ГОУ «ПРИДНЕСТРОВСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ им. Т. Г. ШЕВЧЕНКО»
**Физико-технический институт**
Факультет среднего профессионального образования
(технический колледж им. Ю. А. Гагарина)
*Кафедра интегрированных компьютерных
технологий и систем*

# БАЗЫ ДАННЫХ

***Учебное пособие***

Тирасполь
Издательство Приднестровского Университета
2025

УДК 004.65(075.32)
ББК А635я723
Б17

*Составители:*
**О. М. Фурдуй**, доц.
**О. В. Комарова**, ст. преп.

***Рецензенты:***

*С. В. Уманец*, заместитель начальника отдела администрирования сетей и баз данных открытого акционерного общества «Эксимбанк»

*С. В. Федорченко*, канд. техн. наук каф. программное обеспечение вычислительной техники ФТИ (Приднестровский государственный университет им. Т. Г. Шевченко)

Б17 **Базы данных**: учебное пособие [Электронный ресурс] / ГОУ «Приднестровский государственный университет им. Т. Г. Шевченко» ; Физико-технический институт ; составители : О. М. Фурдуй, О. В. Комарова. – Тирасполь : Изд-во Приднестр. ун-та, 2025. – 132 с.

Минимальные системные требования: CPU (Intel/AMD) 1,5ГГц/ОЗУ 2ГГб/HDD 450Мб/1024*768/Windows 7 и старше/Internet Explorer 11/Adobe Acrobat Reader 6 и старше

*Подготовлено в соответствии с учебным планом специальности 2. 09.02.01 «Компьютерные системы и комплексы». Учебное пособие отличается от известных учебников рациональной компоновкой учебного материала, содержит теоретический материал для более глубокого усвоения новых понятий, в дальнейшем для выполнения лабораторных работ по одноименному предмету. Каждый раздел пособия содержит вопросы для самопроверки.*

*Адресовано обучающимся ФСПО (Технического колледжа им. Ю. А. Гагарина) по специальности 2.09.02.01 «Компьютерные системы и комплексы».*

**УДК 004.65(075.32)**
**ББК А635я723**

Рекомендовано Научно-методическим советом ПГУ им. Т. Г. Шевченко

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

В современном мире, где информация становится одной из самых ценных ресурсов, понимание основ работы с базами данных является ключевым навыком для специалистов в различных областях – от разработки программного обеспечения и аналитики до управления бизнесом и научных исследований.

На сегодняшнем этапе технологического прогресса базы данных играют центральную роль в процессе хранения, организации и обработки информации. Они позволяют нам эффективно управлять большими объемами данных, обеспечивают быструю выборку необходимой информации и поддерживают целостность и безопасность данных.

Изучение баз данных является крайне важным аспектом для специалистов в различных областях, включая информационные технологии, аналитику, бизнес и науку. Вот несколько причин, почему эта тема имеет большое значение:

1. Рост объема данных. В современном мире объем генерируемых данных постоянно увеличивается. Базы данных позволяют эффективно хранить, управлять и обрабатывать эти данные.

2. Навыки для карьерного роста. Знание баз данных является одним из ключевых навыков для специалистов в области ИТ, аналитики данных, разработки программного обеспечения и многих других профессий. Это открывает множество карьерных возможностей.

3. Анализ данных. Базы данных являются основой для работы с большими объемами информации. Умение извлекать и анализировать данные позволяет принимать обоснованные решения, что критично в бизнесе и научных исследованиях.

4. Системы управления данными. Знание систем управления базами данных (СУБД), таких как MySQL, PostgreSQL, Oracle или MongoDB, помогает в разработке, администрировании и оптимизации баз данных, что делает процессы хранения и поиска информации более эффективными.

5. Создание приложений. Многие современные приложения требуют работы с базами данных для хранения пользовательских данных, настроек и другой информации. Разработчики, умеющие работать с базами данных, имеют явное преимущество.

6. Интеграция технологий. Понимание принципов работы с базами данных облегчает интеграцию различных технологий и систем, таких как облачные платформы, аналитические инструменты и приложения для автоматизации бизнес-процессов.

7. Безопасность данных. Знание основ работы с базами данных помогает в обеспечении безопасности данных. Понимание уязвимостей и методов защиты информации крайне важно в свете растущих угроз кибербезопасности.

Всё это делает изучение баз данных не только актуальным, но и необходимым в современном мире, где информация играет ключевую роль в успехе бизнеса и научных исследований.

Курс лекций посвящен изучению данных, таких как реляционные, объектно-ориентированные и NoSQL, знакомству с языком SQL, который является стандартом для работы с реляционными базами данных, а также рассмотрена техника проектирования баз данных, включая нормализацию и оптимизацию.

Изучение баз данных начнется с изучения ключевых концепций и архитектуры баз данных, после чего углубление в практическое применение и современный инструментарий. Этот курс направлен не только на обогащение знаний, но и на дальнейшее изучение важной и увлекательной области – проектирование баз данных.

# Глава 1. ОСНОВНЫЕ ПОНЯТИЯ БАЗ ДАННЫХ

## 1.1. Документальные и фактографические информационные системы

Автоматизированные информационные системы (АИС) начали активно развиваться в 1960-х годах, когда возникла необходимость обработки больших объемов накопленной информации в различных сферах, таких как военная промышленность и бизнес. Эти системы представляют собой программно-аппаратные комплексы, предназначенные для хранения, обработки и предоставления информации пользователям. Первые АИС работали преимущественно с фактографическими данными, такими как характеристики объектов и их взаимосвязи. Однако со временем, благодаря усложнению и развитию технологий, они получили возможность обрабатывать текстовые документы, изображения и другие типы данных.

Несмотря на схожие принципы хранения информации, методы обработки данных в различных системах существенно различаются. В зависимости от вида информации, с которой работает система, принято выделять два основных класса: документальные и фактографические системы.

Документальные информационные системы ориентированы на работу с текстовыми документами на естественном языке, такими как научные монографии, газетные статьи, пресс-релизы, законодательные акты и другие источники. Эти системы проводят анализ содержания документов, используя методы, позволяющие выявлять смысл даже при его неполном представлении. Наиболее распространенным видом документальных систем являются информационно-поисковые системы (ИПС), которые позволяют хранить и находить документы по различным критериям.

Фактографические системы, в свою очередь, оперируют структурированными фактами, представленными в виде формализованных записей. В их основе лежат системы управления базами данных (СУБД), обеспечивающие хранение, обработку и структурирование информации. Фактографические системы не только предоставляют справочную информацию, но и позволяют выполнять задачи обработки данных, такие как ввод, сортировка, фильтрация и группировка записей. В результате пользователи могут получать итоговые отчеты, представленные, например, в табличной форме.

Фактографические системы нашли широкое применение в различных сферах деятельности. Они используются для учета товаров в магазинах и на складах, расчета заработной платы, управления финансами и множества других задач. Современное предприятие или учреждение сложно представить без автоматизированных информационных систем, которые составляют основу информационной деятельности в самых разных областях – от промышленности и телекоммуникаций до личных финансов.

Для эффективного хранения и обработки данных в АИС необходимо грамотно организовать их структуру, обеспечивая целостность и непротиворечивость информации. Обычные файловые системы не способны обеспечить достаточную производительность при работе с большими объемами данных, поэтому основой автоматизированных информационных систем являются СУБД – системы управления базами данных, обеспечивающие эффективное хранение и обработку информации.

## 1.2. Системы управления базами данных

Любая автоматизированная информационная система (АИС) работает с определенной частью реального мира, называемой **предметной областью**. Под этим словом понимается совокупность реальных объектов (сущностей) и связей между ними. Каждый объект обладает определенными характеристиками, называемыми **атрибутами**.

Например, в системе автоматизации бухгалтерии предприятия необходимо хранить информацию о сотрудниках. В этом случае сущностями будут служащие, а их атрибутами – личный номер, имя, должность, стаж работы и другие данные. Каждое из этих свойств принимает конкретное значение: например, атрибут «стаж работы» может быть равен «10», что означает, что сотрудник проработал в организации 10 лет. Предметная область может включать не только физические объекты, но и процессы или абстрактные сущности.

Кроме характеристик, объекты предметной области могут быть связаны между собой. Так, сотрудник организации обязательно относится к какому-либо отделу, поэтому в описании его данных может присутствовать атрибут «отдел».

В автоматизированных системах предметная область представлена в виде **структурированной совокупности данных**, хранящейся в памяти компьютера. Это называется **базой данных (БД)**, которая отражает состав объектов, их свойства и взаимосвязи.

Для создания, обновления и управления базами данных используется специальное программное обеспечение – **системы управления базами**

**данных (СУБД)**, также известные как **DBMS (DataBase Management Systems)**.

Первые СУБД появились в конце 1960-х – начале 1970-х годов и были ориентированы на работу с мэйнфреймами, которые в то время доминировали в сфере вычислений. Однако ранние системы имели множество ограничений и недостатков. Несмотря на это, некоторые из них до сих пор используются. Со временем технологии совершенствовались, разрабатывались новые подходы к хранению и обработке данных, а также к созданию и управлению базами данных.

Сегодня СУБД – это мощные инструменты, которые позволяют эффективно управлять информацией и находят применение во множестве сфер, от бизнеса и производства до науки и государственного управления.

## 1.3. Модели данных

Способ организации сущностей, их атрибутов и взаимосвязей в структуре базы данных определяется используемой **моделью данных**. Выделяют несколько основных моделей: **иерархическую, сетевую, реляционную, объектно-ориентированную**, а также модель, основанную на **инвертированных списках**. Соответственно, в зависимости от используемой модели, различают и **разные типы систем управления базами данных (СУБД)**: иерархические, сетевые, реляционные, объектно-ориентированные и СУБД, основанные на инвертированных списках.

**Иерархическая модель**. В данной модели база данных имеет **древовидную структуру**, где объекты связаны **по принципу иерархии**. Вершина дерева представляет собой **корневой узел**, от которого отходят связи к подчиненным элементам. При этом каждый узел, кроме корневого, имеет **только одного родителя** (рис. 1). Например, на схеме объект **«Организация»** является родительским по отношению к объектам **«Отделы»** и **«Филиалы»**.

Основное преимущество такой структуры – **простота отражения иерархических отношений**, что делает ее удобной для представления **организационных структур**, классификаций и других подобных объектов реального мира.

**Сетевая модель.** Если в структуре данных допускается, что один элемент может быть связан **с несколькими другими элементами на разных уровнях**, то такая модель называется **сетевой**. В отличие от иерархической модели, здесь **нет жесткого ограничения** в виде единственного родительского узла – один объект может иметь **несколько связей** с другими элементами (рис. 2).

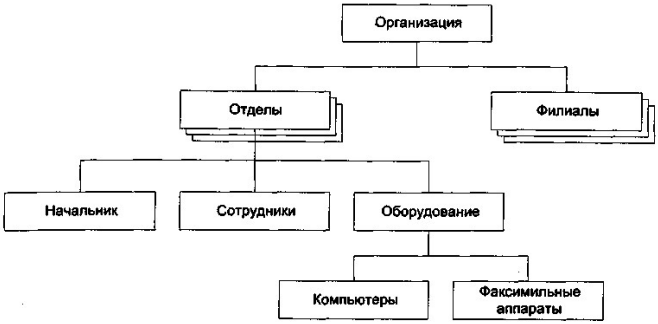


Рис. 1. *Пример иерархической древовидной структуры БД*

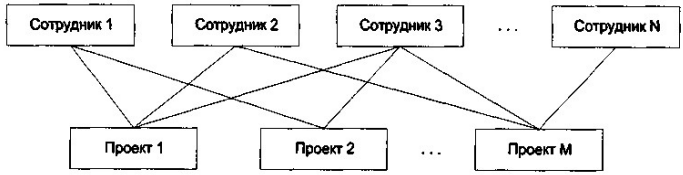


Рис. 2. *Пример сетевой структуры*

Сетевая база данных представляет собой **набор записей**, отражающих объекты предметной области, и **набор связей** между ними. Например, информация об участии сотрудников в различных проектах может быть удобно представлена в сетевой БД. В данном случае сетевая модель демонстрирует, что **один сотрудник может участвовать в нескольких проектах, а в одном проекте может работать несколько сотрудников**.

**Реляционная модель.** В **реляционных базах данных** информация представляется в **табличном виде**. Данная модель была предложена **Эдгаром Коддом** в начале 1970-х годов. За свои разработки в области реляционных баз данных и реляционной алгебры Кодд в 1981 году был удостоен **премии Тьюринга** от Американской ассоциации вычислительной техники.

Реляционная модель данных стала **новым этапом в развитии СУБД**, поскольку она отличается **простотой, гибкостью и универсальностью**. Несмотря на некоторые ограничения, она быстро завоевала популярность и в конечном итоге **стала промышленным стандартом** в области баз данных «**де-факто**».

Реляционная модель баз данных основана на принципах **реляционной алгебры** и включает в себя такие ключевые понятия, как **таблица, отно-**

**шение, строка, столбец и первичный ключ**. Вся работа с реляционной базой данных сводится к операциям над таблицами.

**Структура таблиц**. Каждая таблица представляет собой **совокупность строк и столбцов** и имеет **уникальное имя** в пределах базы данных. Таблица описывает определенный **тип объектов реального мира** (сущность), а ее **строки (кортежи)** хранят информацию о конкретных экземплярах этой сущности (рис. 3).

Например, таблица «**Сотрудники отдела**» содержит сведения обо всех работниках определенного подразделения, при этом **каждая строка** таблицы представляет **отдельного сотрудника** и его характеристики. Значения атрибутов (столбцов) таблицы берутся из определенного **домена (domain)**, который представляет собой множество допустимых значений для конкретного параметра объекта.

Каждый **столбец таблицы имеет уникальное имя**, заданное при ее создании, и располагается в **фиксированном порядке**. Однако **строки не имеют имен**, их порядок не определен и может меняться. Кроме того, таблица может содержать **любое количество строк**, поскольку их число не ограничено логически. Так как строки хранятся **в произвольном порядке**, их нельзя идентифицировать по позиции – в реляционной модели **нет понятий «первая» или «последняя» строка**.

**Ключи и связи между таблицами**. В любой таблице должен быть хотя бы **один столбец**, значения которого **однозначно идентифицируют каждую строку**. Такой столбец (или комбинация нескольких столбцов) называется **первичным ключом**. Например, в таблице «**Сотрудники отдела**» первичным ключом может быть столбец «**Номер пропуска**», поскольку он уникален для каждого работника. В реляционной модели **не допускается наличие строк с одинаковыми значениями первичного ключа**. Если таблица удовлетворяет этому требованию, она называется **отношением**.

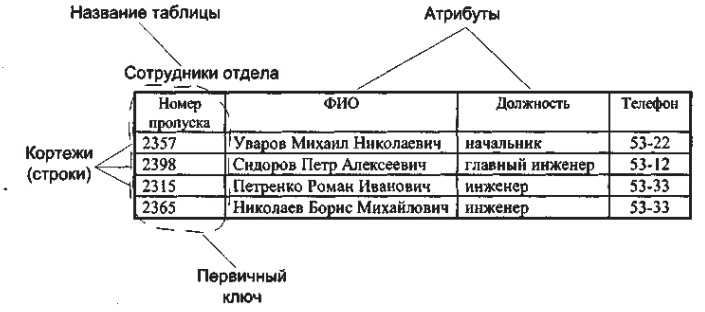


Рис. 3. *Отношение реляционной БД*

Связи между таблицами реализуются с помощью **внешних ключей**. **Внешний ключ** – это столбец, значения которого соответствуют **значениям первичного ключа** в другой таблице. Таким образом, **таблица, содержащая внешний ключ, ссылается на таблицу, где этот атрибут является первичным ключом**.

**Метаданные и дополнительные объекты**. Для работы базы данных необходимо хранить «**данные о данных**», или **метаданные**. Они включают в себя **описания таблиц, столбцов, связей и других параметров**, необходимых для управления структурой данных. Метаданные представлены в **табличной форме** и хранятся в специальном **словаре данных**.

Помимо таблиц, в базе данных могут находиться и другие объекты, такие как **экранные формы, шаблоны отчетов и прикладные программы**, которые обеспечивают обработку и отображение информации.

Для пользователей информационной системы важно, чтобы база данных точно и непротиворечиво отражала предметную область. Если база данных обладает этими свойствами, то говорят, что она **соответствует условиям целостности**. Для обеспечения целостности вводятся специальные **ограничения**, называемые **ограничениями целостности**. Основные из них:

1. **Целостность сущностей** – каждая строка (кортеж) в таблице должна быть уникальной. Это означает, что у каждой таблицы должен быть **первичный ключ**, который однозначно идентифицирует каждую запись. В реляционных базах данных это свойство **автоматически выполняется**, если не нарушены основные принципы реляционной модели.
2. **Целостность ссылок** – значения внешнего ключа должны **обязательно ссылаться** на существующие записи в другой таблице. Это предотвращает появление ссылок на несуществующие данные и помогает поддерживать **логическую связность** между таблицами.

**Преимущества реляционной модели.** Популярность реляционной модели в современных системах управления базами данных объясняется рядом факторов:

1. **Развитая теоретическая основа** – реляционная модель опирается на **глубокие математические исследования**, что делает ее более обоснованной по сравнению с другими моделями.
2. **Гибкость в представлении данных** – существующие **модели данных могут быть преобразованы в реляционную форму**, что расширяет ее возможности.
3. **Высокая скорость доступа** – реляционные СУБД включают **оптимизированные механизмы поиска**, обеспечивающие быстрый доступ к информации.

4. **Абстракция от физической структуры хранения** – пользователи могут работать с данными, **не зная** технических деталей их размещения на носителях.

5. **Стандартизированные языки запросов** – реляционная модель поддерживает **унифицированные языки** работы с данными, такие как **SQL**, что делает работу с базами данных удобной и предсказуемой.

**Инвертированные списки.** Базы данных, организованные на основе **инвертированных списков**, имеют схожесть с реляционной моделью, но обладают рядом особенностей:

1. **Доступность индексов и таблиц** – пользователи могут **напрямую обращаться** к хранимым таблицам и индексам.

2. **Физический порядок хранения** – строки таблиц **располагаются в определенной последовательности**, установленной самой системой.

Объектно-ориентированные СУБД (ООСУБД) развивались параллельно с **объектно-ориентированными языками программирования**. Их основная идея заключается в том, что пользователи работают **не с низкоуровневыми структурами данных** (такими как **записи, поля, байты**), а с **объектами и операциями**, которые описывают реальные сущности.

В отличие от других типов баз данных, **объектно-ориентированные СУБД лучше сохраняют семантику предметной области**, поскольку предоставляют **высокий уровень абстракции**. Это делает их удобными для работы с **сложными данными и моделями**, где важна логическая целостность объектов и их взаимосвязей.

# Глава 2. БАНКИ ДАННЫХ

**Банк данных** (БнД) представляет собой одну из разновидностей информационных систем. Под этим термином подразумевается система, включающая специально организованные базы данных, а также программные, технические, языковые и организационно-методические средства, обеспечивающие централизованное накопление и многопользовательское многоцелевое использование данных.

Основные характеристики БнД:

1. Банк данных создается для решения не одной, а нескольких взаимосвязанных задач, при этом им пользуется не один человек, а целая группа пользователей.

2. В составе БнД предусмотрены специальные инструменты, облегчающие работу с данными (например, системы управления базами данных – СУБД).

**Централизованное управление данными**, по сравнению с традиционной файловой системой, обладает рядом преимуществ:

1. Уменьшает избыточность хранения информации.

2. Снижает трудозатраты на разработку, эксплуатацию и модернизацию информационных систем.

3. Обеспечивает удобный доступ к данным как для специалистов в области обработки информации, так и для конечных пользователей.

Основные требования к БнД:

- соответствие модели предметной области (полнота, целостность, непротиворечивость данных, актуальность информации);
- возможность работы пользователей разных категорий, высокая скорость доступа к данным;
- удобство интерфейса и минимальные затраты на обучение;
- обеспечение конфиденциальности и разграничение доступа к данным в зависимости от прав пользователей;
- надежность хранения и защита данных.

Центральным элементом БнД является база данных (БД). БД – это именованный набор взаимосвязанных данных, находящихся под управлением СУБД.

**Метаинформация** БнД включает:

• описание структуры базы данных (схемы и подсхемы);
• модель предметной области;
• сведения о пользователях и их правах доступа;
• описание входных и выходных документов.

Централизованное хранилище метаинформации называется словарем данных. В системах автоматизированного проектирования ИС словари данных играют особенно важную роль.

Программные компоненты СУБД подразделяются на:

• ядро СУБД, отвечающее за ввод, вывод, обработку и хранение данных в базе;

• трансляторы, выполняющие преобразование языка СУБД во внутренний формат, понятный ядру;

• утилиты, предназначенные для настройки системы, отладки программ, архивации и восстановления данных, сбора статистики;

• прикладные программы, обеспечивающие обработку запросов к базе данных.

**Операционная система** иногда рассматривается как часть банка данных, поскольку в процессе работы СУБД активно взаимодействует с ОС.

Языковые средства обеспечивают связь пользователей с базой данных. Они включают инструменты для спецификации данных, создания отчетов, экранных форм, формирования запросов, а также процедурные механизмы для описания последовательности выполнения задач. Язык СУБД может представлять собой универсальный язык программирования, дополненный специализированным подъязыком для работы с базами данных. Например, в языках программирования общего назначения, таких как DELPHI, Visual Basic 5, Visual C++, встроена поддержка SQL. В то же время существуют СУБД с собственными специализированными языками, например, dBASE, FoxPro, Clipper, Paradox, Access. Некоторые СУБД используют исключительно SQL.

Технические средства включают:

• универсальный компьютер;
• устройства для ввода и вывода информации;
• инструменты для работы в сети.

Организационно-методические средства:

• инструкции;
• методические и регламентные материалы для пользователей.

Персонал включает специалистов, ответственных за создание, эксплуатацию и развитие банка данных.

## 2.1. Пользователи банка данных. Основные задачи персонала банка данных

Как и любой программно-организационно-технический комплекс, банк данных существует во времени и пространстве, проходя определенные этапы развития:

1. Проектирование.
2. Реализация.
3. Эксплуатация.
4. Модернизация и развитие.
5. Полная реорганизация.

На каждом этапе функционирования банка данных задействованы различные группы пользователей.

Пользователи БнД делятся на три основные категории:

• **Первая группа** – конечные пользователи. Они взаимодействуют с системой баз данных напрямую через рабочие станции или терминалы, используя оперативные приложения или встроенные интерфейсы ПО СУБД. Доступ может осуществляться через SQL-запросы, а также через интерфейсы на основе меню и форм. Это основная категория пользователей, для которых и создается банк данных. Среди них выделяют случайных и регулярных пользователей. Основной принцип – конечные пользователи не должны обладать специальными знаниями в области вычислительной техники.

• **Вторая группа** – администраторы банка данных. Эти специалисты на этапе проектирования отвечают за оптимальную организацию системы, учитывая многопользовательский режим работы. В процессе эксплуатации они следят за корректностью функционирования банка данных, а при модернизации и реорганизации обеспечивают возможность внесения изменений без сбоев в текущей работе.

**Администратор данных** (АД) – отвечает за стратегические решения, касающиеся данных предприятия.

**Администратор банка данных** (АБД) – отвечает за техническую поддержку и общее управление системой.

• **Третья группа** – разработчики и администраторы приложений (прикладные программисты). Они создают программные продукты, использующие базу данных. Эти программы могут быть как пакетными обработчиками, так и оперативными приложениями, обеспечивающими работу конечных пользователей.

**Администраторы приложений** (АП) координируют работу разработчиков при создании конкретных приложений или их групп, объединенных в функциональную подсистему.

Не во всех банках данных присутствуют все категории пользователей. При создании информационных систем с персональными СУБД разработчик, администратор БнД и администратор приложений могут совмещать свои функции. Однако в крупных корпоративных базах данных, автоматизирующих основные бизнес-процессы предприятия, обычно существуют отдельные группы администраторов приложений и разработчиков. Наибольшая нагрузка и ответственность ложится на группу администраторов БнД.

В состав группы администраторов банка данных (АБД) должны входить:

- системные аналитики, анализирующие требования к данным и структуре системы;
- специалисты по проектированию структур данных и организации внешней информационной поддержки;
- разработчики технологических процессов обработки информации;
- системные и прикладные программисты, занимающиеся разработкой программного обеспечения;
- операторы и технические специалисты, отвечающие за поддержку и обслуживание системы;
- маркетологи (если банк данных носит коммерческий характер).

Персонал банка данных включает широкий круг специалистов: администраторов БнД, аналитиков, программистов, операторов, технических экспертов, специалистов по маркетингу и др.

Основные задачи, выполняемые персоналом на этапах разработки и эксплуатации БнД:

1. Изучение предметной области: выявление потребностей конечных пользователей, построение информационной модели, определение ограничений целостности данных.
2. Разработка структуры базы данных: определение файловой организации, описание схемы с использованием языка спецификации данных.
3. Определение и настройка ограничений целостности, обеспечивающих корректность данных.
4. Управление базой данных: загрузка, обновление, удаление записей; проектирование механизмов внесения изменений; создание удобных форм ввода и контроль качества вводимых данных.
5. Обеспечение безопасности: настройка прав доступа, проверка механизмов защиты, отслеживание попыток несанкционированного доступа.
6. Разработка и реализация механизмов восстановления данных при сбоях.
7. Оценка эффективности работы БнД и его совершенствование.
8. Взаимодействие с пользователями: обучение, сбор обратной связи, учет предложений по доработке.

