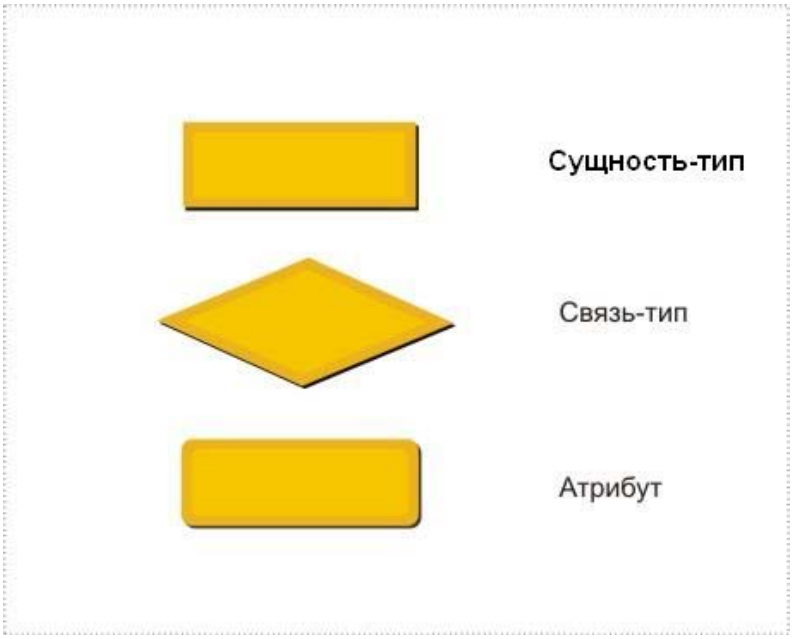


Рис. 3.2. Обозначения элементов диаграммы

Каждая сущность имеет имя и изображается на диаграммах в виде прямоугольника, а экземпляр сущности — в виде точки в прямоугольнике данной сущности (рис. 3.3).

Рис. 3.3. Обозначение сущности на ER-диаграмма

3.2.3. Атрибуты

Атрибут — это поименованная характеристика сущности, с помощью которой моделируется ее свойство. Каждой сущности присущи свои атрибуты. Например, сущность **ТОВАР** должна иметь такие атрибуты: Наименование_товара, Индекс_товара, Цена_товара, Количество. На диаграммах атрибуты сущности соединяются с ней линиями (рис. 3.4).

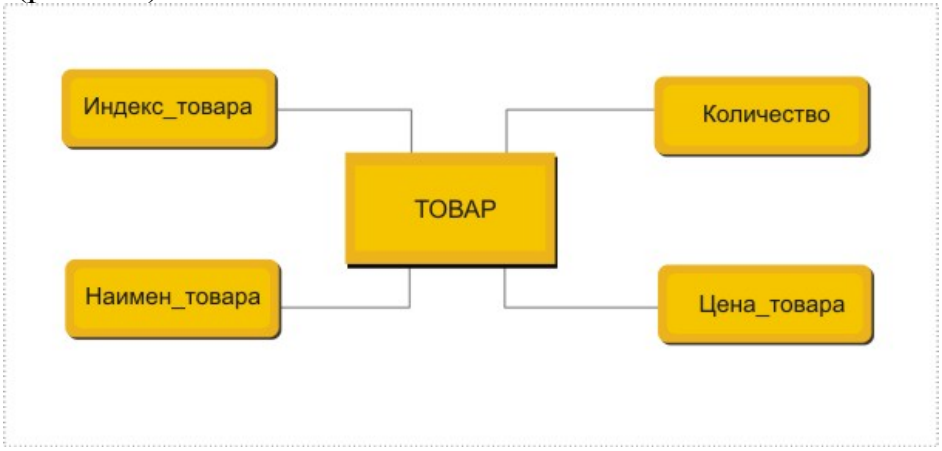


Рис. 3.4. Диаграмма представлений сущности ТОВАР и ее атрибутов

Значения каждого атрибута выбираются из соответствующего множества значений, включающего все потенциальные значения, которые могут быть присвоены атрибуту. Это множество значений называется *доменом.*

Сущность и экземпляр сущности могут быть определены следующим образом: Сущность: **СТУДЕНТ (ФИО, Группа, Год_рождения)**

Экземпляр сущности (Петров П.И., 93-ОА-22, 1992) .

Значения атрибутов могут часто меняться, в то время как описываемая ими сущность остается той же самой. Так, у экземпляра сущности **СТУДЕНТ** может измениться значение атрибута **ФИО,** но сама сущность останется той же.

3.2.4. Ключи

Среди атрибутов особое положение занимают такие, с помощью которых можно идентифицировать экземпляр сущности. Такие атрибуты называются *ключами.* Атрибут или несколько атрибутов, значения которых уникальным образом идентифицируют каждый экземпляр сущности, являются *потенциальным* ключом данной сущности. Потенциальных ключей может быть несколько. Например, экземпляр сущности **ФАКУЛЬТЕТ (Код_факультета, Название_факультета, ФИО_декана)** может однозначно идентифицироваться любым из первых двух указанных атрибутов.

Один из потенциальных ключей может быть выбран в качестве *первичного* ключа. Обычно в качестве первичного ключа выбирается тот, который имеет наименьшую длину. Остальные потенциальные ключи называются *альтернативными.* Тот факт, что атрибут служит первичным ключом, отмечается его подчеркиванием.

Идентификацию некоторых сущностей иногда приходится осуществлять при помощи *составных* ключей, которые включают несколько атрибутов.

Например, сущность:

ЛЕЧЕНИЕ (ФИО врача, ФИО пациента, Дата назначения, Лекарство) ,

однозначно идентифицировать можно только составным ключом: **(ФИО_врача, ФИО_пациента, Дата_назначения).**

3.2.5. Связи между сущностями

Две сущности могут быть связаны между собой. Подобная связь осуществляется через связь экземпляров одной сущности с экземплярами другой сущности, образуя набор экземпляров связи между двумя сущностями, который называется типом связи.

Каждому типу связи присваивается имя, которое должно представлять его функцию. Рассмотрим сущности **ПРЕПОДАВАТЕЛЬ** и **КУРС.** Между этими сущностями можно определить связь **ЧИТАЕТ,** сопоставив каждому преподавателю ту дисциплину, по которой он читает лекции, или, наоборот, каждой дисциплине — преподавателя. Связь **ЧИТАЕТ** составлена из множества пар, в каждой из которых преподаватель — из сущности **ПРЕПОДАВАТЕЛЬ**, а дисциплина — из сущности **КУРС** (рис. 3.5).

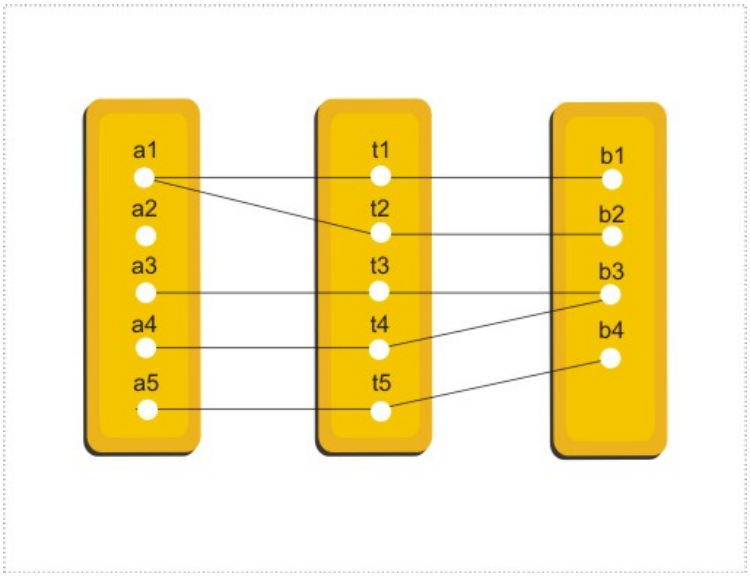


Рис. 3.5. Экземпляры типа связи ЧИТАЕТ

Полученная структура сама по себе является сущностью, состоящей из пар экземпляров, взятых из двух сущностей, связанных между собой. Сущность **ЧИТАЕТ**, полученная путем связи между сущностями **ПРЕПОДАВАТЕЛЬ** и **КУРС**, называется *составной сущностью.*

Описанная ситуация на диаграммах имеет свое графическое изображение, где тип связи обозначается в виде ромбика с указанным на нем именем связи, который соединен линиями со связываемыми сущностями (рис. 3.6).

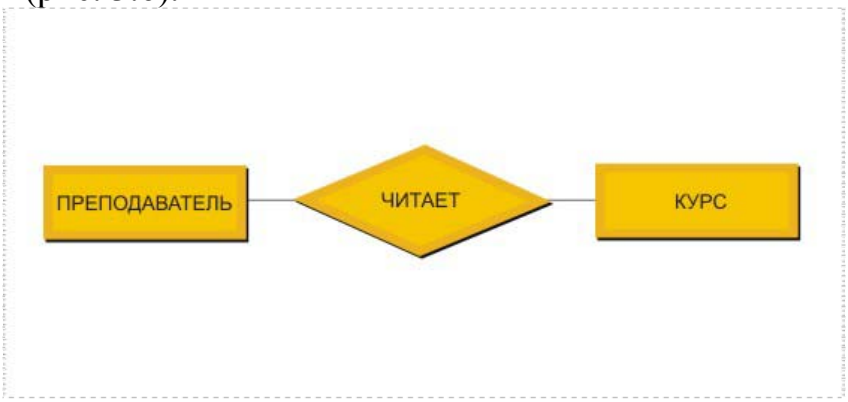


Рис. 3.6. Диаграмма типа связи **ЧИТАЕТ**

В связи могут участвовать не две, а большее количество сущностей, которые в данном случае являются участниками этой связи. Количество участников некоторой связи называется степенью связи.

В подавляющем числе случаев проектирования БД можно ограничиться рассмотрением бинарных связей.

Мощность связи

Мощность обозначает максимальное количество экземпляров одной сущности, связанных с одним экземпляром другой сущности. Например,

если допустить, что у человека может быть только один супруг, то мощность связи ЖЕНАТЫ будет равна одному в каждом направлении (рис. 3.7).

Иногда помимо максимальной мощности полезно определять и минимальную мощность. В рассматриваемом примере не исключаются одинокие мужчины и женщины, поэтому минимальная мощность равна нулю в каждом направлении. Такая ситуация может быть обозначена следующим образом:

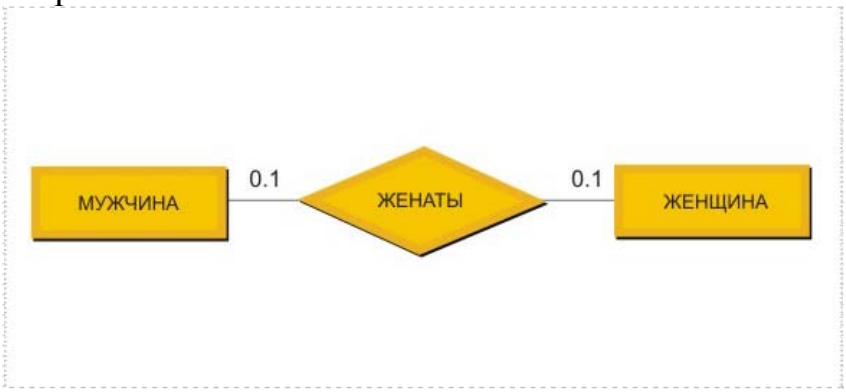


Рис. 3.7. Диаграмма связи **ЖЕНАТЫ** с указание минимальной и максимальной мощности

Некоторые связи не имеют конкретного значения максимальной мощности Например, преподаватель может читать не один курс, а, возможно, больше. Такую мощность обозначают: 1,*, где 1 обозначает минимальную мощность. а * обозначает "много" (существует и другой способ обозначения "много" — вместо * ставится N). С другой стороны, если допустить, что каждый данный курс читается одним и только одним преподавателем, то мощность в обратном направлении будет 1,1.

Показатель кардинальности

Для того чтобы указать количество возможных связей для каждого экземпляра участвующего в связи сущности, используют *показатель кардинальности.*

Для бинарных связей показатель кардинальности может иметь следующие значения:

"один к одному" (1 : 1), *"один ко многим"*(1 : N*), "многие ко многим"* (M : N).

Если максимальная мощность связи в обоих направлениях равна одному, мы называем ее связью *"один к одному"* (1 : 1).

Например, на факультете может быть один декан, и обратно, один и тот же декан может руководить только одним факультетом, что может быть обозначено следующим образом:

ФАКУЛЬТЕТ <-----> ДЕКАН.

Если максимальная мощность в одном направлении равна одному, а в другом — многим, то связь называется *"один ко многим"* (1 : N).

Например, в группе учится много студентов, но каждый студент учится только в одной группе:

ГРУППА <----->> СТУДЕНТ.

Концептуальная модель подобной связи приведена на рис. 3.8.

Рис. 3.8. Диаграмма типа связи **УЧИТСЯ** с указанием показателя кардинальности

На диаграмме использовано два способа обозначения вида бинарной связи: символическая (со стороны сущности **ГРУППА** выход связи помечен символом 1, а со стороны сущности **СТУДЕНТ** — символом N) и стрелками (в направлении, где максимальная мощность равна многим, проставлена двойная стрелка, а со стороны, где она равна единице — одинарная). Реально при построении диаграмм выбирают один из них.

И наконец, если максимальная мощность в обоих направлениях равна многим, то такая связь относится к типу *"многие ко многим"* (M : N). Например, преподаватель работает в разных группах, и в одной и той же группе работают различные преподаватели:

ПРЕПОДАВАТЕЛЬ <<----->> ГРУППА.

Связь между сущностями осуществляется посредством атрибутов. Например, рассмотрим две сущности:

Сущность: **СТУДЕНТ**

Атрибуты: **Номер_зачетной_книжки**

ФИО студента

Сущность: **ГРУППА**

Атрибуты: **Код группы**

Количество студентов

ФИО старосты

Для их связи в число атрибутов сущности **СТУДЕНТ** необходимо добавить код группы, в которой он учится, и значение которого будет использовано для связи экземпляра одной сущности с экземпляром другой сущности.

3.2.6. Супертип и подтип

Для удовлетворения новых требований, выдвигаемых все более усложняющимися приложениями, в семантическое моделирование были введены дополнительные концепции, расширяющие его возможности. Дополнительные концепции базируются на таких понятиях, как супертип и подтип, а также используют процесс наследования атрибутов.

Супертип — это сущность, включающая разные подтипы, которые необходимо представить в модели данных.

Подтип — это сущность, являющаяся членом супертипа, но выполняющая отдельную роль в нем.

Супертип может иметь несколько разных подтипов. Так, например, подтипы: **АССИСТЕНТ, СТАРШИЙ ПРЕПОДАВАТЕЛЬ, ДОЦЕНТ, ПРОФЕССОР** являются членами супертипа **ПРЕПОДАВАТЕЛЬ.** Это означает, что каждый экземпляр подтипа является в то же время и экземпляром супертипа. Связь между супертипом и подтипом относится к типу "один к одному".

Использование понятий супертипа и подтипов позволяет при моделировании выделить для подтипа свои собственные атрибуты и атрибуты, наследуемые им от супертипа.

На диаграмме (рис. 3.9) подтипы соединяются линиями с кружком, который в свою очередь соединяется с супертипом. На каждой линии, идущей от подтипа, располагается U-образный символ, который обозначает направление включения. Верхняя часть **U** "открывается" в сторону супертипа. Внутри кружка располагается буква **D**, если подтипы не пересекаются, и буква **O** — для пересекающихся подтипов. В последнем случае экземпляр супертипа может быть членом сразу нескольких подтипов. Изображенная на диаграмме ситуация исключает пересечение подтипов, поэтому в кружок помещен символ **D**.

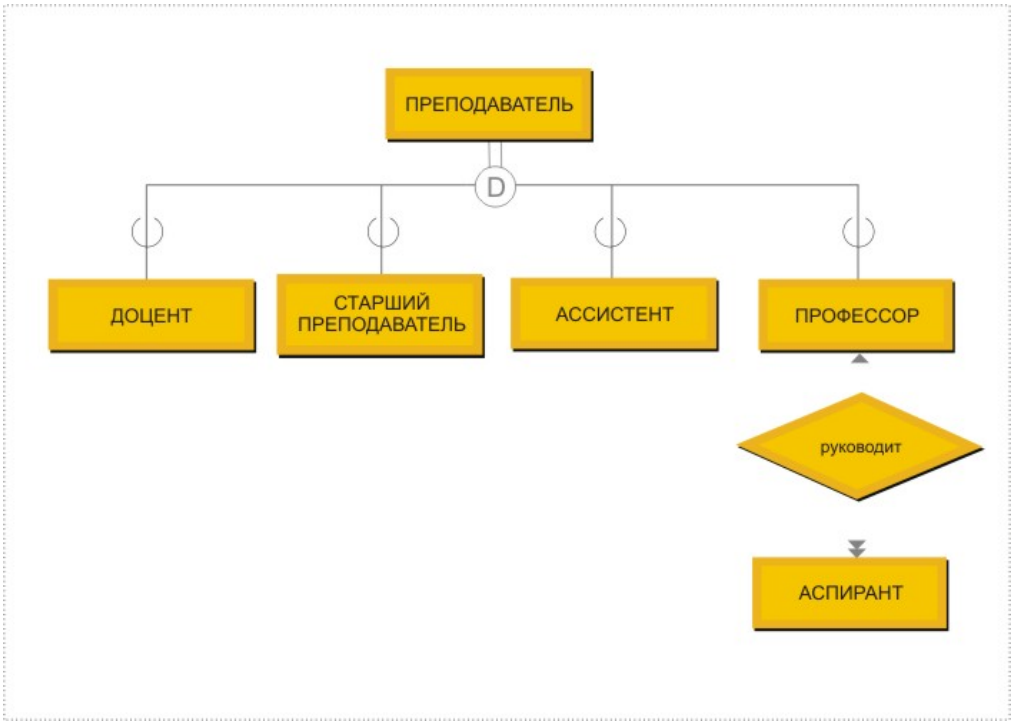


Рис. 3.9. Диаграмма с использованием понятий супертип и подтип

3.3. Пример моделирования локальной ПрО

С помощью рассмотренных выше понятий могут быть получены ER-модели для большинства схем баз данных в традиционных административно-управленческих приложениях. Если ПрО обширная, то построение ее концептуальной модели будет протекать более успешно, если эту ПрО разбить на несколько локальных предметных областей. Объем локальной ПрО выбирается таким образом, чтобы в нее входило не более 6-7 сущностей. Как ранее упоминалось, отправными элементами для построения ER-модели локальной ПрО очень часто являются используемые в организации документы.

Предположим, что определена локальная ПрО: поставка товаров на склад. Пусть используемая форма поставки имеет вид, как на рис. 3.10.

Покажем, как, используя приведенную форму, можно построить концептуальную модель этой небольшой локальной предметной области.

Рис. 3.10. Форма поставки

Итак, анализируемая форма содержит следующую информацию: **Поставщик, Индекс поставщика, Адрес поставщика, Товар. Индекс товара, Цена товара. Количество товара, Поставка, Индекс поставщика, Дата поставки** и **Номер склада.**

Выделим две сущности: **ПОСТАВЩИК** и **ТОВАР** (рис. 3.11).

Оставшиеся атрибуты характеризуют сущность — **ПОСТАВКА.** Сформируем ее и установим определенные типы бинарных связей между тремя сущностями, исходя из следующих рассуждений: один и тот же поставщик может осуществить ряд поставок, но каждая поставка осуществляется только одним поставщиком.

Мощность связи между сущностями **ПОСТАВКА** и **ТОВАР** должна быть установлена M:N, так как каждая поставка может содержать несколько товаров, и один и тот же товар может содержаться в нескольких поставках. Исходя из вышесказанного, диаграмма модели предметной области **ПОСТАВКА** примет такой вид, как на рис. 3.12.

Рис. 3.11. ПрО ПОСТАВКА

Атрибуты **Индекс поставщика, Индекс поставки** и **Индекс товара** были введены для однозначной идентификации экземпляров рассматриваемых сущностей, так как ни один из остальных атрибутов не подходит на эту роль. Как уже упоминалось, такие идентификационные атрибуты называются первичными ключами.

При построении концептуальной модели следует избегать избыточности информации. После того, как выделены сущности, ключи, определяют и удаляют имеющиеся избыточные связи. Большое внимание уделяется анализу атрибутов. Забегая вперед, следует указать на то, что в хорошо спроектированной БД должно соблюдаться правило: среди атрибутов сущности должна наблюдаться зависимость описательного атрибута от ключевого, но не должна существовать зависимость между описательными атрибутами.

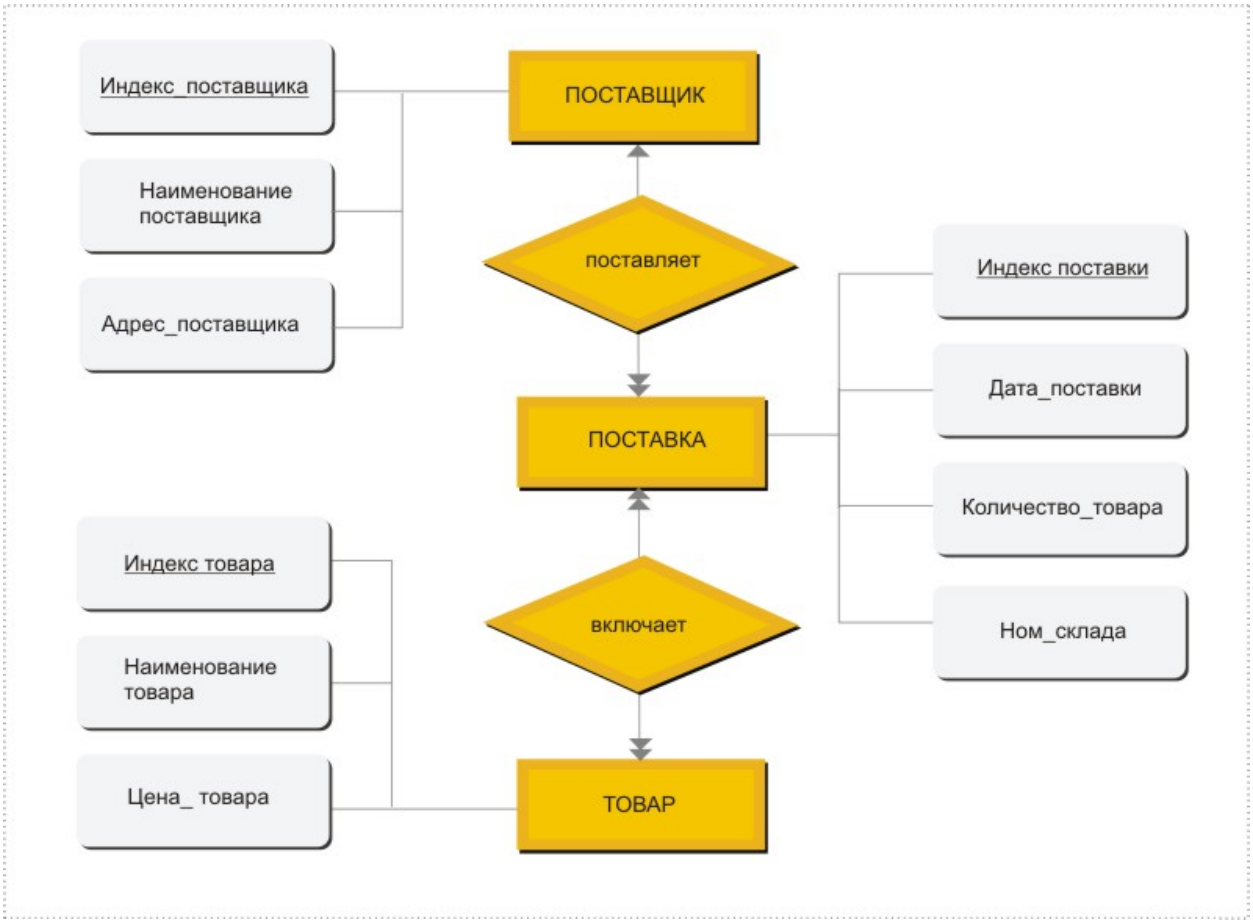


Рис. 3.12. Диаграмма модели предметной области ПОСТАВКА

Завершающим этапом построения концептуальной модели исследуемой ПрО является спецификация всех сущностей, входящих в модель. Для данного примера результаты этого шага должны быть сведены к следующему:

1. Спецификация сущностей:

ПОСТАВЩИК: Индекс_поставщика— идентификационный атрибут
Адрес_поставщика — описательный атрибут
Наименование_поставщика — описательный атрибут
ПОСТАВКА: Индекс_поставки — идентификационный атрибут
Количество_товара— описательный атрибут
Дата_поставки — описательный атрибут
Номер_склада — описательный атрибут
ТОВАР: Индекс_товара — идентификационный атрибут
Наименование_товара — описательный атрибут
Цена_товара — описательный атрибут

2. Спецификация типов связей:

ПОСТАВЛЯЕТ: связь **ПОСТАВЩИК** <-----> **ПОСТАВКА** 1:N
ВКЛЮЧАЕТ: связь **ПОСТАВКА** <-----> **ТОВАР** M:N

3. Спецификация атрибутов:

Индекс_поставщика: символьный, 6 символов

Адрес_поставщика: символьный, 50 символов

Цена_товара: денежный

Сформированные спецификации заносятся в словарь данных.

После создания моделей каждой выделенной предметной области производится объединение локальных концептуальных моделей в одну общую, как правило, довольно сложную схему.

4. Модели данных

Моделью данных называется формализованное описание структуры единиц информации и операций над ними в информационной системе.

Модель данных — это некоторая абстракция, в которой отражаются самые важные аспекты функционирования выделенной предметной области, а второстепенные — игнорируются. Модель данных включает в себя набор понятий для описания данных, связей между ними и ограничений, накладываемых на данные. В модели данных различают три главные составляющие:

- структурную часть, определяющую правила порождения допустимых для данной СУБД видов структур данных;
- управляющую часть, определяющую возможные операции над такими структурами;
- классы ограничений целостности данных, которые могут быть реализованы средствами этой системы.

Каждая СУБД поддерживает ту или иную модель данных.

По существу модель данных, поддерживаемая механизмами СУБД, полностью определяет множество конкретных баз данных, которые могут быть созданы средствами этой системы, а также способы модификации состоянии БД с целью отображения тех изменений, которые происходят в предметной области.

4.1. Классификация моделей данных

В настоящее время описано много разнообразных моделей, построение которых преследует разные цели. Из множества опубликованных моделей данных можно выделить три категории:

- объектные модели данных;
- модели данных на основе записей;
- физические модели данных.

Применительно к трехуровневой архитектуре баз данных следует отметить, что первые две категории используются для описания данных на внешнем и концептуальном уровнях, а последняя категория — на внутреннем уровне.

Объектные модели данных

Среди объектных моделей следует выделить ER-модель, которая наиболее часто используется в методологии проектирования баз данных, а также объектно-ориентированную модель, последнее время широко используемую в технологиях баз данных. Объектно-ориентированная модель расширяет понятие объекта, включая в него не только атрибуты, характеризующие состояние объекта, но и связанные с ним действия.