9. Поддержка системного программного обеспечения: установка, обновление, настройка и сопровождение.

10. Организационно-методическая деятельность: выбор подходов к проектированию и модернизации, стратегическое планирование развития БнД, подготовка документации.

## 2.2. Классификация банков данных

Базы данных классифицируют, учитывая экономико-правовые особенности.

В зависимости от **условий использования** они могут быть бесплатными или платными. Платные, в свою очередь, делятся на коммерческие и некоммерческие, к которым относятся научные, библиотечные и социально значимые ресурсы.

По форме собственности базы данных бывают государственными или частными.

С точки зрения доступности их разделяют на открытые для всех и ограниченные для определённого круга пользователей.

Помимо этого, существуют и другие способы классификации, основанные на различных характеристиках баз данных.

# Глава 3. БАЗЫ ДАННЫХ

**Классификация баз данных**

Неструктурированные базы данных включают системы, основанные на семантических сетях, а частично структурированные – это гипертекстовые и полнотекстовые базы данных.

**Гипертекстовая база данных** – это текстовая база, где записи связаны между собой, что позволяет создавать ансамбли данных на основе их логической взаимосвязи.

**Полнотекстовая база данных** – это база данных, в которой хранятся полные тексты документов или их части.

Самыми распространёнными в экономике являются структурированные реляционные базы данных (рис. 4).

## 3.1. Система управления базами данных (СУБД)

**База данных** – это совокупность данных, имеющая определённое имя, отражающая состояния объектов и их связи в конкретной области знаний.

**Система управления базами данных (СУБД)** представляет собой компьютерную систему для хранения записей, основная цель которой – хранить информацию и предоставлять её по запросу.

Базу данных можно сравнить с электронной картотекой, то есть с хранилищем набора файлов данных, которые занесены в компьютер. Пользователь такой системы может выполнять множество операций с файлами, например:

- создавать новые пустые файлы в базе данных;
- добавлять новые данные в уже существующие файлы;
- искать информацию в существующих файлах;
- изменять данные в существующих файлах;

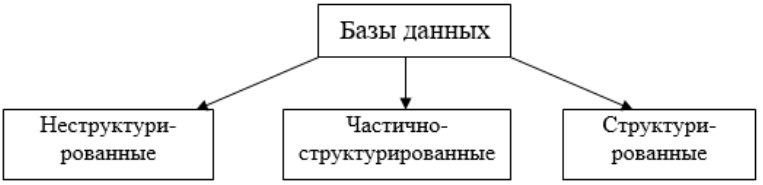


Рис. 4. *Классификация баз данных*

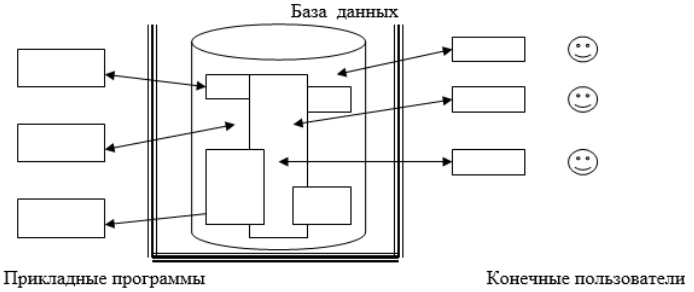


Рис. 5. *Схема системы базы данных*

• удалять данные из существующих файлов;

• удалять файлы из базы данных, что приводит к уничтожению их содержимого.

На **рис. 5** представлена упрощённая схема системы баз данных, включающая четыре ключевых компонента: **данные, аппаратное обеспечение, программное обеспечение и пользователей**.

**Система управления базами данных (СУБД)**

СУБД – это совокупность языковых и программных инструментов, предназначенных для **создания, администрирования и совместного использования базы данных** различными пользователями.

СУБД могут работать как на **персональных компьютерах** (включая ноутбуки), так и на **крупных вычислительных системах**. В связи с этим выделяют два типа таких систем:

**1. Однопользовательские системы (single-user system)** – в таких системах в один момент времени доступ к базе данных может получить только один пользователь.

**2. Многопользовательские системы** – позволяют одновременно работать с базой данных сразу нескольким пользователям.

Главная цель большинства многопользовательских СУБД – **обеспечить каждому пользователю возможность работы с базой так, словно это однопользовательская система**.

**Интегрированные и общие данные**

В большинстве случаев данные в базе данных обладают свойствами **интегрированности и общего доступа**.

**Интегрированные данные** – это представление базы данных в виде совокупности **нескольких независимых файлов**, которые могут как **перекрываться**, так и **не пересекаться** по содержанию.

employee (сотрудник)

| Name | Address | Department | Salary | . . . |
|---|---|---|---|---|

enrollment

| Name | Course | . . . |
|---|---|---|

Под термином «общие данные» понимается возможность одновременного использования одних и тех же данных в базе данных (БД) несколькими различными пользователями. Это означает, что каждый пользователь может получить доступ к той же информации одновременно с другими, при этом они могут использовать данные для разных целей. Например, информация о подразделении в файле сотрудников может использоваться как кадровым отделом, так и отделом обучения.

Обычно данные в БД называют постоянными, хотя на самом деле они могут изменяться. Под понятием «постоянные» понимаются те, которые остаются стабильными в отличие от более изменчивых данных, таких как промежуточные результаты, входные и выходные данные, рабочие очереди и другие транзитные данные.

База данных состоит из набора **постоянных данных**, которые используются прикладными системами для нужд предприятия.

Среди данных, встречающихся в различных предприятиях, можно выделить информацию о продукции, счетах, пациентах, студентах и данные для планирования.

## 3.2. Аппаратное обеспечение

1. Устройства для хранения данных (например, жёсткие диски), которые работают в сочетании с аппаратными средствами для ввода и вывода информации.

2. Центральный процессор (или процессоры), взаимодействующие с основной памятью, которые обеспечивают выполнение программ.

## 3.3. Программное обеспечение

Между физической базой данных (то есть реальными данными, которые хранятся) и пользователями системы находится программный слой – это менеджер базы данных (database manager) или, как чаще говорят, система управления базами данных (СУБД). Все операции, такие как выполнение запросов, добавление файлов, выборка и обновление данных, выполняются через СУБД.

Основная задача СУБД – предоставить пользователю возможность работать с базой данных, не вникая в детали, связанные с аппаратным обеспечением.

Следует отметить, что СУБД является важнейшим, но не единственным программным элементом системы. К другим компонентам можно отнести утилиты, инструменты для разработки приложений, средства проектирования, генераторы отчетов и другие программы.

## 3.4. Архитектура базы данных

В исследованиях, направленных на разработку оптимальной структуры СУБД, было предложено несколько подходов. Наибольший успех имела модель, предложенная американским комитетом по стандартизации ANSI, основанная на трехуровневой архитектуре организации базы данных.

1. **Уровень внешних представлений** – это самый верхний уровень, на котором каждая модель представляет свою собственную картину данных. Этот уровень отображает точку зрения на базу данных, соответствующую конкретным приложениям. Каждое приложение получает доступ только к тем данным, которые необходимы для его работы.
2. **Концептуальный уровень** – центральная часть системы, где база данных представлена в обобщённом виде, объединяя все данные, используемые различными приложениями. На этом уровне отображена общая модель предметной области, для которой создаётся база данных, включая только те характеристики объектов реального мира, которые важны для обработки.
3. **Физический уровень** – данные, которые хранятся в файлах или на структурированных страницах внешних носителей информации.


Рис. 6. *Модель организации БД*

Такая архитектура обеспечивает как логическую (между уровнями 1 и 2), так и физическую (между уровнями 2 и 3) независимость работы с данными. Логическая независимость означает возможность изменения одного приложения без необходимости изменений в других приложениях, работающих с той же базой данных. Физическая независимость позволяет переносить данные между различными носителями, сохраняя при этом работоспособность всех приложений. Это является значительным улучшением по сравнению с использованием файловых систем.

Выделение концептуального уровня позволило создать систему централизованного управления базой данных.

# Глава 4. РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ

Теоретическая база реляционной модели данных основывается на теории отношений, разработанной двумя логиками: американцем Чарльзом Содерсом Пирсом (1839–1914) и немецким ученым Эрнстом Шредером (1841–1902). В работах по теории отношений продемонстрировано, что множество отношений замкнуто относительно определенных операций, что позволяет образовать абстрактную алгебру в сочетании с этими операциями. Это ключевое свойство отношений стало основой для реляционной модели, в которой разработан язык манипулирования данными, связанный с первоначальной алгеброй. В 1970 году американский математик Кодд впервые сформулировал основные принципы и ограничения реляционной модели, обозначив операции четырьмя основными и четырьмя дополнительными.

Основной структурой данных в реляционной модели является отношение, отсюда и название модели (relation). N-арное отношение R представляет собой подмножество декартова произведения множеств D1, D2, …, Dn (где $n \geq 1$), которые могут быть как одинаковыми, так и различными. Эти исходные множества D1, D2, …, Dn в контексте модели называются доменами.

$R \subseteq D1 \times D2 \times \ldots \times Dn$,

где $D1 \times D2 \times \ldots \times Dn$ – полное декартово произведение, которое включает все возможные сочетания элементов, каждый из которых принадлежит своему домену. Например, для трех доменов D1, D2 и D3 это будет выглядеть следующим образом:

D1 = {Иванов, Крылов, Степанов}; D2 = {Теория автоматов, Базы данных}; D3 = {3, 4, 5}.

В таком случае полное **декартово произведение** будет содержать 18 возможных троек, где первый элемент – фамилия, второй – предмет, а третий – оценка. Однако отношение R может не охватывать все эти комбинации, оно может моделировать лишь определенные реальные случаи.

**Графическое** представление отношения довольно простое: его можно изобразить в виде таблицы, где столбцы отражают домены, участвующие в отношении, а строки показывают различные комбинации значений, взятых из этих доменов. Порядок значений в строках строго соответствует заголовкам столбцов (табл. 1).

*Таблица 1*

| R | | |
|---|---|---|
| Фамилия | Дисциплина | Оценка |
| Иванов | Теория автоматов | 4 |
| Иванов | Базы данных | 3 |
| Крылов | Теория автоматов | 5 |
| Степанов | Теория автоматов | 5 |
| Степанов | Базы данных | 4 |

Эта таблица обладает рядом характеристик:

1. В таблице нет повторяющихся строк.
2. Столбцы таблицы соответствуют атрибутам отношения.
3. Каждый атрибут имеет уникальное имя.
4. Порядок строк в таблице не имеет значения.

**Элементы домена**, входящие в отношение, называют атрибутами, а строки отношения – кортежами. Количество атрибутов в отношении называется его степенью или рангом.

Отношение не может содержать два одинаковых кортежа, поскольку оно является подмножеством декартова произведения, в котором все n-ки уникальны.

Таким образом, два отношения, различающиеся только порядком строк или столбцов, будут восприниматься как одно и то же в рамках реляционной модели.

Каждое отношение представляет собой динамическую модель реального объекта внешнего мира. В связи с этим вводятся понятия экземпляра отношения, который отражает текущее состояние объекта, и схемы отношения, определяющей структуру отношения.

Схемой отношения R является список имен атрибутов этого отношения с указанием соответствующего домена.

SR = (A1, A2, …, An), где Ai ∈ Di.

Если атрибуты принадлежат одному и тому же домену, они называются **сравнимыми**, где под «сравнимыми» подразумеваются атрибуты, для которых задано множество операций сравнения, определённых для данного домена. Например, если домен включает числовые значения, тогда допустимыми операциями будут {=, <>, >=, <=, >, <}.

Схемы двух отношений считаются эквивалентными, если они имеют одинаковую степень и можно провести упорядочивание имен атрибутов, при котором на одинаковых позициях окажутся атрибуты, принимающие значения из одного и того же домена, то есть сравнимые между собой.

SR1 = (A1, A2, …, An) – схема отношения R1.

SR2 = (B1i, B2i, …, Bni) – схема отношения R2 после упорядочивания атрибутов.

В таком случае выполняются следующие условия:

1) n = m;

2) SR1 ~ SR2 ⇔ для каждого j, Aj, Bij ⊆ Dj.

Как уже упоминалось, реляционная модель представляет базу данных как совокупность взаимосвязанных отношений, которые поддерживаются неявно. Эта модель поддерживает **иерархические связи** между отношениями, где одно отношение может быть основным, а другое – подчиненным. Это означает, что один кортеж из основного отношения может быть связан с несколькими кортежами из подчиненного отношения. Чтобы поддерживать такие связи, оба отношения должны содержать соответствующие атрибуты. В основном отношении таким атрибутом является **первичный ключ** (PRIMARY KEY), который уникально идентифицирует кортеж этого отношения. В подчиненном отношении для установления связи должен быть набор атрибутов, который соответствует первичному ключу основного отношения, но в этом случае такой набор уже считается **внешним ключом** (FOREIGN KEY), то есть он определяет несколько кортежей подчиненного отношения, которые связываются с единственным кортежем основного.

Пример:

Рассмотрим карьеру одного человека, который сменяет несколько мест работы и выполняет различные должностные обязанности. Для моделирования этих данных потребуется два отношения: одно для всех сотрудников, другое – для записей в их трудовых книжках. В этом случае первичный ключ в отношении «Сотрудник» – атрибут «Паспорт», который будет внешним ключом для отношения «Карьера» (рис. 7).

## Операции над отношениями

Алгебра представляет собой совокупность объектов с набором операций, определённых для этих объектов, при этом операции замкнуты относительно данного множества, которое называется основным.

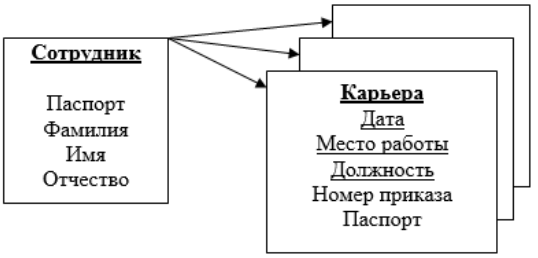


Рис. 7. *Связь между основным и подчиненным отношением*

В **реляционной алгебре** основным множеством является множество отношений. Для работы с этими отношениями используются операции реляционной алгебры. Э.Ф. Кодд предложил восемь операций. Эти операции можно разделить на две категории: теоретико-множественные и специальные. Первая категория включает четыре операции, три из которых являются бинарными, то есть оперируют с двумя отношениями и требуют эквивалентных схем исходных отношений.

Основные операции реляционной алгебры следующие:

**Теоретико-множественные операции:**

1) объединение отношений;
2) пересечение отношений;
3) разность отношений;
4) произведение отношений.

**Специальные операции:**

1) деление отношений;
2) ограничение отношения;
3) проекция отношения;
4) соединение отношений.

Помимо этих операций, в большинстве систем управления базами данных также реализуются операции присваивания, которые позволяют сохранять результаты обработки в базе данных, а также операция переименования атрибутов.

**Операции** реляционной алгебры можно описать следующим образом:

1. Операция объединения двух отношений создаёт новое отношение, которое включает в себя все строки из обеих исходных таблиц. Для выполнения операции атрибуты обеих таблиц должны совпадать.

2. Операция пересечения двух отношений возвращает отношение, состоящее только из тех строк, которые присутствуют в обеих исходных таблицах. Для этого атрибуты в операндах должны быть одинаковыми.

3. Операция разности отношений позволяет выделить строки, которые принадлежат только первому отношению и отсутствуют во втором. Операнды должны содержать одинаковые атрибуты.

Пример:

Рассмотрим три отношения, все из которых имеют одинаковые схемы:

• список абитуриентов, сдавших предварительные экзамены: **R1 = {ФИО, Паспорт, Школа}**

• список абитуриентов, сдавших вступительные экзамены: **R2 = {ФИО, Паспорт, Школа}**

• список абитуриентов, принятых в институт: **R3 = {ФИО, Паспорт, Школа}**

В случае неудачи на предварительных экзаменах абитуриенты могут попробовать поступить в институт через вступительные экзамены. Ответим на несколько вопросов:

• список абитуриентов, сдававших дважды экзамены и не поступивших в вуз: **R = R1 ∩ R2 \ R3**

• список абитуриентов, поступивших в вуз с первого раза: **R = (R1 \ R2 ∩ R3) ∩ (R2 \ R1 ∩ R3)**

• Список абитуриентов, поступивших в вуз со второго раза: **R = R1 ∩ R2 ∩ R3**

• список абитуриентов, поступавших в вуз один раз и не поступивших: **R = (R1 \ R2) ∩ (R2 \ R1) \ R3**

4. Операция произведения двух отношений объединяет каждую строку первого отношения с каждой строкой второго, создавая новое отношение. Количество строк в результирующем отношении будет равно произведению количества строк в исходных отношениях. Атрибуты операндов не должны пересекаться.

5. Операция деления «противоположна» операции произведения. Для объяснения этой операции мы используем несколько обозначений. Пусть есть два отношения: делимое **A** с атрибутами **{a1, a2, ... an, b1, b2, ... bm}** и делитель **B** с атрибутами **{b1, b2, ... bm}**. Атрибуты **b1, b2, ... bm** в обеих таблицах имеют одинаковое имя и определены на одном и том же домене (множестве допустимых значений). Результат деления A на B будет отношением **C** с атрибутами **{a1, a2, ... an}**.

Операция деления заключается в следующем: если в строках отношения **A** атрибуты **b1, b2, ... bm** совпадают с какой-либо строкой в отношении B, то эти атрибуты исключаются, а оставшиеся атрибуты **(a1, a2, ... an)** включаются в итоговое отношение **C**. Пример операции деления можно увидеть на рис. 8.

6. Операция ограничения (или выборки) отношения позволяет выбрать те строки из отношения-операнда, которые удовлетворяют заданному условию или предикату.

| Фамилия сотрудника | Отдел |
|---|---|
| Уваров | Сметный |
| Федоров | Выпуска |
| Омелькин | Сметный |
| Чеповский | Сметный |
| Чибрин | Выпуска |

:

| Отдел |
|---|
| Сметный |
| Выпуска |

=

| Фамилия сотрудника |
|---|
| Уваров |
| Федоров |
| Омелькин |
| Чеповский |
| Чибрин |

Рис. 8. *Пример операции деления*

7. Операция проекции дает возможность выбрать определённые столбцы из отношения-операнда, игнорируя остальные.

8. Операция соединения является одной из самых важных и широко используемых в реляционной алгебре. Она имеет большое практическое значение, поскольку существует два типа соединений: соединение по условию и естественное соединение. При соединении по условию двух отношений происходит объединение строк из обоих отношений, после чего эта объединённая строка проверяется на выполнение заданного условия. Если условие выполняется, строка добавляется в результат. Если оба отношения имеют общий атрибут (возможно, составной), то условие соединения может быть опущено, и предполагается, что будет произведено сравнение на равенство значений этих общих атрибутов. В таких случаях говорят о естественном соединении отношений.

9. Операция переименования используется для изменения имён атрибутов в отношении, что позволяет создать более понятные или удобные для работы наименования.

10. Операция присваивания позволяет сохранить результат выполнения реляционного выражения в виде нового отношения базы данных, что обеспечивает возможность дальнейшего использования этих данных.

Точные определения этих операций можно найти в литературе, посвящённой теории баз данных. На практике пользователи часто работают с ними через синтаксис языков программирования, например, SQL-запросов.

# Глава 5. ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ НА ОСНОВЕ ПРИНЦИПОВ НОРМАЛИЗАЦИИ

Проектирование реляционной базы данных подразумевает создание системы взаимосвязанных таблиц, в которых четко определены все атрибуты, установлены первичные ключи и заданы дополнительные параметры, гарантирующие целостность данных. Взаимосвязь таблиц обусловлена тем, что при выполнении запросов они объединяются, а данные, содержащиеся в разных таблицах, должны быть представлены в едином формате. Например, если в одной таблице оценки выражены числами, а в другой – текстовыми значениями, такими как «отлично» или «хорошо», объединить эти таблицы по полю «Оценка» не удастся, даже несмотря на их схожее значение. Поэтому при проектировании базы данных важно учитывать все детали и стремиться к максимально точному структурированию.

Фактически, создание базы данных – это ключевой этап разработки программного комплекса, который будет использоваться в течение длительного времени и большим числом пользователей.

Жизненный цикл базы данных, представленный на рис. 9, в целом соответствует основным этапам создания программного обеспечения, но обладает рядом уникальных особенностей, характерных именно для работы с базами данных.

Процесс разработки базы данных состоит из нескольких этапов, целью которых является преобразование неформального описания структуры данных, относящихся к определенной предметной области, в формализованное представление с применением модели данных. В общем случае проектирование базы данных включает следующие стадии:

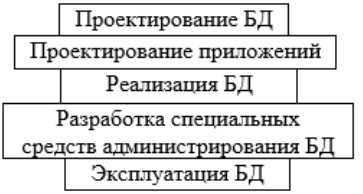


Рис. 9. *Этапы жизненного цикла БД*

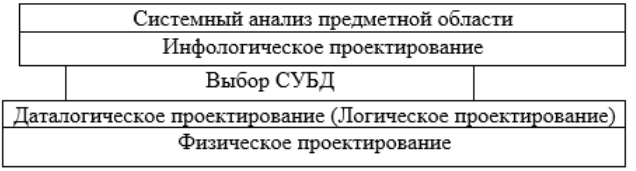


Рис. 10. *Этапы проектирования БД*

1) проведение анализа системы и составление текстового описания информационных объектов, относящихся к рассматриваемой предметной области;

2) создание инфологической модели, которая представляет собой частично формализованное описание объектов предметной области с использованием семантического подхода;

3) этап логического (или даталогического) проектирования, в ходе которого разрабатывается структура базы данных в соответствии с выбранной даталогической моделью;

4) физическое проектирование, включающее выбор наиболее эффективных способов хранения данных на внешних носителях с целью оптимизации производительности системы.

Дополнительно, перед переходом от инфологической к логической модели необходимо определить, какая система управления базами данных (СУБД) будет использоваться для реализации проекта. Таким образом, весь процесс проектирования базы данных можно условно разделить на пять этапов, что наглядно демонстрируется на рис. 10.

Рассмотрим каждый из этих этапов более подробно.

## 5.1. Системный анализ предметной области

В ходе проектирования базы данных на этапе системного анализа необходимо составить детальное текстовое описание объектов предметной области и их взаимосвязей. Это описание должно быть достаточно точным, чтобы чётко отразить все взаимодействия между объектами и их связями в рамках рассматриваемой области.

При определении состава и структуры предметной области можно использовать два основных подхода:

Функциональный подход базируется на принципе «от задач» и применяется в случаях, когда заранее известны функции и требования определенной группы пользователей, для которых создается база данных. В этом слу-

чае можно определить минимально необходимый набор объектов, опираясь на конкретные потребности.

Предметный подход используется в тех ситуациях, когда информационные потребности пользователей сложно заранее зафиксировать, так как они могут быть разнообразными и подвержены изменениям. В этом случае затруднительно точно установить, какие объекты следует описывать. Поэтому при разработке базы данных в первую очередь рассматриваются те объекты и взаимосвязи, которые являются наиболее значимыми для данной предметной области.

Если база данных разрабатывается с применением предметного подхода, она называется предметной. Такие базы данных могут быть использованы для решения широкого спектра задач, которые изначально не были четко определены. Хотя этот метод обеспечивает большую гибкость, сложность учета всех возможных потребностей пользователей может привести к созданию слишком сложной структуры, которая окажется малоэффективной для решения конкретных задач.

На практике наиболее часто применяется компромиссный подход, сочетающий ориентацию на конкретные функциональные потребности пользователей с возможностью дальнейшего расширения базы данных для поддержки новых приложений в будущем.

Завершающим этапом системного анализа является подробное описание информации об объектах предметной области, необходимой для решения текущих задач, которые будут храниться в базе данных. Также следует сформулировать конкретные задачи, которые база данных будет решать, кратко описать алгоритмы их выполнения и определить перечень выходных документов, которые система должна формировать. Важным элементом анализа является также описание входных документов, служащих основой для наполнения базы данных.

## 5.2. Даталогическое проектирование

Логическое (или даталогическое) проектирование реляционных баз данных направлено на создание структуры базы данных в виде набора взаимосвязанных отношений, которые отражают абстрактные объекты предметной области и их семантические связи. Одним из ключевых инструментов проверки правильности такой структуры являются функциональные зависимости между атрибутами базы данных.

Однако данный процесс не ограничивается только разработкой схемы отношений. По его завершении должны быть подготовлены следующие материалы:

• описание концептуальной структуры базы данных с учетом особенностей выбранной системы управления базами данных (СУБД);

• описание внешних моделей, ориентированных на работу с данной СУБД;

• описание декларативных ограничений, гарантирующих целостность данных;

• разработка механизмов, обеспечивающих поддержку семантической целостности данных.

Перед тем как представить схему базы данных в терминах конкретной СУБД, необходимо тщательно ее спроектировать. При этом важно придерживаться принципов реляционной модели данных, чтобы создать корректную структуру базы.

Схема базы данных считается правильно спроектированной, если в ней отсутствуют лишние или нежелательные зависимости между атрибутами отношений.

Процесс построения корректной реляционной схемы называется логическим проектированием. Существует два основных подхода к его реализации:

• **Декомпозиция** (разбиение) – исходное множество отношений заменяется на большее количество новых отношений, которые представляют собой проекции исходных данных.

• **Синтез** – схема базы данных создается путем объединения исходных элементарных зависимостей между объектами предметной области.