Модели данных на основе записей

В модели данных на основе записей база данных состоит из нескольких записей фиксированного формата, которые могут иметь разные типы.

В большинстве коммерческих СУБД используются ставшие классическими два типа такого рода моделей данных: теоретико-графовые (ТГ) и теоретико-множественные (ТМ) модели данных.

К теоретико-графовым моделям относятся две разновидности:

- сетевые модели;
- иерархические модели.

В таких моделях данных предусматриваются характерные для подобного рода структур операции навигации и манипулирования данными.

Аппарат навигации в ТГ-моделях служит для установки тех объектов данных, к которым будет применяться очередная операция манипулирования данными.

Теоретико-множественные модели используют математический аппарат, реляционную алгебру (знаковая обработка множеств), реляционное исчисление. К моделям данного типа относятся реляционные модели.

В соответствии с реляционной моделью данных БД представляется в виде совокупности таблиц, над которыми могут выполняться операции, формируемые в терминах реляционной алгебры и реляционного исчисления.

Физическая модель данных

Физические модели данных описывают то, как данные хранятся в компьютере, представляя информацию о структуре записей, их упорядоченности и существующих путях доступа. Наиболее распространены из них следующие: обобщающая модель и модель памяти кадров.

4.2. Сетевая модель

Сети — естественный способ представления отношений между объектами, всевозможных их взаимосвязей. Сетевая модель опирается на математическую структуру, которая называется направленным графом. Направленный граф состоит из узлов, соединенных ребрами. В контексте

моделей данных узлы представляют собой объекты в виде типов записей данных, а ребра — связи между объектами со степенью кардинальности "один к одному" или "один ко многим.

Иерархическая модель данных является частным случаем сетевой модели.

4.2.1. Структуры данных сетевой модели

В сетевой модели используются несколько различных типовых структур данных, главными из которых являются типы записей и наборы. Для построения этих структур применяются такие конструктивные элементы, как элемент данных и агрегат (рис. 4.1).

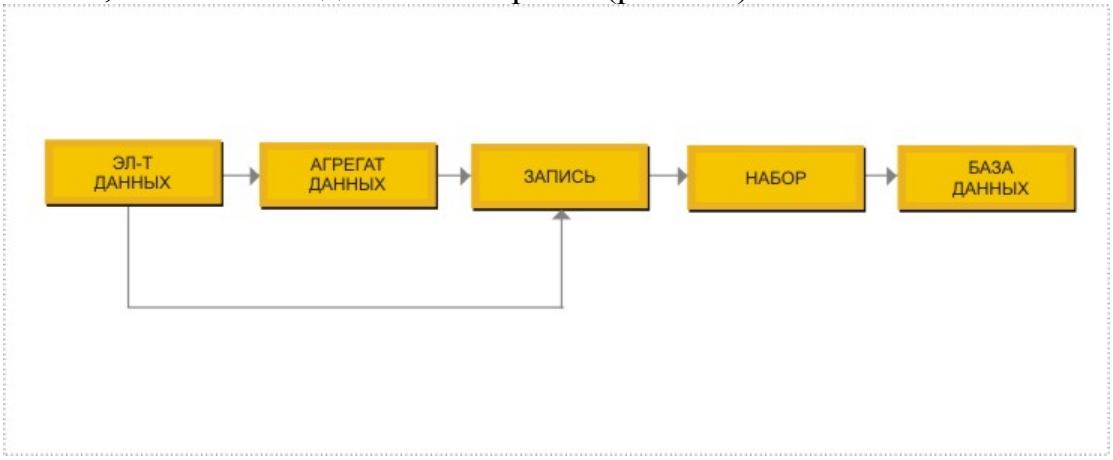


Рис. 4.1. Основные структуры сетевой модели данных

Элемент данных — это наименьшая поименованная информационная единица данных, доступная пользователю (аналог — поле в файловой системе). Элемент данных должен иметь свой тип (не структурный, простой).

Агрегат данных соответствует следующему уровню обобщения — поименованная совокупность элементов данных внутри записи или другого агрегата (рис. 4.2).

Рис. 4.2. Агрегат **Дата**

Запись — конечный уровень агрегации. Каждая запись представляет собой именованную структуру, содержащую один или более именованных элементов данных, каждый из которых обладает своим особым форматом.

Агрегат данных **Дата** входит в состав записи **Сотрудник** (рис. 4.3).

Тип записей — это совокупность логически связанных экземпляров записей. Тип записей моделирует некоторый класс объектов реального мира.

В качестве элемента данных могут быть использованы только простые типы, а в качестве агрегатов могут быть использованы сложные типы: вектор и повторяющаяся группа.

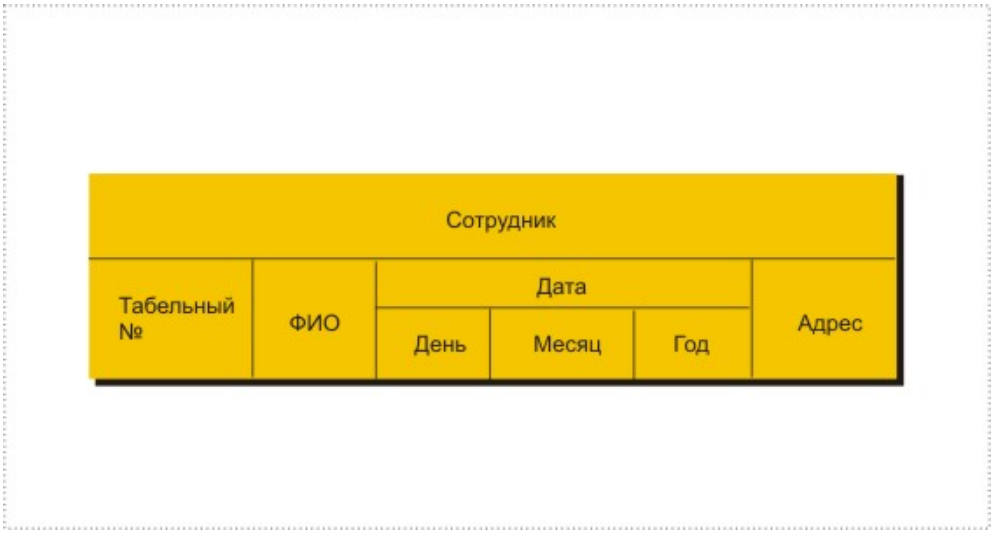


Рис. 4.3. Запись **Сотрудник**

Агрегат типа вектор соответствует линейному набору элементов данных (рис. 4.2). Агрегат типа повторяющаяся группа соответствует совокупности векторов данных. Так, например, в заказе может быть указано несколько видов товаров с числом повторений 10 (рис. 4.4).

Рис. 4.4. Запись **Заказ**

Набор — это поименованная двухуровневая иерархическая структура, которая содержит запись владельца и записи членов. Наборы выражают связи "один ко многим" или "один к одному" между двумя типами записей. Тип набора поддерживает работу с внутренними структурами типов записей.

Набор, приведенный на рис. 4.5, определяет тип записи-владельца **Отдел** и тип записи-члена набора **Сотрудник**, а также тип связи между ними "один ко многим" — с именем **Работает**. Имя набора — это метка, присвоенная стрелке. Связь типа "один ко многим" допускает возможность того, что с данным экземпляром записи-владельца может быть связан ноль, один, или несколько экземпляров записи-члена.

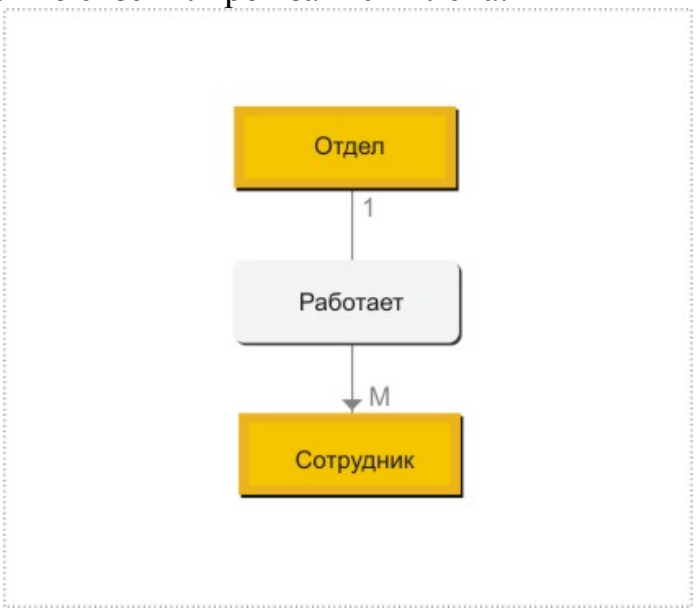


Рис. 4.5. Диаграмма типа набора **Работает**

Используя понятия сетевой модели данных, можно получить другое изображение такого набора (рис. 4.6), где представлены логические типы записей **Отдел** и **Сотрудник**, их структура и связь между типами записей **Работает.**

База данных в сетевой модели данных — это поименованная совокупность экземпляров записей различного типа и экземпляров наборов, содержащих связи между ними.

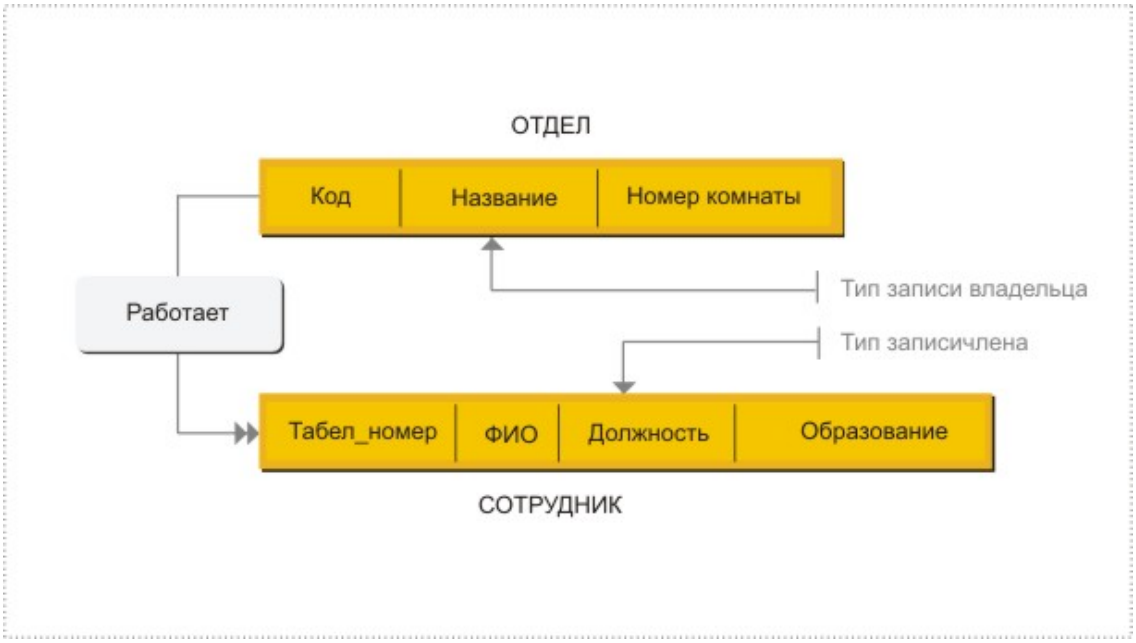


Рис. 4.6. Набор **Работает** между двумя типами записей **Отдел** и **Сотрудник**

4.2.2. Преобразование концептуальной модели в сетевую

Сетевая модель данных может быть без осложнений получена из концептуальной модели. Для этого надо предположить, что в последней используются только бинарные связи. Причем они должны принадлежать к типам "один к одному" или "один ко многим". При этом вместо сущностей концептуальной модели необходимо использовать типы записей сетевой модели, где имена сущностей становятся именами типов записей, атрибуты сущностей становятся полями записей, связь между сущностями превращается в связь между типами записей.

Бинарные связи, принадлежащие к типу "один ко многим", переносятся в сетевую модель следующим образом: тип записи со стороны "один" становится владельцем, а тип записи со стороны "много" становится типом записи-члена. Для связи типа "один к одному" выбор типа записи-владельца и типа записи-члена может быть осуществлен произвольно.

4.2.3. Управляющая часть сетевой модели

Для манипулирования данными в сетевой модели данных определен ряд типичных операций, которые можно подразделить на две группы: навигационные операции и операции модификации.

Навигационные операции осуществляют перемещение по БД путем прохождения по связям, определенным в схеме БД. В результате таких операций определяется запись, которая называется текущей. К подобным операциям относятся:

- найти конкретную запись в наборе однотипных записей и сделать ее текущей;
- перейти от записи-владельца к записи-члену в некотором наборе;
- перейти к следующей записи в некоторой связи;
- перейти от записи-члена к владельцу по некоторой связи.

Операции модификации осуществляют как добавление новых экземпляров отдельных типов записей, так и экземпляров новых наборов, удаление экземпляров записей и наборов, модификацию отдельных компонентов самой записи. Для реализации этих операций в системе текущее состояние детализируется путем запоминания трех его составляющих: текущего набора, текущего типа записи, текущего экземпляра типа записи. В такой ситуации возможны следующие операции:

- извлечь текущую запись в буфер прикладной программы для обработки;
- заменить в извлеченной записи значения указанных элементов данных на заданные новые их значения;
- запомнить запись из буфера в БД;
- создать новую запись;
- уничтожить запись;
- включить текущую запись в текущий экземпляр набора;
- исключить текущую запись из текущего экземпляра набора.

Поддержание ограничений целостности в сетевых моделях в принципе не требуется.

4.3. Иерархическая модель данных

Первые системы управления базами данных использовали иерархическую модель данных, и во времени их появление предшествует появлению сетевой модели.

4.3.1. Структурная часть иерархической модели

Основными информационными единицами в иерархической модели данных являются сегмент и поле. Поле данных определяется как наименьшая неделимая единица данных, доступная пользователю. Для сегмента определяются тип сегмента и экземпляр сегмента. Экземпляр сегмента образуется из конкретных значений полей данных. Тип сегмента — это поименованная совокупность входящих в него типов полей данных.

Как и сетевая, иерархическая модель данных базируется на графовой форме построения данных, и на концептуальном уровне она является просто частным случаем сетевой модели данных. В иерархической модели данных вершине графа соответствует тип сегмента или просто сегмент, а дугам — типы связей предок — потомок. В иерархических структурах сегмент — потомок должен иметь в точности одного предка.

Иерархическая модель представляет собой связный неориентированный граф древовидной структуры, объединяющий сегменты. Иерархическая БД состоит из упорядоченного набора деревьев (рис.4.7).

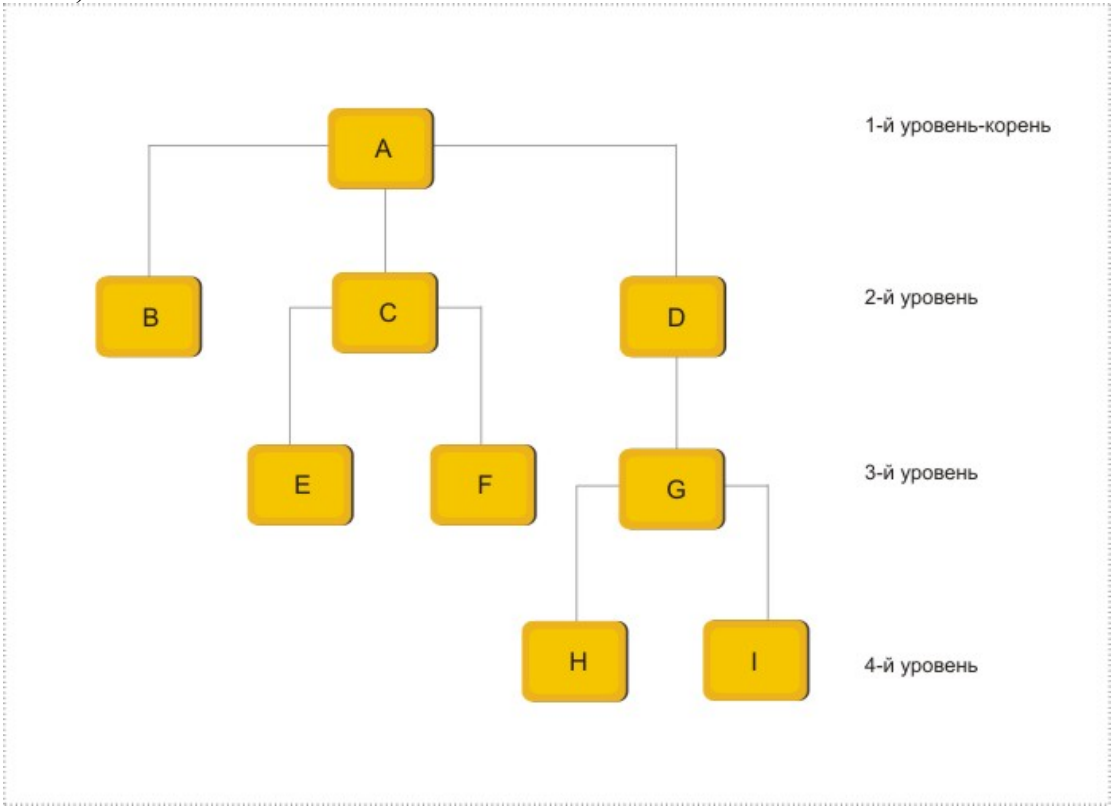


Рис. 4.7. Иерархическая модель данных

4.3.2. Преобразование концептуальной модели в иерархическую модель данных

Преобразование концептуальной модели в иерархическую структуру данных во многом схоже с преобразованием ее в сетевую модель, но и имеет некоторые отличия в связи с тем, что иерархическая модель требует организации всех данных в виде дерева.

Преобразование связи типа "один ко многим" между предком и потомком осуществляется практически автоматически в том случае, если потомок имеет одного предка, и происходит это следующим образом. Каждый объект с его атрибутами, участвующий в такой связи, становится логическим сегментом. Между двумя логическими сегментами устанавливается связь типа "один ко многим". Сегмент со стороны "много" становится потомком, а сегмент со стороны "один" становится предком.

Ситуация значительно усложняется, если потомок в связи имеет не одного, а двух и более предков. Так как подобное положение является невозможным для иерархической модели, то отражаемая структура данных нуждается в преобразованиях, которые сводятся к замене одного дерева, например, двумя (если имеется два предка). В результате такого

преобразования в базе данных появляется избыточность, так как единственно возможный выход из этой ситуации — дублирование данных.

4.3.3. Управляющая часть иерархической модели

В рамках иерархической модели выделяют языковые средства описания данных (ЯОД) и средства манипулирования данными (ЯМД).

Каждая физическая база описывается набором операторов, обусловливающих как ее логическую структуру, так и структуру хранения БД. При этом способ доступа устанавливает способ организации взаимосвязи физических записей. Определены следующие способы доступа:

- иерархически последовательный;
- иерархически индексно-последовательный;
- иерархически прямой;
- иерархически индексно-прямой;
- индексный.

Помимо задания имени БД и способа доступа описания должны содержать определения типов сегментов, составляющих БД, в соответствии с иерархией, начиная с корневого сегмента. Каждая физическая БД содержит только один корневой сегмент, но в системе может быть несколько физических БД.

Среди операторов манипулирования данными можно выделить операторы поиска данных, операторы поиска данных с возможностью модификации, операторы модификации данных. Набор операций манипулирования данными в иерархической БД невелик, но вполне достаточен.

Примеры типичных операторов поиска данных:

- найти указанное дерево БД;
- перейти от одного дерева к другому;
- найти экземпляр сегмента, удовлетворяющий условию поиска;
- перейти от одного сегмента к другому внутри дерева;
- перейти от одного сегмента к другому в порядке обхода иерархии.

Примеры типичных операторов поиска данных с возможностью модификации:

- найти и удержать для дальнейшей модификации единственный экземпляр сегмента, удовлетворяющий условию поиска;
- найти и удержать для дальнейшей модификации следующий экземпляр сегмента с теми же условиями поиска;
- найти и удержать для дальнейшей модификации следующий экземпляр для того же родителя.

Примеры типичных операторов модификации иерархически организованных данных, которые выполняются после выполнения одного

из операторов второй группы (поиска данных с возможностью модификации):

- вставить новый экземпляр сегмента в указанную позицию;
- обновить текущий экземпляр сегмента;
- удалить текущий экземпляр сегмента.

В иерархической модели автоматически поддерживается целостность ссылок между предками и потомками. Основное правило: никакой потомок не может существовать без своего родителя

5. Реляционная модель данных

5.1. История вопроса

Создателем реляционной модели является сотрудник фирмы IBM доктор Э. Ф. Кодд. Будучи по образованию математиком, Э. Кодд предложил использовать для обработки данных аппарат теории множеств. В статье "A Relational Model of Data for Large Shared Data Banks", вышедшей в свет в 1970 году, он показал, что любое представление данных сводится к совокупности двумерных таблиц особого вида, известного в математике как отношение (relation).

Положив теорию отношений в основу реляционной модели, Э. Кодд обосновал реляционную замкнутость отношений и ряда некоторых специальных операций, которые применяются сразу ко всему множеству строк отношения, а не к отдельной строке. Указанная реляционная замкнутость означает, что результатом выполнения операций над отношениями является также отношение, над которым в свою очередь можно осуществить некоторую операцию. Из этого следует, что в данной модели можно оперировать реляционными выражениями, а не только отдельными операндами в виде простых имен таблиц.

Одним из основных преимуществ реляционной модели является ее однородность. Все данные рассматриваются как хранимые в таблицах и только в таблицах. Каждая строка такой таблицы имеет один и тот же формат.

К числу достоинств реляционного подхода можно отнести:

- наличие небольшого набора абстракций, которые позволяют сравнительно просто моделировать большую часть распространенных предметных областей и допускают точные формальные определения, оставаясь интуитивно понятными;
- наличие простого и в то же время мощного математического аппарата, опирающегося главным образом на теорию множеств и математическую логику и обеспечивающего теоретический базис реляционного подхода к организации БД;

- возможность ненавигационного манипулирования данными без необходимости знания конкретной физической организации баз данных во внешней памяти.

Кодд сформулировал двенадцать приведенных ниже правил, которым должна соответствовать настоящая реляционная база данных.

1. Правило информации. Вся информация в базе данных должна быть представлена исключительно на логическом уровне и только одним способом — в виде значений, содержащихся в таблицах.
2. Правило гарантированного доступа. Логический доступ ко всем и каждому элементу данных (атомарному значению) в реляционной базе данных должен обеспечиваться путем использования комбинации имени таблицы, первичного ключа и имени столбца.
3. Правило поддержки недействительных значений. В настоящей реляционной базе данных должна быть реализована поддержка недействительных значений, которые отличаются от строки символов нулевой длины, строки пробельных символов и от нуля или любого другого числа и используются для представления отсутствующих данных независимо от типа этих данных.
4. Правило динамического каталога, основанного на реляционной модели. Описание базы данных на логическом уровне должно быть представлено в том же виде, что и основные данные, чтобы пользователи, обладающие соответствующими правами, могли работать с ним с помощью того же реляционного языка, который они применяют для работы с основными данными.
5. Правило исчерпывающего подъязыка данных. Реляционная система может поддерживать различные языки и режимы взаимодействия с пользователем (например, режим вопросов и ответов). Однако должен существовать, по крайней мере, один язык, операторы которого можно представить в виде строк символов в соответствии с некоторым четко определенным синтаксисом и который в полной мере поддерживает следующие элементы:
   - определение данных;
   - определение представлений;
   - обработку данных (интерактивную и программную);
   - условия целостности;
   - идентификацию прав доступа;
   - границы транзакций (начало, завершение и отмена).
6. Правило обновления представлений. Все представления, которые теоретически можно обновить, должны быть доступны для обновления.
7. Правило добавления, обновления и удаления. Возможность работать с отношением как с одним операндом должна существовать не

только при чтении данных, но и при добавлении, обновлении и удалении данных.

8. Правило независимости физических данных. Прикладные программы и утилиты для работы с данными должны на логическом уровне оставаться нетронутыми при любых изменениях способов хранения данных или методов доступа к ним.
9. Правило независимости логических данных. Прикладные программы и утилиты для работы с данными должны на логическом уровне оставаться нетронутыми при внесении в базовые таблицы любых изменений, которые теоретически позволяют сохранить нетронутыми содержащиеся в этих таблицах данные.
10. Правило независимости условий целостности. Должна существовать возможность определять условия целостности, специфические для конкретной реляционной базы данных, на подъязыке реляционной базы данных и хранить их в каталоге, а не в прикладной программе.
11. Правило независимости распространения. Реляционная СУБД не должна зависеть от потребностей конкретного клиента.
12. Правило единственности. Если в реляционной системе есть низкоуровневый язык (обрабатывающий одну запись за один раз), то должна отсутствовать возможность использования его для того, чтобы обойти правила и условия целостности, выраженные на реляционном языке высокого уровня (обрабатывающем несколько записей за один раз).

5.2. Структурная часть реляционной модели

5.2.1. Отношение

Реляционная база данных — это конечный (ограниченный) набор отношений. Отношения используются для представления сущностей, а также для представления связей между сущностями. Отношение — это двумерная таблица, имеющая уникальное имя и состоящая из строк и столбцов, где строки соответствуют записям, а столбцы — атрибутам. Каждая строка в таблице представляет некоторый объект реального мира или соотношения между объектами.

Атрибут — это поименованный столбец отношения. Свойства сущности, его характеристики определяются значениями атрибутов. Порядок следования атрибутов не влияет на само отношение.

Пусть имеется отношение r. Схемой отношения r называется конечное множество имен атрибутов $R = \{A_1, A_2, \ldots A_n\}$. Заголовки столбцов отношения содержат имена его атрибутов и, следовательно, все вместе отражают его схему.

Схема отношения **ПРЕПОДАВАТЕЛЬ** может быть представлена следующим образом: **{Таб_ном_преп, Фамилия, Должность}**

Или

ПРЕПОДАВАТЕЛЬ
Таб_ном_преп
Фамилия
Должность

Тогда заголовок отношения **ПРЕПОДАВАТЕЛЬ** примет вид

Таб_ном_преп	**Фамилия**	**Должность**

Отношение строится с учетом ряда факторов. Каждому имени атрибута A_i, 1 >= i <= n ставится в соответствие множество допустимых для соответствующего столбца значений. Это множество D_i, называется доменом данного имени атрибута.

Каждая строка отношения является множеством значений, взятых по одному из домена каждого имени атрибута. Домены являются произвольными непустыми конечными или счетными множествами и образуют множество:

$D = D_1 \cup D_2 \cup ... \cup D_n$.

Отношение r со схемой R — это конечное множество отображений $\{t_1, t_2, ... t_p)$ из R в D. Причем каждое отображение $t \in$ r должно удовлетворять следующему ограничению:

$t(A_i)$ принадлежит D_i где 1 <= i <= n.

Эти отображения называются *кортежами.* Каждый кортеж отношения отображает экземпляр сущности, а атрибут отношения отображает атрибут сущности.

Множество кортежей называется *телом отношения.* Тело отношения отражает состояние сущности, поэтому во времени оно постоянно меняется. Тело отношения характеризуется *кардинальным числом,* которое равно количеству содержащихся в нем кортежей.

Одной из главных характеристик отношения является его степень. *Степень отношения* определяется количеством атрибутов, которое в нем присутствует. Эта характеристика отношения имеет еще названия: ранг и арность. Отношение с одним атрибутом называется унарным, с двумя атрибутами — бинарным, с тремя — тернарным, с n атрибутами n-арным. Определение степени отношения осуществляется по заголовку отношения.

Все названные характеристики отношения обозначены на рис. 5.1

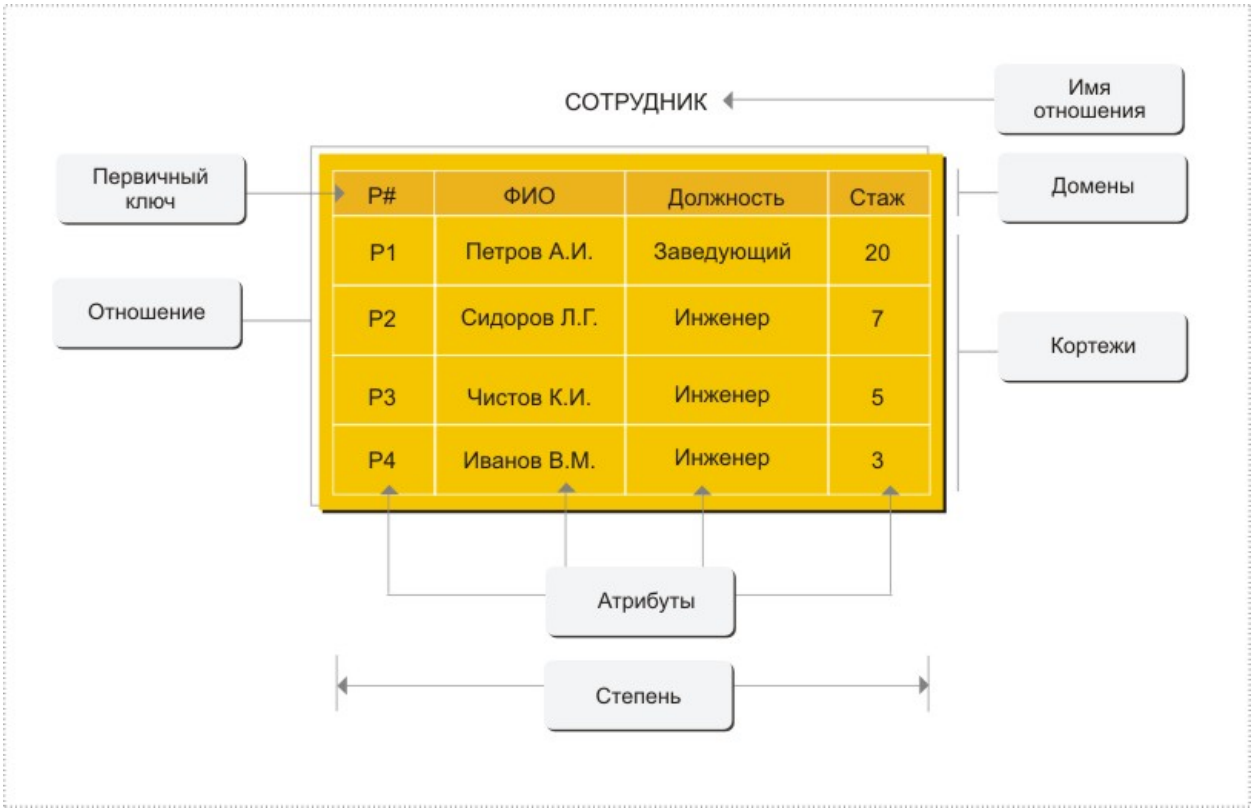


Рис. 5.1. Отношение **СОТРУДНИК**

5.2.2. Свойства и виды отношений

Отношение по структуре подобно таблице, но таблице, обладающей определенными свойствами. Сведем воедино все свойства отношения.

- Отношение имеет имя, которое отличается от имен всех других отношений.
- Отношение представляется в виде табличной структуры.
- Каждый атрибут имеет уникальное имя, его значения берутся из одного и того же домена.
- Каждый компонент кортежа является простым, атомарным значением, не состоящим из группы значений.
- Упорядочение атрибутов теоретически несущественно, однако оно может влиять на эффективность доступа к кортежам.
- Все строки (кортежи) должны быть различны.
- Теоретически порядок следования кортежей не имеет значения.

В реляционной теории встречается несколько видов отношений, но не все они поддерживаются реальными системами. Различают:

- именованное отношение — это переменная отношения, определенная в СУБД посредством специальных операторов;
- базовое отношение — это именованное отношение, являющееся частью базы данных;

- производное отношение — это отношение, определенное посредством реляционного выражения через базовые отношения;
- представление — это именованное виртуальное производное отношение, представленное в системе исключительно через определение в терминах других именованных отношений;
- снимки — это отношения, подобные представлениям, но они сохраняются, доступны для чтения и периодически обновляются;
- результат запроса — это неименованное производное отношение, получаемое в результате запроса, которое для сохранения необходимо преобразовать в именованное отношение;
- хранимое отношение — это отношение, которое поддерживается в физической памяти.

5.2.3. Реляционные ключи

В отношении могут существовать несколько одиночных или составных атрибутов, которые однозначно идентифицируют кортеж отношения. Это — потенциальные ключи.

Говорят, что множество атрибутов К = {A_i, A_j, ..., A_k} отношения r является потенциальным ключом r тогда и только тогда, когда удовлетворяются два независимых от времени условия:

- уникальность: в произвольный заданный момент времени никакие два различных кортежа r не имеют одного и того же значения для A_i, A_j, ..., $A_к$;
- минимальность: ни один из атрибутов A_i, A_j, ... $A_к$ не может быть исключен из К без нарушения уникальности.

Отношение может иметь несколько потенциальных ключей. Ключ, содержащий два и более атрибута, называется составным ключом. Каждое отношение обладает хотя бы одним возможным ключом, поскольку в отношении не может быть одинаковых кортежей, а это значит, что, по меньшей мере, комбинация всех его атрибутов удовлетворяет условию уникальности. Потенциальные ключи, позволяя гарантированно выделить точно один кортеж, обеспечивают основной механизм адресации на уровне кортежей реляционной модели.

Один из возможных ключей (выбранный произвольным образом) принимается за его первичный ключ. Обычно первичным ключом назначается тот возможный ключ, которым проще всего пользоваться при повседневной работе. Остальные возможные ключи, если они есть, называются альтернативными ключами. Для индикации связи между отношениями используются внешние ключи.

Внешний ключ — это набор атрибутов одного отношения, являющийся потенциальным ключом другого отношения.

Благодаря наличию связок между потенциальными и внешними ключами обеспечивается взаимосвязь кортежей определенных отношений.

Отношение, содержащее внешний ключ, называется дочерним или ссылающимся отношением. А отношение, содержащее связанный с внешним ключом потенциальный ключ, — родительским или целевым отношением.

Отношения не могут рассматриваться как статические объекты, так как они предназначены для отражения некоторой части реального мира, а эта часть реального мира может изменяться во времени. Поэтому и отношения изменяются во времени: кортежи могут добавляться, удаляться или модифицироваться. Тем не менее, предполагается, что сама схема отношения инвариантна во времени. Отношение должно восприниматься как множество возможных состояний, которые может принимать отношение.

Пример

Пусть рассматривается концептуальная модель, приведенная на рис. 5.2. Пример относится к предметной области, которую можно назвать "Преподавательская деятельность". Данная модель содержит две сущности: **ЛЕКТОР** и **ПРЕДМЕТ**, между которыми установлена связь **ЧИТАЕТ** типа "многие ко многим". Характеристики сущностей представлены изображенными на рисунке атрибутами. Связь **ЧИТАЕТ** не имеет собственных атрибутов. Для преобразования концептуальной модели в реляционную модель разработан ряд технологий, знакомство с которыми состоится несколько позже.

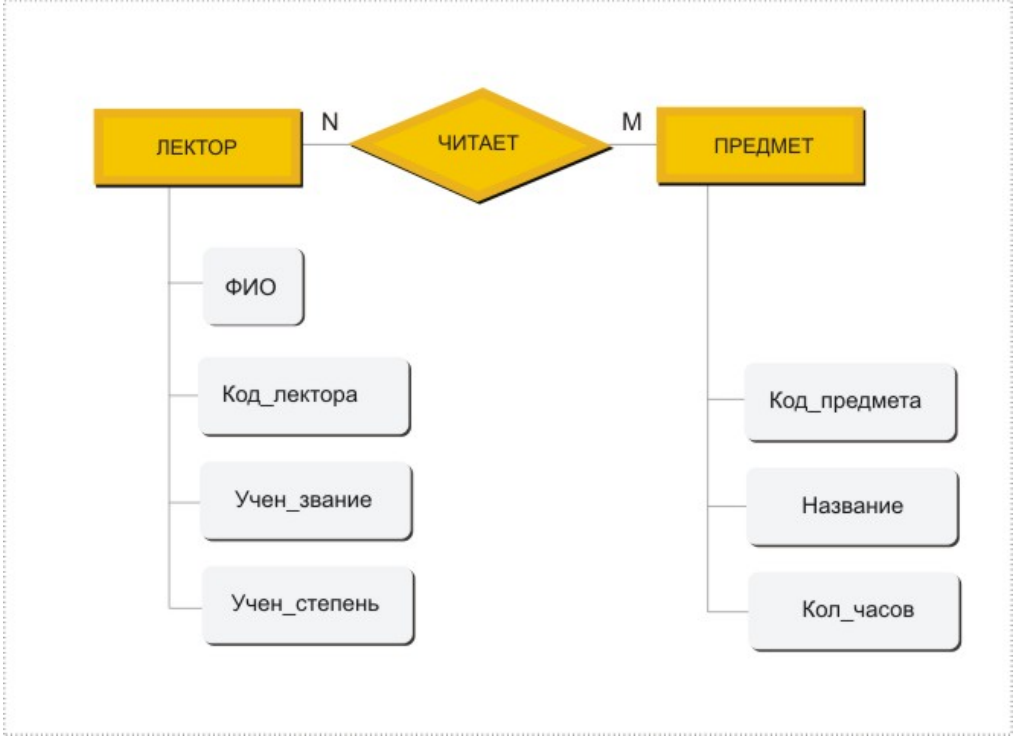


Рис. 5.2. Концептуальная модель для примера

В данный момент, не вдаваясь в подробности причин принятого решения, просто приведем реляционную схему, соответствующую указанной концептуальной модели. Она включает в себя три отношения: **ЛЕКТОР, ПРЕДМЕТ, ЧИТАЕТ**. Схемы отношений и связи между ними изображены на рис. 5.3.

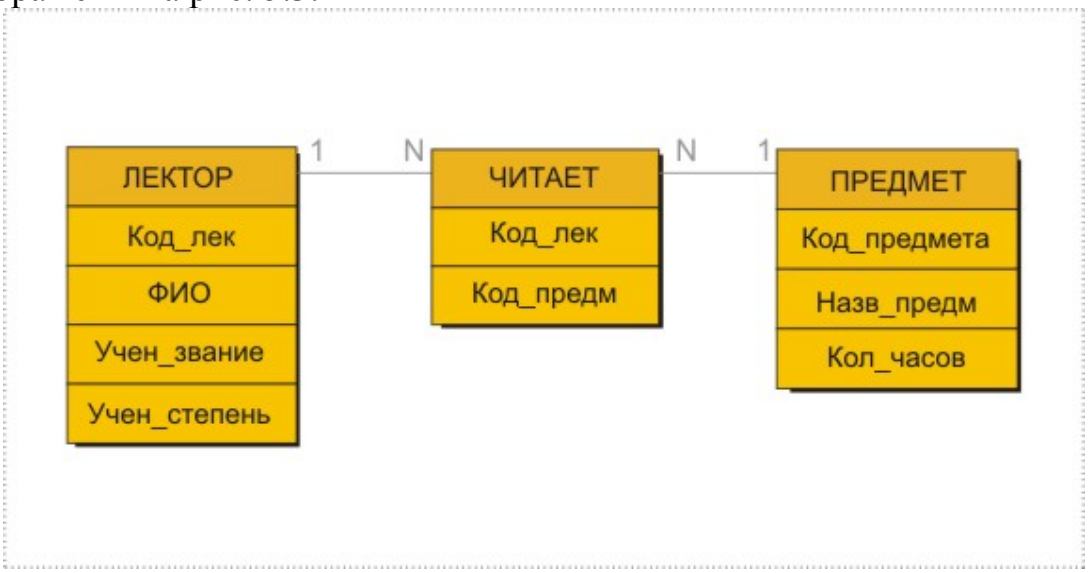
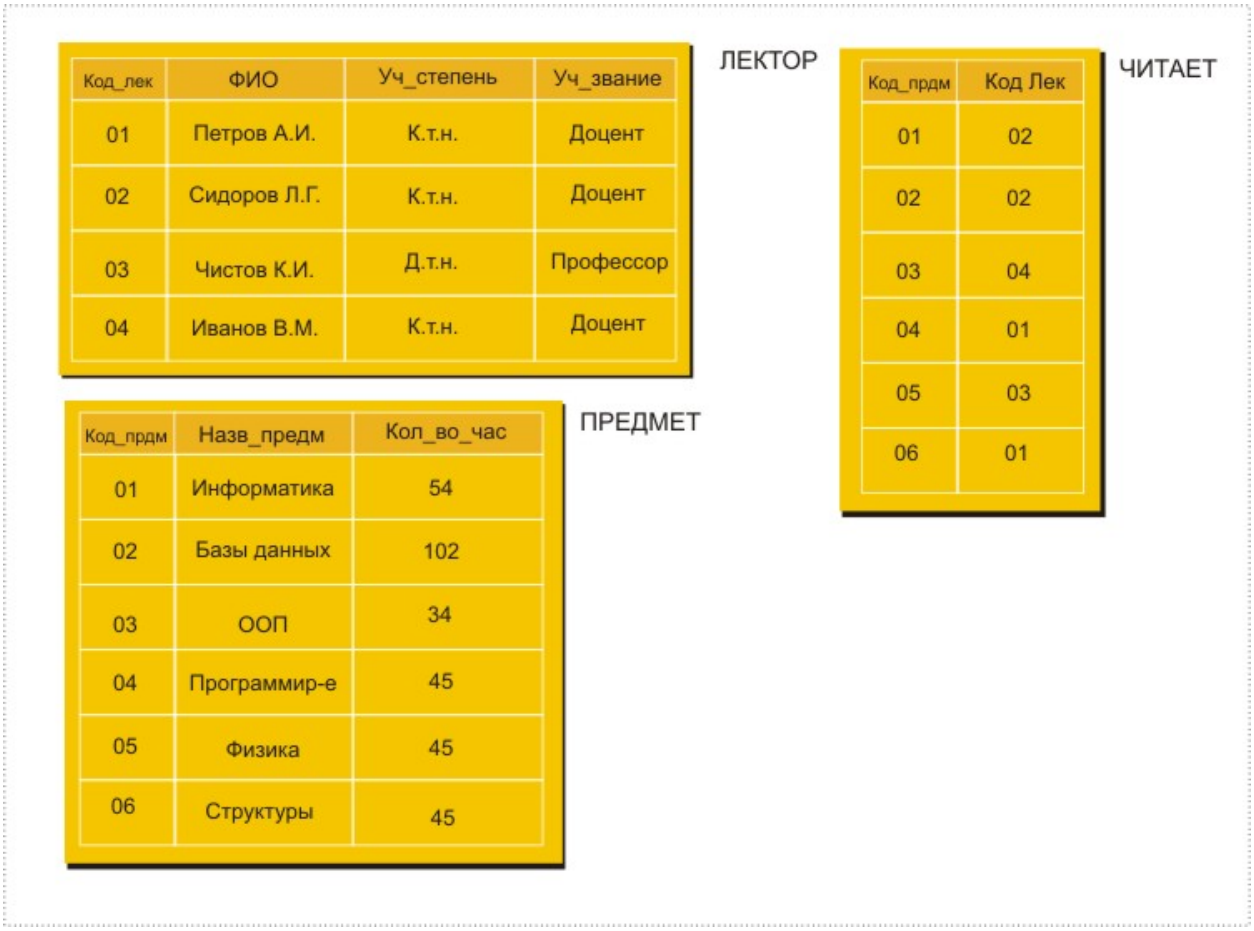


Рис. 5.3. Реляционная схема базы данных для примера

На рис. 5.4 даны соответствующие отношения, заполненные кортежами. Эти отношения имеют следующие характеристики:

- **ЛЕКТОР** — 4-арное отношение с первичным ключом **Код_лек** с кардинальным числом, равным четырем; атрибуты определены на следующих доменах: **Код_лек** — {целые: 1...4}, **ФИО**— {возможные фамилии и инициалы}, **Уч_степень** — {к.т.н., д.т.н., нет степени}, **Уч_звание** — {Доцент, Профессор, Нет_звания};
- **ПРЕДМЕТ** — тернарное отношение с первичным ключом **Код_предм**. с кардинальным числом, равным шести; атрибуты определены на следующих доменах: **Код_предм** — {символьный}. **Назв_предм** — {Информатика, Программирование, Физика, ООП, Базы данных, Базы данных}, **Кол_во_час** — {целые: 54, 102, 36};
- **ЧИТАЕТ** — бинарное отношение с составным первичным ключом **Код_лек, Код_предм**, с кардинальным числом, равным шести, в котором присутствуют первичные ключи только читающих лекторов и первичные ключи только читаемых предметов.



ЛЕКТОР

Код_лек	ФИО	Уч_степень	Уч_звание
01	Петров А.И.	К.т.н.	Доцент
02	Сидоров Л.Г.	К.т.н.	Доцент
03	Чистов К.И.	Д.т.н.	Профессор
04	Иванов В.М.	К.т.н.	Доцент

ЧИТАЕТ

Код_прдм	Код Лек
01	02
02	02
03	04
04	01
05	03
06	01

ПРЕДМЕТ

Код_прдм	Назв_предм	Кол_во_час
01	Информатика	54
02	Базы данных	102
03	ООП	34
04	Программир-е	45
05	Физика	45
06	Структуры	45

Рис. 5.4. Реляционные отношения модели для примера

Все приведенные отношения являются нормализованными, поскольку атрибуты всех отношений имеют атомарные значения. Во всех трех отношениях отсутствуют дублирующие кортежи.

5.3. Обновление отношений

Для обновления отношений необходимо иметь возможность выполнять следующие операции;

- добавление кортежа;
- удаление кортежа;
- изменение кортежа.

Рассмотрим их по порядку.

Операция добавления для отношения r со схемой (A_1, A_2, ..., A_n) имеет вид:

ADD (r: A_l=d_l, A_2=d_2, ..., A_n=d_n).

Если порядок атрибутов фиксирован, возможна более короткая запись:

ADD (r: d_l, d_2,…, d_n).

Выполнение этой операции может стать невозможным в ряде случаев:

- добавляемый кортеж не соответствует схеме определенного отношения;
- некоторые значения кортежей не принадлежат соответствующим доменам;
- описанный кортеж совпадает по ключу с кортежем, уже находящимся в отношении.

Операция удаления предназначена для удаления кортежей. Она может быть записана следующим образом:

DEL (r; $A_1=d_1$, $A_2=d_2$,…, $A_n=d_n$),

или для упорядоченных атрибутов:

DEL (r; d_1, d_2,…, d_n) .

Для удаления некоторого кортежа часто достаточно указать значение некоторого ключа:

DEL (r; ключ) .

Если указанный кортеж в отношении отсутствует, то отношение остается неизменным.

Операция изменения предназначена для модификации части кортежа. Для отношения r ее можно при $\{C_1, C_2, ... ,C_p\} \in \{A_1, A_2, ..., A_n\}$ определить так:

CH (r; $A_1=d_1$, $A_2=d_2$, ..., $A_n=d_n$; $C_1=e_1$, $C_2=e_2$, ..., $C_p=e_p$).

Если $K = \{B_1, B_2, ..., B_k\}$ является ключом, то запись данной операции может быть сокращена:

CH (r; $B_1=d_1$, $B_2=d_2$, ..., $B_k=d_k$; $C_1=e_1$, $C_2=e_2$, ..., $C_p=e_p$).

Возможные ошибки в данном случае те же, что и у предыдущих операций:

- указанный кортеж не существует;
- изменения имеют неправильный формат;
- используемые значения не принадлежат соответствующим доменам.

5.4. Целостность базы данных

Поддержка целостности базы данных реализуется посредством ряда ограничений, накладываемых на данные.

Первый тип ограничений проистекает из того факта, что каждый атрибут определяется на своем домене, или наоборот: домен атрибута задает множество значений, которые может принимать атрибут. Указанное ограничение называется ограничением атрибута.

Важными понятиями в теории реляционных баз данных являются категорная целостность и целостность на уровне ссылок.

Категорная целостность ограничивает набор значений первичных ключей базовых отношений. Такого рода целостность заключается в следующем: кортеж не может записываться в БД до тех пор, пока значения его ключевых атрибутов не будут полностью определены. Иными словами: никакой ключевой атрибут любого кортежа отношения не может

содержать отсутствующего значения, обозначаемого определителем NULL.

Целостность на уровне ссылок. При построении отношений для связывания строк одной таблицы со строками другой таблицы используются внешние ключи. База данных, в которой все непустые внешние ключи ссылаются на текущие значения ключей другого отношения, обладает целостностью на уровне ссылок.

Как правило, определение условия целостности данных осуществляется при установлении взаимосвязи между родительским и дочерним отношениями; при этом пользователю часто предоставляется возможность самому решить, каким путем сохранять целостность базы данных и насколько жестко должно соблюдаться условие целостности при различных операциях изменения данных.

Так, например, при изменении значения первичного ключа в родительском отношении часто разрешены следующие варианты действий, из которых два первых — обеспечивают нахождение базы данных в целостном непротиворечивом состоянии.

- При изменении значений полей первичного ключа в родительском отношении автоматически осуществляется каскадное изменение всех соответствующих значений в дочернем отношении.
- Не позволяется изменять значения полей первичного ключа в родительском отношении, если в дочернем отношении имеется хотя бы одна запись, содержащая ссылку на изменяемую запись.
- Позволяется изменять значения полей первичного ключа в родительском отношении, независимо от существования связанных записей в дочернем отношении. Целостность данных при этом не поддерживается.

При удалении записи в родительском отношении также возможны несколько вариантов действий.

- При удалении записи в родительском отношении автоматически осуществляется каскадное удаление всех записей из дочернего отношения, связанных с удаляемой записью.
- Не позволяется удалять записи в родительском отношении, если в дочернем отношении имеется хотя бы одна запись, содержащая ссылку на удаляемую запись. При попытке удаления записи возникает ошибка, которую можно обработать программно.
- Позволяется удалять записи в родительском отношении независимо от существования связанных записей в дочернем отношении. Очевидно, что целостность данных при этом не поддерживается.

При добавлении новой записи в дочернее отношение или редактировании в нем существующей записи возможно выбрать один из вариантов:

- не позволяется вводить запись, если значение индексного выражения дочернего отношения не соответствует одной из записей в родительском отношении;
- при вводе данных в дочернее отношение не анализируется значение индексного выражения. Целостность данных при этом не поддерживается.

Еще один вид ограничений, накладываемый на информацию, содержащуюся в отношениях, связан с теми бизнес-правилами, которые существуют в данной организации или в моделируемой предметной области. Это так называемые дополнительные правила поддержки целостности данных, определяемые пользователем или администратором данных, которые называются корпоративными ограничениями целостности. Приведем примеры таких правил:

- студентам нельзя планировать занятия продолжительностью более восьми часов в день;
- студенты весь день должны заниматься в одном корпусе вуза, чтобы не требовалось дополнительное время на переходы из одного здания в другое.

6. Проектирование базы данных

6.1. Избыточность данных и аномалии обновления в БД

Избыточность данных в БД относится к нежелательным явлениям, поскольку ведет к увеличению объема памяти, необходимого для физического хранения отношений. Избыточность вызывается, прежде всего, дублированием данных.

Вот характерный пример отношения (табл. 6.1), содержащего нежелательную избыточность:

Таблица 6.1. Отношение **СТУДЕНТ**

Ном_зач_кн	**ФИО_студента**	**Код_группы**	**ФИО_старосты**	**Куратор**
20-Т-201	Иванов С.И.	20-Т-11	Рябов В.С.	Доц. Фок И.И.
20-Т-215	Петров Я.Р.	20-Т-12	Сизов М.М.	Доц. Докин С.С.
20-Т-217	Рябов В.С	20-Т-11	Рябов В.С.	Доц. Фок И.И.
20-Т-211	Сенова А.Л.	20-Т-11	Рябов В.С.	Доц. Фок И.И.

В данном отношении с первичным ключом **Ном_зач_кн** в каждом кортеже о каждом студенте из одной и той же группы повторяются сведения о коде группы, старосте и кураторе.

При работе с отношениями, содержащими избыточные данные, могут возникнуть проблемы, которые называются аномалиями обновления.

Различают три вида аномалий в базе данных:

- аномалии включения;
- аномалии удаления;
- аномалии модификации.

Аномалии включения

В приведенном выше отношении аномалии включения возникают при попытке создать новую группу и ввести ее в отношение при том условии, что в нее еще не зачислен ни один студент. Ввод такой информации в подобной ситуации требует присвоения значения NULL всем атрибутам описания студента, в том числе и атрибуту **Ном_зач_кн**, который является первичным ключом данного отношения. Но реализация такой попытки приведет к нарушению категорней целостности, а значит, система ее обязана отклонить.

Результатом анализа является вывод о том, что в отношении табл. 6.1 присутствуют аномалии включения, а, следовательно, это отношение должно быть преобразовано таким образом, чтобы от них избавиться.

Структура отношений, содержащая ту же информацию, что и отношение **СТУДЕНТ**, но лишенная аномалий включения, представлена в табл. 6.2 и 6.3.

Таблица 6.2 Отношение **СТУДЕНТ**

Ном_зач_кн	ФИО_студента	Код_группы
20-Т-201	Иванов С.И.	20-Т-11
20-Т-215	Петров Я.Р.	20-Т-12
20-Т-217	Рябов В.С.	20-Т-11
20-Т-211	Сенова А.Л.	20-Т-11

Таблица 6.3 Отношение **ГРУППА**

Код_группы	ФИО_старосты	Куратор
20-Т-11	Рябов В.С.	Доц. Фок И.И.
20-Т-12	Сизов М.М.	Доц. Докин С.С.

Аномалии удаления

Вернемся к анализу отношения, представленного в табл. 6.1. При удалении из этого отношения кортежа:

20-Т-215 Петров Я.Р. 20-Т-12 Сизов М.М. Доц. Докин С.С.

из базы данных будут удалены все сведения о группе 20-Т-12. Такая ситуация представляет собой аномалию удаления.