Классический метод проектирования реляционных баз данных основывается на **теории нормализации**, которая предусматривает анализ функциональных зависимостей между атрибутами отношений. Эти зависимости играют ключевую роль в процессе нормализации, поскольку определяют устойчивые связи между объектами и их характеристиками в рамках предметной области.

Метод проектирования, основанный на декомпозиции, представляет собой пошаговую нормализацию отношений. На каждом этапе схема базы данных переводится в нормальную форму более высокого уровня, что улучшает ее структурные свойства и эффективность по сравнению с предыдущим уровнем.

Каждая нормальная форма имеет свой уникальный набор ограничений, и отношение считается соответствующим определенной нормальной форме, если оно удовлетворяет этим ограничениям. В теории реляционных баз данных выделяются несколько нормальных форм:

– первая нормальная форма (1NF);

– вторая нормальная форма (2NF);

– третья нормальная форма (3NF);

– нормальная форма Бойса-Кодда (BCNF);

– четвертая нормальная форма (4NF);

– пятая нормальная форма или форма проекции-соединения (5NF или NF/PJ).

Основные особенности нормальных форм:

1) каждая следующая нормальная форма улучшает характеристики предыдущей в определенных аспектах;

2) переход к более высокой нормальной форме сохраняет все преимущества предыдущих форм.

Традиционное проектирование баз данных включает последовательный переход от одной нормальной формы к другой. В процессе декомпозиции важно обеспечить возможность восстановления исходной схемы, то есть решить задачу обратимости. Это означает, что процесс декомпозиции должен обеспечивать эквивалентность схем баз данных при замене одной схемы на другую.

Схемы считаются эквивалентными, если изначальные данные можно восстановить путем естественного соединения отношений, входящих в результатирующую схему, при этом не появляются дополнительные записи в исходных данных.

При таких эквивалентных преобразованиях сохраняются все исходные функциональные зависимости между атрибутами отношений. Эти зависимости не описывают текущее состояние базы данных, а касаются всех ее возможных состояний. То есть функциональные зависимости отражают связи, присущие реальному объекту, который моделируется с помощью базы данных.

Следовательно, функциональные зависимости можно определить на основе текущего состояния базы данных только в том случае, если ее экземпляр полностью отражает все абстрактные данные (при этом предполагается, что база данных не изменяется и не дополняется). Однако в реальных условиях это требование практически невозможно выполнить.

Процесс нормализации базы данных основан на концепции функциональной зависимости атрибутов.

Атрибут B считается функционально зависимым от атрибута A (обозначается как $A \rightarrow B$), если для каждого значения A существует не более одного значения B в любой момент времени. Важно, что термин «функциональная зависимость» аналогичен понятию функции в математике.

Если неключевой атрибут зависит от всего составного ключа, но не зависит от его отдельных частей, такая зависимость называется полной функциональной зависимостью атрибута от составного ключа.

Если атрибут A зависит от атрибута B, а атрибут B зависит от атрибута C, но обратной зависимости нет, говорят, что атрибут C зависит от атрибута A транзитивно.

В случае наличия нескольких функциональных зависимостей в отношении каждый атрибут или набор атрибутов, от которых зависит другой атрибут, называют детерминантом отношения.

Отношение считается находящимся в первой нормальной форме (1NF), если и только если в каждом пересечении строки и столбца содержатся только атомарные (неделимые) значения атрибутов.

Это определение в некотором роде избыточно, так как оно фактически описывает саму сущность отношения в теории реляционных баз данных. Отношения, удовлетворяющие первой нормальной форме, обычно называют просто нормализованными отношениями. В свою очередь, ненормализованные отношения можно рассматривать как таблицы с нерегулярным заполнением, например, таблица «Расписание», которая может иметь следующий вид:

| Преподаватель | День недели | Номер пары | Название дисциплины | Тип занятий | Группа |
|---|---|---|---|---|---|
| Петров В. И. | Понед. | 1 | Теор. выч. проц. | Лекция | 4906 |
| | Вторник | 1 | Комп. графика | Лаб. раб. | 4907 |
| | Вторник | 2 | Комп. графика | Лаб. раб. | 4906 |

На пересечении строки и столбца таблицы присутствует набор атомарных значений, который включает перечень дней, список учебных пар и дисциплин, по которым занятия проводит один преподаватель.

Для того чтобы привести отношение «Расписание» к первой нормальной форме, необходимо добавить в каждую строку фамилию преподавателя.

Отношение находится во **второй нормальной форме**, если оно соответствует первой нормальной форме и не включает неполных функциональных зависимостей атрибутов, не входящих в первичный ключ, от элементов этого ключа.

| Преподаватель | День недели | Номер пары | Название дисциплины | Тип занятий | Группа |
|---|---|---|---|---|---|
| Петров В. И | Понед. | 1 | Теор. выч. проц. | Лекция | 4906 |
| Петров В. И | Вторник | 1 | Комп. графика | Лаб. раб. | 4907 |
| Петров В. И | Вторник | 2 | Комп. графика | Лаб. раб. | 4906 |

Рассмотрим базу данных, которая моделирует процесс сдачи студентами экзаменов в период сессии. Структура этой базы данных включает такие атрибуты, как: (ФИО, Номер зачётной книжки, Группа, Дисциплина, Оценка).

Поскольку каждый студент сдаёт несколько предметов в рамках одной сессии, в качестве первичного ключа для этого отношения можно выбрать пару атрибутов (Номер зачётной книжки, Дисциплина). Это позволяет однозначно идентифицировать каждую запись в базе данных. Однако атрибуты «ФИО» и «Группа» зависят только от части первичного ключа – от атрибута «Номер зачётной книжки». Это свидетельствует о наличии неполных функциональных зависимостей в отношении.

Чтобы привести это отношение ко второй нормальной форме (2NF), необходимо разделить его на несколько проекций. Важно, чтобы при этом не терялась информация и можно было восстановить исходное отношение. В данном случае возможны следующие проекции:

1. (ФИО, Номер зачётной книжки, Группа).
2. (Номер зачётной книжки, Дисциплина, Оценка).

После разделения отношения на эти два, каждая проекция будет удовлетворять требованиям второй нормальной формы, так как в них не останется неполных функциональных зависимостей.

Для понимания, почему важно привести отношение ко второй нормальной форме, рассмотрим ситуацию, когда студент меняет группу. Если отношение не было разделено, придётся изменить все записи для этого студента и обновить атрибут «Группа» в каждой из них. Но если отношение уже разделено, необходимо изменить только одну запись в первой проекции, что более эффективно и исключает избыточность данных.

Когда отношение приводит к третьей нормальной форме (3NF), оно должно не только удовлетворять требованиям второй нормальной формы, но и не содержать транзитивных зависимостей.

Примером такого отношения может быть связь между студентами и их группами, факультетами, специальностями, например: (ФИО, Номер зачётной книжки, Группа, Факультет, Специальность, Выпускающая кафедра).

В качестве первичного ключа в рассматриваемом отношении используется атрибут «Номер зачётной книжки». Однако для правильного понимания работы с этим отношением важно учесть другие функциональные зависимости. Например, группа, в которой учится студент, однозначно определяет факультет, на котором он обучается, а также его специальность и выпускающую кафедру. Также выпускающая кафедра определяет факультет, который отвечает за выпуск студентов с этой кафедры. Однако если несколько кафедр могут выпускать студентов одной и той же специальности,

то специальность не будет уникально определять выпускающую кафедру. Таким образом, можно выделить следующие функциональные зависимости:

- Номер зачётной книжки → ФИО
- Номер зачётной книжки → Группа
- Номер зачётной книжки → Факультет
- Номер зачётной книжки → Специальность
- Номер зачётной книжки → Выпускающая кафедра
- Группа → Факультет
- Группа → Специальность
- Группа → Выпускающая кафедра
- Выпускающая кафедра → Факультет

Эти зависимости могут привести к возникновению транзитивных связей. Для устранения таких зависимостей можно предложить следующую структуру отношений:

1. (Номер зачётной книжки, ФИО, Специальность, Группа).
2. (Группа, Выпускающая кафедра).
3. (Выпускающая кафедра, Факультет).

Каждое из этих отношений имеет свой первичный ключ, и такая структура отношений будет соответствовать третьей нормальной форме.

Теперь рассмотрим нормальную форму Бойса-Кодда (BCNF). Отношение считается находящимся в BCNF, если оно удовлетворяет требованиям третьей нормальной формы и каждый детерминант в отношении является потенциальным ключом. Важно, чтобы все функциональные зависимости были связаны с ключом и не нарушали нормализацию данных.

Примером отношения, которое необходимо привести к нормальной форме Бойса-Кодда, может быть база данных, которая моделирует процесс сдачи студентами экзаменов в текущей сессии. Студент может сдавать экзамен по одной дисциплине несколько раз, если его оценка неудовлетворительная. Чтобы исключить путаницу с однофамильцами, каждого студента можно однозначно идентифицировать по номеру его зачётной книжки. Кроме того, в рамках системы учёта успеваемости каждому студенту назначается уникальный идентификатор. Структура отношения, которое моделирует сдачу экзаменов, будет следующей: (Номер зачётной книжки, Идентификатор_студента, Дисциплина, Дата, Оценка).

Возможные кандидаты на роль первичного ключа для данного отношения включают:

- Номер зачётной книжки, Дисциплина, Дата
- Идентификатор_студента, Дисциплина, Дата

Соответствующие функциональные зависимости для этого отношения следующие:

- Номер зачётной книжки, Дисциплина, Дата → Оценка

• Идентификатор_студента, Дисциплина, Дата → Оценка
• Номер зачётной книжки → Идентификатор_студента
• Идентификатор_студента → Номер зачётной книжки

Два последних типа зависимости возникаютв связи с тем, что каждому студенту соответствует уникальный номер зачётной книжки и уникальный идентификатор студента. То есть, зная номер зачётной книжки, можно однозначно определить идентификатор студента (это третья зависимость), и наоборот (это четвёртая зависимость).

Эти зависимости также необходимо учитывать при анализе и нормализации данного отношения.

Это отношение соответствует третьей нормальной форме, поскольку в нём отсутствуют неполные функциональные зависимости между непервичными атрибутами и атрибутами возможного ключа, а также нет транзитивных зависимостей. Однако возникает вопрос: не являются ли третья и четвёртая зависимости неполными? Ответ: нет, потому что зависимые атрибуты не относятся к непервичным атрибутам, то есть к атрибутам, не входящим в состав возможных ключей. Таким образом, к данному аспекту претензий нет.

Тем не менее, это отношение не соответствует четвёртой нормальной форме, так как в нём присутствуют два детерминанта – Номер зачётной книжки и Идентификатор_студента, которые не являются возможными ключами этого отношения. Для приведения отношения к нормальной форме Бойса-Кодда его нужно разделить, например, на два подотчёта с такими схемами:

1. (Идентификатор_студента, Дисциплина, Дата, Оценка).
2. (Номер зачётной книжки, Идентификатор_студента).

Или наоборот:

1. (Номер зачётной книжки, Дисциплина, Дата, Оценка).
2. (Номер зачётной книжки, Идентификатор_студента).

Эти схемы эквивалентны с точки зрения теории нормализации, и выбор между ними зависит от дополнительных факторов. Например, если существует вероятность утраты зачётных книжек, нужно подумать, как будет происходить их восстановление: если номер останется прежним, разницы не будет, но если номер изменится, то предпочтительнее будет первая схема.

В большинстве реальных проектов баз данных для удовлетворения требований нормализации достаточно придерживаться третьей нормальной формы или формы Бойса-Кодда. Однако в теории нормализации существуют и более высокие нормальные формы, которые уже не затрагивают функциональные зависимости между атрибутами отношений, а касаются более

глубоких аспектов семантики предметной области и других типов зависимостей.

## 5.3. Инфологическое моделирование

Инфологическая модель применяется на втором этапе разработки базы данных, сразу после того, как была создана начальная словесная характеристика предметной области. Она необходима для того, чтобы представление о системе было ясно не только специалистам по базам данных, но и заказчикам, а также другим экспертам, работающим в конкретной области. Процесс проектирования базы данных обычно требует многочисленных обсуждений с различными участниками проекта. В крупных информационных системах проект базы данных является основой для всей системы, и на основе правильно выполненного инфологического проекта может быть принято решение о кредитовании, например, банком.

Основная цель инфологической модели – представить предметную область в такой форме, которая будет понятна и легко воспринимаема. При этом она должна быть достаточно полной, чтобы позволить оценить, насколько глубоко и корректно проработан проект базы данных, но не должна зависеть от конкретной СУБД. Выбор системы управления базами данных является отдельным шагом, который следует после создания такого универсального проекта, который не привязан к конкретной СУБД.

Инфологическое проектирование связано с тем, что оно направлено на отображение семантики предметной области в структуре базы данных. Реляционная модель, хоть и достаточно проста и универсальна, не предоставляет достаточно средств для того, чтобы выразить смыслы и отношения, которые существуют в реальной предметной области. В 1970-х годах было предложено несколько альтернативных моделей для решения этой задачи, включая семантическую модель, представленную Хаммером и МакЛеоном, функциональную модель Шипмана и модель «сущность-связь» (ER-модель), предложенную Ченом. Несмотря на множество существующих вариантов, ER-модель стала наиболее популярной и используется как стандарт в инфологическом проектировании.

На сегодняшний день ER-модель поддерживается многими современными CASE-системами, которые предлагают инструменты для разработки данных в рамках этой модели. Эти системы также обеспечивают возможности для автоматического преобразования инфологического проекта в реляционную модель, подходящую для использования с конкретной СУБД. Помимо этого, CASE-системы включают функции для детализированного документирования всего процесса разработки базы данных. Автоматизиро-

ванные инструменты для генерации отчетов позволяют создать подробные отчеты о текущем состоянии проекта, описывая объекты и их связи как в графическом, так и в текстовом виде, что значительно упрощает процесс ведения и контроля за проектом.

## 5.4. Модель «сущность-связь»

Как и любая концептуальная модель, модель «сущность-связь» базируется на нескольких ключевых понятиях, которые служат основой для создания более сложных структур в соответствии с определёнными правилами. Этот подход проектирования во многом схож с объектно-ориентированным методом, который сегодня является основным при разработке сложных программных систем. Поэтому элементы ER-модели могут быть знакомы тем, кто уже работает с объектно-ориентированными концепциями. В этом случае освоение технологии проектирования баз данных с использованием ER-модели будет интуитивно понятным и доступным.

Основные понятия ER-модели включают следующее:

**Сущность и её свойства**

Сущность представляет собой модель группы однотипных объектов, которые существуют в рамках информационной системы. Каждая сущность имеет уникальное название в пределах системы. Поскольку сущность отражает класс объектов, можно предположить, что в системе существует несколько её экземпляров.

Каждый экземпляр сущности характеризуется набором атрибутов, которые описывают его свойства. Важно, чтобы этот набор позволял отличать один экземпляр сущности от другого. Например, для сущности «Сотрудник» такими атрибутами могут быть: Табельный номер, Фамилия, Имя, Отчество, Дата рождения, Количество детей и другие.

Одним из атрибутов является ключевой атрибут, который однозначно идентифицирует экземпляр сущности. Например, для сущности «Сотрудник» таким атрибутом является Табельный номер, так как он уникален для каждого сотрудника организации. Таким образом, каждый конкретный сотрудник является экземпляром сущности «Сотрудник».

**Графическое изображение сущности**

Одним из наиболее распространённых способов графически изобразить сущность является прямоугольник, в верхней части которого размещается название сущности, а ниже – перечень её атрибутов. Ключевые атрибуты могут быть выделены, например, подчёркиванием или другим шрифтом (рис. 11).


Рис. 11. *Пример определения сущности в модели ER*

**Связи между сущностями**

Сущности в ER-модели могут быть связаны между собой с помощью бинарных ассоциаций, которые показывают их взаимосвязь и взаимодействие в системе. Эти связи могут быть как между разными сущностями, так и между экземплярами одной и той же сущности, что называется рекурсивной связью. Связи определяют, как экземпляры сущностей соотносятся между собой.

Если между двумя сущностями устанавливается связь, она описывает отношения между экземплярами этих сущностей. Например, рассмотрим связь между сущностью «Студент» и сущностью «Преподаватель», где связь обозначает руководство дипломными проектами.

В данном случае каждый студент может иметь только одного руководителя, тогда как один преподаватель может курировать нескольких студентов-дипломников. Таким образом, данная связь является отношением типа «один ко многим» (1:M), где одна сущность (Преподаватель) связана со множеством экземпляров другой сущности (Студент).

Графическое представление данной связи можно увидеть на рис. 12.

Различные нотации по-разному визуализируют **мощность (кардинальность) связи**. В данном случае используется нотация CASE-системы **PowerDesigner**, в которой множественность обозначается путем разделения линии связи на три части. Связь получает **общее название**, например, «**Дипломное проектирование**», и содержит **ролевые обозначения** для каждой из сущностей. В рассматриваемом примере роль *Студента* обозначается как «**Пишет диплом под руководством**», а роль *Преподавателя* – «**Руководит**».

Графическое представление связи обеспечивает удобочитаемость модели и упрощает анализ взаимосвязей между сущностями. Такой подход делает структуру данных более наглядной и интуитивно понятной.

Связи между сущностями классифицируются в зависимости от количества экземпляров, которые могут быть связаны друг с другом:


Рис. 12. *Пример отношения «один-ко-многим» при связывании сущностей*

1. **Один к одному (1:1)** – один экземпляр одной сущности может быть связан не более чем с одним экземпляром другой сущности.
2. **Один ко многим (1:M)** – один экземпляр первой сущности может быть связан с несколькими экземплярами второй, но каждый экземпляр второй сущности связан только с одним экземпляром первой.
3. **Многие ко многим (M:M)** – один экземпляр первой сущности может иметь связи с несколькими экземплярами второй сущности, и наоборот.

Например, если рассмотреть связь **«Изучает»** между сущностями **«Студент»** и **«Дисциплина»**, то она относится к **типу «многие ко многим» (M:M)**. Это означает, что один студент может изучать сразу несколько дисциплин, а каждая дисциплина, в свою очередь, изучается множеством студентов. Графическое представление данной связи можно увидеть на рис. 13.

В рамках модели между двумя сущностями может быть установлено **неограниченное количество связей**, каждая из которых несет **различную смысловую нагрузку**. Например, между сущностями **«Студент»** и **«Преподаватель»** можно определить несколько взаимосвязей. Одна из них – **«Дипломное проектирование»**, которая была рассмотрена ранее. Другая связь, например, **«Лекции»**, может отражать информацию о том, какие преподаватели читают лекции определенным студентам и какие студенты посещают лекции конкретного преподавателя.

Любая связь может быть **обязательной** или **необязательной**:

**Обязательная связь** означает, что **каждый экземпляр** сущности должен участвовать в данной связи.

**Необязательная связь** допускает, что **не все экземпляры** сущности обязаны входить в эту взаимосвязь.

Связь может быть **обязательной для одной сущности** и одновременно **необязательной для другой**.

В разных нотациях обязательность связи обозначается по-разному. В **PowerDesigner** используется следующая схема:

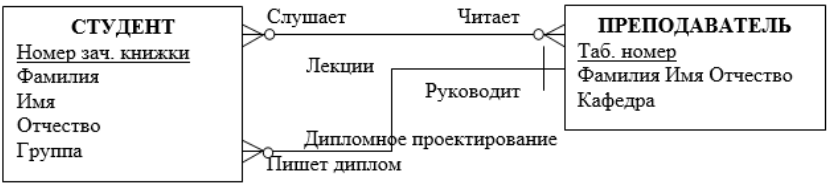


Рис. 13. *Пример моделирования связи «многие-ко-многим»*

• **необязательная связь** изображается **пустым кружком** на конце линии связи;

• **обязательная связь** обозначается **перпендикулярной чертой**, перечеркивающей линию связи.

Пример: «Дипломное проектирование»

Рассмотрим связь «**Дипломное проектирование**». В данной модели каждый студент, работающий над дипломом, обязательно должен иметь научного руководителя. Однако не каждый преподаватель обязан курировать дипломные проекты. Таким образом, связь является обязательной со стороны студента, но необязательной со стороны преподавателя.

В рамках **ER-модели** предусмотрена возможность **категоризации сущностей**, что соответствует концепции **наследования** в объектно-ориентированном программировании. Данный принцип подразумевает, что **сущность может быть разделена на несколько подтипов**, каждый из которых наследует **общие атрибуты и связи** супертипа, а также может содержать **собственные, уникальные характеристики**.

При категоризации сущности **все ее подтипы должны быть взаимоисключающими**, то есть каждый экземпляр супертипа должен принадлежать только одному из подтипов. В случае, если на этапе анализа **невозможно определить полный список подтипов**, вводится **резервный подтип**, условно обозначаемый как **«Прочие»**, который может быть уточнен в дальнейшем. Практика показывает, что в большинстве информационных систем бывает достаточно использовать **два-три уровня подтипизации**.

**Супертип** – сущность, от которой производятся подтипы.

**Подтип** – сущность, являющаяся частью супертипа и обладающая как общими, так и специфическими атрибутами.

**Экземпляр супертипа** всегда относится к одному из его подтипов.

Для визуального представления иерархии подтипов используется **узел-дискриминатор** – специальный графический элемент, обозначающий категоризацию. В **PowerDesigner** дискриминатор отображается в виде **полукруга**, обращенного выпуклой стороной к **суперсущности**.

**Стрелка от выпуклой стороны** указывает на **супертип**.

**Стрелки от диаметра полукруга** соединяют дискриминатор с подтипами сущности.

Графическое представление данного принципа можно увидеть на рис. 14.

Каждый тест в некоторой системе тестирования является либо тестом проверки знаний языка SQL, либо некоторой аналитической задачей, которая выполняется с использованием заранее написанных Java-апплетов, либо тестом по некоторой области знаний, состоящим из набора вопросов и набора ответов, предлагаемых к каждому вопросу. В результате построения модели предметной области в виде набора сущностей и связей получаем связный граф. В полученном графе необходимо избегать циклических связей – они выявляют некорректность модели.

В качестве примера спроектируем инфологическую модель системы, предназначенной для хранения информации о книгах и областях знаний, представленных в библиотеке. Разработку модели начнем с выделения основных сущностей.

Прежде всего, существует сущность «Книги», каждая книга имеет уникальный шифр, который является ее ключом, и ряд атрибутов, которые взяты из описания предметной области. Множество экземпляров сущности определяет множество книг, которые хранятся в библиотеке. Каждый экземпляр сущности «Книги» соответствует не конкретной книге, стоящей на полке, а описанию некоторой книги, которое дается обычно в предметном каталоге библиотеке. Каждая книга может присутствовать в нескольких экземплярах, и это как раз те конкретные книги, которые стоят на полках библиотеки. Для того чтобы отразить это, мы должны ввести сущность «Экземпляры», которая будет содержать описания всех экземпляров книг, которые хранятся в библиотеке. Каждый экземпляр имеет уникальный инвентарный номер, однозначно определяющий конкретную книгу. Кроме того, каждый экземпляр книги может находиться либо в библиотеке, либо на руках у некоторого читателя, и в последнем случае для данного экземпляра указываются дополнительно дата взятия книги читателем и дата предполагаемого возврата книги.

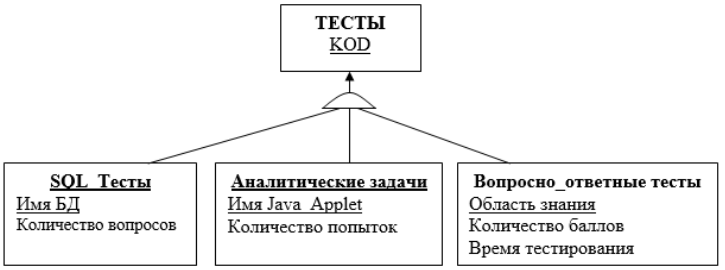


Рис. 14. *Диаграмма подтипов сущности ТЕСТ*

Между сущностями «Книги» и «Экземпляры» существует связь «один ко многим» (1:М), обязательная с двух сторон. Чем определяется данный тип связи? Мы можем предположить, что каждая книга может присутствовать в библиотеке в нескольких экземплярах, поэтому связь «один ко многим». При этом если в библиотеке нет ни одного экземпляра данной книги, то мы не будем хранить ее описание. Это означает, что со стороны книги связь обязательная. Что касается сущности «Экземпляры», то не может существовать в библиотеке ни одного экземпляра, который бы не относился к конкретной книге, поэтому и со стороны «Экземпляры» связь тоже обязательная.

Теперь необходимо определить, как в нашей системе будет представлен читатель. Естественно предложить ввести для этого сущность «Читатели», каждый экземпляр которой будет соответствовать конкретному читателю. В библиотеке каждому читателю присваивается уникальный номер читательского билета, который будет однозначно идентифицировать нашего читателя. Номер читательского билета будет ключевым атрибутом сущности «Читатели». Кроме того, в сущности «Читатели» должны присутствовать дополнительные атрибуты, которые требуются для решения поставленных задач, этими атрибутами будут: «Фамилия, Имя, Отчество», «Адрес читателя», «Телефон домашний» и «Телефон рабочий». Если существует ограничение на возраст читателей, в сущности «Читатели» надо ввести обязательный атрибут «Дата рождения», который позволит нам контролировать возраст наших читателей.

Из описания предметной области мы знаем, что каждый читатель может держать на руках несколько экземпляров книг. Для отражения этой ситуации нам надо провести связь между сущностями «Читатели» и «Экземпляры». А почему не между сущностями «Читатели» и «Книги»? Потому что читатель берет из библиотеки конкретный экземпляр конкретной книги, а не просто книгу. А как же узнать, какая книга у данного читателя? А это можно будет узнать по дополнительной связи между сущностями «Экземпляры» и «Книги», и эта связь каждому экземпляру ставит в соответствие одну книгу, поэтому мы в любой момент можем однозначно определить, какие книги находятся на руках у читателя, хотя связываем с читателем только инвентарные номера взятых книг. Между сущностями «Читатели» и «Экземпляры» установлена связь «один ко многим», и при этом она необязательная с двух сторон. Читатель в данный момент может не держать ни одной книги на руках, а с другой стороны, данный экземпляр книги может не находиться ни у одного читателя.

Теперь нам надо отразить последнюю сущность, которая связана с системным каталогом. Системный каталог содержит перечень всех областей

знаний, сведения по которым содержатся в библиотечных книгах. Название области знаний может быть длинным и состоять из нескольких слов, поэтому для моделирования системного каталога мы введем сущность «Системный каталог» с двумя атрибутами: «Код области знаний» и «Название области знаний». Атрибут «Код области знаний» будет ключевым атрибутом сущности.

Из описания предметной области нам известно, что каждая книга может содержать сведения из нескольких областей знаний, а с другой стороны, из практики известно, что в библиотеке может присутствовать множество книг, относящихся к одной и той же области знаний, поэтому нам необходимо установить между сущностями «Системный каталог» и «Книги» связь «многие ко многим», обязательную с двух сторон. Инфологическая модель предметной области «Библиотека» представлена на рис. 15.

## 5.5. Переход к реляционной модели данных

Инфологическая модель используется на ранних стадиях разработки проекта. Если понимать язык условных обозначений, которые соответствуют категориям ER-модели, то ее можно легко «читать», следовательно, она доступна для анализа программистам-разработчикам, которые будут разрабатывать отдельные приложения. Она имеет однозначную интерпретацию, и поэтому здесь не может быть никакого недопонимания со стороны разработчиков.

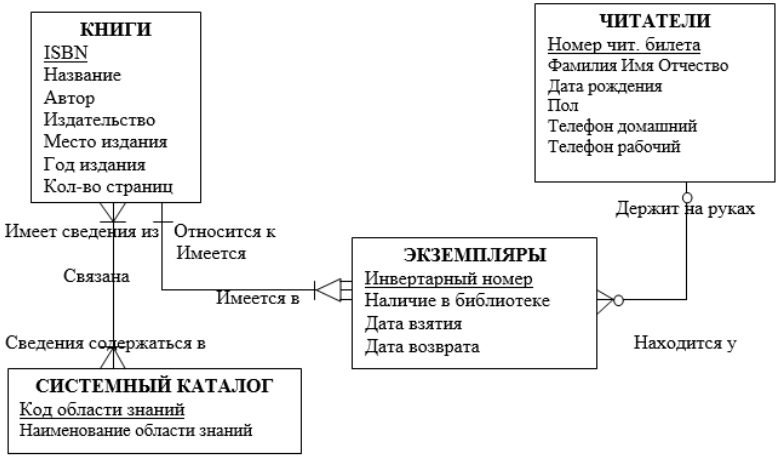


Рис. 15. *Инфологическая модель «Библиотека»*

Все специалисты предпочитают выражать свои мысли на некотором формальном языке, который обеспечивает однозначную их трактовку. Таким языком для программистов раньше был язык алгоритмов. Любой алгоритм имел однозначную интерпретацию. Он мог быть реализован на разных языках программирования, но сам алгоритм был и оставался одним и тем же. Для описания алгоритмов могли использоваться разные формализмы. Одним из таких формализмов был метаязык, в котором использовались слова на естественном языке и каждый мог прочесть эти слова, но смысл самого алгоритма мог понять только тот, кто владел знаниями трактовки алгоритмов.

Вот таким условным общепринятым языком описания базы данных и стал язык ER-модели. Для ER-модели существует алгоритм однозначного преобразования ее в реляционную модель данных, что позволило в дальнейшем разработать множество инструментальных систем, поддерживающих процесс разработки информационных систем, базирующихся на технологии баз данных.

Рассмотрим правила преобразования ER-модели в реляционную.

1. Каждой сущности ставится в соответствие отношение реляционной модели данных. При этом имена сущности и отношения могут быть различными, потому что на имена сущностей могут не накладываться дополнительные синтаксические ограничения, кроме уникальности имени в рамках модели. Имена отношений могут быть ограничены требованиями конкретной СУБД, чаще всего эти имена являются идентификаторами в некотором базовом языке, они ограничены по длине и не должны содержать пробелов и некоторых специальных символов. Например, сущность может быть названа «Книжный каталог», а соответствующее ей отношение желательно назвать, например, BOOKS (без пробелов и латинскими буквами).

2. Каждый атрибут сущности становится атрибутом соответствующего отношения. Переименование атрибутов должно происходить в соответствии с теми же правилами, что и переименование отношений в п. 1. Для каждого атрибута задается конкретный допустимый в СУБД тип данных и обязательность или необязательность данного атрибута (то есть допустимость или недопустимость NULL значений для него).

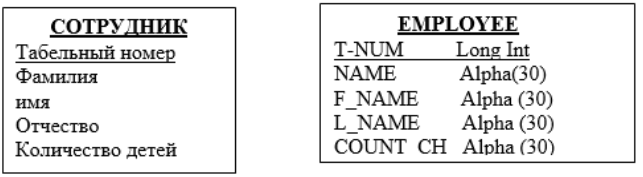


Рис. 16. *Преобразование сущности СОТРУДНИК к отношению EMPLOYEE*

| Column Code | Type |
|---|---|
| COUNT_CH | Short (30) Null |
| F_NAME | Alpha (30) Not Null |
| L_NAME | Alpha (30) Null |
| NAME | Alpha (30) Null |
| T_NUM | Long Int Not Null |

Рис. 17. *Свойства атрибутов отношения EMPLOYEE*

Первичный ключ сущности становится PRIMARY KEY соответствующего отношения. Атрибуты, входящие в первичный ключ отношения, автоматически получают свойство обязательности (NOT NULL).

4. В каждое отношение, соответствующее подчиненной сущности, добавляется набор атрибутов основной сущности, являющейся первичным ключом основной сущности. В отношении, соответствующем подчиненной сущности, этот набор атрибутов становится внешним ключом (FOREING KEY).

5. Для моделирования необязательного типа связи на физическом уровне у атрибутов, соответствующих внешнему ключу, устанавливается свойство допустимости неопределенных значений (признак NULL). При обязательном типе связи атрибуты получают свойство отсутствия неопределенных значений (признак NOT NULL).

6. Для отражения категоризации сущностей при переходе к реляционной модели возможны несколько вариантов представления:

а) возможно создать только одно отношение для всех подтипов одного супертипа. В него включают все атрибуты всех подтипов. Однако тогда для ряда экземпляров ряд атрибутов не будет иметь смысла. И даже если они будут иметь неопределенные значения, то потребуются дополнительные правила различения одних подтипов от других. Достоинством такого представления является то, что создается всего одно отношение;

б) при втором способе для каждого подтипа и для супертипа создаются свои отдельные отношения. Недостатком такого способа представления является то, что создается много отношений, однако достоинств у такого способа больше, так как вы работаете только со значимыми атрибутами подтипа. Кроме того, для возможности переходов к подтипам от супертипа необходимо в супертип включить идентификатор связи.

7. Дополнительно при описании отношения между типом и подтипами необходимо указать тип дискриминатора. Дискриминатор может быть взаимоисключающим (M/E, mutually exclusive) или нет. Если установлен данный тип дискриминатора, то это значит, что один экземпляр сущности супертипа связан только с одним экземпляром сущности подтипа и для каждого экземпляра сущности супертипа существует потомок. Кроме того, необходимо

указать для второго способа, наследуется ли только идентификатор супертипа в подтипы или наследуются все атрибуты супертипа.

8. Если задать наследование только идентификатора, то получим следующее преобразование (рис. 18). Наследование всех атрибутов суперсущности (рис. 19).

Разрешение связей типа «многие ко многим». Так как в реляционной модели данных поддерживаются между отношениями только связи типа «один ко многим», а в ER-модели допустимы связи «многие ко многим», то необходим специальный механизм преобразования, который позволит отразить множественные связи, неспецифические для реляционной модели, с помощью допустимых для нее категорий. Это делается введением специального дополнительного отношения, которое связано с каждым исходным связью «один ко многим», атрибутами этого отношения являются первичные ключи связываемых отношений.

Так, например, в схеме «Библиотека» присутствует связь такого типа между сущностью «Книги» и «Системный каталог». Для разрешения этой неспецифической связи при переходе к реляционной модели должно быть введено специальное дополнительное отношение, которое имеет всего два атрибута: ISBN (шифр книги) и KOD (код области знаний). При этом каждый из атрибутов нового отношения является внешним ключом (FOREING KEY), а вместе они образуют первичный ключ (PRIMARY KEY) новой связующей сущности.

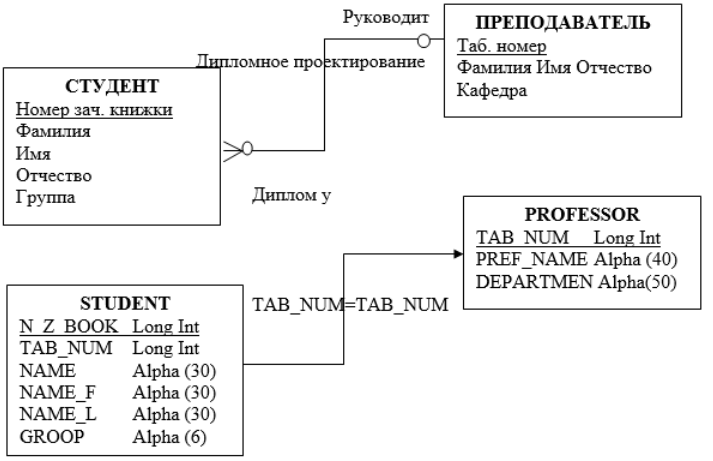


Рис. 18. *Преобразование взаимосвязанных сущностей СТУДЕНТ и ПРЕПОДАВАТЕЛЬ к взаимосвязанным отношениям STUDENT и PROFESSOR*

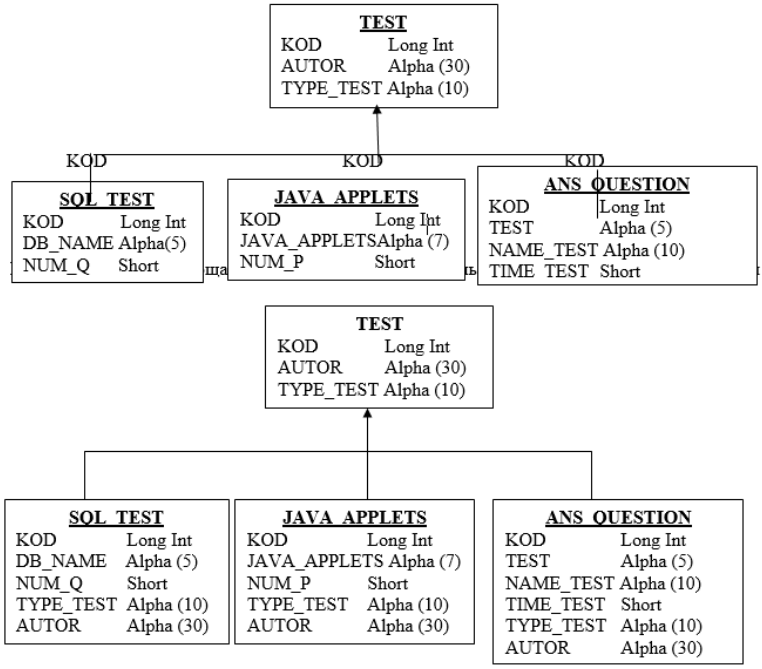


Рис. 19. *Результирующая модель с наследованием всех атрибутов суперсущности*

Теория нормализации применима и к модели «сущность-связь». Поэтому нормализацию можно проводить и на уровне инфологической (семантической) модели, смысл ее аналогичен нормализации реляционной модели.

# Глава 6. ПРИНЦИПЫ ПОДДЕРЖКИ ЦЕЛОСТНОСТИ В РЕЛЯЦИОННОЙ МОДЕЛИ ДАННЫХ

Одним из основополагающих понятий в технологии баз данных является понятие *целостности*. В общем случае это понятие прежде всего связано с тем, что база данных отражает в информационном виде некоторый объект реального мира или совокупность взаимосвязанных объектов реального мира. В реляционной модели объекты реального мира представлены в виде совокупности взаимосвязанных отношений. *Под целостностью будем понимать соответствие информационной модели предметной области, хранимой в базе данных, объектам реального мира и их взаимосвязям в каждый момент времени.* Любое изменение в предметной области, значимое для построенной модели, должно отражаться в базе данных, и при этом должна сохраняться однозначная интерпретация информационной модели в терминах предметной области.

## Общие понятия и определения целостности

Поддержка целостности в реляционной модели данных в ее классическом понимании включает в себя 3 аспекта.

Во-первых, это поддержка *структурной целостности*, которая трактуется как то, что реляционная СУБД должна допускать работу только с однородными структурами данных типа «реляционное отношение». При этом понятие «реляционного отношения» должно удовлетворять всем ограничениям, накладываемым на него в классической теории реляционной БД (отсутствие дубликатов кортежей, соответственно обязательное наличие первичного ключа, отсутствие понятия упорядоченности кортежей).

Во-вторых, это поддержка *языковой целостности*, которая состоит в том, что реляционная СУБД должна обеспечивать языки описания и манипулирования данными не ниже стандарта SQL. Не должны быть доступны иные низкоуровневые средства манипулирования данными, не соответствующие стандарту.

Именно поэтому доступ к информации, хранимой в базе данных, и любые изменения этой информации могут быть выполнены только с использованием операторов языка SQL.

В-третьих, это поддержка *ссылочной целостности* (Declarative Referential Integrity, DRI), что означает обеспечение одного из заданных принципов взаимосвязи между экземплярами кортежей взаимосвязанных отношений:

• кортежи подчиненного отношения уничтожаются при удалении кортежа основного отношения, связанного с ними;

• кортежи основного отношения модифицируются при удалении кортежа вспомогательного отношения, связанного с ними, при этом на месте ключа родительского отношения ставится неопределенное Null значение.

Ссылочная целостность обеспечивает поддержку непротиворечивого состояния БД в процессе модификации данных при выполнении операций добавления или удаления.

Кроме указанных ограничений целостности, которые в общем виде не определяют семантику БД, вводится понятие *семантической поддержки целостности*.

Структурная, языковая и ссылочная целостность определяют правила работы СУБД с реляционными структурами данных. Требования поддержки этих трех видов целостности говорят о том, что каждая СУБД должна уметь это делать, а разработчики должны это учитывать при построении баз данных с использованием реляционной модели. Но с другой стороны, эти аспекты никак не касаются содержания базы данных. Для определения некоторых ограничений, которые связаны с содержанием базы данных, требуются другие методы. Именно эти методы и сведены в поддержку семантической целостности.

Рассмотрим конкретный пример. То, что мы можем построить схему базы данных или ее концептуальную модель только из совокупности нормализованных таблиц, определяет структурную целостность. И мы построили нашу схему библиотеки из пяти взаимосвязанных отношений. Но мы не можем с помощью перечисленных трех методов поддержки целостности обеспечить ряд правил, которые определены в нашей предметной области и должны в ней соблюдаться. К таким правилам могут быть отнесены следующие:

1. В библиотеке должны быть записаны читатели не моложе 17 лет.

2. В библиотеке присутствуют книги, изданные начиная с 1960 по текущий год.

3. Каждый читатель может держать на руках не более 5 книг.

4. Каждый читатель при регистрации в библиотеке должен дать телефон для связи: он может быть рабочим или домашним.

Принципы семантической поддержки целостности как раз и позволяют обеспечить автоматическое выполнение тех условий, которые перечислены ранее.

Семантическая поддержка может быть обеспечена двумя путями: декларативным и процедурным.

*Декларативный путь* связан с наличием механизмов в рамках СУБД, обеспечивающих проверку и выполнение ряда декларативно заданных правил-ограничений, называемых чаще всего «бизнес-правилами» (Business Rules) или *декларативными ограничениями целостности*.

Выделяются следующие виды *декларативных ограничений целостности:*

1. *Ограничения целостности атрибута: значение по умолчанию*, задание обязательности или необязательности значений (Null), задание условий на значения атрибутов.

Задание значения по умолчанию означает, что каждый раз при вводе новой строки в отношение, при отсутствии данных в указанном столбце этому атрибуту присваивается именно значение по умолчанию. Например, при вводе новых книг разумно в качестве значения по умолчанию для года издания задать значение текущего года.

2. *Ограничения целостности, задаваемые на уровне доменов*, при поддержке доменной структуры. Эти ограничения удобны, если в базе данных присутствуют несколько столбцов разных отношений, которые принимают значения из одного и того же множества допустимых значений. Некоторые СУБД поддерживают подобную доменную структуру, то есть разрешают определять отдельно домены, задавать тип данных для каждого домена и задавать соответственно ограничения в виде бизнес-правил для доменов. А для атрибутов задается их принадлежность тому или другому домену. Иногда доменная структура выражена неявно и в ряде СУБД применяется специальная терминология для этого. Так, например, в MS SQL Server 7.0 вместо понятия домена вводится понятие типа данных, определенных пользователем, но смысл этого типа данных фактически эквивалентен смыслу домена. В этом случае действительно удобно задать ограничение на значение прямо на уровне домена, тогда оно автоматически будет выполняться для всех атрибутов, которые принимают значения из этого домена. Если мы зададим это ограничение для каждого атрибута, входящего в домен, разве наша система будет работать неправильно? Нет, конечно, она будет работать правильно, но представьте себе, что у вас в организации изменились правила работы, которые выражены в виде декларативных ограничений на значения. Если это ограничение задано не на один столбец, то надо просматривать все отношения и во всех отношениях менять старое правило на новое. Не легче ли заменить его один раз в домене, а все атрибуты, которые принимают значения из этого домена, будут автоматически работать по новому правилу.

3. *Ограничения целостности, задаваемые на уровне отношения*. Некоторые семантические правила невозможно преобразовать в выражения, которые будут применимы только к одному столбцу. В примере с библиотекой не сможем выразить требование наличия по крайней мере одного телефонного номера для быстрой связи с читателем. Под телефоны отведены два столбца, это в некотором роде искусственно, но специально так сделано, чтобы показать другой тип ограничений. Каждый из атрибутов является в общем случае необязательным и может принимать неопределенные значения: необязательно должен быть задан и рабочий, и домашний телефон. Потребуем, чтобы из двух по крайней мере один телефон был бы задан обязательно.

4. *Ограничения целостности, задаваемые на уровне связи между отношениями:* задание обязательности связи, принципов каскадного удаления и каскадного изменения данных, задание поддержки ограничений по мощности связи. Эти виды ограничений могут быть выражены заданием обязательности или необязательности значений внешних ключей во взаимосвязанных отношениях.

Декларативные ограничения целостности относятся к ограничениям, которые являются немедленно проверяемыми. Есть ограничения целостности, которые являются откладываемыми. Эти ограничения целостности поддерживаются механизмом транзакций и триггеров.

# Глава 7. ФИЗИЧЕСКИЕ МОДЕЛИ БАЗ ДАННЫХ

Физические модели баз данных определяют способы размещения данных в среде хранения и способы доступа к этим данным, которые поддерживаются на физическом уровне. Исторически первыми системами хранения и доступа были файловые структуры и системы управления файлами (СУФ), которые фактически являлись частью операционных систем. СУБД создавала над этими файловыми моделями свою надстройку, которая позволяла организовать всю совокупность файлов таким образом, чтобы она работала как единое целое и получала централизованное управление от СУБД. Однако непосредственный доступ осуществлялся на уровне файловых команд, которые СУБД использовала при манипулировании всеми файлами, составляющими хранимые данные одной или нескольких баз данных.

Однако механизмы буферизации и управления файловыми структурами неприспособлены для решения задач собственно СУБД, эти механизмы разрабатывались просто для традиционной обработки файлов, и с ростом объемов хранимых данных они стали неэффективными для использования СУБД. Тогда постепенно произошел переход от базовых файловых структур к непосредственному управлению размещением данных на внешних носителях самой СУБД. И пространство внешней памяти уже выходило из-под владения СУФ и управлялось непосредственно СУБД. При этом механизмы, применяемые в файловых системах, перешли во многом и в новые системы организации данных во внешней памяти, называемые чаще страничными системами хранения информации.

## 7.1. Файловые структуры, используемые для хранения информации в базах данных

В каждой СУБД по-разному организованы хранение и доступ к данным, однако существуют некоторые файловые структуры, которые имеют общепринятые способы организации и широко применяются практически во всех СУБД.

В системах баз данных файлы и файловые структуры, которые используются для хранения информации во внешней памяти, можно классифицировать следующим образом (рис. 20).


Рис. 20. *Классификация файлов, используемых в системах баз данных*

С точки зрения пользователя, *файлом* называется поименованная линейная последовательность записей, расположенных на внешних носителях.

В связи с тем, что файл – это линейная последовательность записей, то всегда в файле можно определить текущую запись, предшествующую ей и следующую за ней. Всегда существует понятие первой и последней записи файла.

В соответствии с методами управления доступом различают устройства внешней памяти с *произвольной адресацией* (магнитные и оптические диски) и устройства с *последовательной адресацией* (магнитофоны, стримеры).

На устройствах с произвольной адресацией теоретически возможна установка головок чтения-записи в произвольное место мгновенно. Практически существует время позиционирования головки, но оно весьма мало по сравнению со временем считывания-записи.

В устройствах с последовательным доступом для получения доступа к некоторому элементу требуется «перемотать (пройти)» все предшествующие ему элементы информации. На устройствах с последовательным доступом вся память рассматривается как линейная последовательность информационных элементов.

Файлы с постоянной длиной записи, расположенные на устройствах прямого доступа (УПД), являются *файлами прямого доступа*.

В этих файлах физический адрес расположения нужной записи может быть вычислен по номеру записи (NZ).

Каждая файловая система СУФ – поддерживает некоторую иерархическую файловую структуру, включающую чаще всего неограниченное количество уровней иерархии в представлении внешней памяти.

Для каждого файла в системе хранится следующая информация:

• имя файла;
• тип файла (например, расширение или другие характеристики);
• размер записи;
• количество занятых физических блоков;
• базовый начальный адрес;
• ссылка на сегмент расширения;
• способ доступа (код защиты).

Для файлов с постоянной длиной записи адрес ее размещения с номером К может быть вычислен по формуле:

$$BA + (K - 1) \times LZ + 1,$$

где BA – базовый адрес, LZ – длина записи.

Если можно всегда определить адрес, на который необходимо позиционировать механизм считывания-записи, то устройства прямого доступа делают это практически мгновенно, поэтому для таких файлов чтение произвольной записи практически не зависит от ее номера. Файлы прямого доступа обеспечивают наиболее быстрый доступ к произвольным записям, и их использование считается наиболее перспективным в системах баз данных.

На устройствах последовательного доступа могут быть организованы файлы только последовательного доступа.

Файлы с переменной длиной записи всегда являются файлами последовательного доступа. Они могут быть организованы двумя способами:

Конец записи отличается специальным маркером.

| | | | | | |
|---|---|---|---|---|---|
| Запись 1 | x | Запись 2 | x | Запись 3 | x |

В начале каждой записи фиксируется ее длина.

| | | | | | |
|---|---|---|---|---|---|
| LZ1 | Запись 1 | LZ2 | Запись 2 | LZ3 | Запись 3 |

Здесь LZN – длина N-й записи.

Не всегда возможно хранить информацию в виде файлов прямого доступа, но главное – это то, что доступ по номеру записи в базах данных весьма неэффективен. Чаще всего в базах данных необходим поиск по первичному или возможному ключам, иногда необходима выборка по внешним ключам, но во всех этих случаях известно значение ключа, но неизвестен номер записи, который соответствует этому ключу.

При организации файлов прямого доступа в некоторых очень редких случаях возможно построение функции, которая по значению ключа однозначно вычисляет адрес (номер записи файла).

$$NZ – F(K),$$

где NZ – номер записи, K – значение ключа, F( ) – функция.

Функция F() при этом должна быть линейной, чтобы обеспечивать однозначное соответствие.

Однако далеко не всегда удается построить взаимно-однозначное соответствие между значениями ключа и номерами записей. В подобных случаях применяют различные методы хэширования (рандомизации) и создают специальные хэш-функции.

Суть методов хэширования состоит в том, что значения ключа (или некоторые его характеристики) используются для начала поиска, то есть вычисляется некоторая хэш-функция h(k) и полученное значение берется в качестве адреса начала поиска. Это означает, что не требуется полного взаимно-однозначного соответствия, но, с другой стороны, для повышения скорости ограничивается время этого поиска (количество дополнительных шагов) для окончательного получения адреса. Таким образом, допускается, что нескольким разным ключам может соответствовать одно значение хэш-функции (то есть один адрес). Подобные ситуации называются *коллизиями*. Значения ключей, которые имеют одно и то же значение хэш-функции, называются *синонимами*.

Поэтому при использовании хэширования как метода доступа необходимо принять два независимых решения:

- выбрать хэш-функцию;
- выбрать метод разрешения коллизий.

Существует множество различных стратегий разрешения коллизий, но для примера рассмотрим две – достаточно распространенные.