Для исключения из базы данных аномалии удаления это отношение должно быть преобразовано. Причем преобразования должны быть проведены точно такие же, какие были проведены для исключения аномалии включения.

Аномалии модификации

Такая аномалия возникает при попытке изменить что-либо касающееся сведений о группе обучения студента. Допустим, что в группе 20-Т-11 решили назначить нового старосту, например, Сенову А.Л.

В такой ситуации необходимо просмотреть все кортежи отношения и в каждом кортеже значение атрибута **ФИО_старосты** заменить Рябов В.С. на Сенова А.Л.

Появление аномалии модификации можно заблокировать, если опять же прибегнуть к преобразованию отношения из табл. 6.1. Эти преобразования точно такие же, которые были использованы для исключения аномалий включения и удаления. Действительно, смена старосты группы требует изменения значения атрибута **ФИО_старосты** только в одном кортеже отношения табл. 6.3.

Проблема обратимости

Чтобы исключить различного рода аномалии из отношения, его подвергают процессу декомпозиции. При решении задачи декомпозиции возникают две проблемы.

Первая проблема — это проблема обратимости, заключающаяся в возможности восстановления исходной схемы, а именно — восстановления любого кортежа исходного отношения, используя кортежи полученных в результате декомпозиции отношений. Декомпозиция, удовлетворяющая этому требованию, называется декомпозицией, гарантирующей отсутствие потерь.

Вторая проблема связана с сохранением зависимостей. Следует напомнить, что проектирование базы данных включает в себя и определение ограничений, накладываемых на ее отношения. В процессе декомпозиции получаются новые отношения и ограничения, которые приписываются им, должны быть такими, чтобы были сохранены исходные ограничения.

Все сказанное означает, что проводимая декомпозиция должна сохранять эквивалентность схем при замене одной схемы на другую, т. е. нужна декомпозиция, гарантирующая отсутствие потерь и сохранение зависимостей.

6.2. Нормализация отношений

Для устранения рассмотренных выше недостатков и применяется процесс нормализация отношений. Данный процесс — это формальный метод анализа отношений на основе их первичных или потенциальных ключей и существующих функциональных зависимостей. Он включает ряд формальных правил, используемых для проверки всех отношений базы данных. Различают:

- 1НФ — первую нормальную форму;

- 2НФ — вторую нормальную форму;
- 3НФ — третью нормальную форму;
- НФБК — нормальную форму Бойса — Кодда;
- 4НФ — четвертую нормальную форму;
- 5НФ — пятую нормальную форму.

Каждая нормальная форма налагает определенные ограничения на данные. Эти ограничения вводятся в каждом конкретном отношении, и соблюдение этих ограничений в отношении связано уже с наличием нормальной формы.

- 1НФ, 2НФ, 3НФ— ограничивают зависимость непервичных атрибутов от ключей.
- НФБК — ограничивает зависимость первичных атрибутов.
- 4НФ — формулирует ограничения на виды многозначных зависимостей.
- 5НФ — вводит другие типы зависимостей: зависимости соединений.

Процесс перехода от нормальной формы более низкого уровня к нормальной форме более высокого уровня и называется нормализацией отношений (НО).

Для реляционных баз данных необходимо, чтобы все отношения базы данных обязательно находились в 1НФ. Нормальные формы более высокого порядка могут использоваться разработчиками по своему усмотрению. Однако следует стремиться к тому, чтобы довести уровень нормализации базы данных хотя бы до 3НФ, тем самым, исключив из базы данных избыточность данных и аномалии обновления.

6.2.1. Функциональные зависимости

Функциональная зависимость определяется для атрибутов, находящихся в одном и том же отношении в 1НФ. Функциональная зависимость является семантическим свойством атрибутов отношения.

Пусть дана схема отношения $R(A_1, A_2, ..., A_n)$.

Атрибут A_2 функционально зависит от атрибута A_1 если каждому значению A_1 соответствует единственное значение A_2 (т. е. каждый кортеж, имеющий одно и то же значение A_1, должен иметь одно и то же значение A_2)

Обозначается подобная ситуация так: $A_1 \rightarrow A_2$. Левая часть функциональной зависимости называется детерминантом, а правая часть — зависимой частью. Отсутствие функциональной зависимости обозначается $A \not\rightarrow B$.

Рассмотренная выше функциональная зависимость — это F-зависимость.

Дадим более общее формальное определение функциональных зависимостей.

Пусть r-отношение со схемой R, а X, Y— подмножества R. Отношение r удовлетворяет функциональной зависимости $X \to Y$, если для любых двух кортежей t_1 и t_2 в r $t_1(X) = t_2(X)$, то $t_1(Y) = t_2(Y)$.

6.2.2. Аксиомы вывода

Для отношения r(R) в любой момент существует некоторое семейство F-зависимостей, которым это отношение удовлетворяет. Причем одно состояние отношения может удовлетворять F-зависимости, а другое — нет. Требуется выявить семейство F-зависимостей F, которому удовлетворяют все допустимые состояния r.

Множество функциональных зависимостей, применимых к отношению r(R), конечно, так как существует только конечное число подмножеств множества R. Таким образом, можно найти все F-зависимости, которым удовлетворяет r, перебрав все возможные. Время поиска можно сократить в том случае, если известны некоторые F-зависимости из F. В подобной ситуации сокращение времени поиска происходит часто из-за того, что остальные F-зависимости можно получить, использую аксиомы вывода.

Аксиома вывода — это правило, устанавливающее, что если отношение удовлетворяет определенным F-зависимостям, то оно должно удовлетворять к некоторым другим F-зависимостям.

Ниже приведены шесть аксиом вывода для F-зависимостей. В их формулировках используется обозначение r для отношения на R, и W, X, У, Z для подмножеств R.

Запишем кратко все аксиомы.

- F1. Рефлексивность: $X \to X$.
- F2. Пополнение: $X \to Y$ влечет за собой $XZ \to Y$.
- F3. Аддитивность: $X \to Y$ и $X \to Z$ влечет за собой $X \to YZ$.
- F4. Проективность: $X \to YZ$ влечет за собой $X \to Y$.
- F5-Транзитивность: $X \to Y$ и $Y \to Z$ влечет за собой $X \to Z$.
- F6. Псевдотранзитивность: $X \to Y$ и $YZ \to W$ влечет за собой $XZ \to W$.

6.2.3. Первая нормальная форма

Отношение находится в первой нормальной форме, если все его атрибуты имеют простые (атомарные) значения. Другими словами, значения в домене каждого атрибута отношения не являются ни списками, ни множествами простых или сложных значений.

Определить понятия атомарности трудно. Значение, атомарное в одном приложении, может быть неатомарным в другом. Можно руководствоваться общим принципом, что значение не атомарно, если в приложении оно используется по частям. Рассмотрим пример отношения, представленного в табл. 6.4.

Таблица 6.4. Отношение **РОЖДЕНИЕ**

Имя	**Дата рождения**
Анна	5 марта 1986
Александр	25 января 1987
Ольга	1 ноября 1987
Федор	14 сентября 1986

Если значение атрибута **Дата рождения** предполагается использовать целиком, то в этом случае данное отношение находится в 1НФ. Если бы потребовалось выделить и отдельно использовать, скажем, год, число, месяц, то это отношение не находилось бы в 1НФ, так как требуемые данные являются только частями значения атрибута **Дата рождения**. Чтобы перевести такое отношение в 1НФ, атрибут **Дата рождения** должен быть разбит на части так, как показано в табл. 6.5.

Таблица 6.5. Отношение **РОЖДЕНИЕ**

Имя	**День рождения**	**Месяц рождения**	**Год рождения**
Анна	5	Март	1986
Александр	25	Январь	1987
Ольга	1	Ноябрь	1987
Федор	14	Сентябрь	1986

Или, например, табл. 6.6 является ненормализованной, и она не находится в 1НФ потому, что включает величины, являющиеся совокупностью атомарных значений. Чтобы получить отношение РОД, находящееся в 1НФ, необходимо его представить так, как это сделано в табл. 6.7.

Таблица 6.6. Ненормализованная таблица **РОД**

Имя	**Пол**
{Александр, Федор}	Мужской
Ольга	Женский

Таблица 6.7. Отношение **РОД**

Имя	**Пол**
Александр	Мужской
Федор	Мужской
Ольга	Женский

6.2.4. Вторая нормальная форма

Вторая и третья нормальные формы возникли в результате стремления избежать аномалий обновления данных при работе с БД и избавиться от информационной избыточности в отношениях.

Вторая нормальная форма применяется к отношениям с составными ключами, т. е. к таким отношениям, первичный ключ которых состоит из двух или более атрибутов. Отношение, у которого первичный ключ включает только один атрибут, всегда находится во 2НФ.

Дадим определения новых понятий.

Определение 1. Атрибут называется посторонним для функциональной зависимости $X \rightarrow Y$, если он может быть удален из правой или левой части функциональной зависимости без изменений транзитивного замыкания F+ множества F.

Определение 2. Функциональная зависимость $X \rightarrow Y$ называется редуцированной слева, если X не содержит атрибута Z, постороннего для функциональной зависимости $X \rightarrow Y$. Если функциональная зависимость $X \rightarrow Y$ редуцирована слева, то Y является полностью зависящим от X. В противном случае Y частично зависит от X.

Определение 3. Для данной схемы отношения R атрибут A в R и множество функциональных зависимостей F на R атрибут A называется первичным в R относительно F, если A содержится в каком-нибудь ключе схемы R. В противном случае A называется непервичным в R.

Итак, схема отношения R находится во 2НФ относительно F, если она находится в 1НФ, и каждый непервичный атрибут функционально полно зависит от первичного ключа.

Схема всей БД имеет 2НФ относительно F, если каждая ее схема отношения находится относительно F во 2НФ.

Рассмотрим отношение **КОНСУЛЬТАЦИИ_ДИПЛОМНИКОВ** со схемой:

(Таб_Ном_преп, Ном_зач_кн, Дата, ФИО_преп, Должность, ФИО_студ, Тема_диплома, Время, Аудитория, Вместимость).

Это отношение содержит информацию о том, какой преподаватель консультирует определенного дипломника, а также дату, время консультации и аудиторию с ее вместимостью, где она должна проводиться. Обозначим основные зависимости этого отношения.

(Таб_Ном_преп, Ном_зач_кн, Дата) → (ФИО_преп, Должность, ФИО_студ, Тема_липлома, Время, Аудитория, Вместимость).

Описательные атрибуты преподавателя **ФИО_преп**, **Должность** зависят только от части первичного ключа. Данную ситуацию определяет зависимость:

Таб-Ном_преп → (ФИО_преп, Должность).

Описательные атрибуты студента **ФИО_студ, Тема_диплома** также зависят только от части первичного ключа и не зависят от остальных атрибутов ключа, то есть имеется зависимость вида:

Ном_зач_кн → (ФИО_студ. Тема_диплома).

Отсутствие полной функциональной зависимости каждого непервичного атрибута отношения от первичного ключа, как и в других рассмотренных здесь примерах, является источником аномалий обновления и вносит свою долю избыточности в базу данных. Устранение данных отрицательных явлений осуществляется путем декомпозиции исходного отношения на три со следующими схемами:

ПРЕПОДАВАТЕЛЬ (Таб_Ном_преп, ФИО_преп, Должность);
СТУДЕНТ (Ном_зач_кн, ФИО_студ, Тема_диплома);
КОНСУЛЬТАЦИИ (Таб_Ном_преп, Ном_зач_кн, Дата, Время, Аудитория, Вместимость).

6.2.5. Третья нормальная форма

Рассмотрим транзитивную зависимость следующего типа: если A → B, B -/→ A (B не является ключом), B → C, то A → C.

В этом случае считается, что C транзитивно зависит от A.

Транзитивная зависимость вызвана наличием в отношении двух семантических зависимостей различных типов.

Схема отношения находится в третьей нормальной форме относительно множества функциональных зависимостей F, если она находится в 1НФ и ни один из непервичных атрибутов в R не является транзитивно зависимым от ключа для R.

Схема всей БД находится в ЗНФ относительно F, если каждая схема отношения находится в ЗНФ относительно F.

Рассмотрим схему базы данных примера предыдущего раздела.

ПРЕПОДАВАТЕЛЬ (Таб_Ном_преп, ФИО_преп, Должность);
СТУДЕНТ (Ном_зач_кн, ФИО_студ, Тема_диплома);
КОНСУЛЬТАЦИИ (Таб_Ном_преп, Ном_зач_кн, Дата, Время, Аудитория, Вместимость).

Последнее отношение содержит транзитивную зависимость:

(Таб_Ном_преп, Ном_зач_кн, Дата) → Аудитория → Вместимость.

Следовательно, это отношение не находится в ЗНФ со всеми вытекающими из этого последствиями, и прежде всего следующими:

- если аудитория исключается из расписания консультаций, то о ней вообще теряются сведения;
- если аудитория перестроена и в результате изменилась ее вместимость, то придется просмотреть все кортежи и провести модификацию значений атрибута.

Для устранения транзитивной зависимости необходимо провести декомпозицию последнего отношения, удалив из него транзитивно-зависимый атрибут и поместив его в новое отношение вместе с копией того атрибута, от которого он зависит.

Таким образом, база данных этого примера, лишенная транзитивных зависимостей, в ЗНФ будет выглядеть так:

ПРЕПОДАВАТЕЛЬ (Таб_Ном_преп, ФИО_преп, Должность);

СТУДЕНТ Ном_зач_кн, ФИО_студ, Тема_диплома);

КОНСУЛЬТАЦИИ (Таб_Ном_преп, Ном_зач_кн, Дата, Время, Аудитория);

АУДИТОРИЯ (Аудитория, Вместимость).

При проектировании структуры реляционной базы данных считается корректной установка, что любая БД должна находиться как минимум в ЗНФ. На практике третья нормальная форма схем отношений достаточна в большинстве случаев, и приведением к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается.

6.2.6. Нормальная форма Бойса — Кодда

Следует отметить, что определение для ЗНФ было дано Коддом для ситуаций с упрощающим картину допущением того, что отношение имеет только один потенциальный ключ, который, естественно, и является первичным ключом. Естественно, что не все отношения могут быть уложены в данные довольно жесткие рамки. Более обобщающими являются случаи, когда в наличии имеются следующие условия:

- отношение имеет два (или более) потенциальных ключа;
- два потенциальных ключа являются составными;
- два потенциальных ключа перекрываются, т. е. имеют, по крайней мере, один общий атрибут.

Схема отношения R находится в НФБК относительно множества F-зависимостей тогда и только тогда, когда детерминанты являются потенциальными ключами.

Допустим, что при проектировании базы данных **ПОСТАВКИ_ТОВАРОВ** рассматривается отношение:

ПОСТАВКА (Индекс_поставщ, Имя_поставщ, Индекс_товара, Колич_товара).

Допустим также, что значения атрибута **Имя_поставщ** уникальны и могут быть использованы наряду с атрибутом **Индекс_поставщ** для идентификации поставщика. В такой ситуации можно выделить два составных потенциальных ключа:

(Индекс поставщ, Индекс_товара);

(Имя_поставщ, Иидекс_товара).

В рассматриваемом отношении есть два атрибута **Индекс_поставщ** и **Имя_поставщ**, которые идентифицируют один и тот же экземпляр

сущности, а, значит, они определяют друг друга. В таком случае отношение содержит два детерминанта, но эти детерминанты не являются потенциальными ключами отношения. Сложившаяся картина противоречит данному выше определению НФБК, и следовательно, данное отношение не находится в НФБК.

Для схемы отношения, не находящейся в НФБК, можно провести декомпозицию в схему БД в НФБК. Из исходного отношения убирается и переносится в новое отношение зависимая часть вместе с копией детерминанта.

Для рассматриваемого примера решение проблемы можно осуществить, разбив исходное отношение на два. Причем поскольку два детерминанта **Индекс_поставщ** и **Имя_поставщ** определяют друг друга, то возможны два равносильных варианта декомпозиции, приводящей к НФБК.

Первый вариант получается, если учитывается зависимость

Индекс_поставщ → Имя_поставщ,

в результате чего имеем следующих два отношения:

ПОСТАВКА (Иидекс_поставщ, Индекс_товара, Колич_товара);
ПОСТАВЩИК (Индекс_поставщ, Имя_поставщ),

Второй вариант исходит из зависимости

Имя_поставщ → Индекс_поставщ,

в результате чего получаем альтернативную группу отношений;

ПОСТАВКА (Имя_поставщ, Индекс_товара, Колич_товара);
ПОСТАВЩИК (Индекс_поставщ, Имя_поставщ).

6.2.7. Четвертая нормальная форма

Итак, НФБК позволяет устранить любые аномалии обновления, вызванные функциональными зависимостями.

Рассмотрим следующую схему отношения: **НИР (Номер_НИР, Сотр, Задан_НИР).**

Отношение НИР содержит номера тем научно-исследовательских работ, для каждой темы — список сотрудников, которые могут выполнять работы по теме, и список заданий темы. Сотрудники могут участвовать в нескольких темах, и разные темы могут включать одинаковые задания. В такой ситуации единственно возможным ключом отношения является составной атрибут:-

(Номер_НИР, Сотр, Задан_НИР)

Отношение характеризуется значительной избыточностью и приводит к возникновению аномалий обновления. Все рассмотренные до сих пор приемы нормализации, опирающиеся на функциональные зависимости, оказываются неприменимыми, поскольку этих зависимостей в отношении вовсе нет.

В 1971 году Фейгин предложил строго теоретически обоснованный выход из этой ситуации с помощью понятия многозначной зависимости (МЗ).

Определим формально условие существования многозначной зависимости: многозначная зависимость имеет место в том отношении, в котором содержится две независимые связи типа 1 : М. И все проблемы данной ситуации вызваны именно этой независимостью связей.

В отношении R(A, B, C) существует многозначная зависимость A $\rightarrow$ B в том и только в том случае, если множество значений B, соответствующее паре значений A и C, зависит только от A и не зависит от C.

В отношении НИР существуют следующие две многозначные зависимости:

Номер_НИР ->> Сотр;

Номер_НИР ->> Задан_НИР.

Многозначные зависимости всегда образуют связанные пары и поэтому их обычно представляют вместе в символьном виде так: A ->> B | C.

Дальнейшая нормализация таких отношений должна проходить по пути разделения двух независимых повторяющихся групп. Это разделение основывается на следующей теореме Фейгина.

Отношение R (A, B, C) можно спроецировать без потерь в отношения R1 (A, B) и R2 (A, C) тогда и только тогда, когда для отношения R выполняется МЗ-зависимость: A ->> B | C. Такая зависимость называется нетривиальной МЗ-зависимостью.

Отношение находится в четвертой нормальной форме (4НФ) тогда и только тогда, когда существуют такие подмножества A и B атрибутов отношения R, что выполняется нетривиальная многозначная зависимость A ->> B. Тогда все атрибуты отношения R также функционально зависят от атрибута A.

Итак, поскольку проблема многозначных зависимостей возникает в связи с многозначными атрибутами, то решить проблему можно, поместив каждый многозначный атрибут в свою собственную таблицу вместе с ключом, от которого атрибут зависит.

В рассматриваемом примере можно произвести декомпозицию отношения НИР в два отношения **НИР-СОТРУДНИКИ** и **НИР-ЗАДАНИЯ**:

НИР-СОТРУДНИКИ (Номер_НИР, Сотр);

НИР-ЗАДАНИЯ (Номер_НИР, Задан_НИР).

6.2.8. Пятая нормальная форма

Во всех рассмотренных до этого момента ситуациях нормализация отношений производилась декомпозицией одного отношения на два.

Иногда нормализовать отношение путем декомпозиции на два отношения без потерь не удается, но просматривается возможность декомпозиции исходного отношения без потерь на большее число отношений, каждое из которых обладает лучшими свойствами. Такое отношение называется термином "n-декомпозитируемое отношение" для некоторого n > 2. Это значит, что для данного от ношения возможна декомпозиция без потерь на n проекций, а на меньшее число проекций декомпозиция без потерь невозможна.

Если в процессе естественного соединения декомпозированных отношений в сравнении с первоначальным отношением генерируются ложные кортежи то такая декомпозиция характеризуется зависимостью соединения.

В отношении R (X, У, ..., Z) отсутствует зависимость соединения *(X, У, ..., Z), в том и только в том случае, когда R восстанавливается без потерь путем соединения своих проекций на X, У, ..., Z

Отношение находится в пятой нормальной форме, если оно не содержит зависимостей соединения.

6.3. Проектирование реляционной базы данных

Концептуальная модель данных состоит из ряда компонентов: сущностей, связей, атрибутов. При переходе к реляционной схеме базы данных каждый из этих компонентов должен быть проанализирован и, если это окажется необходимым, то даже и преобразован. Изменения, вносимые в процессе преобразования, должны быть такими, чтобы их результат полностью отвечал требованиям, выдвигаемым реляционной моделью данных.

Таким образом, фаза логического проектирования предполагает следующие действия:

- преобразование концептуальной модели данных в логическую модель, в результате которого будет определена схема реляционной модели данных;
- проверка модели с помощью концепций последовательной нормализации;
- проверка поддержки целостности данных.

Рассмотрим последовательно каждое действие.

6.2.1. Преобразование сущностей и атрибутов

Общий подход к преобразованию сущностей концептуальной модели ПрО в отношения реляционной базы данных состоит в следующем:

- построить набор предварительных отношений и указать первичные ключи для каждого отношения;
- подготовить список всех представляющих интерес атрибутов (тех из них, которые не были перечислены в диаграмме в качестве

первичных ключей сущностей) и назначить каждый из этих атрибутов одному из предварительных отношений с тем условием, чтобы эти отношения находились в НФБК. На этом шаге для каждого отношения должны быть определены межатрибутные функциональные зависимости, с помощью которых проверяется соответствие отношений НФБК. Если полученные отношения в итоге не находятся в НФБК или если некоторым атрибутам не находится логически обоснованных мест в предварительных отношениях, то в этих случаях диаграммы необходимо пересмотреть.

6.2.2. Преобразование бинарных связей

Каждая сущность преобразуется в определенное отношение, а значит, связь между сущностями преобразуется в связь между отношениями.

Напомним, что связи между отношениями в реляционной модели данных реализуются посредством механизма первичных и внешних ключей. Чтобы этот механизм действовал, необходимо в первую очередь определиться с тем, которое из двух отношений является родительским, а которое — дочерним. Родительским считается такое отношение, которое передает копию набора значений своего первичного ключа другому отношению, где этот набор значений будет представлять внешний ключ. Последнее отношение в этом случае будет являться дочерним отношением.

Рассмотрим бинарную связь **ЧИТАЕТ** между сущностями **ПРЕПОДАВАТЕЛЬ** и **КУРС** (рис. 6.1).

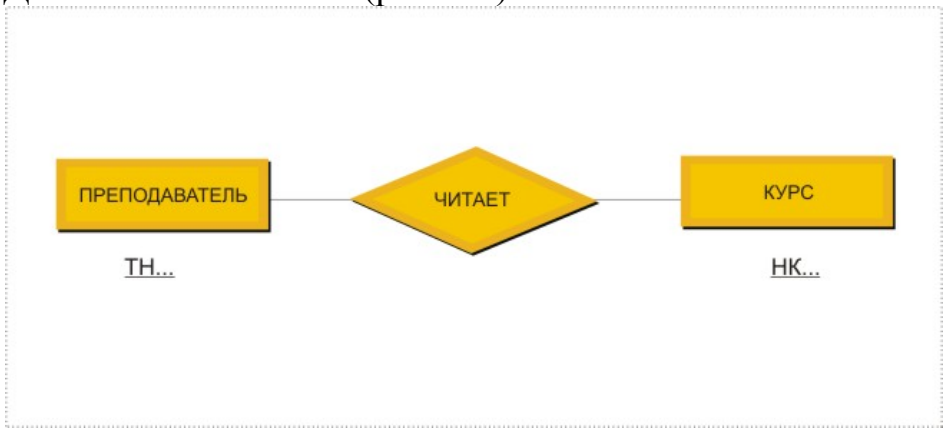


Рис. 6.1. Диаграмма бинарной связи

Как известно, эта связь может быть изображена с помощью диаграммы, которая содержит всю информацию, необходимую для генерации соответствующих отношений РБД.

Сущности **ПРЕПОДАВАТЕЛЬ** и **КУРС** однозначно идентифицируются с помощью **ТН** — табельного номера преподавателя и **НК** — номера курса.

Напомним, что, если все элементы данной сущности должны участвовать в связи, то такое участие называется обязательным. Например, если каждый преподаватель должен читать один какой-либо курс, то класс принадлежности такой сущности — обязательный. Класс принадлежности сущностей, а также мощность связи между сущностями являются факторами, определяющими структуру проектируемой БД.

6.2.3. Предварительные отношения для бинарных связей типа 1 :1

Предварительные отношения для бинарных связей с показателем кардинальности, равным 1 : 1 (рис. 6.2) могут быть получены вследствие просмотра нескольких логических альтернатив и выбора из них наиболее подходящей.

Пусть в проектируемой БД должна храниться следующая информация:

ТН — номер преподавателя;
Ф_И_О — фамилия, имя, отчество преподавателя;
Тел_П — телефон преподавателя;
НК — номер курса;
Назв_К — название курса.

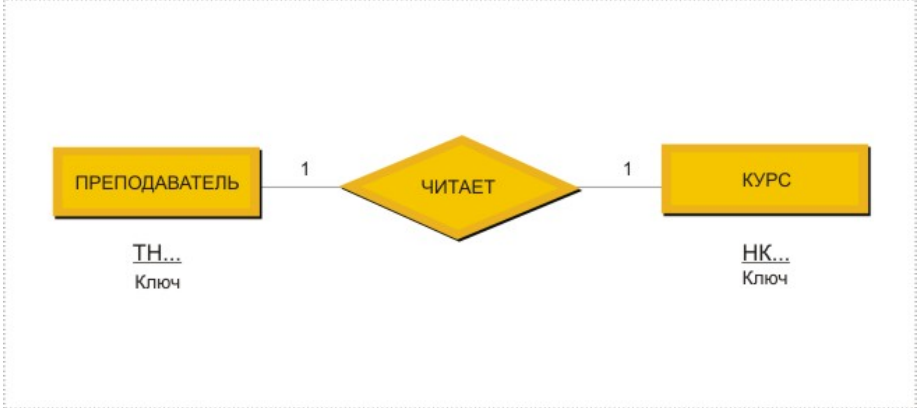


Рис. 6.2. Диаграмма бинарной связи типа 1 : 1

Считая, что класс принадлежности является обязательным для обеих сущностей, получаем отношение:

ПРЕПОДАВАТЕЛЬ (ТН, Ф_И_О, Тел_П, НК, Назв_К).

Первичным ключом этого отношения может быть ключ любой из двух сущностей. Ситуация будет другая, если класс принадлежности одной из сущностей (**ПРЕПОДАВАТЕЛЬ**) — обязательный, второго (**КУРС**) — нет.

В такой ситуации требуется построение двух отношений: по одному под каждую сущность. При этом ключ каждой сущности должен служить первичным ключом для соответствующего отношения. Сущность, для которой класс принадлежности является необязательным, преобразуется в

родительское отношение, а сущность, участвующая в связи с обязательным классом принадлежности, преобразуется в дочернее отношение.