## 7.2. Стратегия разрешения коллизий с областью переполнения

Первая стратегия условно может быть названа стратегией с областью переполнения. При выборе этой стратегии область хранения разбивается на 2 части L (рис. 21):

- основную область;
- область переполнения.

Для каждой новой записи вычисляется значение хэш-функции, которое определяет адрес ее расположения, и запись заносится в основную область в соответствии с полученным значением хэш-функции.

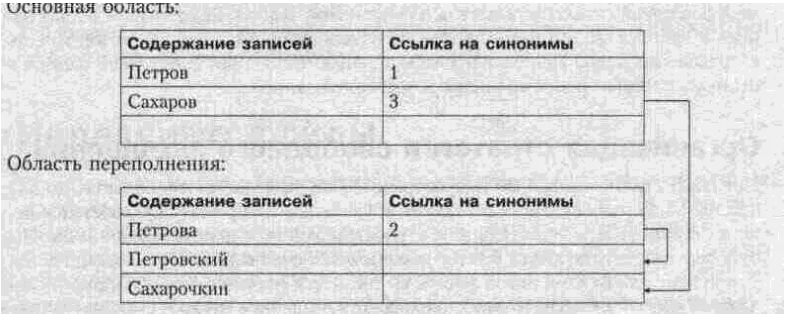


Рис. 21. *Область хранения*

Если вновь заносимая запись имеет значение функции хэширования такое же, которое использовала другая запись, уже имеющаяся в БД, то новая запись заносится в область переполнения на первое свободное место, а в записи-синониме, которая находится в основной области, делается ссылка на адрес вновь размещенной записи в области переполнения. Если же уже существует ссылка в записи-синониме, которая расположена в основной области, то тогда новая запись получает дополнительную информацию в виде ссылки и уже в таком виде заносится в область переполнения.

При таком алгоритме время размещения любой новой записи составляет не более двух обращений к диску, с учетом того, что номер первой свободной записи в области переполнения хранится в виде системной переменной.

Рассмотрим механизмы поиска произвольной записи и удаления записи для этой стратегии хэширования.

При поиске записи также сначала вычисляется значение ее хэш-функции и считывается первая запись в цепочке синонимов, которая расположена в основной области. Если искомая запись не соответствует первой в цепочке синонимов, то далее поиск происходит перемещением по цепочке синонимов, пока не будет обнаружена требуемая запись. Скорость поиска зависит от длины цепочки синонимов, поэтому качество хэш-функции определяется максимальной длиной цепочки синонимов. Хорошим результатом может считаться наличие не более 10 синонимов в цепочке.

При удалении произвольной записи сначала определяется ее место расположения. Если удаляемой является первая запись в цепочке синонимов, то после удаления на ее место в основной области заносится вторая (следующая) запись в цепочке синонимов, при этом все указатели (ссылки на синонимы) сохраняются.

Если же удаляемая запись находится в середине цепочки синонимов, необходимо провести корректировку указателей: в записи, предшествую-

щей удаляемой, в цепочке ставится указатель из удаляемой записи. Если это последняя запись в цепочке, то все равно механизм изменения указателей такой же, то есть в предшествующую запись заносится признак отсутствия следующей записи в цепочке, который ранее хранился в последней записи.

## 7.3. Организация стратегии свободного замещения

При этой стратегии файловое пространство не разделяется на области, но для каждой записи добавляется 2 указателя: указатель на предыдущую запись в цепочке синонимов и указатель на следующую запись в цепочке синонимов. Отсутствие соответствующей ссылки обозначается специальным символом, например, нулем. Для каждой новой записи вычисляется значение хэш-функции, и если данный адрес свободен, то запись попадает на заданное место и становится первой в цепочке синонимов. Если адрес, соответствующий полученному значению хэш-функции, занят, то по наличию ссылок определяется, является ли запись, расположенная по указанному адресу, первой в цепочке синонимов. Если да, то новая запись располагается на первом свободном месте и для нее устанавливаются соответствующие ссылки: она становится второй в цепочке синонимов, на нее ссылается первая запись, а она ссылается на следующую, если таковая есть.

Если запись, которая занимает требуемое место, не является первой записью в цепочке синонимов, значит, она занимает данное место «незаконно» и при появлении «законного владельца» должна быть «выселена», то есть перемещена на новое место. Механизм перемещения аналогичен занесению новой записи, которая уже имеет синоним, занесенный в файл. Для этой записи ищется первое свободное место и корректируются соответствующие ссылки: в записи, которая является предыдущей в цепочке синонимов для перемещаемой записи, заносится указатель на новое место перемещаемой записи, указатели же в самой перемещаемой записи остаются прежние.

После перемещения «незаконной» записи вновь вносимая запись занимает свое законное место и становится первой в новой цепочке синонимов.

Механизмы удаления записей во многом аналогичны механизмам удаления в стратегии с областью переполнения. Однако еще раз кратко опишем их.

Если удаляемая запись является первой записью в цепочке синонимов, то после удаления на ее место перемещается следующая (вторая) запись из цепочки синонимов и проводится соответствующая корректировка указателя третьей записи в цепочке синонимов, если таковая существует.

Если же удаляется запись, которая находится в середине цепочки синонимов, то производится только корректировка указателей: в предшествующей записи указатель на удаляемую запись заменяется указателем на следующую за удаляемой запись, а в записи, следующей за удаляемой, указатель на предыдущую запись заменяется на указатель на запись, предшествующую удаляемой.

## 7.4. Индексные файлы

Несмотря на высокую эффективность хэш-адресации, в файловых структурах далеко не всегда удается найти соответствующую функцию, поэтому при организации доступа по первичному ключу широко используются индексные файлы. Индексные файлы можно представить как файлы, состоящие из двух частей. Это необязательно физическое совмещение этих двух частей в одном файле, в большинстве случаев индексная область образует отдельный индексный файл, а основная область образует файл, для которого создается индекс.

Предполагаем, что сначала идет индексная область, которая занимает некоторое целое число блоков, а затем основная область, в которой последовательно расположены все записи файла.

В зависимости от организации индексной и основной областей различают 2 типа файлов: с *плотным индексом* и с *неплотным индексом*. Эти файлы имеют еще дополнительные названия, которые напрямую связаны с методами доступа к произвольной записи, которые поддерживаются данными файловыми структурами.

Файлы с плотным индексом называются также индексно-прямыми файлами, а файлы с неплотным индексом – индексно-последовательными файлами.

## 7.5. Файлы с плотным индексом, или индексно-прямые файлы

Рассмотрим *файлы с плотным индексом*. В этих файлах основная область содержит последовательность записей одинаковой длины, расположенных в произвольном порядке, а структура индексной записи в них имеет следующий вид:

| Значение ключа | Номер записи |
|---|---|

Здесь *значение ключа* – это значение первичного ключа, а номер записи – это 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 них не может быть двух записей, имеющих одинаковые значения первичного ключа. В индексных файлах с плотным индексом для каждой записи в основной области существует одна запись из индексной области. Все записи в индексной области упорядочены по значению ключа, поэтому можно применить более эффективные способы поиска в упорядоченном пространстве.

Длина доступа к произвольной записи оценивается не в абсолютных значениях, а в количестве обращений к устройству внешней памяти, которым обычно является диск. Именно обращение к диску является наиболее длительной операцией по сравнению со всеми обработками в оперативной памяти.

Наиболее эффективным алгоритмом поиска на упорядоченном массиве является логарифмический, или бинарный, поиск. Максимальное количество шагов поиска определяется двоичным логарифмом от общего числа элементов в искомом пространстве поиска:

$$T_n = {}_{log2}N,$$

где N – число элементов.

Однако в нашем случае является существенным только число обращений к диску при поиске записи по заданному значению первичного ключа. Поиск происходит в индексной области, где применяется двоичный алгоритм поиска индексной записи, а потом путем прямой адресации мы обращаемся к основной области уже по конкретному номеру записи. Для того чтобы оценить максимальное время доступа, нам надо определить количество обращений к диску для поиска произвольной записи.

На диске записи файлов хранятся в блоках. В одном блоке могут размещаться несколько записей. Поэтому нам надо определить количество индексных блоков, которое потребуется для размещения всех требуемых индексных записей, а потому максимальное число обращений к диску будет равно двоичному логарифму от заданного числа блоков плюс единица. Зачем нужна единица? После поиска номера записи в индексной области мы должны еще обратиться к основной области файла. Поэтому формула для вычисления максимального времени доступа в количестве обращений к диску выглядит следующим образом: $T_n = {}_{log2}N_{бл.инд.} + 1$.

Давайте рассмотрим конкретный пример и сравним время доступа при последовательном просмотре и при организации плотного индекса.

Допустим, что мы имеем следующие исходные данные:

Длина записи файла (LZ) – 128 байт. Длина первичного ключа (LK) – 12 байт. Количество записей в файле (KZ) – 100000. Размер блока (LB) – 1024 байт.

Рассчитаем размер индексной записи. Для представления целого числа в пределах 100000 нам потребуется 3 байта, можем считать, что у нас допустима только четная адресация, поэтому нам надо отвести 4 байта для хранения номера записи, тогда длина индексной записи будет равна сумме размера ключа и ссылки на номер записи, то есть:

LI = LK + 4 = 12 + 4 = 16 байт.

Определим количество индексных блоков, которое требуется для обеспечения ссылок на заданное количество записей. Для этого сначала определим, сколько индексных записей может храниться в одном блоке:

KIZB = LB/LI = 1024/16 = 64 индексных записи в одном блоке.

Теперь определим необходимое количество индексных блоков:

KIB = KZ/KZIB = 100000/64 = 1563 блока.

Мы округлили в большую сторону, потому что пространство выделяется целыми блоками, и последний блок у нас не будет заполнен.

А теперь мы уже можем вычислить максимальное количество обращений к диску при поиске произвольной записи:

$T_{поиска}$ = log2KIB + 1 = log2l563 + 1 = (10,61)11 + 1 = 12 обращений к диску.

Логарифм мы тоже округляем, так как считаем количество обращений, а оно должно быть целым числом.

Следовательно, для поиска произвольной записи по первичному ключу при организации плотного индекса потребуется не более 12 обращений к диску. А теперь оценим, какой выигрыш мы получаем, ведь организация индекса связана дополнительными накладными расходами на его поддержку, поэтому такая организация может быть оправдана только в том случае, когда она действительно дает значительный выигрыш. Если бы мы не создавали индексное пространство, то при произвольном хранении записей в основной области нам бы в худшем случае было необходимо просмотреть все блоки, в которых хранится файл, временем просмотра записей внутри блока мы пренебрегаем, так как этот процесс происходит в оперативной памяти.

Количество блоков, которое необходимо для хранения всех 100000 записей, мы определим по следующей формуле:

KBO = KZ/(LB/LZ) = 100000/(1024/128) = 12500 блоков.

И это означает, что максимальное время доступа равно 12500 обращений к диску. Действительно, выигрыш существенный.

Рассмотрим, как осуществляются операции добавления и удаления новых записей.

При операции добавления осуществляется запись в конец основной области. В индексной области необходимо произвести занесение информации в конкретное место, 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. 22).

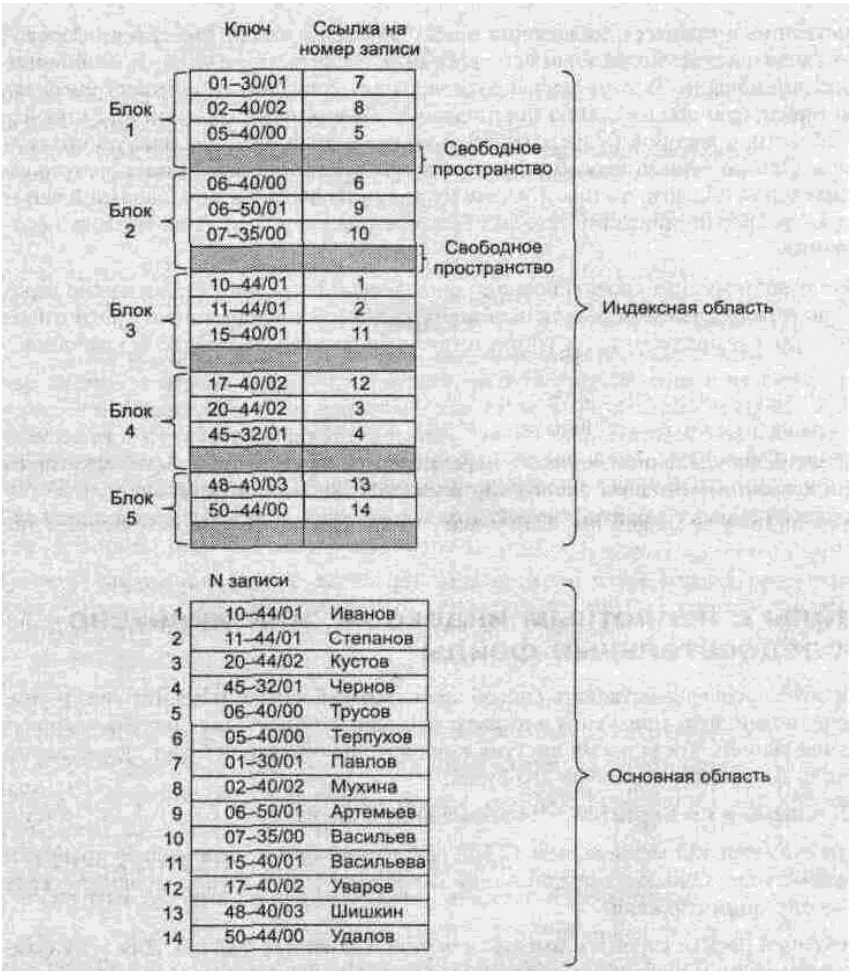


Рис. 22. *Пример организации файла с плотным индексом*

После определения блока, в который должен быть занесен индекс, этот блок копируется в оперативную память, там он модифицируется путем вставки в нужное место новой записи (благо, в оперативной памяти это делается на несколько порядков быстрее, чем на диске) и, измененный, записывается обратно на диск.

Определим максимальное количество обращений к диску, которое требуется при добавлении записи, – это количество обращений, необходимое для поиска записи плюс одно обращение для занесения измененного индексного блока и плюс одно обращение для занесения записи в основную область.

$$T_{\text{добавления}} = \log 2N + 1 + 1 + 1.$$

Естественно, в процессе добавления новых записей процент расширения постоянно уменьшается. Когда исчезает свободная область, возникает переполнение индексной области. В этом случае возможны два решения: либо перестроить индексную область, либо организовать область переполнения для индексной области, в которой будут храниться не поместившиеся в основную область записи. Однако первый способ потребует дополнительного времени на перестройку индексной области, а второй увеличит время на доступ к произвольной записи и потребует организации дополнительных ссылок в блоках на область переполнения.

Именно поэтому при проектировании физической базы данных очень 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 (отсутствующая), в индексной области соответствующий индекс уничтожается физически, то есть записи, 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.

## 7.6. Файлы с неплотным индексом, или индексно-последовательные файлы

Попробуем усовершенствовать способ хранения файла: будем хранить его в упорядоченном виде и применим алгоритм двоичного поиска для доступа к произвольной записи. Тогда время доступа к произвольной записи будет существенно меньше. Для нашего примера это будет:

T = log2КВО = log212500 = 14 обращений к диску.

И это существенно меньше, чем 12 500 обращений при произвольном хранении записей файла. Однако и поддержание основного файла в упорядоченном виде также операция сложная.

Неплотный индекс строится именно для упорядоченных файлов. Для них используется принцип внутреннего упорядочения для уменьшения количества хранимых индексов. Структура записи индекса для таких файлов имеет следующий вид:

| Значение ключа первой записи блока | Номер блока с этой записью |
|---|---|

В индексной области мы теперь ищем нужный блок по заданному значению первичного ключа. Так как все записи упорядочены, то значение первой записи блока позволяет нам быстро определить, в каком блоке находится искомая запись. Все остальные действия происходят в основной области. На рис. 23 представлен пример заполнения основной и индексной областей, если первичным ключом являются целые числа.

Время сортировки больших файлов весьма значительно, но поскольку файлы поддерживаются сортированными с момента их создания, накладные расходы в процессе добавления новой информации будут гораздо меньше.

Оценим время доступа к произвольной записи для файлов с неплотным индексом. Алгоритм решения задачи аналогичен.

Сначала определим размер индексной записи. Если ранее ссылка рассчитывалась исходя из того, что требовалось ссылаться на 100000 записей, то теперь нам требуется ссылаться всего на 12 500 блоков, поэтому для ссылки достаточно двух байт. Тогда длина индексной записи будет равна:

LI = LK + 2 – 14 + 2 = 14 байт.

Количество индексных записей в одном блоке будет равно:

KIZB = LB/LI – 1024/14 = 73 индексные записи в одном блоке.

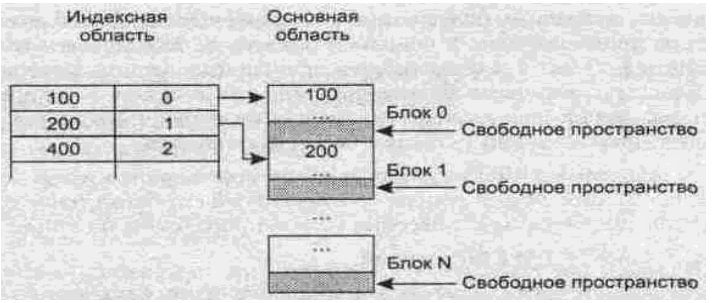


Рис. 23. *Заполнение основной и индексной областей*

Определим количество индексных блоков, которое необходимо для хранения требуемых индексных записей:

KIB = KBO/KZIB = 12500/73 = 172 блока.

Тогда время доступа по прежней формуле будет определяться:

$T_{поиска}$ = log2K IB + 1 = log2l72 + 1=8 + 1 = 9 обращений к диску.

Мы видим, что при переходе к неплотному индексу время доступа уменьшилось практически в полтора раза. Поэтому можно признать, что организация неплотного индекса дает выигрыш в скорости доступа.

Рассмотрим процедуры добавления и удаления новой записи при подобном индексе.

Здесь механизм включения новой записи принципиально отличен от ранее рассмотренного. Новая запись должна заноситься сразу в требуемый блок на требуемое место, которое определяется заданным принципом упорядоченности на множестве значений первичного ключа. Поэтому сначала ищется требуемый блок основной памяти, в который надо поместить новую запись, а потом этот блок считывается, затем в оперативной памяти корректируется содержимое блока и он снова записывается на диск на старое место. Здесь, так же как и в первом случае, должен быть задан процент первоначального заполнения блоков, но только применительно к основной области. В MS SQL server этот процент называется Full-factor и используется при формировании кластеризованных индексов. Кластеризованными называются как раз индексы, в которых исходные записи физически упорядочены по значениям первичного ключа. При внесении новой записи индексная область не корректируется.

Количество обращений к диску при добавлении новой записи равно количеству обращений, необходимых для поиска соответствующего блока плюс одно обращение, которое требуется для занесения измененного блока на старое место.

$T_{добавления}$ = log2N + 1 + 1 обращений.

Уничтожение записи происходит путем ее физического удаления из основной области, при этом индексная область обычно не корректируется, даже если удаляется первая запись блока. Поэтому количество обращений к диску при удалении записи такое же, как и при добавлении новой записи.

## 7.7. Организация индексов в виде B-tree (B-деревьев)

Построение B-деревьев связано с простой идеей построения индекса над уже построенным индексом. Действительно, если мы построим неплотный индекс, то сама индексная область может быть рассмотрена нами как

основной файл, над которым надо снова построить неплотный индекс, а потом снова над новым индексом строим следующий и так до того момента, пока не останется всего один индексный блок.

Мы в общем случае получим некоторое дерево, 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 (balanced) именно потому, что путь от корня до любого листа в этом древе одинаков. Именно термин «сбалансированное» от английского «balanced» – «сбалансированный, взвешенный» и дал название данному методу организации индекса.

Построим подобное дерево для нашего примера и рассчитаем для него количество уровней и, соответственно, количество обращений к диску.

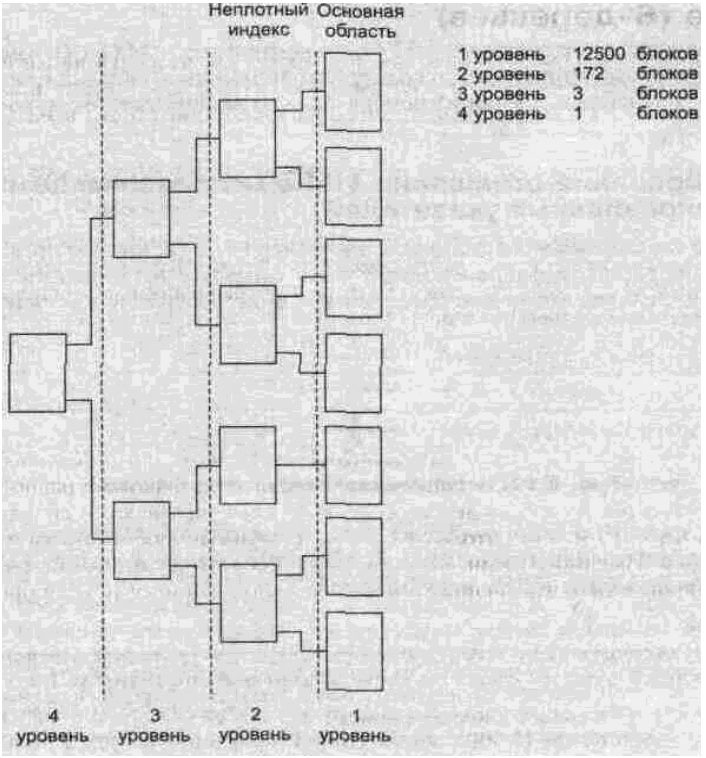


Рис. 24. *Построенное B-дерево*

На первом уровне число блоков равно числу блоков основной области, это нам известно, – оно равно 12500 блоков. Второй уровень образуется из неплотного индекса, его тоже уже строили и вычислили, что количество блоков индексной области в этом случае равно 172 блокам. А теперь над этим вторым уровнем снова построим неплотный индекс.

Не меняя длину индексной записи, считаем ее прежней, равной 14 байтам. Количество индексных записей в одном блоке нам тоже известно, и оно равно 73. Поэтому сразу определим, сколько блоков необходимо для хранения ссылок на 172 блока.

$KIB_3 = KIB_2/KZIB = 172/73 = 3$ блока

Снова округляем в большую сторону, потому что последний, третий, блок будет заполнен не полностью.

И над третьим уровнем строим новый, на котором будет всего один блок, имеющий три записи. Поэтому число уровней в построенном дереве равно четырем, и соответственно количество обращений к диску для доступа к произвольной записи равно четырем. Это не максимально возможное число обращений, а всегда одно и то же, одинаковое для доступа к любой записи.

$T_д = R_{уровн} = 4$

Механизм добавления и удаления записи при организации индекса в виде B-дерева аналогичен механизму, применяемому в случае с неплотным индексом.

И наконец, последнее, что хотелось бы прояснить, – это наличие вторых названий для плотного и неплотного индексов.

В случае плотного индекса после определения местонахождения искомой записи доступ к ней осуществляется прямым способом по номеру записи, поэтому этот способ организации индекса и называется индексно-прямым.

В случае неплотного индекса после нахождения блока, в котором расположена искомая запись, поиск внутри блока требуемой записи происходит последовательным просмотром и сравнением всех записей блока. Поэтому способ индексации с неплотным индексом называется еще и индексно-последовательным.

# Глава 8. МОДЕЛИРОВАНИЕ ОТНОШЕНИЙ «ОДИН-КО-МНОГИМ» НА ФАЙЛОВЫХ СТРУКТУРАХ

Отношение иерархии является типичным для баз данных, поэтому моделирование иерархических связей является типичным для физических моделей баз данных.

Для моделирования отношений 1:M (один ко многим) и M:M (многие ко многим) на файловых структурах используется принцип организации цепочек записей внутри файла и ссылки на номера записей для нескольких взаимосвязанных файлов.

## 8.1. Моделирование отношения 1:M с использованием однонаправленных указателей

В этом случае связываются два файла, например F1 и F2, причем предполагается, что одна запись в файле F1 может быть связана с несколькими записями в файле F2. Условно это можно представить в виде, изображенном на рис. 25.

При этом файл F1 в этом комплексе условно называется «Основным», а файл F2 – «зависимым» или «подчиненным». Структура основного файла может быть условно представлена в виде трех областей.

«Основной файл» F1.

| Ключ | Запись | Ссылка-указатель на первую запись в «Подчиненном» файле, с которой начинается цепочка записей файла F2, связанных с данной записью файла F1 |
|---|---|---|

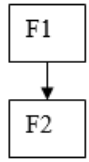


Рис. 25. *Иерархическая связь между файлами*

В подчиненном файле также к каждой записи добавляется специальный указатель, в нем хранится номер записи, которая является следующей в цепочке записей «подчиненного» файла, связанной с одной записью «основного» файла.

Таким образом, каждая запись «подчиненного файла» делится на две области: область указателя и область, содержащую собственно запись.

Структура записи «подчиненного» файла.

| Указатель на следующую запись в цепочке | Содержимое записи |
|---|---|

В качестве примера рассмотрим связь между преподавателями и занятиями, которые эти преподаватели проводят. В файле F1 приведен список преподавателей, а в файле F2 – список занятий, которые они ведут (рис. 26).