Для связи полученных отношений ключ родительского отношения добавляется в качестве атрибута (внешнего ключа) в дочернее отношение.

В результате получаем следующую реляционную схему:

ПРЕПОДАВАТЕЛЬ (ТН, Ф_И_О, Тел_П, НК);

КУРС (НК, Назв_К).

Если класс принадлежности для обеих сущностей — необязательный, то лучшим решением является определение трех отношений — по одному для каждой сущности и одного для связи:

ПРЕПОДАВАТЕЛЬ (ТН, Ф_И_О, Тел_П);

КУРС (НК, Назв_К);

ЧИТАЕТ (ТН, НК)

Отношение **ПРЕПОДАВАТЕЛЬ** содержит сведения обо всех преподавателях, а отношение **КУРС** — обо всех курсах.

Отношение для связи должно иметь среди своих атрибутов по одному ключу от каждой сущности.

6.2.4. Предварительные отношения для бинарных связей типа 1: N

Пусть в рассмотренной выше концептуальной модели существует бинарная связь типа 1: N (рис. 6.3).

Для таких связей существует только два правила. Вариант определяется классом принадлежности N-связной сущности, класс принадлежности 1-связной сущности не влияет на конечный результат в обоих случаях.

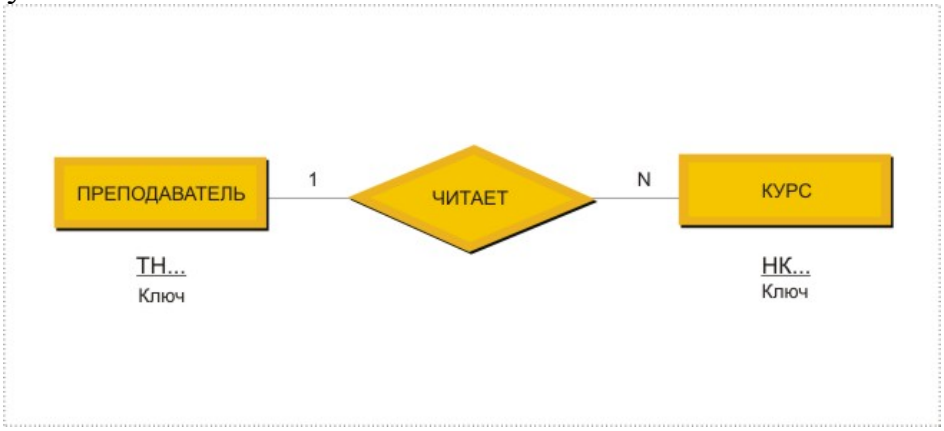


Рис. 6.3. Диаграмма бинарной связи типа 1 : N

Первый вариант. **ПРЕПОДАВАТЕЛЬ** — класс принадлежности необязательный, **КУРС** — обязательный.

Решение задачи становится возможным, если использовать два отношения, по одному на каждую сущность, при условии, что ключ

каждой сущности служит в качестве первичного ключа для соответствующего отношения. В качестве родительского назначается отношение, соответствующее "единичной" стороне связи, а в качестве дочернего назначается отношение, представляющее "множественную" сторону связи. Для представления этой связи необходимо скопировать первичный ключ родительского отношения в дочернее отношение, где данный ключ должен быть описан как внешний.

Окончательно в БД войдут два отношения:

ПРЕПОДАВАТЕЛЬ (ТН, Ф_И_О, Тел_П);
КУРС (НК, Назв_К, НП).

Второй вариант. Класс принадлежности для обеих сущностей — необязательный. Решение — в формировании трех отношений: по одному на каждую сущность (причем ключ каждой сущности служит первичным ключом соответствующего отношения) и одного отношении для связи. Отношение для связи должно иметь среди своих атрибутов ключ каждой сущности:

КУРС (НК, Назв_К);
ПРЕПОДАВАТЕЛЬ (ТН, Ф_И_О, Тел_П);
ЧИТАЕТ (ТН, НК)

6.2.5. Преобразование связи типа "суперкласс/подкласс"

Для каждой присутствующей в логической модели данных связи типа "супертип/подтип" сущность супертипа необходимо определить как родительскую, а сущность подтипа — как дочернюю. Существуют различные варианты представления подобных связей в виде одного или нескольких отношений. Выбор наиболее подходящего варианта зависит от ограничений участия и пересечения, наложенных на участников связи типа "суперкласс/подкласс".

Если суперкласс с его подклассами имеет тотальные и непересекающиеся связи, где каждый экземпляр суперкласса обязательно должен быть членом одного и только одного подкласса, то самым целесообразным решением является представление каждого из подклассов в виде отдельного отношения, содержащего копию первичного ключа суперкласса.

Рассмотрим подклассы **АССИСТЕНТ, СТАРШИЙ ПРЕПОДАВАТЕЛЬ, ДОЦЕНТ, ПРОФЕССОР**, которые являются членами суперкласса **ПРЕПОДАВАТЕЛЬ** (рис. 6.4). Это означает, что каждый экземпляр подкласса является в то же время и экземпляром суперкласса. Причем каждый преподаватель обязательно принадлежит одному и только одному подклассу.

Подобная диаграмма преобразуется в следующую реляционную схему отношений:

ПРЕПОДАВАТЕЛЬ (Табельный_номер. Ф_И_О, Адрес, Педагог_стаж).

ПРОФЕССОР (Табельный_номер, Номер_Диплома_профессора);

ДОЦЕНТ (Табельный_номер. Номер_диплома_доцента);

СТАРШИЙ_ПРЕПОДАВАТЕЛЬ (Табельный_номер):

АССИСТЕНТ (Табельный_номер).

Рис. 6.4. Диаграмма связи "суперкласс/подкласс"

6.2.6. Предварительные отношения для бинарных связей типа M: N

При такой степени связи требуется три отношения вне зависимости от класса принадлежности обеих сущностей: по одному для каждой сущности, причем ключ каждой сущности используется в качестве первичного ключа соответствующего отношения, и одного отношения для связи. Последнее должно иметь в числе своих атрибутов ключ каждой сущности. Единственно допустимый вариант в сложившейся ситуации — представить БД тремя отношениями:

КУРС (НК, Назв_К);

ПРЕПОДАВАТЕЛЬ (ТН, Ф_И_О, Тел_П);

ЧИТАЕТ (ТН, НК).

Вся информация о курсе будет содержаться в отношении **КУРС**, информация о преподавателе — в отношении **ПРЕПОДАВАТЕЛЬ**, а отношение **ЧИТАЕТ** будет содержать только экземпляры связи, имеющиеся в модели.

Использование прямого преобразования концептуальных связей в логические структуры оказывается очень полезным при моделировании составных сущностей.

6.2.6. Проверка модели с помощью концепций последовательной нормализации

Созданный на предыдущем этапе предварительный набор отношений логической модели данных должен обязательно быть

подвергнут анализу на корректность объединения атрибутов в каждом из отношений. Проверка корректности состава каждого из отношений должна проводиться посредством применения к ним процедуры последовательной нормализации. Целью применения этой процедуры является получение гарантий того, что каждое из отношений, полученных на основе концептуальной модели, находится, по крайней мере, в НФБК.

Если в процессе анализа отношений модели будут найдены отношения не отвечающие требованиям НФБК, то это будет означать, что где-то на предыдущих этапах были допущены ошибки. Возможно, ошибки появились при построении концептуальной модели, а возможно — в процессе ее преобразования в логическую модель. Для обеспечения корректности логической модели в такой ситуации придется вернуться на ранние этапы проектирования и перестроить ошибочно созданные фрагменты модели.

6.2.7. Проверка поддержки целостности данных

Следует обратить внимание на следующие вопросы:

- возможность для атрибутов иметь пустые значения;
- ограничения для доменов атрибутов;
- категорная целостность;
- ссылочная целостность;
- бизнес-правила в данной предметной области.

Необходимо выполнить работу по проверке каждой составляющей применительно к каждой сущности, к каждому атрибуту, к каждой связи логической модели предметной области. В том случае, если и данная проверка даст положительный результат, можно переходить к следующему этапу проектирования базы данных — физическому проектированию.

7. Физическая организация данных

7.1. Страничная организация данных в СУБД

Для функционирования СУБД во внешней памяти базы данных возникает необходимость хранить следующие разновидности объектов:

- строки отношений — основная часть базы данных, большей частью непосредственно видимая пользователям;
- управляющие структуры — индексы, создаваемые по инициативе пользователя (администратора) или верхнего уровня системы из соображений повышения эффективности выполнения запросов и обычно автоматически поддерживаемые нижним уровнем системы;
- журнальную информацию, поддерживаемую для удовлетворения потребности в надежном хранении данных;
- служебную информацию, поддерживаемую для удовлетворения внутренних потребностей нижнего уровня системы.

Поиск и предоставление данных пользователю осуществляются с помощью нескольких программ доступа данных и включают в себя три основных этапа (рис. 7.1).

- Определяется искомая запись, для извлечения которой запрашивается диспетчер файлов.
- Диспетчер файлов определяет страницу, на которой находится искомая запись, и для извлечения этой страницы запрашивает диспетчер дисков.
- Диспетчер дисков определяет физическое положение искомой страницы на диске и посылает соответствующий вопрос на ввод-вывод данных.

Рис. 7.1. Схема доступа к базе данных

Основными единицами осуществления операций обмена являются страницы данных, управляемые диспетчером дисков. Все данные хранятся постранично. Таким образом, с точки зрения СУБД база данных выглядит как набор записей, которые просматриваются с помощью диспетчера файлов. С точки зрения диспетчера файлов база данных выглядит как набор страниц, которые могут просматриваться с помощью диспетчера дисков.

На одной странице хранятся однородные данные, например, только данные или только индексы. Все страницы данных имеют одинаковую структуру, представленную на рис. 7.2.

Заголовок страницы содержит информацию о физическом дисковом адресе страницы, которая логически следует за данной страницей (рис. 7.3 — правый верхний угол).

Заголовки страниц и указатели на следующие страницы обрабатываются только диспетчером дисков и скрыты от диспетчера файлов. Для выполнения своих функций в распоряжении диспетчера дисков имеется каталог (страница 0), содержащий информацию обо всех имеющихся на данном диске наборах страниц вместе с указателями на первые страницы этих наборов.

Рис. 7.2. Структура страницы данных

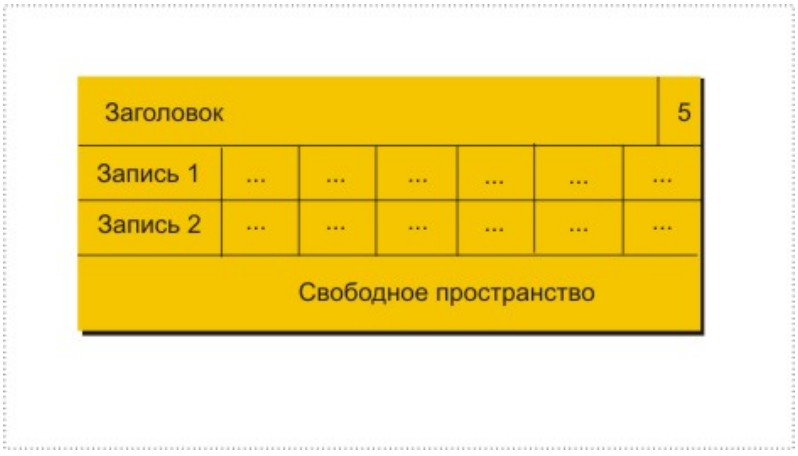


Рис. 7.3. Определение логической последовательности страниц

Слоты характеризуют размещение строк данных на странице.

Каждая запись обладает уникальным идентификатором, не изменяемым во все время ее существования, он имеет определенную длину и состоит из номера страницы, на которой данная запись находится, и

байта смещения слота от конца страницы, который в свою очередь содержит байт смещения записи от начала страницы (рис. 7.4).

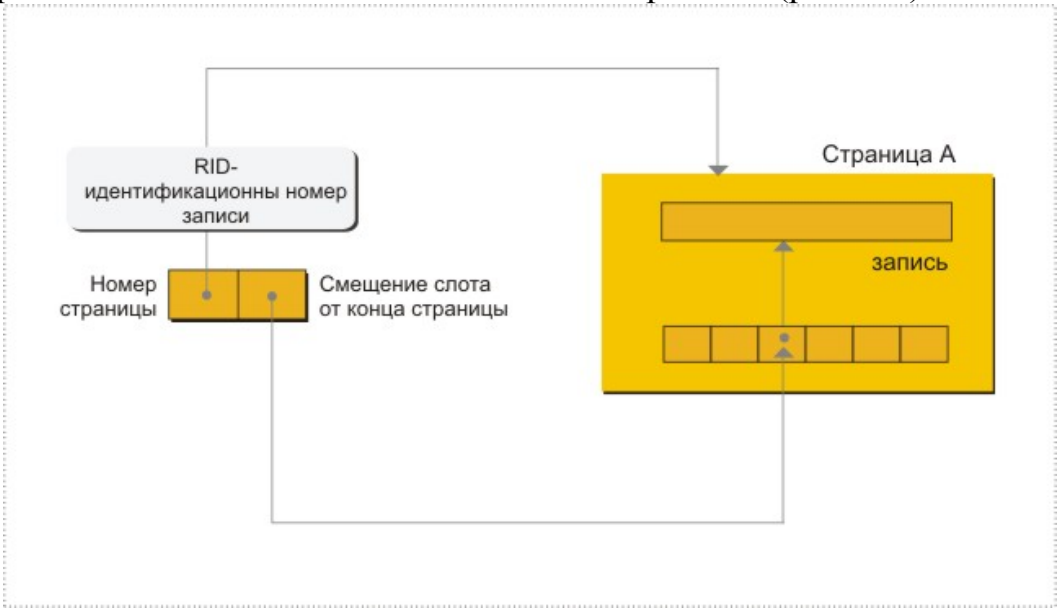


Рис. 7.4. Структура идентификационного номера записи

7.2. Индексирование

Основное назначение индексов состоит в обеспечении эффективного прямого доступа к записи таблицы по ключу. Различают индексированный файл и индексный файл (рис. 7.5). Индексированный файл — это основной файл, содержащий данные отношения, для которого создан индексный файл.

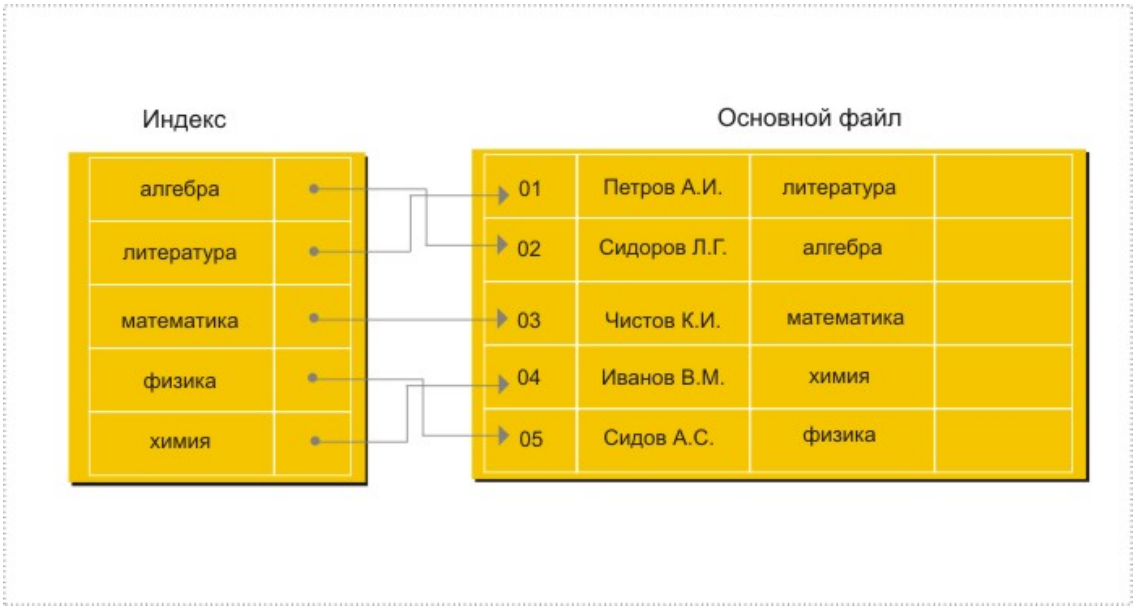


Рис. 7.5. Индексированный и индексный файлы

Индексный файл — это файл особого типа, в котором каждая запись состоит из двух значений: данных и указателя. Данные представляют поле,

по которому производится индексирование, а указатель осуществляет связывание с соответствующим кортежем индексированного файла. Если индексирование осуществляется по ключевому полю, то индекс называется первичным. Такой индекс к тому же обладает свойством уникальности, т. е. не содержит дубликатов ключа.

Основное преимущество использования индексов заключается в значительном ускорении процесса выборки данных, а основной недостаток — в замедлении процесса обновления данных. Действительно, при каждом добавлении новой записи в индексированный файл потребуется также добавить новый индекс в индексный файл.

7.2.1. Индексно-прямые файлы

В индексно-прямых файлах основная область содержит последовательность записей одинаковой длины, расположенных в произвольном порядке, а индексная запись содержит значение первичного ключа и порядковый номер записи в основной области, которая имеет данное значение первичного ключа.

Так как индексные файлы строятся для первичных ключей, однозначно определяющих запись, то в индексно-прямых файлах для каждой записи в основной области существует только одна запись из индексной области. Такой индекс называется плотным. Все записи в индексной области упорядочены по значению ключа.

Наиболее эффективным алгоритмом поиска на упорядоченном массиве является бинарный поиск. При этом все пространство поиска разбивается пополам, и так как оно строго упорядочено, то сначала определяется, не является ли срединный элемент искомым, а если нет, то дается оценка в какой половине его надо искать. Далее установленная половина также делится пополам и производятся аналогичные действия, и так до тех пор, пока не будет обнаружен искомый элемент.

Потом путем прямой адресации происходит обращение к основной области уже по конкретному номеру записи.

Операция добавления осуществляет запись в конец основной области. В индексной области при этом производится занесение информации так, чтобы не нарушать упорядоченности. Поэтому вся индексная область файла разбивается на блоки и при начальном заполнении в каждом блоке остается свободная область (процент расширения) (рис. 7.6).

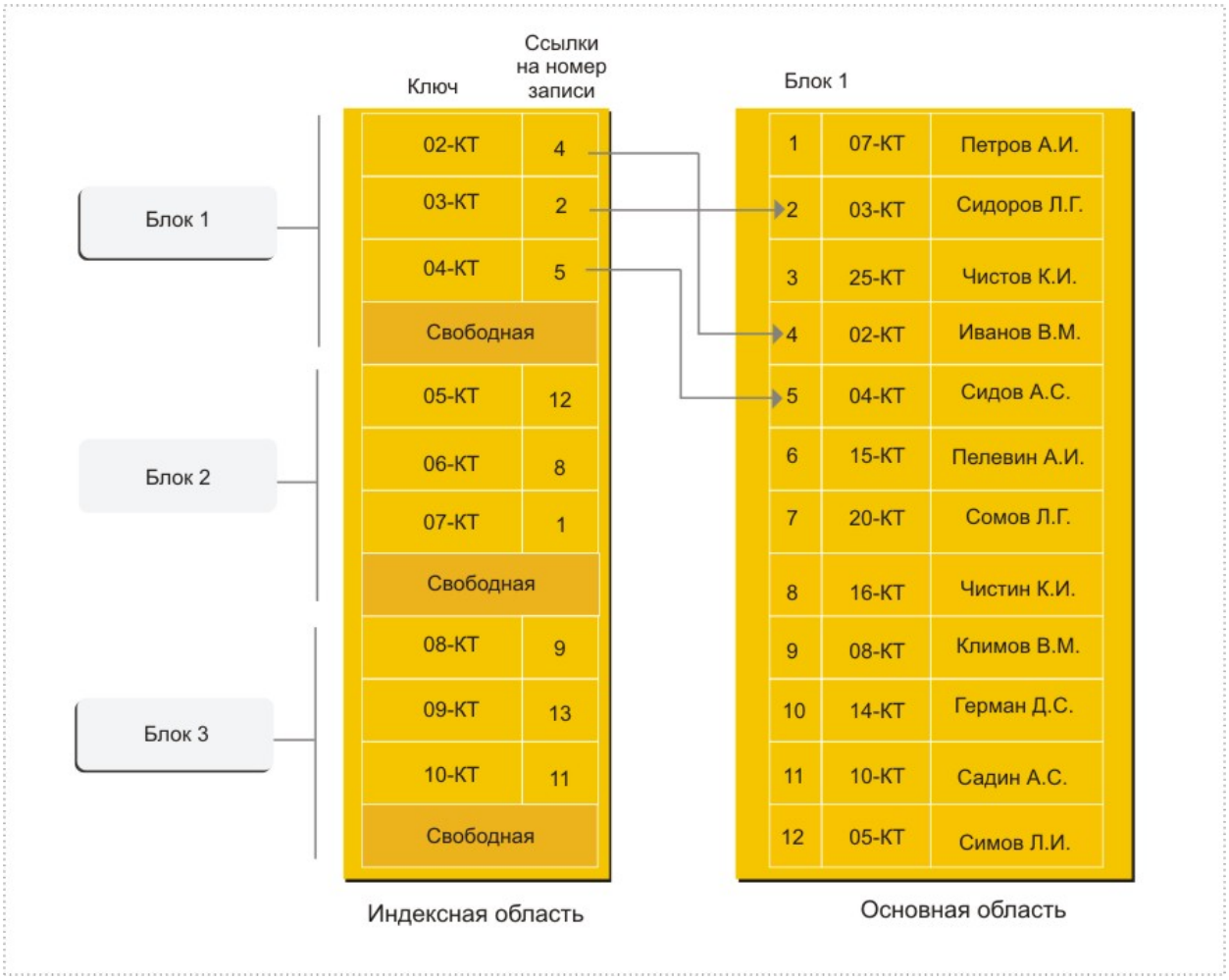


Рис. 7.6. Организация индексно-прямой адресации

При проектировании физической базы данных так важно заранее как можно точнее определить объемы хранимой информации, спрогнозировать ее рост и предусмотреть соответствующее расширение области хранения.

При удалении записи возникает следующая последовательность действий: запись в основной области помечается как удаленная, в индексной области соответствующий индекс уничтожается физически, то есть записи индексного файла, следующие за удаленной записью, перемещаются на ее место, и блок, в котором хранился данный индекс, заново записывается на диск. При этом количество обращений к диску для этой операции такое же, как и при добавлении новой записи.

7.2.3. Индексно-последовательные файлы

Если файлы поддерживаются в отсортированном состоянии с момента их создания, то для работы с ними может быть использован другой подход. Принципы внутреннего упорядочения и блочности построения таких файлов позволяют уменьшить количество хранимых индексов за счет того, что в индексном файле не содержатся указатели на

все записи индексированного файла. Таким образом, в этом случае индекс получается *неплотным* или *разреженным.*

Индексная запись для таких файлов должна содержать: значение ключа первой записи блока и номер блока с этой записью.

Теперь по заданному значению первичного ключа в индексной области требуется отыскать уже нужный блок. Все остальные действия происходят в основной области (рис. 7.7). При переходе к неплотному индексу время доступа уменьшается практически в полтора раза.

При таком подходе новая запись должна заноситься сразу в требуемый блок на требуемое место. Естественно, что для добавления записей уже блок основной области должен иметь свободное место. При внесении новой записи индексная область не корректируется.

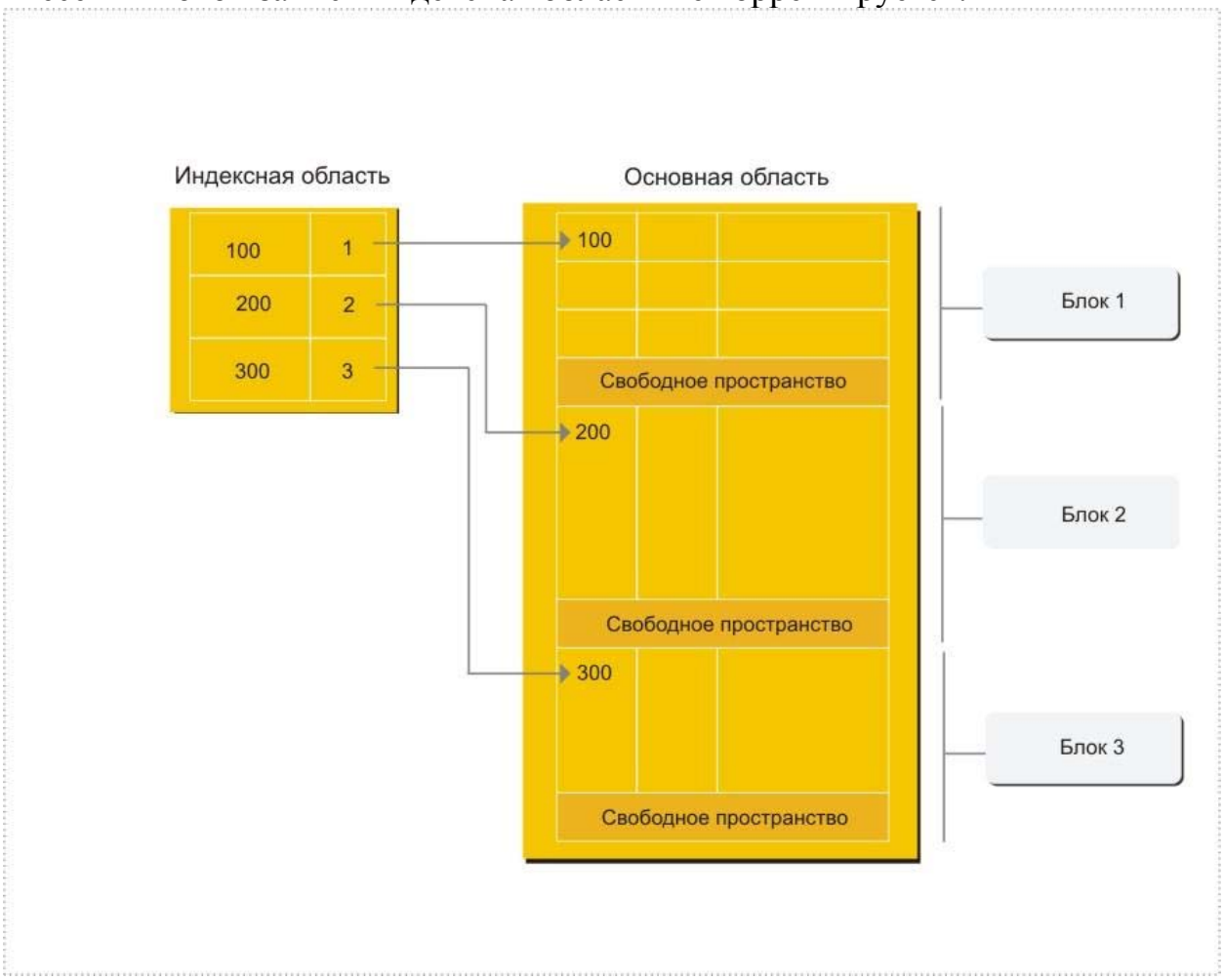


Рис. 7.7. Организация индексно-последовательной адресации

Уничтожение записи происходит путем ее физического удаления из основной области, при этом индексная область обычно не корректируется.

Однако следует отметить:

- с помощью одного неплотного индекса нельзя выполнить проверку наличия некоторого значения;

- в данном хранимом файле может быть, по крайней мере, один неплотный индекс, который организуется по полю, по которому этот файл отсортирован, а остальные индексы обязательно должны быть плотными.

7.2.3. Организация индексов в виде Б-деревьев

Структура типа Б-дерева является частным случаем бинарного индекса древовидного типа, которая используется для построения наиболее важных и распространенных индексов. Бинарные индексы обладают в большинстве случаев сравнительно высокой производительностью и поэтому в настоящее время их использование предусмотрено почти во всех СУБД, а некоторые СУБД работают только на основе такого индекса.

Высокая производительность бинарных индексов обеспечивается тем, что при их использовании удается избежать обязательного просмотра всего содержимого файла согласно его физической последовательности, которое требует больших затрат времени. Дело в том, что если индексированный файл имеет большой размер, то и его индекс также очень велик и последовательный просмотр даже только одного индекса занимает значительное время.

Построение Б-деревьев связано с простой идеей построения индекса над уже построенным индексом. Действительно, если уже построен плотный (но может быть и неплотный, если в индексированном файле проведена кластеризация на основе индекса) индекс для реальных данных, то сама индексная область может рассматриваться как основной файл, над которым надо снова построить неплотный индекс.

Записи внутри индекса сгруппированы по страницам, а страницы связаны в цепочку таким образом, чтобы логическое упорядочение на основе индекса осуществлялось внутри первой страницы, затем с помощью физической последовательности записей внутри второй страницы этой последовательной цепочки и т. д. Таким образом, с помощью набора последовательностей можно организовать быстрый последовательный доступ к индексированным данным.

Потом снова над новым индексом строится следующий неплотный индекс и так до того момента, пока не останется всего один индексный блок. Эта операция обычно применяется трижды, поскольку создание большого количества иерархических уровней индексирования требуется для очень больших файлов. При этом индекс на каждом из уровней будет неплотным по отношению к нижнему индексируемому уровню. Таким образом, на самом верхнем уровне такого индекса находится только один элемент структуры (страница, содержащая множество записей), который называется корневым.

Набор индексов обеспечивает быстрый непосредственный доступ к набору последовательностей (а значит, и к данным).

В результате такого построения получится некоторое дерево (рис. 7.8), каждый родительский блок которого связан с одинаковым количеством подчиненных блоков, число которых равно числу индексных записей, размещаемых в одном блоке. Количество обращений к диску при этом для поиска любой записи одинаково и равно количеству уровней в построенном дереве. Такие деревья называются сбалансированными потому, что путь от корня до любого листа в этом древе одинаков.

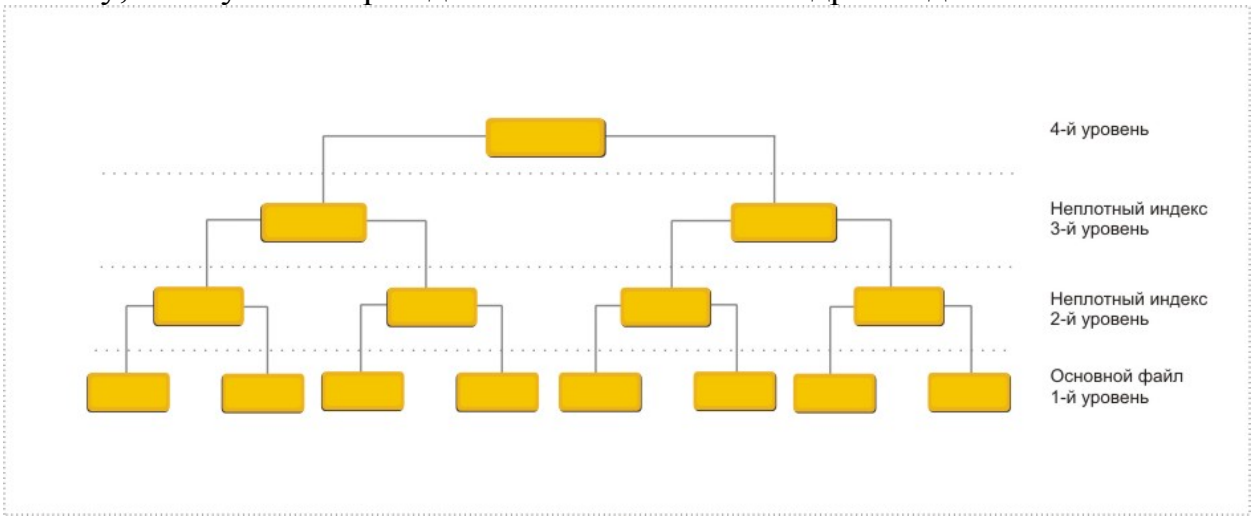


Рис. 7.8. Б-дерево

Структуры типа Б-дерева позволяют, используя специальные, алгоритмы, осуществлять сбалансированную вставку или удаление значений. К. Дж. Дейт приводит такой краткий алгоритм вставки нового значения V в структуру типа Б-дерева порядка . Он рассчитан на вставку значения только лишь в набор индексов, но может быть достаточно просто расширен для вставки записи с данными в набор последовательностей.

- На самом низком уровне набора индексов следует найти элемент (допустим, что это элемент N), с которым логически связано вставляемое значение V. Если элемент N содержит свободное пространство, то значение V вставляется в него, и на этом процесс завершается.
- В противном случае (если свободного пространства нет, т. е. придется создать еще один уровень) элемент N (допустим, что он содержит 2n индексных записей) разделяется на два элемента —N1 и N2. Обозначим символом S множество из 2n + 1 значений, в котором 2n исходных значений и одно новое значение V. Тогда n первых значений этой логической (уже упорядоченной) последовательности необходимо поместить в элемент N1, n последних — в элемент N2, а среднее между ними значение W— в родительский элемент P на более высоком структурном уровне. Впоследствии, при

осуществлении поиска значения U и достижении элемента P, поиск будет перенаправлен в сторону элемента N1, если $U \leq W$, либо в сторону элемента N2, если $U > W$.

- Далее этот процесс следует повторить для вставки среднего значения W в родительский элемент P на более высоком структурном уровне.

В худшем случае процесс разделения элементов структуры может продлиться вплоть до корневого элемента всей структуры с образованием нового иерархического уровня.

Для удаления некоторого значения следует применить аналогичный алгоритм, но только в обратном порядке. А для изменения значения его можно удалить, а затем вставить новое.

7.2.4. Инвертированные списки

Базы данных должны предоставлять возможность проводить операции доступа к данным не только по первичным, но и по вторичным индексам. Для обеспечения ускорения доступа по вторичным индексам используются структуры, называемые инвертированными списками.

При организации инвертированного списка можно выделить три уровня (рис. 7.9).

Самый нижний уровень представлен собственно основным файлом.

Над этим уровнем строится еще два уровня, которые и представляют собой непосредственно инвертированный список.

- На первом уровне этой структуры находится файл, в который помешаются значения вторичных индексов основного файла, причем в упорядоченном состоянии. В этом файле предусмотрено поле, куда помешается ссылка на второй уровень.
- На втором уровне для каждого значения вторичного индекса строится цепочка блоков, содержащих номера записей основного файла с этим значением вторичного индекса. Адрес первого блока такой цепочки и помещается в поле ссылки первого уровня. При этом блоки второго уровня также упорядочены по значениям вторичного индекса.

Механизм доступа к записям по вторичному индексу при подобной организации записей состоит в следующем:

- найти в области первого уровня заданное значение вторичного индекса;
- по ссылке считать блоки второго уровня, содержащие номера записей с заданным значением вторичного индекса;
- прямым доступом загрузить в рабочую область пользователя содержимое всех записей, содержащих заданное значение вторичного индекса.

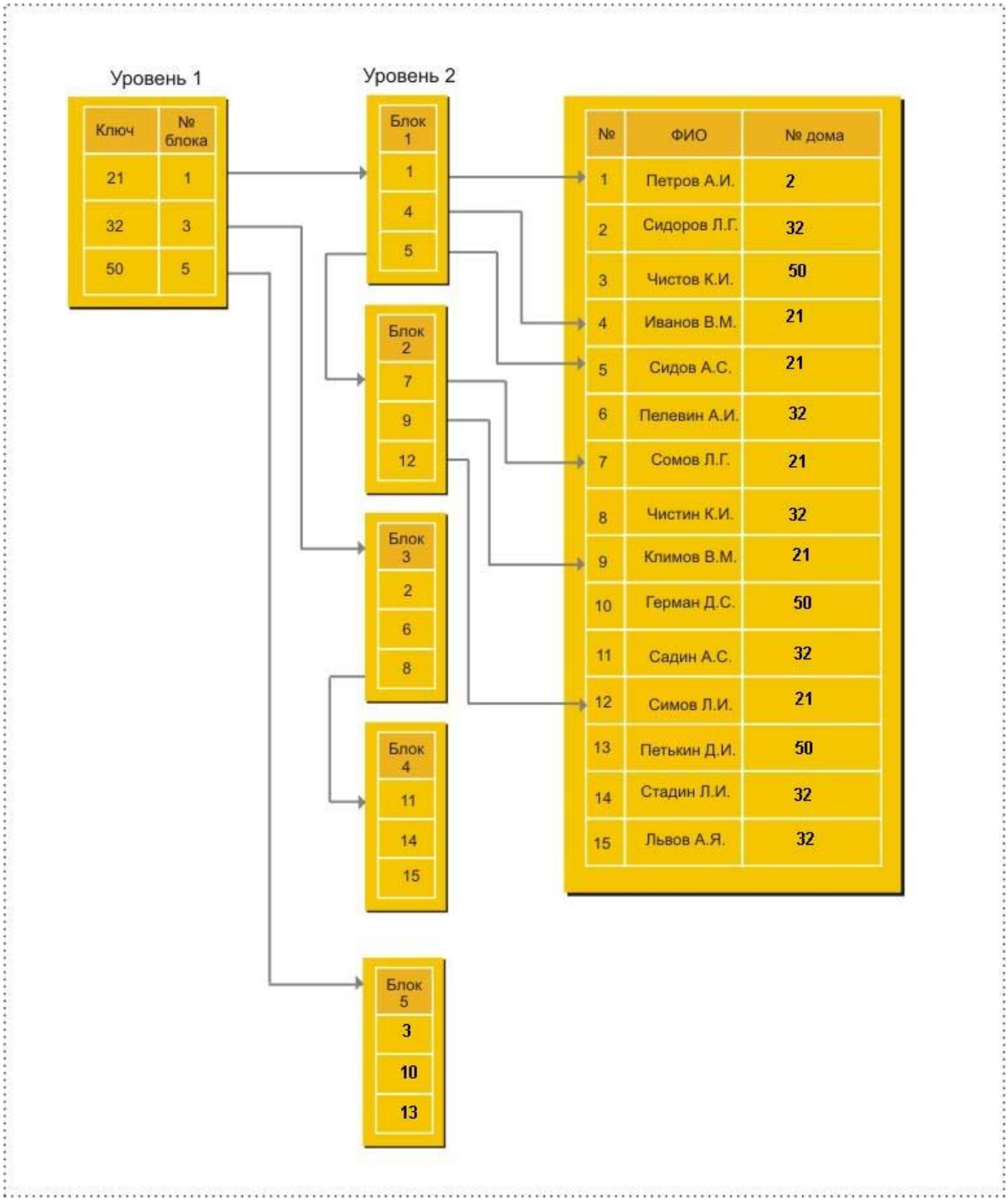


Рис.7.9. Уровни инвертированного списка

Для одного основного файла может быть создано несколько инвертированных списков по разным вторичным индексам.

Если же база данных постоянно изменяется, дополняется, модифицируется содержимое записей, то наличие большого количества инвертированных списков или индексных файлов по вторичным индексам может резко замедлить процесс обработки информации.

Действительно, модификация основного файла в такой ситуации требует:

- изменить запись основного файла;
- исключить старую ссылку на предыдущее значение вторичного индекса;
- добавить новую ссылку на новое значение вторичного индекса.

8. Управление реляционной базой данных

Для управления реляционной базой данных Э. Ф. Кодд ввел реляционные языки обработки данных — реляционную алгебру и реляционное исчисление.

Реляционная алгебра — это процедурный язык обработки реляционных таблиц. Это означает, что в реляционной алгебре используется пошаговый подход к созданию реляционных таблиц, содержащих ответы на запросы.

Реляционное исчисление — непроцедурный язык. В реляционном исчислении запрос создается путем определения таблицы запроса за один шаг.

Кодд показал логическую эквивалентность реляционной алгебры и реляционного исчисления. Это означает, что любой запрос, который можно сформулировать при помощи реляционного исчисления, также можно сформулировать, пользуясь реляционной алгеброй, и наоборот.

И реляционная алгебра, и реляционное исчисление в том виде, как они были сформулированы Коддом, являются теоретическими языками.

8.1. Реляционная алгебра

Ранее были определены основные операции по обновлению информации в реляционной базе данных. Данные операции обновления — это операции не над отношениями, а над кортежами отношения. Операторы реляционной алгебры используют одно или два из существующих отношений для создания нового отношения. Реляционная алгебра (или алгебра отношений) представляет собой совокупность операций высокого уровня над отношениями. Реляционная алгебра определяет следующие операции:

- объединение;
- разность;
- произведение;
- пересечение;
- проекция;
- выбор;
- соединение;
- деление.

Первые четыре операции взяты Коддом из математической теории множеств и практически совпадают с операциями теории множеств. Следующие четыре — новые операции, относящиеся только к реляционной модели данных.

8.1.1. Объединение (Union)

Пусть имеются отношения r и s, тогда отношение t = r ∪ s называется объединением r и s, если каждый кортеж, принадлежащий t, принадлежит или r, или s, или им обоим.

Пример
Пусть даны отношения:

r— Изделие 1

Код_дет	Название	Вес
01	A	1
03	B	2
04	C	3

s — Изделие 2

Код_дет	Название	Вес
02	Д	2
03	B	2
04	C	3

Необходимо сформировать ответ на следующий запрос: какие типы деталей входят в состав обоих изделий? Для достижения этой цели необходимо выполнить операцию t=r ∪ s. Результирующее отношение содержит все детали, которые входят в состав обоих изделий.

Код_дет	Название	Вес
01	A	1
02	Д	2
03	B	2
04	C	3

8.1.2. Разность

Пусть имеются два отношения r и s, тогда отношение t = r—s называется разностью r и s, если каждый кортеж, принадлежащий t, принадлежит r, но не принадлежит s. Операция применяется к отношениям одной арности. Пусть отношение r представляет потребности в некоторых видах деталей, а отношение s — сведения о тех видах деталей, которые фирма может произвести сама, тогда отношение t = r—s содержит сведения о тех видах деталей, которые нужно приобрести.

r - ПОТРЕБНОСТИ

Код_дет	Название	Вес
01	А	1
02	Д	2
03	В	2
04	С	3
05	Е	1

s - ВОЗМОЖНОСТИ

Код_дет	Название	Вес
02	Д	2
03	В	2
04	С	3

t= r- s

Код _дет	Названне	Вес
01	А	1
05	Е	1

8.1.3. Декартово произведение

Эта операция не накладывает никаких ограничений на схемы исходных отношений, и поэтому она допустима для любых двух отношений.

Под декартовым произведением двух отношений понимается множество упорядоченных пар кортежей. Пусть имеются два отношения r и s, тогда отношение t = r * s арности к = $к_1$ + k_2, где $к_1$ - арность r, а k_2 — арность s, называется декартовым произведением r и s, если оно состоит из кортежей, первые k_1 компонентов которых образуют кортежи из r, а остальные k_2 — из s.

Пример

Пусть r → **СТУДЕНТЫ (Ном_зач_кн, ФИО);**

s → **ЭКЗАМЕНЫ (Код_дисц, Назв_дисц, Дата, Оценка),**

тогда r * s → **ЭКЗАМ_ВЕД (Ном_зач_кн, ФИО, Код_дисц, Назв_дисц, Дата, Оценка).**

r – СТУДЕНТЫ

Ном_зач_кн	ФИО
02-Э-01	Иванов И.И.
02-Э-02	Петров Т.Т.
02-Э-05	Серов С.С.

s - ЭКЗАМЕНЫ

Код_дисц	Назв_дисц	Дата	Оценка
01	Математика	10.01.03	
02	Физика	15.01.03	
03	Ин. язык	20.01.03	

ЭКЗАМЕНАЦИОННАЯ ВЕДОМОСТЬ ПО ВСЕМ ДИСЦИПЛИНАМ

Ном_зач_кн	ФИО	Код_дисц	Назв_днсц	Дата	Оценка
02-Э-01	Иванов И.И.	01	Математика	10.01.03	
02-Э-01	Иванов И.И.	02	Физика	15.01.03	
02-Э-01	Иванов И.И.	03	Ин. язык	20.01.03	
02-Э-02	Петров Т.Т.	01	Математика	10.01.03	

02-Э-02	Петров Т.Т.	02	Физика	15.01.03	
02-Э-02	Петров Т.Т.	03	Ин. язык	20.01.03	
02-Э-05	Серов С.С.	01	Математика	10.01.03	
02-Э-05	Серов С.С.	02	физика	15.01.03	
02-Э-05	Серов С.С.	03	Ин. язык	20.01.03	

8.1.4. Пересечение

Пусть имеются два отношения r и s, тогда отношение $t = r \cap s$ называется пересечением r и s, если каждый кортеж, принадлежащий t, одновременно принадлежит r и s. Операция применяется к отношениям одной арности. Справедлива следующая формула: $t = r \cap s = r - (r - s)$.

Пример

Пусть имеются отношения:

r - ИЗДЕЛИЕ 1

Код_дет	Название	Вес
01	А	1
02	Д	2
03	В	2
04	С	3
05	Е	1

s - ИЗДЕЛИЕ 2

Код_дет	Название	Вес
02	Д	2
04	С	3
03	В	2
06	К	1

Сформируем ответ на такой запрос: определить детали, входящие в состав обоих изделий. Для этого необходимо выполнить операцию пересечения двух исходных отношений. Результат представляется отношением:

Код_дет	Название	Вес
02	Д	2
04	С	3
03	В	2

8.1.5. Проекция (Project)

Оператор проекции (вертикальное подмножество) является унарным оператором на отношениях. Он осуществляет выбор на множестве столбцов.

Пусть в отношении r(R) выделено некоторое множество атрибутов Y, тогда отношение $t = \pi_Y(r)$ называется проекцией отношения r, если оно является вертикальным подмножеством столбцов отношения r из множества R.

Иными словами, проекция R на Y есть также отношение, полученное вычеркиванием столбцов, соответствующих атрибутам R — Y, и

исключением, по определению отношения, из оставшихся столбцов повторяющихся строк.

Пусть дано отношение r:

A	B	C
P	1	a
P	2	b
Q	2	c

Тогда:

$\pi_{AC}(r)$

A	C
P	a
P	b
Q	c

8.1.6. Выбор (Select)

Выбор или селекция — это одна из важнейших операций обработки информации. Она также как и предыдущая, относится к унарным операциям над отношением. Результатом ее применения к отношению r является другое отношение, которое представляет собой подмножество кортежей отношения r, с определенным значением в выделенном атрибуте.

Итак, результатом селекции отношения r по некоторому Θ будем считать отношение $t = \delta_{\Theta}(r)$, которое включает в себя кортежи отношения r, удовлетворяющие указанному условию Θ.

Условие Θ — это формула, по которой определяется выборка. Операндами в такой формуле являются атрибуты отношения, а знаками операций — логические операции и операции отношений.

$\delta_{C \neq b}(r)$

A	B	C
P	1	a
Q	2	c

$\delta_{A \neq q \wedge B > 1}(r)$

A	B	C
P	2	b

8.1.7. Соединение (Join)

Пусть имеются два отношения r (X, У) и s (Y, Z) и некоторое условие Θ, где X, У, Z-— непересекающиеся множества атрибутов и Y— множество атрибутов, общих для r и s, тогда отношение

$t = r >_{\Theta}< s$

называется Θ-соединением r и s, если каждый кортеж, принадлежащий t, состоит из кортежей r и s, при выполненном условии Θ. Справедлива следующая формула:

$t = \delta_{\Theta} (r * s)$,

то есть Θ-соединение представляет собой декартово произведение r и s, над которым выполнена селекция по условию Θ.

Пример

Из общей экзаменационной ведомости по всем дисциплинам получим экзаменационную ведомость по дисциплине математика. Для этого выполним операцию соединение (Join) при условии Код_дисц = “01”

ЭКЗАМЕНАЦИОННАЯ ВЕДОМОСТЬ ПО ДИСЦИПЛИНЕ МАТЕМАТИКА

Ном_зач_кн	ФИО	Код_дисц	Назв_дисц	Дата	Оценка
02-Э-01	Иванов И.И.	01	Математика	10.01.03	
02-Э-02	Петров Т.Т.	01	Математика	10.01.03	
02-Э-05	Серов С.С.	01	Математика	10.01.03	

8.1.8 Деление

Деление — это также бинарная несимметричная операция для получения некоторого отношения из двух исходных, причем степень результирующего отношения не совпадает со степенью ни одного из операндов, а вычисляется как разность между степенью отношения-делимого и степенью отношения-делителя.

Пусть имеются отношения r(X, У) арности k_1 и p(Z) арности k_2, где Y и Z определены на одном домене, тогда отношение t = r ÷ p арности k1 – k2 называется делением r на p, если любой кортеж из t вместе с любым кортежем из p образуют кортеж, имеющийся в r.

Пример

Даны отношения, содержащие сведения об экзаменах, которые должны были сдавать студенты, и сведения о результатах сдачи этих экзаменов:

VEDOM

Ном_зач_кн	Фамилия	Назв_дисц	Дата
02-Э-01	Иванов	Физика	10.01.03
02-Э-01	Иванов	Химия	14.01.03
02-Э-02	Сидоров	Физика	10.01.03
02-Э-02	Сидоров	Химия	14.01.03
02-Э-05	Коровин	Химия	14.01.03

RASP

Назв_дисц	Дата
Химия	14.01.03
Физика	10.01.03

Сформируем ответ на такой запрос: дать сведения о студентах, сдавших все экзамены. Для получения ответа на данный запрос необходимо выполнить следующую операцию деления:

t = r ÷ p

STUD = VEDOM ÷ RASP

Ном_зач_кн	Фамилия
02-Э-01	Иванов
02-Э-02	Сидоров

8.2. Реляционное исчисление

Большинство работающих языков запросов основано на реляционном исчислении. Реляционные исчисления — непроцедурные системы. Исчисления выражают только то, каким должен быть результат вычисления, но не то, каким образом проводить вычисления. Эта обязанность возлагается на процессор языка запросов данной СУБД.

Различают реляционное исчисление кортежей и реляционное исчисление доменов. Поскольку реляционное исчисление доменов сходно с реляционным исчислением кортежей за исключением того, что переменные принимают значения в доменах, а не являются кортежами, а исчисление кортежей используется чаще, рассмотрим только исчисление кортежей. В дальнейшем, когда речь будет идти о реляционном исчислении, будет подразумеваться именно реляционное исчисление кортежей.

Реляционное исчисление кортежей является, по сути, формализацией системы обозначений, предназначенной для образования множеств. В реляционном исчислении используются булевы операции (И, ИЛИ, НЕ) над условиями, которые могут быть истинными или ложными. В нем также используются кванторы существования и всеобщности, означающие, соответственно, что элемент определенного типа существует или что условие истинно для каждого элемента определенного типа.

Рассмотрим отношение:

r — Деталь

Код_дет	Назв_дет	Вес
01	А	1
02	Д	2
03	В	2
04	С	3

Запрос: Какие детали имеют вес, равный 2?

В реляционной алгебре для выполнения этого запроса необходимо использовать алгебраическое выражение, которое должно содержать следующие операции:

- операцию Выбор над отношением r, результатом ее применения к отношению r является другое отношение, которое представляет собой подмножество кортежей отношения r со значением, равным 2 в атрибуте Вес:

Код_дет	Назв_дет	Вес
02	Д	2
03	В	2

- проецирование результата предыдущей операции на атрибут **Назв_дет.** Окончательный результат запроса будет выглядеть следующим образом:

Назв_дет
Д
В

В реляционном же исчислении формулировка этого запроса должна иметь следующий вид:

{ t.Назв_дет | t in r and t.Вес = 2},

где t — это переменная, обозначающая произвольную строку. Отношение, из которого берется t, определяется выражением “ in r”которое означает, что t— строка отношения.

t.Назв_дет — значение атрибута **Назв_дет** в строке r; символ (|) — разделяет целевой список и определяющее выражение. В данном случае:

t.Назв_дет — целевой список;

t in r and t.Вес = 2 — определяющее выражение;

t.Вес = 2 означает, что значение атрибута **Вес** в строке t равно 2.

Фигурные скобки "{}", в которые заключено выражение, определяют результат запроса, как множество значений данных. Что именно входит в это множество, описывает выражение в скобках. Приведенное здесь решение иллюстрирует почти все элементы реляционного исчисления.

8.2.1. Целевой список и определяющее выражение

Решением каждого запроса в реляционном исчислении является отношение, которое задается целевым списком и определяющим выражением. Целевой список определяет атрибуты отношения решения. Определяющее выражение — это условие, на основании которого отбираются значения из базы данных, которые войдут в отношение решения.

В предыдущем примере **Назв_дет** берется из строки и помешается в строку решения, если строка t удовлетворяет условию

t in r and t. Вес = 2.

Система просматривает строки отношения r одну за другой. Первой строке временно присваивается имя t и проверяется истинность определяющего выражения. Поскольку в этом случае t.Вес = 1, определяющее выражение, ложно, поэтому А — значение атрибута t.Назв_дет в этой строке не помещается в отношение решения. Затем система переходит к следующей строке, дает ей имя t и снова проверяет истинность определяющего выражения. На этот раз выражение истинно, поэтому Д помещается в отношение решения. Процесс повторяется для каждой строки отношения r. В результате получается следующее отношение, идентичное полученному ранее:

Назв_дет
Д
В

Целевой список может состоять из нескольких атрибутов, разделенных запятыми.

{t.Код_дет, t.Назв_дет, t.Вес | t in r and t.Вес = 2}.

Рассмотрим еще один пример. Пусть даны отношения:

r - ИЗДЕЛИЕ 1

Код_дет	**Назв_дет**	**Вес**
01	А	1
02	Д	2
03	В	2
04	С	3
05	Е	1

s -ИЗДЕЛИЕ 2

Код_дет	**Назв_дет**	**Вес**
02	Д	2
04	С	3
03	В	2
06	К	1

Согласно алгебраическому выражению $\pi_{\text{Назв\_дет}}(r \cap s)$ реляционной алгебры, содержащему операции пересечения и проецирования, получим отношение:

Назв_дет
Д
С
В

В реляционном же исчислении, если t кортеж r, а y кортеж s, этот запрос может включать в себя следующие компоненты:

t.Назв_дет - целевой список;

t in r and y in s and t.Код_дет = y.Код_дет and t.Вес = y.Вес and t.Назв_дет = y.Назв_дет — определяющее выражение.