В этом случае содержимое двух взаимосвязанных файлов F1 и F2 может быть расшифровано следующим образом: первая запись в файле F1 связана с цепочкой записей файла F2, которая начинается с записи номер 1, следующая запись номер 4 и последняя запись в цепочке – запись номер 5. Последняя – потому что пятая запись не имеет ссылки на следующую запись в цепочке. Аналогично можно расшифровать и остальные связи. Если мы проведем интерпретацию данных связей на уровне предметной области, то можно утверждать, что преподаватель Иванов ведет предмет «Вычислительные сети» в группе 4306, «Моделирование» в группе 84305 и «Вычислительные сети» в группе 4309.

Аналогично могут быть расшифрованы и остальные взаимосвязанные записи.

**F1**

| Номер записи | Ключ и остальная запись | Указатель |
|---|---|---|
| 1 | Иванов И. Н. ... | 1 |
| 2 | Петров А. А. | 3 |
| 3 | Сидоров П. А. | 2 |
| 4 | Яковлев В. В. | |

**F2**

| Номер записи | Указатель на следующую запись в цепочке | Содержимое записи |
|---|---|---|
| 1 | 4 | *4306 Вычислительные сети* |
| 2 | – | 4307 Контроль и диагностика |
| 3 | 6 | 4308 Вычислительные сети |
| 4 | 5 | 84305 Моделирование |
| 5 | – | 4309 Вычислительные сети |
| 6 | – | 84405 Техническая диагностика |
| 7 | – | |

Рис. 26. *Связь преподавателей и предметов*

## 8.2. Алгоритм нахождения нужных записей «подчиненного» файла

**Шаг 1.** Ищется запись в «основном» файле в соответствии с его организацией (с помощью функции хэширования, или с использованием индексов, или другим образом). Если требуемая запись найдена, то переходим к шагу 2, в противном случае выводим сообщение об отсутствии записи основного файла.

**Шаг 2.** Анализируем указатель в основном файле. Если он пустой, то есть стоит прочерк, значит, для этой записи нет ни одной связанной с ней записи в «подчиненном файле», и выводим соответствующее сообщение, в противном случае переходим к шагу 3.

**Шаг 3.** По ссылке-указателю в найденной записи основного файла переходим прямым методом доступа по номеру записи на первую запись в цепочке «Подчиненного» файла. Переходим к шагу 4.

**Шаг 4.** Анализируем текущую запись на содержание. Если это искомая запись, то мы заканчиваем поиск, в противном случае переходим к шагу 5.

**Шаг 5.** Анализируем указатель на следующую запись в цепочке. Если он пуст, то выводим сообщение, что искомая запись отсутствует, и прекращаем поиск, в противном случае по ссылке-указателю переходим на следующую запись в «подчиненном файле» и снова переходим к шагу 4.

Использование цепочек записей позволяет эффективно организовывать модификацию взаимосвязанных файлов.

## 8.3. Алгоритм удаления записи из цепочки «подчиненного» файла

**Шаг 1.** Ищется удаляемая запись в соответствии с ранее рассмотренным алгоритмом. Единственным отличием при этом является обязательное сохранение в специальной переменной номера предыдущей записи в цепочке, допустим, это переменная NP.

**Шаг 2.** Запоминаем в специальной переменной указатель на следующую запись в найденной записи, например, заносим его в переменную NS. Переходим к шагу 3.

**Шаг 3.** Помечаем специальным символом, например, символом *звездочка* (*), найденную запись, то есть в позиции указателя на следующую запись в цепочке ставим символ «*» – это означает, что данная запись отсутствует, а место в файле свободно и может быть занято любой другой записью.

**Шаг 4.** Переходим к записи с номером, который хранится в NP, и заменяем в ней указатель на содержимое переменной NS.

Для того чтобы эффективно использовать дисковое пространство при включении новой записи в «подчиненный файл», ищется первое свободное

место, т. е. запись, помеченная символом «*», и на ее место заносится новая запись, после этого производится модификация соответствующих указателей. При этом необходимо различать 3 случая:

1. Добавление записи на первое место в цепочке.
2. Добавление записи в конец цепочки.
3. Добавление записи на заданное место в цепочке.

## 8.4. Инвертированные списки

Достаточно часто в базах данных требуется проводить операции доступа по вторичным ключам. Напомним, что вторичным ключом является набор атрибутов, которому соответствует набор искомых записей. Это означает, что существует множество записей, имеющих одинаковые значения вторичного ключа. Например, в случае нашей БД «Библиотека» вторичным ключом может служить место издания, год издания. Множество книг могут быть изданы в одном месте, и множество книг могут быть изданы в один год.

Для обеспечения ускорения доступа по вторичным ключам используются структуры, называемые инвертированными списками, которые послужили основой организации индексных файлов для доступа по вторичным ключам.

Инвертированный список в общем случае – это двухуровневая индексная структура. Здесь на первом уровне находится файл или часть файла, в которой упорядоченно расположены значения вторичных ключей. Каждая запись с вторичным ключом имеет ссылку на номер первого блока в цепочке блоков, содержащих номера записей с данным значением вторичного ключа. На втором уровне находится цепочка блоков, содержащих номера записей одним и тем же значением вторичного ключа. При этом блоки второго уровня упорядочены по значениям вторичного ключа.

И, наконец, на третьем уровне находится собственно основной файл.

Механизм доступа к записям по вторичному ключу при подобной организации записей весьма прост. На первом шаге мы ищем в области первого уровня заданное значение вторичного ключа, а затем по ссылке считываем блоки второго уровня, содержащие номера записей с заданным значением вторичного ключа, а далее уже прямым доступом загружаем в рабочую область пользователя содержимое всех записей заданным значением вторичного ключа.

На рис. 27 представлен пример инвертированного списка, составленного для вторичного ключа «Номер группы» в списке студентов некоторого учебного заведения. Для более наглядного представления мы ограничили размер блока пятью записями (целыми числами).

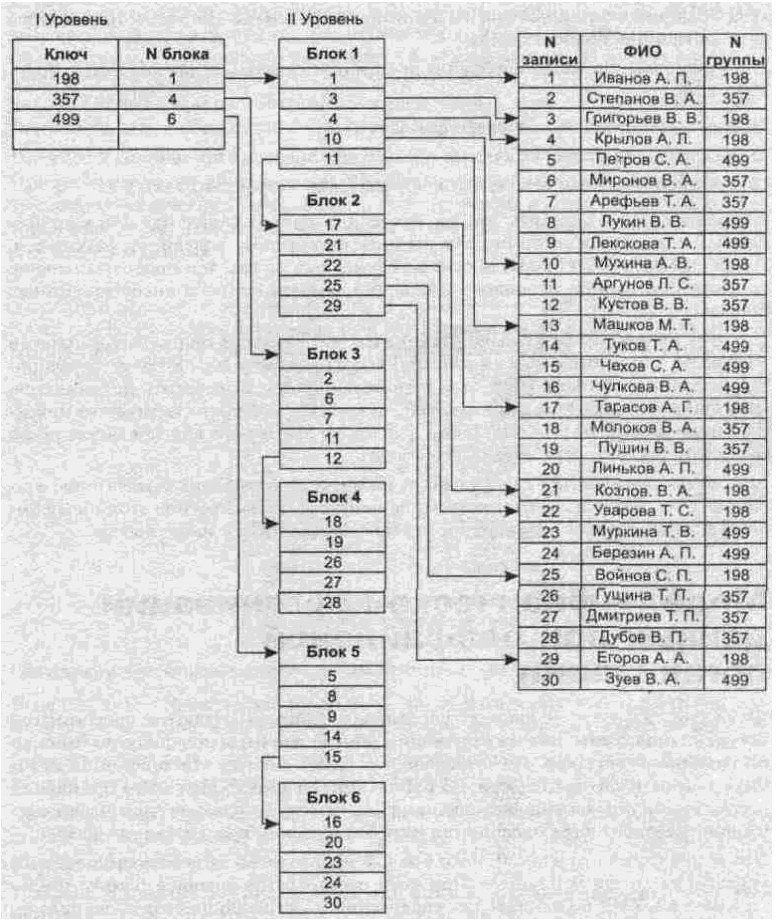


Рис. 27. *Построение инвертированного списка по номеру группы*

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

Следует отметить, что организация вторичных списков действительно ускоряет поиск записей с заданным значением вторичного ключа. Но рассмотрим вопрос модификации основного файла.

При модификации основного файла происходит следующая последовательность действий:

– изменяется запись основного файла;

– исключается старая ссылка на предыдущее значение вторичного ключа;

– добавляется новая ссылка на новое значение вторичного ключа.

При этом следует отметить, что два последних шага выполняются для всех вторичных ключей, по которым созданы инвертированные списки. И, разумеется, такой процесс требует гораздо больше временных затрат, чем просто изменение содержимого записи основного файла без поддержки всех инвертированных списков.

Не следует безусловно утверждать, что введение индексных файлов (в том числе и инвертированных списков) всегда ускоряет обработку информации в базе данных. Отнюдь, если база данных постоянно изменяется, дополняется, модифицируется содержимое записей, то наличие большого количества инвертированных списков или индексных файлов по вторичным ключам может резко замедлить процесс обработки информации.

Можно придерживаться следующей позиции: если база данных достаточно стабильна и ее содержимое практически не меняется, то построение вторичных индексов действительно может ускорить процесс обработки информации.

## 8.5. Модели физической организации данных при бесфайловой организации

Файловая структура и система управления файлами являются прерогативой операционной среды, поэтому принципы обмена данными подчиняются законам операционной системы. По отношению к базам данных эти принципы могут быть далеки от оптимальности. СУБД подчиняется несколько иным принципам и стратегиям управления внешней памятью, чем те, которые поддерживают операционные среды для большинства пользовательских процессов или задач.

Это и послужило причиной того, что СУБД взяли на себя непосредственное управление внешней памятью. При этом пространство внешней памяти предоставляется СУБД полностью для управления, а операционная среда не получает непосредственного доступа к этому пространству.

Физическая организация современных баз данных является наиболее закрытой, она определяется как коммерческая тайна для большинства поставщиков коммерческих СУБД. И здесь не существует никаких стандартов, поэтому в общем случае каждый поставщик создает свою уникальную структуру и пытается обосновать ее наилучшие качества по сравнению со своими конкурентами. Физическая организация является в настоящий момент наиболее динамичной частью СУБД. Стремительно расширяются возможность устройств внешней памяти, дешевеет оперативная память, увеличивается ее объем и поэтому изменяются сами принципы организа-

ции физических структур данных. И можно предположить, что и в дальнейшем эта часть современных СУБД будет постоянно меняться. Поэтому при рассмотрении моделей данных, используемых для физического хранения и обработки, мы коснемся только наиболее общих принципов и тенденций.

При распределении дискового пространства рассматриваются две схемы структуризации: физическая, которая определяет хранимые данные, и логическая, которая определяет некоторые логические структуры, связанные с концептуальной моделью данных (рис. 28).

Определим некоторые понятия, используемые в указанной классификации.

*Чанк* (chank) – представляет собой часть диска, физическое пространство на диске, которое ассоциировано одному процессу (online процессу обработки данных).

Чанком может быть назначено неструктурированное устройство, часть этого устройства блочно-ориентированное устройство или просто файл UNIX.

Чанк характеризуется маршрутным именем, смещением (от физического начала устройства до начальной точки на устройстве, которая используется как чанк), размером, заданным в Кбайтах или Мбайтах.

При использовании блочных устройств и файлов величина смещения считается равной нулю.

Логические единицы образуются совокупностью экстентов, то есть таблица моделируется совокупностью экстентов.

*Экстент* – это непрерывная область дисковой памяти.

Для моделирования каждой таблицы используется 2 типа экстентов: первый и последующие.

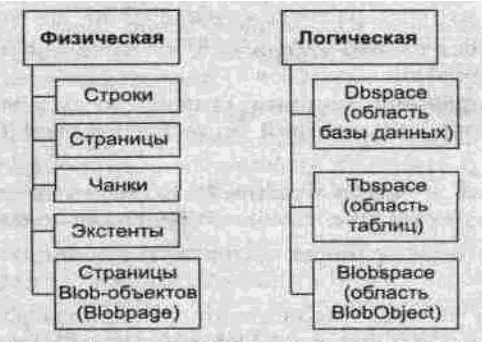


Рис. 28. *Классификация объектов при статичной организации физической модели данных*

Первый экстент задается при создании нового объекта (типа таблицы), его размер задается при создании. *EXTENTSIZE* – размер первого экстента, *NEXT SIZE* – размер каждого следующего экстента.

Минимальный размер экстента в каждой системе свой, но в большинстве случаев он равен 4 страницам, максимальный – 2 Гбайтам.

Новый экстент создается после заполнения предыдущего и связывается с ним специальной ссылкой, которая располагается на последней странице экстента. В ряде систем экстенты называются сегментами, но фактически эти понятия эквиваленты.

При динамическом заполнении БД данными применяется специальный механизм адаптивного определения размера экстентов.

Внутри экстента идет учет свободных страниц.

Между экстентами, которые располагаются друг за другом без промежутков, производится своеобразная операция конкатенации, которая просто увеличивает размер первого экстента.

*Механизм удвоения размера экстента:* если число выделяемых экстентов для процесса растет в пропорции, кратной 16, то размер экстента удваивается каждые 16 экстентов.

Например, если размер текущего экстента 16 Кбайт, то после заполнения 16 экстентов данного размера размер следующего будет увеличен до 32 Кбайт.

Совокупность экстентов моделирует логическую единицу – таблицу-отношение (tblspace).

Экстенты состоят из четырех типов страниц: страницы данных, страницы индексов, битовые страницы и страницы blob-объектов. Blob – это сокращение Binary Larg Object, и соответствует оно неструктурированным данным. В ранних СУБД такие данные относились к типу Memo. В современных СУБД к этому типу относятся неструктурированные большие текстовые данные, картинки, просто наборы машинных кодов. Для СУБД важно знать, что этот объект надо хранить целиком, что размеры этих объектов от записи к записи могут резко отличаться и этот размер в общем случае неограничен.

Основной единицей осуществления операций обмена (ввода-вывода) является страница данных. Все данные хранятся постранично. При табличном хранении данные на одной странице являются однородными, то есть страница может хранить только данные или только индексы.

Все страницы данных имеют одинаковую структуру.

*Слот* – это 4-байтовое слово, 2 байта соответствуют смещению строки на странице и 2 байта – длина строки. Слоты характеризуют размещение строк данных на странице. На одной странице хранится не более 255 строк. В базе данных каждая строка имеет уникальный идентификатор в рамках

всей базы данных, часто называемый RowID – номер строки, он имеет размер 4 байта и состоит из номера страницы и номера строки на странице. Под номер страницы отводится 3 байта, поэтому при такой идентификации возможна адресация к 16 777 215 страницам.

При упорядочении строк на страницах не происходит физического перемещения строк, все манипуляции происходят со слотами.

При переполнении страниц создается специальный вид страниц, называемых страницами остатка. Строки, не уместившиеся на основной странице, связываются (линкуются) со своим продолжением на страницах остатка с помощью ссылок-указателей «вперед» (то есть на продолжение), которые содержат номер страницы и номер слота на странице.

Страницы индексов организованы в виде B-деревьев. Страницы Blob предназначены для хранения слабоструктурированной информации, содержащей тексты большого объема, графическую информацию, двоичные коды. Эти данные рассматриваются как потоки байтов произвольного размера, в страницах данных делаются ссылки на эти страницы.

Битовые страницы служат для трассировки других типов страниц. В зависимости от трассируемых страниц битовые страницы строятся по 2-битовой или 4-битовой схеме. 4-битовые страницы служат для хранения сведений о столбцах типа Varchar, Byte, Text, для остальных типов данных используются 2-битовые страницы.

Битовая структура трассирует 32 страницы. Каждая битовая структура представлена двумя 4-байтными словами. Каждая i-я позиция описывает одну i-ю страницу. Сочетание разрядов в i-х позициях двух слов обозначает состояние данной страницы: ее тип и занятость.

При обработке данных СУБД организует специальные структуры в оперативной памяти, называемые разделяемой памятью, и специальные структуры во внешней памяти, называемые журналами транзакций. Разделяемая память служит для кэширования данных при работе с внешней памятью с целью сокращения времени доступа, кроме того, разделяемая память служит для эффективной поддержки режимов одновременной параллельной работы пользователей с базой данных.

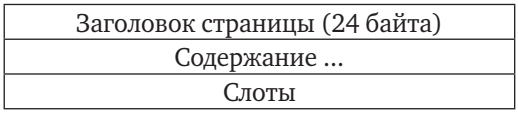


Рис. 29. *Обобщенная структура страницы данных*

# Глава 9. ЯЗЫК SQL. ФОРМИРОВАНИЕ ЗАПРОСОВ К БАЗЕ ДАННЫХ

## 9.1. История развития SQL

База данных бесполезна без инструментов для доступа к содержащейся в ней информации. Чтобы получить необходимые данные, пользователи отправляют запросы в систему управления базами данных (СУБД), которая их обрабатывает и возвращает результаты. Для этого используется специальный «язык запросов». В современных реляционных СУБД фактически стандартом такого языка является SQL.

SQL (Structured Query Language) – это структурированный язык запросов, предназначенный для работы с реляционными базами данных. Он появился после разработки реляционной алгебры, а его прототип был создан в конце 1970-х годов в исследовательском центре IBM. Первая реализация SQL была представлена в рамках проекта реляционной СУБД System R. В 1989 году был утверждён первый международный стандарт SQL (известный как SQL89 или SQL1), который поддерживается большинством существующих СУБД. Этот стандарт также называют ANSI/ISO.

В 1992 году появился новый, более детализированный и полный стандарт SQL92 (или SQL2). Производители СУБД постепенно адаптировали свои продукты к этому стандарту.

Следующее значительное обновление произошло в 1999 году, когда был принят стандарт SQL3. В него были добавлены новые типы данных, позволяющие создавать более сложные структуры, ориентированные на объектную модель. Также в этом стандарте впервые официально закреплены механизмы работы с событиями и триггерами, которые уже давно применялись в коммерческих системах. Триггеры теперь рассматриваются как сочетание события и соответствующего действия, которым может быть последовательность SQL-операторов или управляющие конструкции для контроля выполнения программы.

Кроме того, в SQL3 были внесены изменения в механизм управления транзакциями: вернулась поддержка точек сохранения (savepoints), а в команде отката ROLLBACK появилась возможность указания конкретной точки возврата. Это позволяет не откатывать всю транзакцию целиком, а вернуться к заранее сохранённому состоянию, что делает обработку данных более гибкой.

SQL нельзя отнести к классическим языкам программирования, поскольку в нём отсутствуют конструкции для управления выполнением программ, определения типов данных и другие элементы, характерные для традиционных языков. SQL представляет собой набор стандартных команд для взаимодействия с данными в базе. Эти команды можно встраивать в другие языки программирования, такие как C++, PL, COBOL и др. Также SQL-команды могут выполняться в интерактивном режиме.

Хотя большинство разработчиков придерживаются стандартов SQL, они часто дополняют его возможностями для специфической обработки данных. SQL настолько распространён, что даже некоторые нереляционные СУБД, такие как Adabas, включают поддержку SQL-интерфейса.

Запросы на языке SQL формируются с использованием различных операторов. Они могут разделяться либо переводом строки, либо символом точки с запятой.

## 9.2. Структура SQL

В отличие от реляционной алгебры, где были представлены только операции запросов к БД, SQL является полным языком, в нем присутствуют не только операции запросов, но и операторы, соответствующие DDL – Data Definition Language – языку описания данных. Кроме того, язык содержит операторы, предназначенные для управления (администрирования) БД.

*Таблица 2*

**Операторы определения данных DDL**

| Оператор | Смысл | Действие |
|---|---|---|
| CREATE TABLE | Создать таблицу | Создает новую таблицу в БД |
| DROP TABLE | Удалить таблицу | Удаляет таблицу из БД |
| ALTER TABLE | Изменить таблицу | Изменяет структуру существующей таблицы или ограничения целостности, задаваемые для данной таблицы |
| CREATE VIEW | Создать представление | Создает виртуальную таблицу, соответствующую некоторому SQL-запросу |
| ALTER VIEW | Изменить представление | Изменяет ранее созданное представление |
| DROP VIEW | Удалить представление | Удаляет ранее созданное представление |
| CREATE INDEX | Создать индекс | Создает индекс для некоторой таблицы для обеспечения быстрого доступа по атрибутам, входящим в индекс |
| DROP INDEX | Удалить индекс | Удаляет ранее созданный индекс |

*Таблица 3*

**Операторы манипулирования данными**
**Data Manipulation Language (DMP)**

| Оператор | Смысл | Действие |
|---|---|---|
| DELETE | Удалить строки | Удаляет одну или несколько строк, соответствующих условиям фильтрации, из базовой таблицы. Применение оператора согласуется с принципами поддержки целостности, поэтому этот оператор не всегда может быть выполнен корректно, даже если синтаксически он записан правильно |
| INSERT | Вставить строку | Вставляет одну строку в базовую таблицу. Допустимы модификации оператора, при которых сразу несколько строк могут быть перенесены из одной таблицы или запроса в базовую таблицу |
| UPDATE | Обновить строку | Обновляет значения одного или нескольких столбцов в одной или нескольких строках, соответствующих условиям фильтрации |

*Таблица 4*

**Язык запросов Data Query Language (DQL)**

| Оператор | Смысл | Действие |
|---|---|---|
| SELECT | Выбрать строки | Оператор, заменяющий все операторы реляционной алгебры и позволяющий сформировать результирующее отношение, соответствующее запросу |

*Таблица 5*

**Средства управления транзакциями**

| Оператор | Смысл | Действие |
|---|---|---|
| COMMIT | Завершить транзакцию | Завершить комплексную взаимосвязанную обработку информации, объединенную в транзакцию |
| ROLLBACK | Откатить транзакцию | Отменить изменения, проведенные в ходе выполнения транзакции |
| SAVEPOINT | Сохранить промежуточную точку выполнения транзакции | Сохранить промежуточное состояние БД, пометить его для того, чтобы можно было в дальнейшем к нему вернуться |

*Таблица 6*

**Средства администрирования данных**

| Оператор | Смысл | Действие |
|---|---|---|
| ALTER DATABASE | Изменить БД | Изменить набор основных объектов в базе данных, ограничений, касающихся всей базы данных |

| Оператор | Смысл | Действие |
|---|---|---|
| ALTER DBAREA | Изменить область хранения БД | Изменить ранее созданную область хранения |
| ALTER PASSWORD | Изменить пароль | Изменить пароль для всей базы данных |
| CREATE DATABASE | Создать БД | Создать новую базу данных, определив основные параметры для нее |
| CREATE DBAREA | Создать область хранения | Создать новую область хранения и сделать ее доступной для размещения данных |
| DROP DATABASE | Удалить БД | Удалить существующую базу данных (только в том случае, когда вы имеете право выполнить это действие) |
| DROP DBAREA | Удалить область хранения БД | Удалить существующую область хранения (если в ней на настоящий момент не располагаются активные данные) |
| GRANT | Предоставить права | Предоставить права доступа на ряд действий над некоторым объектом БД |
| REVOKE | Лишить прав | Лишить прав доступа к некоторому объекту или некоторым действиям над объектом |

*Таблица 7*

**Программный SQL**

| Оператор | Смысл | Действие |
|---|---|---|
| DECLARE | Определяет курсор для запроса | Задает некоторое имя и определяет связанный с ним запрос к БД, который соответствует виртуальному набору данных |
| OPEN | Открыть курсор | Формирует виртуальный набор данных, соответствующий описанию указанного курсора и текущему состоянию БД |
| FETCH | Считать строку из множества строк, определенных курсором | Считывает очередную строку, заданную параметром команды из виртуального набора данных, соответствующего открытому курсору |
| CLOSE | Закрыть курсор | Прекращает доступ к виртуальному набору данных, соответствующему указанному курсору |
| PREPARE | Подготовить оператор SQL к динамическому выполнению | Сгенерировать план выполнения запроса, соответствующего заданному оператору SQL |

| Оператор | Смысл | Действие |
|---|---|---|
| EXECUTE | Выполнить оператор SQL, ранее подготовленный к динамическому выполнению | Выполняет ранее подготовленный план запроса |

В коммерческих СУБД набор основных операторов расширен. В большинство СУБД включены операторы определения и запуска хранимых процедур и операторы определения триггеров.

## 9.3. Типы данных

Стандарт языка SQL определяет типы данных, которые можно использовать при создании БД и работе с ней. В табл. 8 перечислены основные типы данных, используемые в SQL, а также указаны соответствующие им типы языка С.

*Таблица 8*

| Тип данных SQL/92 | Описание | Тип языка С |
|---|---|---|
| CHARACTER | Строка символов фиксированной длины | char[] |
| INTEGER | Целое число | long |
| SMALLINT | Целое число | short |
| REAL | Число с плавающей запятой | float |
| DOUBLE PRECISION | Число с плавающей запятой двойной точности | double |

Стандарт SQL включает в себя определение нескольких специальных типов данных:

**MONEY** – используется для хранения денежных значений;

**DATE** и **TIME** – предназначены для работы с датами и временем;

**FLOAT, NUMERIC, DECIMAL** – числовые типы, позволяющие задавать масштаб и точность.

С помощью SQL можно создавать таблицы, содержащие данные любого из этих типов. Однако при работе с приложениями, написанными на традиционных языках программирования, таких как С или Pascal, могут возникнуть сложности, поскольку в этих языках отсутствуют многие типы данных, используемые в SQL. Для их корректного использования требуются специальные механизмы преобразования.