В рассмотренных примерах были использованы операции выбора, пересечении и проецирования реляционной алгебры. Аналогично можно получить конструкции реляционного исчисления, соответствующие ряду других операций: объединению, разности и произведению.

Несколько иначе выглядят аналоги только двух операций реляционной алгебры — операций соединения и деления. Для построения аналогов этих операций требуются кванторы: квантор существования для соединения и квантор всеобщности для деления.

8.2.2. Квантор существования

Квантор существования означает, что существует хотя бы один экземпляр определенного типа вещей. В реляционном исчислении квантор существования используется для задания условия того, что определенный тип строк в отношении существует.

Рассмотрим отношения:

ПОТРЕБНОСТИ

Код_дет	Назв_дет	Количество
01	А	1
02	Д	2
03	В	2
04	С	3
05	Е	1

СКЛАД

Код_дет	Стеллаж	Кол_дет
01	05	100
03	10	250
04	02	2

Запрос: Перечислить названия деталей, потребности в которых покрываются деталями, хранящимися на складе.

Очевидно, что решением этой задачи будет отношение, содержащее названия некоторых деталей. Это отношение состоит из одного столбца, так что в этом случае целевым списком является:

t.Назв_дет,

где t— строка из отношения **ПОТРЕБНОСТИ**.

Пусть s — строка отношения **СКЛАД**.

Формирование определяющего выражения осуществляется, исходя из следующего. Для того чтобы деталь вошла в отношение решения, должно удовлетворяться условие, что на складе эта деталь имеется в достаточном количестве. Другими словами, если **Код_дет** встречается в строке отношения **СКЛАД** с **Кол_дет > Количество** в строке отношения **ПОТРЕБНОСТИ**, то эта деталь должна быть включена в решение. Таким образом, условие таково: существует хотя бы одна строка в отношении

СКЛАД, содержащая требуемый Код_дет и где s.Кол_дет > t.Количество. Это формируется следующим образом:

exists s in СКЛАД

(s.Код_дет = t.Код_дет and s.Кол дет > t.Количество).

Такое выражение читается: "Существует строка s в отношении **СКЛАД** такая, что s.Код_дет = t.Код_дет и s. Код_дет > t.Количество".

Приведенное выражение определяет строку t. Если оно истинно, то есть для строки t существует такая строка s, то t.Назв_дет помещается в результирующее отношение. Если выражение ложно — то есть такой строки s не существует — тогда t.Назв_дет не помещается в результирующее отношение.

Полное решение в реляционном исчислении выглядит следующим образом:

{t.Назв_дет | t in ПОТРЕБНОСТИ and exists s in СКЛАД (s.Код_дет = t.Код_дет and s.Кол_дет > t.Количество)}.

Оно описывает отношение, состоящее из одного столбца и содержащее названия деталей, взятых из строк отношения **ПОТРЕБНОСТИ**. Данное название помещается в отношение решения, если его строка t удовлетворяет условию после знака " |".

Рассмотрим подробнее вышеописанный механизм обработки нескольких строк отношения **ПОТРЕБНОСТИ**, чтобы понять, как будет применяться условие.

Первая строка отношения **ПОТРЕБНОСТИ** (которая обозначена t) имеет **Назв_дет = А**, и оно будет помещено в результирующее отношение, если в отношении **СКЛАД** существует строка, в которой **Код_дет** = ‘01’, а Кол_дет > t.Количество. Такая строка действительно существует, и она обозначена как s. Итак, t удовлетворяет определяющему условию, поэтому t.Назв_дет помещается в отношение решения. Этот процесс должен повториться для каждой строки отношения **ПОТРЕБНОСТИ**. Когда закончена обработка первой строки, вторая строка обозначена t, и теперь уже для нее ищется соответствующая строка s в отношении **СКЛАД**. Такой строки не существует, поэтому Д не помещается в результирующее отношение. Продолжая дальше обработку строк отношений по указанному алгоритму, получим множество решения, которое составит новое отношение, и будет выглядеть следующим образом:

Назв_дет
А
В

В реляционной алгебре для выполнения этого запроса требуется соединение. Таким образом, было показано, как квантор существования

используется в реляционном исчислении, для того чтобы выполнять функцию соединения.

8.2.3. Квантор всеобщности

Квантор всеобщности означает, что некоторое условие применяется ко всем строкам или к каждой строке некоторого типа. Он используется в тех же целях, что и операция деления реляционной алгебры. Проиллюстрируем его применение тем же запросом, который рассматривался в разделе, описывающем операцию деления. Пусть даны два отношения:

VEDOM

Ном_зач_кн	Фамилия	Назв_дисц	Дата
02-Э-01	Иванов	Физика	10.01.03
02-Э-01	Иванов	Химия	14.01.03
02-Э-02	Сидоров	Физика	10-01.03
02-Э-02	Сидоров	Химия	14.01.03
О2-Э-О5	Коровин	Химия	14.01.03

RASP

Назв_дисц	Дата
Химия	14.01.03
Физика	10.01.03

Требуется определить фамилии тех студентов, каждый из которых сдавал все экзамены, перечисленные в отношении **RASP**. Необходимо обратить внимание на то, что условие выбора студента содержит определение каждый. В результирующее отношение нключаются только те студенты, которые сдавали каждый экзамен. Если взглянуть на полученный результат, то можно увидеть, что условию запроса удовлетворяют только два студента.

Полное решение в реляционном исчислении с использованием квантора всеобщности FORALL таково:

{t.Фамилия | t in VEDOM and s in RASP and
FORALL s {t. Назв_дисц_на = s.Назв_дисц and t.Дата = s.Дата)},
где t — кортеж отношения **VEDOM**;
s — кортеж отношения **RASP**.
Результат выполнения запроса:

Фамилия
Иванов
Сидоров

Имя студента из строки t таблицы **VEDOM** помещается в результирующее отношение, если определяющее выражение истинно для строки t, а определяющее выражение истинно, если для каждой строки s отношения **RASP** должна существовать строка t в отношении **VEDOM**,

удовлетворяющая условию, то есть в данном случае для каждой строки s отношения **RASP** должна существовать строка t в отношении **VEDOM** с одной и то же фамилией:

- Иванов присутствует в строках t (1) и t (2): t (1) соответствует s (2) и t (2) соответствует s (1);
- Сидоров присутствует в строках t (3) и t (4): t (3) соответствует s (2) и t (4) соответствует s (1);
- Коровин присутствует только в строке t (5): t (5) соответствует s (1), а для s (2) нет соответствующей строки t.

Полное соответствие всем строкам отношения RASP есть только у Иванова и Сидорова. Таким образом, они и вошли в результирующее отношение.

9. Язык SQL

9.1. Оператор выбора SELECT. Формирование запросов к базе данных

Назначение оператора SELECT состоит в выборке и отображении данных одной или нескольких таблиц БД.

Синтаксис оператора SELECT:

SELECT [DISTINCT| ALL] {* ⏐[<СПИСОК СТОЛБЦОВ>]} FROM <СПИСОК ТАБЛИЦ>

[WHERE <предикат-условие выборки или соединения;>] [GROUP BY <список полей результата>]

[HAVING <предикат-условие для группы>]

[ORDER BY <список полей, по которым требуется упорядочить вывод>]

Поясним каждую фразу данного оператора.

Фраза SELECT:

- наличие ключевого слова ALL (по умолчанию) означает, что в результирующую таблицу включаются все строки, удовлетворяющие условиям запроса, что может привести к появлению в результирующей таблице одинаковых строк;
- ключевое слово DISTINCT предназначено для приведения таблицы в соответствие с принципами теории отношений, где предполагается отсутствие дубликатов строк;
- символ "*" определяет очень часто встречаемую ситуацию, когда в результирующий набор включаются все столбцы из исходной таблицы
- запроса.

Во фразе FROM задается перечень исходных таблиц запроса.

Во фразе WHERE определяются условия отбора строк результата или условия соединения строк исходных таблиц, подобно операции условного

соединения в реляционной алгебре. В качестве условий отбора могут быть использованы следующие операторы:

- сравнения " = , <>, >, <, >=, <=" — для сравнения результатов вычисления двух выражений; более сложные выражения строятся с помощью логических операторов AND, OR, NOT;
- BETWEEN A AND B — предикат истинен, когда вычисленное значение выражения попадает в заданный диапазон;
- IN — предикат истинен тогда, когда сравниваемое значение входит в множество заданных значений;
- LIKE и NOT LIKE — предикаты, смысл которых противоположен, требуют задания шаблона, с которым сравнивается заданное значение;
- IS NULL — предикат, применяющийся для выявления равенства значения некоторого атрибута неопределенному значению:
- EXIST и NOT EXIST, используемые во встроенных подзапросах.

Во фразе GROUP BY задается список полей группировки.
Во фразе HAVING задаются предикаты-условия, накладываемые на каждую группу.
Во фразе ORDER BY задается список полей упорядочения результата, то есть список полей, который определяет порядок сортировки в результирующей таблице.

9.1.1. Простые запросы

Рассмотрим ряд простых запросов.

Запрос 1

Вывести номера телефонов кафедр университета.

Результат такого запроса должен содержать только два столбца: Name_kaf и Nom_telef, поэтому сам запрос должен выглядеть следующим образом:

```
SELECT Name_kaf, Nom_telef
FROM kafedra;
```

Результирующая таблица приведена ниже:

Name_kaf	Nom_telef
Физики	23-34-24
Общей математики	23-65-43
Истории	23-78-72
Графики	23-99-77
Прикладной математики	23-66-62

Запрос 2

Вывести сведения о кафедре Графики. Запрос будет выглядеть следующим образом:

SELECT *
FROM kafedra
WHERE Name_kaf = 'Графики';
Ответ на такой запрос будет содержать только одну строку:

Kod_kaf	Name_kaf	Nom_telef	Nom_Auditoria	Col_sotr	Zav_kaf
004	Графики	23-99-77	385	18	Фирсов С.С.

Запрос 3

Вывести сведения о кафедрах университета, находящихся па первом этаже, учитывая тот факт, что номера аудиторий первого этажа лежат в диапазоне от 1 до 99.

Запрос будет выглядеть следующим образом:
SELECT *
FROM kafedra
WHERE NonuAuditoria BETWEEN 1 AMD 99;
Результат запроса:

Kod_kaf	Name_kaf	Nomtetef	Nom_Auditoria	Col_sotr	Zavjcaf
002	Общей математики	23-65-43	003	22	Махов
005	Прикладной	23-66-62	028	24	Ляхова

Запрос 4

Вывести сведения о кафедрах университета в виде, отсортированном по столбцу Name_kaf в порядке возрастания.

Запрос будет выглядеть следующим образом:
SELECT *
FROM kafedra
ORDER BY Namejtaf ASC;
Результат данного запроса:

Kod_kaf	Name_kaf	Nom_telef	Nom_Auditoria	Col_sotr	Zav__kal
004	Графики	23-99-77	385	18	Фирсов
003	Истории	23-78-72	465	16	Росс
002	Общей ма	23-65-43	003	22	Махов
005	Прикладной	23-66-62	028	24	Ляхова
001	Физики	23-34-24	132	25	Иванов Т.М.

9.1.2. Агрегатные функции языка

В стандарте языка SQL определено несколько агрегатных функций:

- COUNT — возвращает количество значений в указанном столбце;
- SUM — возвращает сумму значений в указанном столбце;
- AVG — возвращает усредненное значение в указанном столбце;
- MIN — возвращает минимальное значение в указанном столбце;
- MAX — возвращает максимальное значение в указанном столбце.

В качестве операнда данных функций может использоваться наименование только одного столбца, и все они возвращают единственное значение. С функциями SUM и AVG могут использоваться только числовые поля. С функциями COUNT, MAX и MIN могут использоваться как числовые, так и символьные поля. При вызове всех перечисленных выше функций, кроме функции COUNT (*), осуществляется исключение всех пустых значений» только после этого операция применяется к оставшимся значениям столбца. Функция COUNT (*) призвана осуществлять подсчет всех строк таблицы независимо от того, какие значения в них находятся.

Запрос 5

Подсчитать и вывести общее число кафедр университета. Запрос будет выглядеть следующим образом:

```
SELECT COUNT (*) AS count
FROM kafedra;
```

Ответ на данный запрос будет выглядеть:

count
5

Запрос 6

Определить среднее число сотрудников, работающих на кафедрах университета.

Запрос будет выглядеть следующим образом:

```
SELECT AVG(Col_sotr) AS avg
FROM kafedra;
```

Ответ на запрос:

avg
21

9.1.3. Группирование результатов

Часто встречаются ситуации, когда в отчет необходимо поместить и промежуточные результаты, опирающиеся на вычисления обобщенных групповых значений. Для применения агрегатных функций в подобных случаях предполагается предварительная операция группировки. Суть операции группировки состоит в том, что все множество строк таблицы разбивается на группы, в каждой из которых собираются строки, имеющие одинаковые значения атрибутов, которые заданы в списке группировки. Обработка такой информации реализуется путем применения агрегатных функций уже к каждой отдельной группе и выдаче полученных итогов.

В языке SQL для осуществления операции группировки в оператор SELECT включается фраза GROUP BY. Запрос, в котором присутствует фраза

GROUP BY, называется группирующим запросом, а столбцы, перечисленные в этой фразе, называются группирующими столбцами.

В дальнейшем в качестве примера будем работать с двумя БД: НИР и Сессия.

БД НИР состоит из одной таблицы, в которой хранится информация о производимых выплатах специалистам за проделанную работу по определенным этапам НИР: R= (**ФИО, Этап, Начисления**).

Пусть таблица содержит следующие данные.

r

ФИО	**Этап**	**Начисления (руб)**
Семенов Т.Т.	Этап 1	1000
Просов С.М.	Этап 1	2000
Мехова И.И.	Этап 1	500
Семенов Т.Т.	Этап 2	500
Просов С.М.	Этап 2	500
Мехова И.И.	Этап 2	1000
Просов С.М.	Этап 3	1000
Мехова И.И.	Этап 3	1000
Чемцов Я.Ю.	Этап 3	2000
Чемцов Я.Ю.	Этап 4	2000
Яров И.М.	Этап 4	3000

БД Сессия включает в себя сводную таблицу, где представлены экзаменационные оценки студентов, полученные ими в сессию по определенным дисциплинам:

S = (**ФИО, Дисциплина, Оценка**);

s

ФИО	**Дисциплина**	**Оценка**
Мур С.М.	Физика	4
Цуканов Т.Т.	Физика	5
Думская М.Т.	Физика	3
Дрозд Г.Р.	Физика	4
Мур С.М.	История	4
Цуканов Т.Т.	История	5
Думская М.Т.	История	3
Цуканов Т.Т.	Математика	5
Думская М.Т.	Математика	4
Дрозд Г.Р.	Математика	5
Петрова С.О.	Электротехника	5
Часов И.И.	Электротехника	4
Иванова Я.С.	Электротехника	5
Крисс Р.О.	Электротехника	3
Часов И.И.	Иностр. язык	5
Иванова Я.С.	Иностр. язык	4
Часов И.И.	Экономика	4
Иванова Я.С.	Экономика	4
Крисс Р.О.	Экономика	5
Фирсова Л.Р.	Экономика	3

Сформируем к базам данных несколько запросов.

Запрос 7

БД НИР. Для каждого специалиста определить сумму, выплаченную за работу по данной теме, и количество сделанных ему выплат.

Для формирования запроса включим в предложение SELECT следующую информацию: **ФИО, COUNT (Начисления) AS count, SUM (Начисления) AS sum,** где в качестве имен для двух вычисляемых столбцов используются псевдонимы. Группировку будем производить по столбцу **ФИО**. Для того чтобы проще было просматривать результаты, выводимые данные представим в отсортированном по столбцу **ФИО** виде

SELECT ФИО, COUNT (Начисления) AS count, SUM (Начисления) AS sum FROM r GROUP BY ФИО

ORDER BY ФИО;

Результат запроса:

ФИО	count	sum
Мехова И.И.	3	2500
Просов С.М.	3	3500
Семенова Т.Т.	2	1500
Чемцов Я.Ю.	2	4000
Яров И.М.	1	3000

Запрос 8

БД Сессия. Для каждой дисциплины определить количество студентов, сдавших экзамен.

Запрос будет выглядеть следующим образом:

SELECT Дисциплина, COUNT (*) AS count

FROM s

GROUP BY Дисциплина

ORDER BY Дисциплина;

Результат запроса:

Дисциплина	count
Иностр.язык	2
История	3
Математика	3
Физика	4
Экономика	4
Электротехника	4

Запрос 9

БД НИР. В условиях предыдущего запроса вывести информацию, касающуюся только тех специалистов, которым производились начисления более одного раза.

Для вывода такой информации в текст предыдущего запроса необходимо добавить фразу HAVING COUNT (Начисления! > 1. И в этом случае весь запрос примет вид

SELECT ФИО, СОШТ (Начисления) AS count, SUM (Начисления) AS sum

FROM r

GROUP BY ФИО

HAVING COUNT(Начисления) > 1

ORDER BY ФИО,-

Результаты выполнения запроса представлены ниже.

ФИО	count	sum
Мехова И.И.	3	2500
Просов С.М.	3	3500
Семенов Т.Т.	2	1500
Чемцов Я.Ю.	2	4000

9.1.4. Вложенные запросы

Стандарт языка позволяет в тело одного оператора SELECT внедрять другой оператор SELECT. Если внутренний оператор запроса помешен в предложения WHERE И HAVING внешнего оператора SELECT, ТО создается ситуация вложенных запросов (подзапросов).

Запрос 10

БД НИР. Вывести список платежей, где величина единовременных начислении превысила среднее значение.

Запрос будет выглядеть следующим образом:

SELECT ФИО, Этап, Начисления

FROM r

WHERE Начисления > (SELECT avg(Начисления) FROM r);

Результат запроса:

ФИО	Этап	Начисления (руб)
Просов С. М.	Этап 1	2000
Чемцов Я.Ю.	Этап 3	2000
Чемцов Я.Ю.	Этап 4	2000
Яров И.М.	Этап 4	3200

9.1.5. Многотабличные запросы

При работе с базами данных потребности пользователей не ограничиваются только реализацией простых запросов данных из одной

таблицы. Во многих случаях для получения ответа на запрос необходимо объединить информацию из нескольких исходных таблиц. Для того чтобы осуществить такое объединение в результирующей таблице, необходимо выполнить операцию соединения, при которой объединение информации из двух таблиц происходит посредством образования пар связанных строк, выбранных из каждой таблицы. Таблицам можно присвоить имена-псевдонимы, что бывает полезно для осуществления операции соединения таблицы с самой собою и в ряде других ситуаций.

Если в операторе SELECT указано более одного имени таблицы, неявно подразумевается, что над перечисленными таблицами осуществляется операции декартова произведения. Самый простой запрос SELECT такого рода без необязательных частей выглядит следующим образом:

```
SELECT *
FROM rl, r2;
```

и соответствует декартову произведению таблиц r1 и r2.

Выражение

```
SELECT rl.A, r2.B
FROM rl, r2;
```

соответствует проекции декартова произведения двух таблиц на два столбца A из таблицы r1 и B из таблицы r2.

Рассмотрим базу данных, в которой хранится информация о производимых выплатах специалистам за проделанную работу по определенным этапам НИР. Пусть она состоит из трех отношений r1, r2 и r3. Будем считать, что они представлены таблицами r1, r2 и r3 соответственно.

R1 = **(ФИО, Отдел);**
R2= **(Отдел, Этап);**
R3= **(ФИО, Этап, Начисления).**

r1

ФИО	**Отдел**
Семенов Т.Т.	03
Просов СМ.	03
Мехова И.И.	03
Чемцов Я.Ю.	04
Яров И.М.	04

r2

Отдел	**Этап**
03	Этап 1
03	Этап 2
03	Этап 3
04	Этап 3
04	Этап 4

r3

ФИО	Этап	Начисления (руб)
Семенов Т. Т.	Этап 1	1000
Просов СМ.	Этап 1	2000
Мехова И.И.	Этап 1	500
Семенов Т. Т.	Этап 2	500
Просов СМ.	Этап 2	500
Мехова И.И.	Этап 2	1000
Просов СМ.	Этап 3	1000
Мехова И.И.	Этап 3	1000
Чемцов Я.Ю.	Этап 3	2000
Чемцов Я.Ю.	Этап 4	2000
Яров И.М.	Этап 4	3000

Для запросов будет использоваться и расширенная база данных Сессия, представленная таблицами со схемами:

S1 = (**ФИО, Дисциплина, Оценка**) — содержащей сведения о результатах сессии;

S2= (**ФИО, Группа**) — содержащей сведения о составе групп;

S3 = (**Группа, Дисциплина**) — содержащей перечень экзаменов, подлежащих сдаче.

s1

ФИО	Дисциплина	Оценка
МурС.М.	Физика	4
Цуканов Т.Т.	Физика	5
Думская М.Т.	Физика	3
Дрозд Г.Р.	Физика	4
МурС.М.	История	4
Цуканов Т.Т.	История	5
Думская М.Т.	История	3
Цуканов Т.Т.	Математика	5
Думская М.Т.	Математика	4
Дрозд Г.Р.	Математика "	5
Петрова СО.	Экономика	5
Часов И.И.	Электротехника	4
Иванова Я. С.	Электротехника	5
Крисе Р.О.	Электротехника	3
Часов И.И.	Иностр_язык	5
Иванова Я.С.	Иностр_язьгк	4
Часов И.И.	Экономика	4
Иванова Я.С.	Экономика	4
Крисе Р.О.	Экономика	5
Фирсова Л.Р.	Экономика	3

s2

ФИО	Группа
МурС.М.	02-КТ-21
Цуканов Т.Т.	02-КТ-21
Думская М.Т.	02-КТ-21
Дрозд Г.Р.	02-КТ-21
Петров С.О.	02-КТ-12
Часв И.И.	02-КТ-12
Иванова Я.С.	02-КТ-12
Крисс Р.О.	02-КТ-12
Фирсова Л.Р.	02-КТ-12

s3

Группа	Дисциплина
02-КТ-21	Физика
02-КТ-21	История
02-КТ-21	Математика
02-КТ-12	Экономика
02-КТ-12	Электротехника
02-КТ-12	Иностр. язык

Запрос 11

БД НИР. Вывести список сотрудников отдела 03, которые участвовали в выполнении Этапа_3.

Запрос будет выглядеть следующим образом:

```
SELECT r3.ФИО, r3.Этап
FROM rl, r3
WHERE      rl.Отдел = '03' AND
        rl.ФИО = r3.ФИО AND
        r.Этап = 'Этап_3';
```

Результат запроса:

ФИО	Этап
ПросовС.М.	Этап_3
Мехова И.И.	Этап_3

Запрос 12

Вывести группы, в которых по одной дисциплине на экзаменах получено больше одной пятерки.

Запрос будет выглядеть следующим образом:

```
SELECT s2.Группа
FROM s1, s2
WHERE s1.ФИО = s2.ФИО AND
s1.Оценка = 5
GROUP BY s2.Группа , s1.Дисциплина
HAVING count (*)> 1;
```

Результатом выполнения раздела HAVING является сгруппированная таблица, содержащая только те группы строк, для которых результат вычисления условия поиска есть TRUE.

Результат запроса:

Группа
02-КТ-21
02-КТ-12

Дадим пример запроса, где будет использован предикат NOT EXISTS. Его возможности удобно проиллюстрировать в тексте многотабличного запроса.

Запрос 13

Вывести список тех студентов, кто должен был сдавать экзамен по истории, но пока еще не сдавал,

Запрос будет выглядеть следующим образом:

```
SELECT ФИО
FROM s2,S3
WHERE s2.Группа=s3.Группа AND
Дисциплина = 'История' AND NOT EXISTS (SELECT ФИО
FROM SI
WHERE ФИО = a.ФИО AND
Дисциплина = 'История');
```

Результат запроса:

ФИО
Дрозд Г. Р.

Напомним, что предикат EXISTS истинен, когда подзапрос не пуст, то есть содержит хотя бы один кортеж, в противном случае предикат EXISTS ложен. Предикат ют EXISTS — истинен только тогда, когда подзапрос пуст.

Обработка такого запроса состоит в том, что для каждого студента, обучающегося в группе, студентам которой необходимо сдавать экзамен по истории, проверяется истинность предиката NOT EXISTS. ИСТИННОСТЬ его устанавливается по факту присутствия во внутреннем запросе значений. Если подзапрос пуст, то данный студент еще не сдавал экзамен по истории, предикат кот EXISTS имеет значение TRUE, И ФИО студента помешается в результирующую таблицу вывода.

9.2. Операторы манипулирования данными

9.2.1. Оператор ввода данных INSERT

Оператор ввода данных INSERT имеет следующий синтаксис:

INSERT INTO имя таблицы [(<список столбцов>)] VALUES (<список значений}

Подобный синтаксис позволяет ввести только одну строку в таблицу. Например, введем нового студента в таблицу s2 БД Сессия:

```
INSERT INTO s2 ( ФИО, Группа) VALUES ('Сидоров П.П.', '02-КТ-21');
```

Задание списка столбцов необязательно тогда, когда, как в данном случае, вводится строка с заданием значений всех столбцов. При таком вводе предполагается, что информация будет вводиться в том порядке, в

котором они описаны в операторе CREATE TABLE. Так как в рассматриваемом примере вводится полная строка, то можно не задавать список столбцов, ограничиться только заданием перечня значений, в этом случае оператор ввода будет выглядеть следующим образом:

```
INSERT INTO S2
VALUES ('Сидоров П.П.', '02-КТ-21');
```

Между списком имен столбцов и списком значений должно быть следующее соответствие:

- количество элементов в обоих списках должно быть одинаковым;
- между положением элементов в списках должно быть строгое соответствие, которое определяется слева направо: первый элемент одного списка соответствует первому элементу второго списка и т. д.;
- типы данных соответствующих элементов списков должны быть одинаковые и принадлежать к одному и тому же домену.

9.2.2. Оператор удаления данных DELETE

Оператор удаления данных позволяет удалить одну или несколько строк из таблицы в соответствии с условиями, которые задаются для удаляемых строк. Синтаксис оператора DELETE следующий:

```
DELETE FROM имя_таблицы
[WHERE условия_Отбора]
```

Условия отбора определяют, какие строки должны быть удалены. Если условия отбора не задаются, то из таблицы удаляются все существующие в ней строки. Однако это не означает, что удаляется вся таблица. Исходная таблица остается, но она остается пустой, незаполненной.

Например, если нам надо удалить результаты прошедшей сессии, то мы можем удалить все строки из отношения s1 следующим оператором:

```
DELETE FROM s1;
```

Условия отбора в части WHERE имеют тот же вид, что и условия фильтрации в операторе SELECT. Эти условия определяют, какие строки из исходного отношения будут удалены. Например, исключение по какой-либо причине студента Крисса Р.О. из таблицы s2 можно выполнить оператором:

```
DELETE FROM s2
WHERE ФИО = 'Крисс Р.О.';
```

9.2.3. Операция обновления данных UPDATE

Операция обновления данных UPDATE требуется тогда, когда требуется изменить содержимое базы данных. Данный оператор, также как и другие операторы обновления информации БД, применяется к одной конкретной таблице и имеет следующий формат:

UPDATE имя_таблицы

SET имя_столбца1 = новое_значение1 [,имя__стол0ца2 = новое_значение2...]