Кроме стандартных типов, коммерческие СУБД часто поддерживают дополнительные, не включённые в официальный стандарт. Например, практически все СУБД предлагают формат для хранения больших объёмов

неструктурированного текста. Этот тип данных можно сравнить с **MEMO**, используемым в настольных базах данных.

В первой версии стандарта SQL (SQL1) встроенные функции не были предусмотрены. Однако многие коммерческие СУБД реализовывали их самостоятельно. Впоследствии, в стандарте SQL2, был официально закреплён набор стандартных встроенных функций, которые стали частью языка.

## 9.4. Оператор выбора SELECT

В языке SQL для запросов к данным используется единственный оператор – **SELECT**, который охватывает все операции реляционной алгебры. Написание корректных SQL-запросов может быть сложным, особенно на первых этапах обучения. Один и тот же запрос можно реализовать разными способами, причем все они будут правильными, но могут существенно различаться по скорости выполнения. Это особенно важно при работе с крупными базами данных.

**Синтаксис оператора SELECT**

SELECT [ALL | DISTINCT] (<Список полей> | *)
FROM <Список таблиц>
[WHERE <Условие отбора или соединения>]
[GROUP BY <Список полей для группировки>]
[HAVING <Условие отбора для групп>]
[ORDER BY <Список полей для сортировки>]

• **ALL** – включает в итоговый результат все строки, соответствующие условиям запроса, даже если они повторяются. Это противоречит принципам реляционной алгебры, где дубликаты по умолчанию исключаются.

• **DISTINCT** – обеспечивает отбор только уникальных строк, исключая повторяющиеся.

• * (звёздочка) – означает, что в результат будут включены все столбцы из таблиц, указанных в запросе.

• **FROM** – определяет список таблиц, из которых берутся данные.

• **WHERE** – задаёт условия фильтрации строк или правила объединения данных из нескольких таблиц, аналогично операции соединения в реляционной алгебре.

• **GROUP BY** – определяет поля, по которым будет выполняться группировка данных.

• **HAVING** – применяется для фильтрации группированных данных.

• **ORDER BY** – задаёт порядок сортировки результатов. Например, если сначала указать поле «Фамилия», а затем «Номер группы», то студенты будут

отсортированы по фамилии в алфавитном порядке, а однофамильцы – по номеру группы.

**Предикаты в условиях WHERE**

**Операторы сравнения:** {=, <>, >, <, >=, <=} – работают традиционным образом, сравнивая значения между собой.

**BETWEEN A AND B** – проверяет, находится ли значение в диапазоне [A, B], включая границы. Обратный вариант – NOT BETWEEN A AND B, который возвращает TRUE, если значение выходит за указанный диапазон.

**IN** (список значений) – проверяет, входит ли значение в заданное множество. Значения могут задаваться как простым перечислением, так и вложенным подзапросом. Обратный предикат – NOT IN, который проверяет, что значение не входит в указанное множество.

**LIKE** и **NOT LIKE** – используются для поиска значений, соответствующих шаблону.

**_** (подчёркивание) – заменяет один произвольный символ.

**%** (процент) – заменяет любую последовательность символов.

**IS NULL** и **IS NOT NULL** – применяются для работы с неопределёнными значениями (NULL).

**Особенности работы с NULL**

Концепция неопределённых значений (NULL) была добавлена в реляционную модель позже. NULL интерпретируется как неизвестное или отсутствующее значение. Если в будущем появится дополнительная информация, NULL может быть заменён конкретным значением.

Стандартные правила сравнения к NULL не применяются: два неопределённых значения никогда не считаются равными друг другу. Для проверки используется:

- **<имя атрибута> IS NULL** – возвращает TRUE, если значение атрибута неизвестно.
- **<имя атрибута> IS NOT NULL** – возвращает TRUE, если атрибут имеет определённое значение.

Введение NULL привело к необходимости модификации традиционной двузначной логики (TRUE/FALSE), добавив третье состояние – **UNKNOWN**. Теперь все логические операции с NULL подчиняются трёхзначной логике согласно специальной таблице истинности.

| A | B | Not A | $A \wedge B$ | $A \vee B$ |
|---|---|---|---|---|
| TRUE | TRUE | FALSE | TRUE | TRUE |
| TRUE | FALSE | FALSE | FALSE | TRUE |
| TRUE | Null | FALSE | Null | TRUE |
| FALSE | TRUE | TRUE | FALSE | TRUE |

| A | B | Not A | A ∧ B | A ∨ B |
|---|---|---|---|---|
| FALSE | FALSE | TRUE | FALSE | FALSE |
| FALSE | Null | TRUE | FALSE | Null |
| Null | TRUE | Null | Null | TRUE |
| Null | FALSE | Null | FALSE | Null |
| Null | Null | Null | Null | Null |

Предикаты существования **EXIST** и несуществования **NOT EXIST**. Эти предикаты относятся к встроенным подзапросам.

В условиях поиска могут быть использованы все рассмотренные ранее предикаты.

Рассмотрим детально первые три строки оператора **SELECT**:

1. Самый простой запрос **SELECT** без необязательных частей соответствует просто декартову произведению. Например, выражение

SELECT *
FROM Rl, R2;

соответствует декартову произведению таблиц R1 и R2.

2. 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, при этом дубликаты всех строк сохранены, в отличие от операции проектирования в реляционной алгебре, где при проектировании по умолчанию все дубликаты кортежей уничтожаются.

3. Рассмотрим базу данных, которая моделирует сдачу сессии в некотором учебном заведении. Пусть она состоит из трех отношений $R_1$, $R_2$, $R_3$. Будем считать, что они представлены таблицами Rl, R2 и R3 соответственно.

$R_1$ = (ФИО, Дисциплина, Оценка);

$R_2$ = (ФИО, Группа);

$R_3$ = (Группы, Дисциплина )

| R1 | | |
|---|---|---|
| **ФИО** | **Дисциплина** | **Оценка** |
| Петров Ф. И. | Базы данных | 5 |
| Миронов А. В. | Базы данных | 2 |
| Степанова К. Е. | Базы данных | 2 |
| Степанова К. Е. | Теория информации | 2 |
| Миронов А.В. | Теория информации | Null |
| Петров Ф. И. | Теория информации | 4 |

| R1 | | |
|---|---|---|
| **ФИО** | **Дисциплина** | **Оценка** |
| Трофимов П.А. | Сети и телекоммуникации | 5 |
| Иванова Е.А. | Сети и телекоммуникации | 5 |
| Уткина Н.В. | Сети и телекоммуникации | 4 |
| Трофимов П.А. | Английский язык | 5 |
| Иванова Е.А. | Английский язык | 3 |

| R2 | |
|---|---|
| **ФИО** | **Группа** |
| Петров Ф. И. | 4906 |
| Миронов А. В. | 4906 |
| Степанова К.Е. | 4906 |
| Трофимов П.А. | 4907 |
| Иванова Е.А. | 4907 |
| Уткина Н.В. | 4907 |

| R3 | |
|---|---|
| **Группа** | **Дисциплина** |
| 4906 | Базы данных |
| 4906 | Теория информации |
| 4906 | Английский язык |
| 4907 | Английский язык |
| 4907 | Сети и телекоммуникации |

Приведем несколько примеров использования оператора **SELECT**.

```
SELECT DISTINCT Группа
FROM R3;
```

Результат:

| **Группа** |
|---|
| 4906 |
| 4907 |

Вывести список студентов, которые сдали экзамен по дисциплине «Базы данных» на «отлично»

```
SELECT ФИО
FROM R1
WHERE Дисциплина = ‘Базы данных’ AND Оценка = 5;
```

Результат:

| ФИО |
|---|
| Петров Ф. И. |

Вывести весь список студентов, которым надо сдавать экзамены с указанием названия дисциплин, по которым должны проводиться эти экзамены.

```
SELECT ФИО, Дисциплина
FROM R2, R3
WHERE R2.Группа = R3.Группа ;
```

Здесь часть **WHERE** задает условия соединения отношений R2 и R3, при отсутствии условий соединения в части **WHERE** результат будет эквивалентен расширенному декартову произведению, и в этом случае каждому студенту будут приписаны все дисциплины из отношения R3, а не те, которые должна сдавать его группа.

Результат:

| ФИО | Дисциплина |
|---|---|
| Петров Ф. И. | Базы данных |
| Миронов А. В. | Базы данных |
| Степанова К. Е. | Базы данных |
| Петров Ф. И. | Теория информации |
| Миронов А. В. | Теория информации |
| Степанова К. Е. | Теория информации |
| Петров Ф. И. | Английский язык |
| Миронов А. В. | Английский язык |
| Степанова К. Е. | Английский язык |
| Трофимов П. А. | Сети и телекоммуникации |
| Иванова Е. А. | Сети и телекоммуникации |
| Уткина Н. В. | Сети и телекоммуникации |
| Трофимов П. А. | Английский язык |
| Иванова Е. А. | Английский язык |
| Уткина Н: В. | Английский язык |

Вывести список студентов, имеющих несколько двоек.

```
SELECT DISTINCT Rl.ФИО
FROM Rl a, Rl b
WHERE a.ФИО = b.ФИО AND a.Дисциплина <> b.Дисциплина AND
a.Оценка <= 2 AND b.Оценка <= 2;
```

Здесь мы использовали *псевдонимы* для именования отношения R1 a и b, так как для записи условий поиска нам необходимо работать сразу с двумя экземплярами данного отношения.

Результат:

| ФИО |
|---|
| Степанова К. Е. |

SQL изначально разрабатывался для применения конечными пользователями, и его стремились сделать возможно ближе к языку естественному, а не к языку алгоритмическому. По этой причине SQL на первых порах вызывает путаницу и раздражение у начинающих его изучать профессиональных программистов, которые привыкли разговаривать с машиной именно на алгоритмических языках.

Наличие неопределенных (Null) значений повышает гибкость обработки информации, хранящейся в БД. В наших примерах мы можем предположить ситуацию, когда студент пришел на экзамен, но не сдавал его по некоторой причине, в этом случае оценка по некоторой дисциплине для данного студента имеет неопределенное значение.

В данной ситуации можно поставить вопрос: «Найти студентов, пришедших на экзамен, но не сдававших его с указанием названия дисциплины». Оператор SELECT будет выглядеть следующим образом:

```
SELECT ФИО, Дисциплина
FROM R1
WHERE Оценка IS NULL;
```

Результат:

| ФИО | Дисциплина |
|---|---|
| Миронов А. | Теория информации |

## 9.5. Применение агрегатных функций и вложенных запросов в операторе выбора

В SQL добавлены дополнительные функции, которые позволяют вычислять обобщенные групповые значения. Для применения агрегатных функций предполагается предварительная операция группировки. В чем состоит *суть операции группировки*? При группировке все множество кортежей отношения разбивается на группы, в которых собираются кортежи, имеющие одинаковые значения атрибутов, которые заданы в списке группировки.

Например, сгруппируем отношение R1 по значению столбца **Дисциплина**. Мы получим 4 группы, для которых можем вычислить некоторые групповые значения, например, количество кортежей в группе, максимальное или минимальное значение столбца **Оценка**.

Это делается с помощью агрегатных функций. Агрегатные функции вычисляют одиночное значение для всей группы таблицы. Список этих функций представлен в табл. 9.

*Таблица 9*

**Агрегатные функции**

| Функция | Результат |
|---|---|
| COUNT | Количество строк или непустых значений полей, которые выбрал запрос |
| SUM | Сумма всех выбранных значений данного поля |
| AVG | Среднеарифметическое значение всех выбранных значений данного поля |
| MIN | Наименьшее из всех выбранных значений данного поля |
| MAX | Наибольшее из всех выбранных значений данного поля |

```
SELECT ФИО, Дисциплина, Оценка
FROM R1
GROUP BY R1.Дисциплина;
```

| R1 | | | |
|---|---|---|---|
| | **ФИО** | **Дисциплина** | **Оценка** |
| Группа 1 | Петров Ф. И. | Базы данных | 5 |
| | Миронов А. В. | Базы данных | 2 |
| | Степанова К. Е. | Базы данных | 2 |
| Группа 2 | Степанова К. Е. | Теория информации | 2 |
| | Петров Ф. И. | Теория информации | 4 |
| | Миронов А. В. | Теория информации | Null |
| Группа 3 | Трофимов П. А. | Сети и телекоммуникации | 4 |
| | Иванова Е. А. | Сети и телекоммуникации | 5 |
| | Уткина Н. В. | Сети и телекоммуникации | 5 |
| Группа 4 | Трофимов П. А. | Английский язык | 5 |
| | Иванова Е. А. | Английский язык | 3 |

Агрегатные функции в SQL применяются так же, как и имена полей в операторе **SELECT**, но с одной особенностью: они принимают имя поля в качестве аргумента.

Функции **SUM** и **AVG** работают только с числовыми значениями.

Функции **COUNT**, **MAX** и **MIN** могут применяться как к числовым, так и к текстовым полям.

При работе с текстовыми данными функции **MAX** и **MIN** используют порядок символов в кодировке ASCII, сортируя строки в алфавитном порядке.

Некоторые системы управления базами данных (СУБД) допускают использование вложенных агрегатных функций, однако это не соответствует стандарту ANSI и может приводить к непредвиденным последствиям.

**Пример использования агрегатных функций**

Предположим, необходимо подсчитать количество студентов, сдававших экзамены по каждой дисциплине. Для этого выполняется группировка по полю «**Дисциплина**», а в результате запроса выводится название дисциплины и число записей в каждой группе.

При передаче * в функцию **COUNT**, она подсчитывает все строки в группе.

```
SELECT R1.Дисциплина, COUNT(*)
FROM R1
GROUP BY R1.Дисциплина;
```

Результат:

| Дисциплина | COUNT(*) |
|---|---|
| Базы данных | 3 |
| Теория информации | 3 |
| Сети и телекоммуникации | 3 |
| Английский язык | 2 |

Если же мы хотим сосчитать количество сдававших экзамен по какой-либо дисциплине, то нам необходимо исключить неопределенные значения из исходного отношения перед группировкой. В этом случае запрос будет выглядеть следующим образом:

```
SELECT R1.Дисциплина, COUNT(*)
FROM R1
WHERE R1.Оценка IS NOT NULL
GROUP BY Rl.Дисциплина;
```

Получим результат:

| Дисциплина | COUNT(*) |
|---|---|
| Базы данных | 3 |
| Теория информации | 2 |
| Сети и телекоммуникации | 3 |
| Английский язык | 2 |

Можно применять агрегатные функции также и без операции предварительной группировки, в этом случае все отношение рассматривается как одна группа и для этой группы можно вычислить одно значение на группу.

Обратившись снова к базе данных «Сессия» (таблицы Rl, R2, R3), найдем количество успешно сданных экзаменов:

```
SELECT COUNT(*)
FROM Rl
WHERE Оценка > 2;
```

Возвращается одиночное значение 7.

В результат можно включить значение поля группировки и несколько агрегатных функций, а в условиях группировки можно использовать несколько полей. При этом группы образуются по набору заданных полей группировки. Операции с агрегатными функциями могут быть применены к объединению множества исходных таблиц. Например, поставим вопрос: определить для каждой группы и каждой дисциплины количество успешно сдавших экзамен и средний балл по дисциплине.

```
SELECT R2.Группа, R1.Дисциплина, COUNT(*), AVG(Оценка)
FROM R1,R2
WHERE Rl.ФИО = R2.ФИО AND Rl.Оценка IS NOT NULL AND Rl.Оценка > 2
GROUP BY R2.Группа, Rl.Дисциплина;
```

Результат:

| Группа | Дисциплина | COUNT(*) | AVG(Оценка) |
|---|---|---|---|
| 4906 | Базы данных | 1 | 5.0 |
| 4906 | Теория информации | 1 | 4.0 |
| 4907 | Сети и телекоммуникации | 3 | 4.67 |
| 4907 | Английский язык | 2 | 4.0 |

Агрегатные функции могут применяться как в выражении вывода результатов строки SELECT, так и в выражении условия обработки сформированных групп HAVING. В этом случае каждая агрегатная функция вычисляется для каждой выделенной группы. Значения, полученные при вычислении агрегатных функций, могут быть использованы для вывода соответствующих результатов или для условия отбора групп.

Построим запрос, выводящий группы, в которых по одной дисциплине на экзаменах получено больше одной двойки:

```
SELECT R2.Группа
FROM R1,R2
WHERE Rl.ФИО = R2.ФИО AND R1.Оценка = 2
GROUP BY R2.Группа, Rl.Дисциплина
HAVING COUNT(*) > 1;
```

Результат:

| Группа |
|---|
| 4906 |

## 9.6. Вложенные запросы

Теперь вернемся к БД «Сессия» и рассмотрим на ее примере использование вложенных запросов.

С помощью SQL можно вкладывать запросы внутрь друг друга. Обычно внутренний запрос генерирует значение, которое проверяется в предикате внешнего запроса (в предложении **WHERE** или **HAVING**), определяющего, верно оно или нет. Совместно с подзапросом можно использовать предикат **EXISTS**, который возвращает истину, если вывод подзапроса не пуст.

Например, покажем, как выглядят на SQL некоторые запросы к БД «Сессия»:

*1. Список тех, кто сдал все положенные экзамены.*

```
SELECT ФИО
FROM Rl AS a
WHERE Оценка > 2
GROUP BY ФИО
HAVING COUNT(*) = (SELECT COUNT(*)
                   FROM R2, R3
                   WHERE R2.Группа=R3.Группа
                   AND ФИО=a.ФИО);
```

Здесь во вложенном запросе определяется общее число экзаменов, которые должен сдавать каждый студент, обучающийся в данной группе, и это число сравнивается с числом экзаменов, которые сдал данный студент.

*2. Список тех, кто должен был сдавать экзамен по Английскому языку, но пока еще не сдавал.*

```
SELECT ФИО
FROM R2 a, R3
WHERE R2.Группа=R3.Группа AND Дисциплина = 'Английский язык'
AND NOT EXISTS (SELECT ФИО
                FROM Rl
                WHERE ФИО=a.ФИО AND Дисциплина = 'Английский язык');
```

Предикат **EXISTS** (SubQuery) истинен, когда подзапрос SubQuery не пуст, то есть содержит хотя бы один кортеж, в противном случае предикат **EXISTS** ложен.

Предикат **NOT EXISTS** обратно – истинен только тогда, когда подзапрос SubQuery пуст.

В стандарте SQL2 операторы сравнения расширены до многократных сравнений с использованием ключевых слов **ANY** и **ALL**. Это расширение используется при сравнении значения определенного столбца со столбцом данных, возвращаемым вложенным запросом.

Ключевое слово **ANY**, поставленное в любом предикате сравнения, означает, что предикат будет истинен, если хотя бы для одного значения из подзапроса предикат сравнения истинен. Ключевое слово **ALL** требует, что-

бы предикат сравнения был бы истинен при сравнении со всеми строками подзапроса.

Например, найдем студентов, которые сдали все экзамены на оценку не ниже чем «хорошо».

```
SELECT R1.ФИО
FROM R1
WHERE 4 > = All (Select R1.Оценка
                 FROM R1 AS R11
                 WHERE R1.ФИО = R11.ФИО);
```

Рассмотрим еще один пример:

Работаем с той же базой «Сессия», но добавим к ней еще одно отношение R4, которое характеризует сдачу лабораторных работ в течение семестра:

R4 = (ФИО, Дисциплина, Номер_лаб_раб, Оценка);

Выбрать студентов, у которых оценка по экзамену не меньше, чем хотя бы одна оценка по сданным им лабораторным работам по данной дисциплины:

```
SELECT R1.ФИО
FROM R1
WHERE RL.ОЦЕНКА >= ANY (SELECT R4.Оценка
                        FROM R4
                        WHERE RL.Дисциплина = R4. Дисциплина AND Rl.
ФИО = R4. ФИО);
```

## 9.7. Операторы манипулирования данными

Существует три операции манипулирования данными: операция удаления записей – ей соответствует оператор **DELETE**, операция добавления или ввода новых записей – ей соответствует оператор **INSERT** и операция изменения (обновления записей) – ей соответствует оператор **UPDATE**. Рассмотрим каждый из операторов подробнее.

Все операторы манипулирования данными позволяют изменить данные только в одной таблице.

1. 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. Задание списка столбцов необязательно тогда, когда мы вводим строку с заданием значений всех столбцов. Например, введем новую книгу в таблицу BOOKS:

INSERT INTO BOOKS (ISBN, TITLE, AUTOR, COAUTOR,YEARIZD, PAGES)
VALUES (‘5-88782-290-2’,’Аппаратные средства IBM PC. Энциклопедия’,
‘Тук М. ‘,”, 2000, 816);

В этой книге только один автор, нет соавторов, но мы в списке столбцов задали столбец COAUTOR, поэтому мы должны были ввести соответствующее значение в разделе VALUES. Мы ввели пустую строку, потому что мы знаем точно, что нет соавтора. Мы могли бы ввести неопределенное значение NULL.

Так как мы вводим полную строку, мы можем не задавать список столбцов, ограничиться только заданием перечня значений, в этом случае оператор ввода будет выглядеть следующим образом:

INSERT INTO BOOKS VALUES (‘5-88782-290-2’,‘Аппаратные средства IBM PC. Энциклопедия ‘, ‘Тук М.’,”, 2000, 816) ;

Результаты работы обоих операторов одинаковые.

Наконец, мы можем ввести неполный перечень значений, то есть не вводить соавтора, так как он отсутствует для данного издания. Но в этом случае мы должны задать список вводимых столбцов, тогда оператор ввода будет выглядеть следующим образом:

INSERT INTO BOOKS (ISBN, TITLE, AUTOR,YEARIZD, PAGES)
VALUES (‘5-88782-290-2’,’Аппаратные средства IBM PC. Энциклопедия’,
‘Тук М.’,2000,816)

Столбцу COAUTOR будет присвоено в этом случае значение NULL.

Если столбец или атрибут имеет признак обязательный (NOT NULL) при описании таблицы, то оператор **INSERT** должен обязательно содержать данные для ввода в каждую строку данного столбца. Поэтому если в таблице все столбцы обязательные, то каждая вводимая строка должна содержать полный перечень вводимых значений, а указание имен столбцов в этом случае необязательно. В противном случае, если имеется хотя бы один необязательный столбец и вы не вводите в него значений, задание списка имен столбцов – обязательно.

В набор значений могут быть включены специальные функции и выражения. Ограничением здесь является то, что значения этих функций должны быть определены на момент ввода данных. Поэтому, например, мы можем сформировать оператор ввода данных в таблицу **EXEMPLAR** следующим образом:

INSERT INTO EXEMPLAR (INV, ISBN, YES_NO, NUM_READER, DATE_IN, DATE_OUT)
VALUES (1872, ‘5-88782-290-2’, NO, 344, GetDate(), DateAdd(d,GetDate(),14));

И это означает, что мы выдали экземпляр книги с инвентарным номером 1872 читателю с номером читательского билета 344, отметив, что этот

экземпляр не присутствует с этого момента в библиотеке, и определили дату выдачи книги как текущую дату (функция **GetDate**()), а дату возврата задали двумя неделями позднее, использовав при этом функцию **DateAdd**(), которая позволяет к одной дате добавить заданное количество интервалов даты и тем самым получить новое значение типа «дата». Мы добавили 14 дней к текущей дате.

Оператор ввода данных позволяет ввести сразу множество строк, если их можно выбрать из некоторой другой таблицы. Допустим, что у нас есть таблица со студентами и в ней указаны основные данные о студентах: их фамилии, адреса, домашние телефоны и даты рождения. Тогда мы можем сделать всех студентов читателями нашей библиотеки одним оператором:

```
INSERT INTO READER (NAME_READER, ADRESS, HOOM_PHONE, BIRTH_
DAY)
SELECT (NAME_STUDENT, ADRESS, HOOM_PHONE, BIRTH_DAY)
FROM STUDENT;
```

При этом номер читательского билета может назначаться автоматически, поэтому мы не вводим значения этого столбца в таблицу. Кроме того, мы предполагаем, что у студентов дневного отделения еще нет работы и поэтому нет рабочего телефона, и мы его не вводим.