[WHERE условие_отбора]

Здесь в предложении UPDATE указывается имя обновляемой таблицы, в предложении SET указываются имена столбцов и новые данные. Новые данные должны быть совместимы с теми данными, которые они призваны заменить.

Часть WHERE является необязательной, также как и в операторе DELETE. Она играет здесь ту же роль, что и в операторе DELETE, — позволяет отобрать строки, к которым будет применена операция модификации. Если условие отбора не задается, то операция модификации будет применена ко всем строкам таблицы.

Рассмотрим операцию обновления данных таблицы базы данных НИР. Предположим, что решено все начисления специалистам увеличить на 10%. Операция обновления информации в связи с этим будет выглядеть следующим образом:

UPDATE r3

SET Начисления = Начисления * 1.1;

В том случае, когда модификацию информации необходимо производить

выборочно, требуется использование предложения WHERE. Допустим, что в БД Сессия следует произвести изменение данных, поскольку студентка Думская М.Т. пересдала экзамен по физике и получила оценку "отлично" вместо "удовлетворительно". Для решения поставленной задачи необходимо выполнить следующую операцию:

UPDATE sI SET s1.Оценка = 5

WHERE sI.ФИО = 'Думская М.Т.' AND

s1.Дисциплина = 'ФИЗИКА';

9.3. Операторы определения данных

9.3.1. Создание таблиц

Создание таблицы осуществляется посредством оператора CREATE TABLE. Его упрошенная версия выглядит следующим образом:

CREATE TABLE Имя_таблииы

(Имя_столбца Тип_данъи [NULL | NOT NULL] [,...]}

Оператор такого вида приведет к созданию таблицы с именем <Имя_таблицы>, которая будет содержать столько столбцов, сколько их задано в операторе. При определении столбца необходимо задать его имя, тип данных, к которому будут относиться значения этого столбца, а также определить, можно ли в качестве значения рассматриваемого столбца использовать ключевое слово NULL. Ключевым словом NULL помечается

такой столбец, который может содержать неопределенные значения. Определения столбцов первичных ключей отношений всегда должны содержать ключевые слова ROT NULL.

Для того чтобы создать таблицу s1 БД СЕССИЯ, необходимо использовать оператор вида

```
CREATE TABLE s1 (
        ФИО             VARCHAR (20)        NOT NULL,
        Дисциплина          VARCHAR (20)        NOT NULL,
        Оценка          SMALLINT                NOT NULL);
```

Полное описание оператора CREATE TABLE ДОЛЖНО включать средства поддержки целостности данных. Такие средства представляют собой спецификаторы, позволяющие задать ограничения для предотвращения попыток нарушить согласованность данных. Базовое определение оператора CREATE TABLE имеет следующий формат:

```
CREATE TABLE имя_таблицы
({ имя_столбца тип_даных [NOT NULL] [UNIQUE]
[DEFAULT значение по умолчанию]
[CHECK (условие проверки на допустимость) [,...] }
[PRIMARY KEY (список столбцов),]
{[UNIQUE (список столбцов),] [,...]}
{[FORING KEY {список столбцов внешних ключей)
REFERENCES имя родительской таблицы [(список столбцов ключей-кандидатов)],
[MATCH {PARTIAL | FULL}
[ON UPDATE правило ссылочной целостности]
[ON DELETE правило ссылочной целостности]] [,...]}
{[CHECK (условие проверки на допустимость)] [,...]})
```

Перепишем оператор создания таблицы s1 БД Сессия следующим образом:

```
CREATE TABLE sI (
ФИО             VARCHAB (20)        NOT NULL,
Дисциплина          VAHCHAR (20)        NOT NULL,
Оценка          SMALLINT                NOT NULL);
PRIMARY KEY (ФИО, Дисциплина),
FORING KEY ФИО REFERENCES S2
ON UPDATE CASCADE
ON DELETE CASCADE);
```

Учитывая то, что операторы языка SQL транслируются в режиме интерпретации, создавать таблицы необходимо в определенном порядке: вначале родительские, а затем дочерние. В противном случае появятся сообщения об ошибке в том случае, когда в определении дочерней

таблицы будут присутствовать ссылки на еще не существующую родительскую таблицу.

9.3.2. Обновление таблиц

В уже созданную таблицу изменения могут быть внесены с помощью оператора ALTER TABLE, который имеет следующий обобщенный формат:

ALTER TABLE имя_таблицы
[ADD [COLUMN] имя столбца тип даных [NOT NULL] [UNIQUE]
[DEFAULT значение по умолчанию] [CHECK (условие проверки на допустимость)]]
[DROP [COLUMN]] имя_столбца [RISTRICT | CASCADE]]
[ADD [CONSTRAINT [имя ограничения)] ограничение]
[DROP CONSTRAINT имя ограничения [RISTRICT I CASCADE]]
[ALTER [COLUMN] SET DEFAULT значение по умолчанию]
[ALTER (COLUMN] DROP DEFAULT]

В данном формате предусмотрены возможности для выполнения ряда действий:

- добавить новый столбец в существующую таблицу — ADD COLUMN;
- удалить столбец из существующей таблицы — DROP COLUMN;
- добавить в определение таблицы новое ограничение — ADD CONSTRAINT;
- удалить из определения таблицы существующее ограничение — DROP
- CONSTRAINT;
- задать для существующего столбца значение по умолчанию — ALTER [COLUMN] SET DEFAULT;
- отменить установленное для столбца значение по умолчанию ALTER [COLUMN] DROP DEFAULT.

Добавить в таблицу s1 столбец Группа, содержащий символьный тип данных, можно с помощью оператора:

ALTER TABLE s1
ADD Группа varchar (7) NOT NULL;

9.3.2. Удаление таблиц

Ставшая ненужной таблица может быть удалена из базы данных оператором

DROP TABLE имя таблицы [RISTRICT I CASCADE].

Ключевые слова RISTRICT И CASCADE используются для определения условий удаления таблицы в том случае, если в базе данных присутствуют ее дочерние таблицы. Ключевое слово RISTRICT при наличии в базе данных зависимых от удаляемой таблицы объектов вызовет отмену удаления. Ключевое слово CASCADE в этой ситуации вызовет автоматическое

удаление всех объектов базы данных, существование которых зависит отданной таблицы.

Удалим таблицу s1:

DROP TABLE s1;

9.3.3. Операторы создания и удаления индексов

Поскольку базы данных предназначены для хранения больших объемов информации, эффективность их использования в информационных системах во многом определяется скоростью выборки данных. Для увеличения скорости выборки в БД обычно используют специальную структуру, которая называется индексом. Стандарт языка SQL не предусматривает использование индексов. Но тем не менее разработчики СУБД охотно идут на включение средств поддержки индексов в систему, несмотря на то, что наличие индекса увеличивает нагрузку на систему из-за необходимости обновлять его при каждом изменении данных таблицы, поскольку существенное повышение скорости запросов окупает данные затраты.

Операторы создания и удаления индекса имеют следующий формат. Создать индекс:

CREATE [UNIQUE] INDEX имя_индекса

ОК имя_таблицы (столбец [ASC| DESC] [,_.])

Удалить индекс:

DROP INDEX имя_индекса

Если в операторе CREATE INDEX используется квалификатор UNIQUE, то уникальность значений индекса автоматически поддерживается системой. Для каждого из ключевых столбцов можно указать порядок следования значений: по возрастанию — ASC (используется по умолчанию) и по убыванию — DESC

10. Обеспечение функционирования баз данных

Восстановление

Восстановление в системе управления базами данных (СУБД) означает восстановление самой базы данных, т.е. возвращение БД в правильное состояние. Основной принцип, на котором строится такое восстановление – это избыточность, которая организуется на физическом уровне.

Транзакции

Транзакция – это логическая единица работы. Рассмотрим пример. Предположим, что отношение P (отношение деталей) включает атрибут TOTQTY, представляющий собой общий объем поставок для каждой детали. Значение TOTQTY для любой определенной детали предполагается равным сумме всех значений QTY для всех поставок

данной детали. На рис. 10.1. показано добавление в базу данных новой поставки со значением 1000 для поставщика S5 и детали P1.

```
BEGIN TRANSACTION;

                INSERT ({S#:'S5', P#:'P1', QTY:1000}) INTO SP;
                IF ошибка THEN GO TO UNDO;

                UPDATE P WHERE P# = 'P1' TOTQTY:=TOTQTY+1000;
                IF ошибка THEN GO TO UNDO;

                COMMIT TRANSACTION;
                GO TO FINISH;

UNDO:           ROLLBACK TRANSACTION;

FINISH:         RETURN;
```

Рис.10.1 Пример транзакции

В приведенном примере предполагается, что речь идет об атомарной операции. На самом деле добавление новой поставки – это выполнение двух обновлений в базе данных (Insert добавляет новую поставку к отношению SP, а Update обновляет значение TOTQTY для детали P1). Кроме того, в базе данных между двумя обновлениями временно нарушается требование, что значение TOTQTY для детали P1 равно сумме всех значений QTY для этой детали. Таким образом, транзакция – не просто одиночная операция системы баз данных, а скорее согласование нескольких таких операций. В общем, это преобразование одного согласованного состояния базы данных в другое, причем в промежуточных точках база данных находится в несогласованном состоянии.

Системный компонент, обеспечивающий атомарность называется администратором транзакций, а ключами к его выполнению служат операторы Commit transaction и Rollback Transaction.

- Оператор Commit transaction сигнализирует об успешном окончании транзакции и что база данных находится вновь в согласованном состоянии, а все обновления могут быть зафиксированы, т.е. стать постоянными.
- Оператор Rollback Transaction сигнализирует о неудачном окончании транзакции и что база данных находится в несогласованном состоянии, а все обновления могут быть отменены.

Система поддерживает файл регистрации (журнал регистрации), где записываются детали всех операций обновления, в частности новое и старое значения модифицируемого объекта. Таким образом, при

необходимости отмены некоторого обновления система использует соответствующий файл регистрации для возвращения объекта в первоначальное состояние.

10.1. Восстановление транзакции

Транзакция начинается с успешного выполнения оператора Begin Transaction и заканчивается успешным выполнение либо оператора Commit, либо оператора Rollback. Оператор Commit устанавливает так называемую точку фиксации, которая соответствует концу логической единице работы и , следовательно, точке, в которой база данных находится в согласованном состоянии. Выполнение же оператора Rollback возвращает базу данных в состоянии, в котором она находилась во время выполнения оператора Begin Transaction, т.е. в предыдущую точку фиксации.

Из этого следует, что транзакция – это не только логическая единица работы, но и также единица восстановления при неудачном выполнении операций.

Таким образом, транзакции обладают четырьмя важными свойствами: атомарность, согласованность, изоляция и долговечность.

Атомарность. Транзакции атомарны (выполняется все или ничего).

Согласованность. Транзакции защищают БД согласованно, т.е. транзакции переводят одно согласованное состояние БД в другое без обязательной поддержки согласованности в промежуточных точках.

Изоляция. Транзакции отделены друг от друга. Это означает, что при запуске множества конкурирующих друг с другом транзакций, любое обновление определенной транзакции будет скрыто от остальных до тех пор, пока эта транзакция выполняется.

Долговечность. Когда транзакция выполнена, ее обновления сохраняются, даже если в следующий момент произойдет сбой системы.

10.2 Восстановление системы

Система должна быть готова к восстановлению не только после небольших локальных нарушений, таких как невыполнение операции в пределах определенной транзакции, но также и после глобальных нарушений типа сбоев питания ЦПУ. Местное нарушение по определению поражает только транзакцию, в которой оно произошло. Глобальное нарушение поражает сразу все транзакции, что приводит к значительным для системы последствиям.

Существует два вида глобальных нарушений:

Отказы системы (например, сбои в питании), поражающие все выполняющиеся в данный момент транзакции, но физически не разрушающие базу данных в целом. Такие нарушения в системе также называют аварийным отказом программного обеспечения.

Отказы носителей (например, поломка головок дискового накопителя), которые могут представлять угрозу для БД или для какой либо ее части и поражать, по крайней мере, те транзакции, которые используют эту часть БД. Отказы носителей называют также аварийным отказом аппаратуры.

Критической точкой в отказе системы является потеря содержимого оперативной памяти (рабочих буферов БД). Поскольку точное состояние какой либо выполняющейся в момент нарушения транзакции не известно, транзакция может не завершиться успешно и, таким образом, будет отменена при перезагрузке системы.

Более того, возможно, потребуется повторно выполнить определенную успешно завершившуюся до аварийного отказа транзакцию при перезагрузке системы, если не были физически выполнены обновления этой транзакции.

Решение вопроса о том, какую транзакцию следует отменить, а какую выполнить повторно, при перезагрузке системы заключается в следующем. В некотором предписанном интервале (когда в журнале накапливается определенное число записей) система автоматически принимает контрольную точку. Принятие контрольной точки включает физическую запись содержимого рабочих буферов БД непосредственно в БД и специальную физическую запись контрольной точки, которая предоставляет список всех осуществляемых в данный момент транзакций. На рис.10.2. рассматривается пять возможных вариантов выполнения транзакций до аварийного сбоя системы.

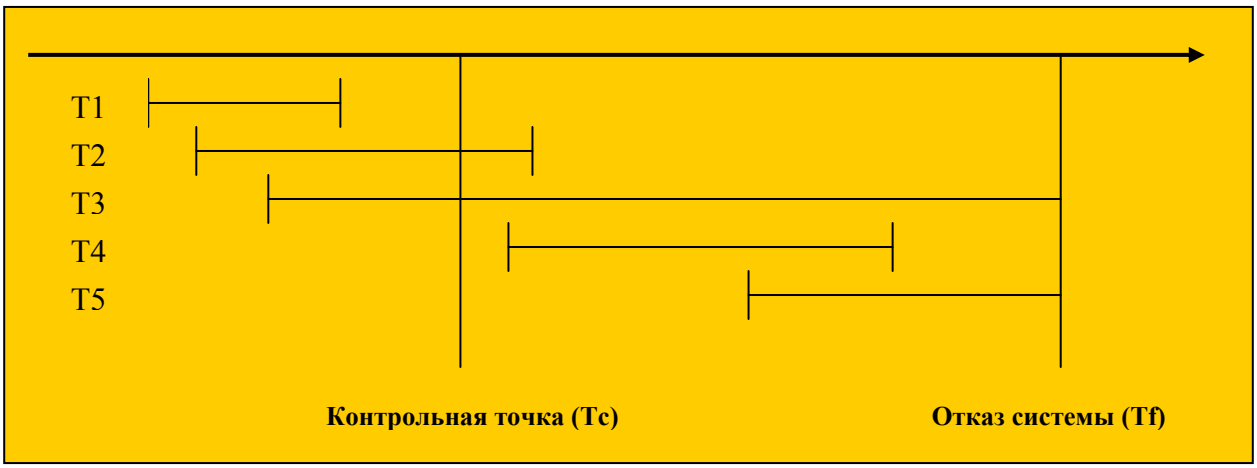


Рис. 10.2. Варианты транзакции

Очевидно, что при перезагрузке системы транзакции типа T3 и T5 должны быть отменены, а транзакции типа T2 и T4 – выполнены повторно, а транзакции типа T1 вообще не включаются в процесс перезагрузки, так как обновления попали в БД еще до момента времени Tc, т.е. зафиксированы еще до принятия контрольной точки.

Следовательно, во время перезагрузки система вначале проходит через процедуру идентификации всех транзакций типа T2-T5. при этом выполняются перечисленные ниже шаги.

1. Создаются два списка транзакций: будем их называть UNDO (отменить) и REDO (повторить). В список UNDO заносятся все транзакции, предоставленные из записи контрольной точки (т.е. все транзакции, выполняющиеся в момент времени Tc), а список REDO остается пустым.
2. Осуществляется поиск в файле регистрации (журнале), начиная с записи контрольной точки.
3. Если в файле регистрации обнаружена запись Begin Transaction о начале транзакции T, то эта транзакция также добавляется в список UNDO.
4. Если в файле регистрации обнаружена запись Commit об окончании транзакции T, то эта транзакция добавляется в список REDO.
5. Когда достигается конец файла регистрации, списки и REDO анализируются для идентификации транзакций типа T2 и T4, появившихся в списке REDO и транзакций типа T3 и T5, оставшихся в списке UNDO.

После этого система просматривает файл регистрации, отменяя транзакции из списка UNDO, а затем просматривает снова вперед, повторно выполняя транзакции из списка REDO.

10.3. Восстановление носителей

При отказе носителей некоторая часть БД разрушается физически. Восстановление после такого нарушения включает перезапись (или восстановление) БД с резервной копии (или дампа) и последующее использование файла регистрации – как его активной, так и архивной части. Такое восстановление – это повторное выполнение всех транзакций, вызванное применением резервной копии. При этом нет необходимости отмены транзакций, которые выполнялись в момент отказа носителей, поскольку по определению все обновления таких транзакций полностью утеряны.

Таким образом, восстановление подразумевает наличие утилиты дамп/восстановление. Дамп-часть этой утилиты используется для сохранения резервной копии. Такие копии могут сохраняться либо на ленте, либо другим архивным способом. После отказа носителей восстановительная часть утилиты используется для создания БД со специализированной резервной копии.

10.4. Параллелизм

Параллелизм означает возможность одновременной обработки в СУБД многих транзакций с доступом к одним и тем же данным, причем в

одно и то же время. В такой системе для корректной обработки параллельных транзакций без возникновения конфликтных ситуаций необходимо использовать некоторый метод управления параллелизмом.

Три проблемы параллелизма

Прежде всего следует уточнить, что каждый метод управления параллелизмом предназначен для решения некоторой конкретной задачи. Тем не менее, при обработке правильно составленных транзакций возникают ситуации, которые могут привести к получению неправильного результата из-за взаимных помех среди некоторых транзакций. Основные проблемы, возникающие при параллельной обработке транзакций следующие:

потеря результатов обновления;
незафиксированная зависимость;
несовместимый анализ.

Проблема потери результатов обновления

Рассмотрим ситуацию, показанную на рис.10.3.

Транзакция A	Время	Транзакция B
Извлечение кортежа **p**	**t1**	
	t2	Извлечение кортежа **p**
Обновление кортежа **p**	**t3**	
	t4	Обновление кортежа **p**

Рис.10.3. Потеря в момент времени t4 результатов обновления, выполненного транзакцией A

Результат операции обновления, выполненной транзакцией A, будет утерян, поскольку в момент времени t4 она не будет учтена и потому будет «отменена» операцией обновления, выполненной транзакцией B.

Проблема незафиксированной зависимости

Проблема незафиксированной зависимости появляется, если с помощью некоторой транзакции осуществляется извлечение (или, что еще хуже, обновление) некоторого кортежа, который в данный момент обновляется другой транзакцией, но это обновление еще не закончено. Таким образом, если обновление не завершено, существует некоторая вероятность того, что оно не будет завершено никогда. В таком случае в первой транзакции будут принимать участие данные, которых больше не существует (в том смысле, что они «никогда» не существовали). Эта ситуация показана на рис.10.4 и рис.10.5.

Транзакция A	Время	Транзакция B
	t1	Обновление кортежа **p**
Извлечение кортежа **p**	**t2**	
	t3	Отмена выполнения транзакции

Рис.10.4. Транзакция A становится зависимой от невыполненного изменения в момент времени t2

Транзакция A	Время	Транзакция B
	t1	Обновление кортежа **p**
Обновление кортежа **p**	**t2**	
	t3	Отмена выполнения транзакции

Рис.5. Транзакция A обновляет невыполненное изменение в момент времени t2, и результаты этого обновления утрачиваются в момент времени t3

Проблема несовместимого анализа

На рис.6. показаны транзакции A и B, которые выполняются для кортежей со счетами. При этом транзакция A суммирует балансы, транзакция B производит перевод суммы 10 со счета 3 на счет 1. полученный в итоге транзакции A результат 110, очевидно, неверен, и если он будет записан в базе данных, то в ней может возникнуть проблема несовместимость. В таком случае говорят, что транзакция A встретилась с несовместимым состоянием и на его основе был выполнен несовместимый анализ.

Счет 1 – 40, Счет2 – 50, Счет3 - 30		
Транзакция A	Время	Транзакция B
Извлечение кортежа **Счет 1 Sum=40**	**t1**	
Извлечение кортежа **Счет 2 Sum=90**	**t2**	
	t3	Извлечение кортежа **Счет 3**
	t4	Обновление кортежа **Счет 3 30 → 20**
	t5	Извлечение кортежа **Счет 1**
	t6	Обновление кортежа **Счет 1 40 → 50**
	t7	Завершение транзакции
Извлечение кортежа **Счет 3 Sum = 110 (а не 120)**	**t8**	

Рис.10.6. Транзакция A выполнила несовместимый анализ

10.5. Блокировка

Описанные выше проблемы могут быть разрешены с помощью методики управления параллельным выполнением процессов под названием блокировка. Ее основная идея проста: в случае, когда для

выполнения некоторой транзакции необходимо чтобы некоторый объект (обычно это кортеж базы данных) не изменялся непредсказуемо и без ведома этой транзакции, такой объект блокируется.

Более подробно функционирование блокировки заключается в следующем:

1. Прежде всего, предположим, что в системе поддерживается два типа блокировок: блокировка без взаимного доступа (монопольная блокировка), называемая X-блокировкой, и блокировка с взаимным доступом, называемая S-блокировкой.
2. Если транзакция A блокирует кортеж p без возможности взаимного доступа (X-блокировка), то запрос другой транзакции B с блокировкой этого кортежа p будет отменен.
3. Если транзакция A блокирует кортеж p с возможностью взаимного доступа (S-блокировка), то
   - запрос со стороны некоторой транзакции B на X-блокировку кортежа будет отвергнут;
   - запрос со стороны некоторой транзакции B на S-блокировку кортежа будет принят (т.е. транзакция B также будет блокировать кортеж p с помощью S-блокировки).

Теперь следует ввести протокол доступа к данным, который на основе X и S-блокировок позволяет избежать возникновения проблем параллелизма, описанных выше.

1. Транзакция, предназначенная для извлечения кортежа, прежде всего должна наложить S-блокировку на этот кортеж.
2. Транзакция, предназначенная для обновления кортежа, прежде всего должна наложить X-блокировку на этот кортеж. Иначе говоря, если, например, для последовательности действий типа извлечение/обновление для кортежа уже задана S-блокировка, то ее необходимо заменить X-блокировкой.
3. Если запрашиваемая блокировка со стороны транзакции B отвергается из-за конфликта с некоторой другой блокировкой со стороны транзакции A, то транзакция B переходит в состояние ожидания. Причем транзакция B будет находится в состоянии ожидания до тех пор, пока не будет снята блокировка, заданная транзакцией A.
4. X-блокировки сохраняются вплоть до конца выполнения транзакции (до операции «завершение выполнения»). S –блокировки также обычно сохраняются вплоть до этого момента.

10.6. Решение проблем параллелизма

Рассмотрим описанные выше проблемы, возникающие при параллельной обработке кортежей.

Проблема потери результатов обновления

На рис. 10.7. приведена измененная версия процесса, показанного на рис.3., с учетом применения протокола блокировки для чередующихся операций.

Транзакция A	Время	Транзакция B
Извлечение кортежа **p** (задание S-блокировки)	**t1**	
	t2	Извлечение кортежа **p** (задание S-блокировки)
Обновление кортежа **p** (задание X - блокировки)	**t3**	
ожидание	**t4**	Обновление кортежа **p** (задание X-блокировки)
ожидание		ожидание

Рис.10.7. Хотя обновления не утрачиваются, но в момент времени t4 возникает тупиковая ситуация

Проблема незафиксированной зависимости

На рис.10.8 и рис.10.9 приведены в измененном виде примеры, показанные ранее на рис.10.4 и рис.10.5 соответственно. Они демонстрируют чередующееся выполнение операций согласно описанному выше протоколу блокировки.

Транзакция A	Время	Транзакция B
	t1	Обновление кортежа **p** (задание X-блокировки)
Извлечение кортежа **p** (задание S-блокировки)	**t2**	
ожидание	**t3**	Окончание или отмена выполнения (снятие X-блокировки)
Итог: извлечение кортежа p (задание S-блокировки)		

Рис.10.8. Транзакция A предохраняется от выполнения операций с незафиксированным изменением в момент времени t2

Транзакция A	Время	Транзакция B
	t1	Обновление кортежа **p** (задание X-блокировки)
Обновление кортежа **p** (задание X-блокировки)	**t2**	
ожидание	**t3**	Окончание или отмена выполнения (снятие X-блокировки)
Итог: обновление кортежа p (задание X-блокировки)		

Рис.10.9. Транзакция A предохраняется от выполнения операций с незафиксированным изменением в момент времени t2

Проблема несовместимого анализа

На рис.10.10 приведена измененная версия рис.10.6 с перечислением чередующихся транзакций согласно протоколу блокировки.

Счет 1 – 40, Счет2 – 50, Счет3 - 30		
Транзакция A	Время	Транзакция B
Извлечение кортежа **Счет 1** (задание S-блокировки для кортежа Счет 1) **Sum=40**	**t1**	
Извлечение кортежа **Счет 2** **Sum=90**	**t2**	
	t3	Извлечение кортежа **Счет 3** (задание S-блокировки для кортежа Счет 3)
	t4	Обновление кортежа **Счет 3** (задание X-блокировки для кортежа Счет 3) **30 → 20**
	t5	Извлечение кортежа **Счет 1** (задание S-блокировки для кортежа Счет 1)
	t6	Обновление кортежа **Счет 1** (задание X-блокировки для кортежа Счет 1) **40 → 50**
Извлечение кортежа **Счет 3** (задание S-блокировки для кортежа Счет 3)	**t7**	ожидание
ожидание		ожидание

Рис.10. Проблема несовместимого анализа разрешается, но в момент времени t7 возникает тупиковая ситуация

10.7. Тупиковая ситуация

Как было показано выше, блокировку можно использовать для разрешения трех основных проблем, возникающих при параллельной обработке кортежей. К сожалению, использование блокировок приводит к возникновению другой проблемы – тупиковой ситуации. Два примера таких ситуаций были приведены выше.

Тупиковая ситуация возникает тогда, когда две или более транзакции одновременно находятся в состоянии ожидания, причем для продолжения работы каждая из транзакция ожидает прекращения выполнения другой транзакции.

Желательно, чтобы при возникновении тупиковой ситуации система могла обнаружить ее и найти из нее выход. Для обнаружения тупиковой

ситуации следует обнаружить цикл в диаграмме состояний ожидания, т.е. в перечне «транзакций, которые ожидают окончания выполнения других транзакций». Поиск выхода из тупиковой ситуации состоит в выборе одной из заблокированных транзакций в качестве жертвы и отмене ее выполнения. Таким образом, с нее снимается блокировка, а выполнение другой транзакции может быть возобновлено.