2. 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]
```

Если условия отбора не задаются, то из таблицы удаляются все 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 R1 командой

```
DELETE FROM Rl;
```

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, если мы исключим студента Миронова А.В., то мы должны написать следующую команду:

```
DELETE FROM R2
WHERE ФИО = 'Миронов А.В.'
```

В части **WHERE** может находиться встроенный запрос.

3. Операция обновления данных **UPDATE** требуется тогда, когда происходят изменения во внешнем мире и их надо адекватно отразить в базе данных, так как надо всегда помнить, что база данных отражает некоторую предметную область. Например, Степанова К. Е. пересдала экзамен по дис-

циплине «Базы данных» с двойки на четверку. В этом случае надо выполнить соответствующую корректировку таблицы R1. Операция обновления имеет следующий формат:

UPDATE имя_таблицы

SET имя_столбца = новое_значение [WHERE условие_отбора]

Часть **WHERE** является необязательной, так же 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.

Для решения ранее поставленной задачи нам необходимо выполнить следующую операцию

UPDATE Rl

SET Rl.Оценка = 4

WHERE Rl. ФИО = ‘Степанова К.Е.’ AND Rl.Дисциплина = ‘Базы данных’

В каких случаях требуется провести изменение в нескольких строках? Это не такая уж редкая задача. Например, если мы расширим нашу учебную базу данных еще одним отношением, которое содержит перечень курсов, на которых учатся наши студенты, то можно с помощью операции обновления промоделировать операцию перевода групп на следующий курс. Пусть новое отношение R4 имеет следующую схему:

R4 = <Группа, Курс>

| R4 | |
|---|---|
| **Группа** | **Курс** |
| 4906 | 3 |
| 4807 | 4 |

В этом случае перевод на следующий курс можно выполнить следующей операцией обновления:

UPDATE R4

SET R4.Курс = R4.Курс + 1

И результат будет выглядеть следующим образом:

| **Группа** | **Курс** |
|---|---|
| 4906 | 4 |
| 4807 | 5 |

Операция модификации, так же как и операция удаления, может использовать сложные подзапросы.

# Глава 10. РАСПРЕДЕЛЕННАЯ ОБРАБОТКА ДАННЫХ

Когда база данных размещена на персональном компьютере, который не подключен к сети, она обычно используется в монопольном режиме. Даже если с базой данных работают несколько пользователей, они могут взаимодействовать с ней только поочередно. В таком случае проблемы поддержания корректности данных не возникают, поскольку порядок работы пользователей с базой данных определяется организационными мерами, например, графиком работы.

Однако на сегодняшний день использование базы данных только на изолированном компьютере с небольшим объемом информации становится все менее характерным для большинства приложений. База данных, отражающая информационную модель реальной предметной области, со временем растет, увеличивается количество задач, решаемых с ее помощью, а значит, увеличивается количество приложений, которые используют единую базу данных. В связи с этим компьютеры часто объединяются в локальные сети, и необходимость распределения приложений, работающих с одной базой данных по сети, становится очевидной.

Например, при создании базы данных для небольшой торговой фирмы у вас появляются разные пользователи с конкретными бизнес-функциями, которые могут находиться в разных частях здания, но все они должны работать с одной информационной моделью организации, т. е. с общей базой данных.

Параллельный доступ нескольких пользователей к базе данных, если она размещена на одном компьютере, соответствует распределенному доступу к централизованной базе данных. Такие системы называют системами распределенной обработки данных.

Если же база данных распределена по нескольким компьютерам, подключенным к сети, и возможен параллельный доступ нескольких пользователей, то речь идет о параллельном доступе к распределенной базе данных. Такие системы называют системами распределенных баз данных. В общем случае режимы использования базы данных можно представить следующим образом (рис. 30).

Определим ключевые термины, которые нам понадобятся в дальнейшей работе. Некоторые из этих терминов уже знакомы, но повторим их для ясности.


Рис. 30. *Режимы работы с БД*

**Терминология:**

**Пользователь БД** – это человек или программа, которые взаимодействуют с базой данных, используя язык манипулирования данными (ЯМД).

**Запрос** – это процесс, при котором пользователь обращается к базе данных для ввода, получения или изменения данных.

**Транзакция** – последовательность операций, которые изменяют данные в базе данных, переходя из одного ее корректного состояния в другое.

**Логическая структура БД** – это описание базы данных, которое не зависит от физической реализации и наиболее близко к концептуальной модели данных.

**Топология БД / Структура распределенной БД** – схема, которая определяет, как физическая база данных распределяется по сети.

**Локальная автономность** – это принцип, согласно которому информация в локальной базе данных и связанные с ней определения данных находятся в ведении и управлении локального владельца.

**Удаленный запрос** – запрос, который выполняется с использованием связи через модем или другие удаленные средства.

**Возможность реализации удаленной транзакции** – это возможность обработать транзакцию, которая состоит из нескольких SQL-запросов, на одном удаленном узле.

**Поддержка распределенной транзакции** позволяет обрабатывать транзакцию, которая состоит из нескольких SQL-запросов, выполняемых на разных узлах сети, будь то удаленные или локальные узлы. Каждый запрос при этом обрабатывается только на одном узле, то есть сами запросы не являются распределенными, но в рамках одной транзакции они могут быть выполнены на разных узлах.

**Распределенный запрос** – это запрос, который использует данные из нескольких баз данных, расположенных на различных узлах сети.

Системы распределенной обработки данных в основном ассоциируются с первыми поколениями СУБД, которые создавались для работы на мультипрограммных операционных системах. Эти системы использовали централизованное хранение баз данных на внешних устройствах центральных ЭВМ и обеспечивали многопользовательский терминальный доступ. Терминалы пользователей не имели собственных ресурсов (таких как процессоры или память), которые могли бы использоваться для хранения и обработки данных. Одной из первых полностью реляционных СУБД, работающих в многопользовательском режиме, была СУБД **SYSTEM R**, разработанная компанией IBM. В этой СУБД был реализован язык манипулирования данными SQL, а также основные принципы синхронизации, применяемые при распределенной обработке данных, которые до сих пор являются основой для многих коммерческих СУБД.

Общая тенденция перехода от централизованных **mainframe-систем** к распределенным открытым системам, объединяющим компьютеры среднего класса, получила название **DownSizing**. Этот процесс оказал значительное влияние на развитие архитектур СУБД и поставил перед разработчиками множество сложных задач. Главной проблемой был технологический перенос от централизованного управления данными на одном компьютере к распределенной обработке данных в неоднородной вычислительной среде, состоящей из множества соединенных в сеть компьютеров разных моделей и производителей.

В противоположность этому происходил процесс **UpSizing** – активное развитие персональных компьютеров и локальных сетей, что также повлияло на эволюцию СУБД. Высокие темпы роста функциональности и производительности ПК привели к тому, что разработчики начали распространять профессиональные СУБД на платформе настольных компьютеров.

Сегодня наблюдается тенденция создания информационных систем, которые полностью соответствуют их масштабу и задачам. Этот процесс называется RightSizing – подбор платформы с учетом точных нужд системы.

Несмотря на это, большие ЭВМ все еще используются и продолжают сосуществовать с современными открытыми системами. Причина этого заключается в том, что в аппаратное и программное обеспечение больших ЭВМ были вложены значительные средства, поэтому многие организации продолжают их использовать, несмотря на устаревшую архитектуру. Кроме того, перенос данных и программ с больших ЭВМ на более современные компьютеры – это сложная техническая задача, требующая значительных ресурсов.

## 10.1. Модели «клиент-сервер» в технологии баз данных

Вычислительная модель **«клиент-сервер»** изначально связана с концепцией открытых систем, которая появилась в 90-х годах и быстро развивалась. Термин **«клиент-сервер»** был применен к архитектуре программного обеспечения, которая описывает распределение процесса выполнения между двумя программами, называемыми «клиентом» и «сервером». Клиентский процесс запрашивает определенные услуги, а серверный процесс отвечает за их выполнение. В этой модели предполагается, что один сервер может обслуживать несколько клиентов.

Ранее приложение (пользовательская программа) выполнялось как единый монолитный блок на одном компьютере. Однако возникла идея более эффективного использования ресурсов сети. При монолитной модели использовались ресурсы лишь одного компьютера, в то время как другие компьютеры в сети служили только терминалами. В отличие от эпохи **mainframe-систем**, все компьютеры в сети теперь имеют свои собственные ресурсы, и необходимо разумно распределять нагрузку, чтобы максимально эффективно их использовать.

Основной принцип технологии **«клиент-сервер»** в контексте баз данных заключается в разделении стандартных функций интерактивного приложения на пять различных групп с разной природой:

1. **Функции ввода и отображения данных (Presentation Logic)** – ответственные за отображение данных и взаимодействие с пользователем.
2. **Прикладные функции (Business Logic)** – определяют основные алгоритмы для решения задач приложения.
3. **Функции обработки данных внутри приложения (Database Logic)** – связаны с управлением и манипуляциями с данными, которые происходят непосредственно внутри приложения.
4. **Функции управления информационными ресурсами (Database Manager System)** – выполняются системой управления базами данных (СУБД), которая обеспечивает хранение и управление данными.
5. **Служебные функции** – играют роль связующих элементов между функциями других групп приложения.

Типичная структура приложения, взаимодействующего с базой данных, представлена на рис. 31.

**Презентационная логика (Presentation Logic)** – это часть приложения, с которой взаимодействует пользователь, и которая отображает информацию на экране. Сюда входят все интерфейсные формы, которые

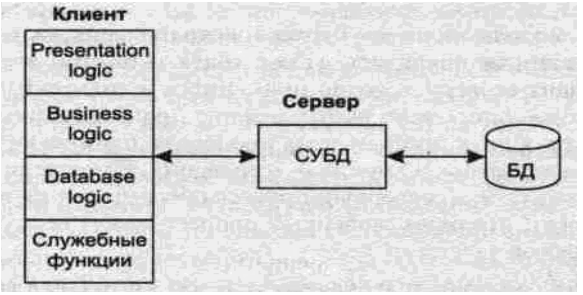


Рис. 31. *Структура типового интерактивного приложения*

пользователь видит и с которыми взаимодействует в процессе работы. Презентационная логика отвечает за:

– формирование экранных изображений;

– чтение и запись данных в формы;

– управление экраном;

– обработку действий пользователя (движения мыши, нажатие клавиш).

**Бизнес-логика (Business Logic)** – это часть кода приложения, которая реализует алгоритмы решения задач, специфичных для данного приложения. Этот код обычно пишется с использованием языков программирования, таких как C, C++, Cobol, SmallTalk или Visual Basic.

**Логика обработки данных (Data Manipulation Logic)** – представляет собой часть приложения, отвечающую за обработку данных внутри самого приложения. Данные управляются системой управления базами данных (СУБД), и для их доступа используются языки запросов, такие как SQL. SQL-запросы обычно встраиваются в код приложения, написанный на языках третьего или четвертого поколения (3GL, 4GL).

**Процессор управления данными (Database Manager System Processing)** – это сама СУБД, которая обеспечивает хранение, организацию и управление данными. Хотя в идеале функции СУБД должны быть скрыты от бизнес-логики, для понимания архитектуры приложения их стоит выделить в отдельную часть.

В **централизованной архитектуре** (Host-based processing) все компоненты приложения находятся в одной среде и выполняются внутри единой исполняемой программы.

В **децентрализованной архитектуре** эти задачи могут быть разделены между серверными и клиентскими процессами. В зависимости от того, как именно распределяются функции, можно выделить различные модели распределения, как показано на рис. 32.


Рис. 32. *Распределение функций приложения в моделях «клиент-сервер»*

**Распределенная презентация (Distributed presentation, DP)**: задачи отображения и взаимодействия с пользователем выполняются распределенно между клиентом и сервером.

**Удаленная презентация (Remote Presentation, RP)**: клиентский интерфейс работает на удаленной машине, в то время как вся обработка и хранение данных происходит на сервере.

**Распределенная бизнес-логика (Remote Business Logic, RBL)**: логика, определяющая алгоритмы решения задач приложения, распределена между несколькими процессами, которые могут работать на разных платформах.

**Распределенное управление данными (Distributed Data Management, DDM)**: управление данными и их хранение разделены между различными узлами сети.

**Удаленное управление данными (Remote Data Management, RDM)**: управление данными осуществляется удаленно, в то время как данные могут храниться на другом сервере.

Эта классификация помогает понять, как различные функции могут быть распределены между сервером и клиентами в децентрализованных системах. Важно отметить, что **бизнес-логика** не может быть полностью

удалена, хотя она и может быть частично распределена. В случае её распределения различные процессы, выполняющие логику, должны корректно взаимодействовать друг с другом, даже если они находятся на разных платформах.

## 10.2. Двухуровневые модели

Двухуровневая модель фактически является результатом распределения пяти указанных функций между двумя процессами, которые выполняются на двух платформах: на клиенте и на сервере. В чистом виде почти никакая модель не существует, однако рассмотрим наиболее характерные особенности каждой двухуровневой модели.

### *10.2.1. Модель файлового сервера*

Модель **удаленного управления данными** также известна как модель **файлового сервера** (File Server, FS) (рис. 33). В рамках этой модели презентационная и бизнес-логика находятся на клиенте, а сервер выполняет роль хранилища данных и обеспечивает доступ к этим данным. Управление информационными ресурсами в такой системе также осуществляется на клиентской стороне.

В данной модели база данных хранится на сервере, а клиент взаимодействует с сервером, отправляя файловые команды для доступа к данным. При этом управление метаданными, а также функции, связанные с базой данных, выполняются непосредственно на клиенте.

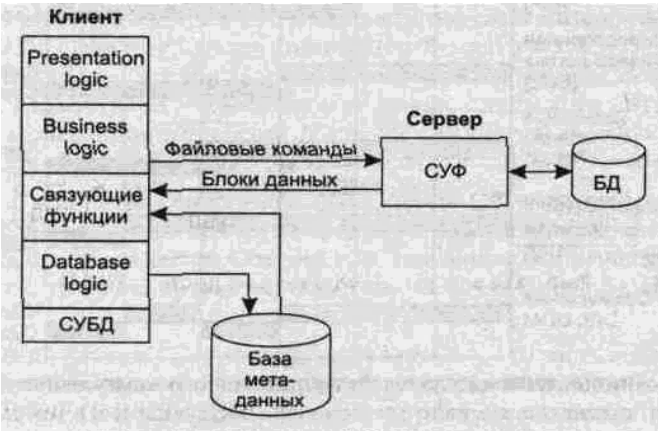


Рис. 33. *Модель файлового сервера*

Преимущество этой модели заключается в том, что приложение разделяется на два взаимодействующих процесса, что позволяет серверу (серверному процессу) обслуживать несколько клиентов одновременно. В этой модели СУБД должна располагаться на стороне клиента.

**Алгоритм выполнения запроса клиента:**

1. Клиент формирует запрос с использованием языка запросов базы данных (ЯМД).
2. СУБД преобразует запрос в последовательность файловых команд.
3. Каждая файловая команда вызывает передачу блока данных с сервера на клиент.
4. На клиенте СУБД анализирует полученные данные, и если в текущем блоке нет ответа на запрос, решается необходимость передачи следующего блока данных.
5. Этот процесс продолжается до получения ответа на запрос клиента.

**Недостатки модели:**

• **Высокий сетевой трафик:** поскольку данные передаются по сети в виде множества блоков и файлов, это может привести к значительным задержкам и нагрузке на сеть.

• **Ограниченные возможности обработки данных:** операции с данными ограничены лишь файловыми командами, что ограничивает гибкость работы с информацией.

• **Низкий уровень безопасности:** защита данных реализована только на уровне файловой системы, что не обеспечивает достаточной безопасности для защиты базы данных от несанкционированного доступа.

### *10.2.2. Модель удаленного доступа к данным*

В модели **удаленного доступа** (Remote Data Access, RDA) база данных и ядро СУБД находятся на сервере. Клиент, в свою очередь, располагает презентационной логикой и бизнес-логикой приложения. Клиент обращается к серверу с запросами, написанными на языке SQL. Эта модель предполагает, что все данные хранятся и обрабатываются на сервере, а клиент лишь отправляет запросы и получает результаты. **Структура модели удаленного доступа** приведена на рис. 34.

**Преимущества модели удаленного доступа (RDA):**

1) перенос компонентов представления и прикладного функционала на клиентский компьютер значительно снижает нагрузку на сервер базы данных, минимизируя количество процессов в операционной системе;

2) сервер базы данных освобождается от ненужных функций, полностью сосредоточив свои ресурсы на обработке данных, запросов и транзакций. Это стало возможным благодаря переходу от терминалов, не обладающих вычислительными ресурсами, к клиентским компьютерам, которые обладают собственной вычислительной мощностью;

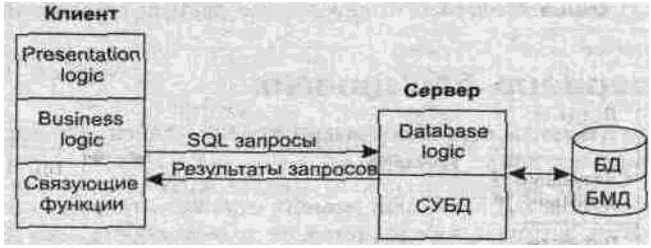


Рис. 34. *Модель удаленного доступа*

3) сеть значительно разгружается, так как теперь передаются не запросы в терминах файловой системы, а запросы на языке SQL, объем которых гораздо меньше. В ответ на запросы клиент получает только необходимые данные, а не целые блоки файлов, как это происходит в модели файлового сервера (FS).

Основным преимуществом модели удаленного доступа (RDA) является унификация интерфейса «клиент-сервер», где стандартом общения между клиентом и сервером становится язык SQL.

**Недостатки модели удаленного доступа (RDA):**

1) при интенсивной работе клиентских приложений запросы на SQL могут значительно загрузить сеть;

2) поскольку в модели удаленного доступа презентационная логика и бизнес-логика приложения находятся на клиенте, это приводит к необходимости повторять код бизнес-логики для каждого отдельного клиентского приложения. Это вызывает излишнее дублирование кода;

3) сервер в этой модели выполняет пассивную роль, и функции управления информационными ресурсами остаются на клиенте. Например, если необходимо контролировать запасы товаров на складе, каждое приложение, связанное с изменением состояния склада, должно проверять остатки после выполнения операций с данными. Это увеличивает сложность клиентских приложений и может привести к необоснованным заказам товаров несколькими приложениями одновременно.

### *10.2.3. Модель сервера баз данных*

Для того чтобы устранить недостатки модели удаленного доступа, необходимо выполнить несколько условий:

**1. Непротиворечивость данных.** База данных (БД) должна всегда отображать текущее состояние предметной области, что предполагает, что не только данные, но и связи между объектами данных должны быть не-

противоречивыми. Данные, хранящиеся в БД, должны всегда быть в консистентном состоянии.

**2. Отражение бизнес-правил.** БД должна содержать правила, которые описывают функционирование предметной области (business rules). Например, для функционирования завода требуется наличие достаточного запаса (страхового запаса) деталей на складе. Кроме того, деталь может быть произведена только при наличии на складе необходимого материала для ее изготовления.

**3. Постоянный контроль за состоянием БД.** БД должна постоянно отслеживать изменения данных и оперативно реагировать на них. Например, если некий параметр достигает критического значения, должна быть отключена соответствующая аппаратура. Если товарный запас на складе опускается ниже допустимой нормы, должна автоматически сформироваться заявка на поставку товаров.

**4. Оперативность реагирования на изменения.** Ситуации в БД должны влиять на выполнение прикладных задач в режиме реального времени.

**5. Контроль типов данных.** Важной задачей СУБД является контроль типов данных. Современные СУБД, как правило, проверяют только синтаксические типы данных, которые определяются в языке описания данных (DDL) и стандартах SQL. Однако в реальных предметных областях существуют данные, которые имеют семантическое значение, такие как координаты объектов или специфические единицы измерений, например, рабочая неделя (отличающаяся от обычной).

Эти требования поддерживаются большинством современных СУБД, таких как Informix, Ingres, Sybase, Oracle, MS SQL Server. Основной принцип этой модели заключается в использовании **механизма хранимых процедур** для программирования SQL-сервера, **триггеров** для отслеживания состояния данных и **ограничений на пользовательские типы данных**, что иногда называют поддержкой доменной структуры. Структура модели сервера баз данных изображена на рис. 35.


Рис. 35. *Модель активного сервера*

В этой модели бизнес-логика разделяется между сервером и клиентом. На сервере она реализована в виде **хранимых процедур** – это программные модули, хранящиеся непосредственно в БД и управляемые самой СУБД. Клиент отправляет запрос на выполнение хранимой процедуры на сервер, который выполняет её, регистрируя все изменения в БД, предусмотренные в процедуре. После этого сервер возвращает клиенту необходимые данные, которые могут быть использованы как для отображения на экране, так и для выполнения части бизнес-логики, которая расположена на клиенте. Это значительно снижает объем трафика между клиентом и сервером.

**Централизованный контроль** в этой модели обеспечивается с помощью **триггеров**, которые также являются частью БД. Триггер – это механизм, отслеживающий специальные события, связанные с состоянием БД. Он работает по аналогии с тумблером, который срабатывает при определенных изменениях в базе. Система мониторит все события, которые активируют триггеры, и при наступлении соответствующего события запускает связанный с ним триггер. Каждый триггер является программой, которая выполняется в контексте БД. Триггеры могут вызывать хранимые процедуры.

Использование триггеров предполагает, что при срабатывании одного триггера могут быть инициированы другие события, что, в свою очередь, активирует дополнительные триггеры. Этот механизм требует тщательной настройки, чтобы избежать бесконечных циклов срабатывания триггеров.

В этой модели сервер имеет активную роль, поскольку не только клиент, но и сам сервер, используя триггеры, может инициировать обработку данных в БД.

Как **хранимые процедуры**, так и **триггеры** сохраняются в словаре БД, и могут быть использованы несколькими клиентами. Это значительно уменьшает дублирование алгоритмов обработки данных в различных приложениях.

Для написания этих процедур и триггеров используется расширение стандартного SQL, называемое **встроенный SQL**.

**Недостатки этой модели:**

• Высокая нагрузка на сервер. Сервер обслуживает несколько клиентов и выполняет следующие задачи:

– мониторинг событий, связанных с триггерами;

– автоматическое срабатывание триггеров при наступлении определенных событий;

– исполнение программ, связанных с триггерами;

– запуск хранимых процедур по запросам пользователей;

– запуск хранимых процедур из триггеров;

– возвращение данных клиенту;

– обеспечение всех функций СУБД: доступ к данным, контроль целостности данных, контроль доступа, поддержка корректной параллельной работы пользователей с одной БД.

С учетом того, что сервер выполняет большую часть бизнес-логики, требования к клиентам значительно снижаются. Такая модель называется **моделью с «тонким клиентом»**, в отличие от **моделей с «толстым клиентом»**, где клиент несет более тяжелую нагрузку.

Для разгрузки сервера была предложена трехуровневая модель, которая более эффективно распределяет задачи между клиентом, сервером и дополнительными уровнями.

### *10.2.4. Модель сервера приложений*

Эта модель является расширением двухуровневой модели и в ней вводится дополнительный промежуточный уровень между клиентом и сервером. Архитектура трехуровневой модели приведена на рис. 36. Этот промежуточный уровень содержит один или несколько серверов приложений.

В этой модели компоненты приложения делятся между тремя исполнителями:

1. *Клиент* обеспечивает логику представления, включая графический пользовательский интерфейс, локальные редакторы; клиент может запускать локальный код приложения клиента, который может содержать обращения к локальной БД, расположенной на компьютере-клиенте. Клиент исполняет коммуникационные функции front-end части приложения, которые обеспечивают доступ клиенту в локальную или глобальную сеть. Дополнительно реализация взаимодействия между клиентом и сервером может включать в себя управление распределенными транзакциями, что соответствует тем случаям, когда клиент также является клиентом менеджера распределенных транзакций.

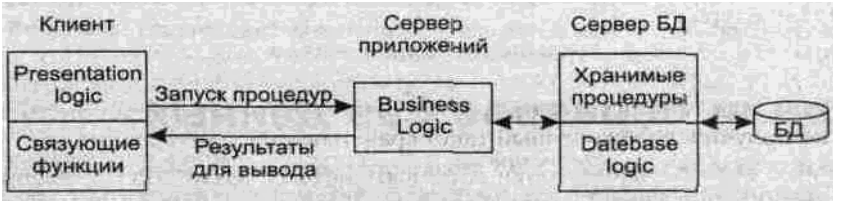


Рис. 36. *Модель сервера приложений*

2. *Серверы приложений* составляют новый промежуточный уровень архитектуры. Они спроектированы как исполнения общих незагружаемых функций для клиентов. Серверы приложений поддерживают функции клиентов как частей взаимодействующих рабочих групп, поддерживают сетевую доменную операционную среду, хранят и исполняют наиболее общие правила бизнес-логики, поддерживают каталоги с данными, обеспечивают обмен сообщениями и поддержку запросов, особенно в распределенных транзакциях.

3. *Серверы баз данных* в этой модели занимаются исключительно функциями СУБД: обеспечивают функции создания и ведения БД, поддерживают целостность реляционной БД, обеспечивают функции хранилищ данных (warehouse services). Кроме того, на них возлагаются функции создания резервных копий БД и восстановления БД после сбоев, управления выполнением транзакций и поддержки устаревших (унаследованных) приложений (legacy application).

Отметим, что эта модель обладает большей гибкостью, чем двухуровневые модели. Наиболее заметны преимущества модели сервера приложений в тех случаях, когда клиенты выполняют сложные аналитические расчеты над базой данных, которые относятся к области OLAP-приложений. (Online analytical processing.) В этой модели большая часть бизнес-логики клиента изолирована от возможностей встроенного SQL, реализованного в конкретной СУБД, и может быть выполнена на стандартных языках программирования, таких как C, C++, SmallTalk, Cobol. Это повышает переносимость системы, ее масштабируемость.

Функции промежуточных серверов могут быть в этой модели распределены в рамках глобальных транзакций путем поддержки XO-протокола (X/Open transaction interface protocol), который поддерживается большинством поставщиков СУБД.

### *10.2.5. Модели серверов баз данных*

На ранних этапах создания СУБД технология «клиент-сервер» только начинала развиваться. Вследствие этого, изначально в архитектуре систем не было эффективного механизма для организации взаимодействия между процессами, выполняющими роль «клиента», и процессами «сервера». Однако сегодня этот механизм является основой для большинства современных СУБД, и от того, как он реализован, зависит производительность системы в целом.

Рассмотрим эволюцию этих механизмов. Они, в основном, зависят от структуры серверных процессов и архитектуры серверов баз данных.

На **нулевом этапе** развития серверов БД, как уже упоминалось, функции управления данными и взаимодействия с пользователем были объединены в одной программе. Это было начальным этапом, который можно назвать первой моделью.

Затем, в дальнейшем развитии, функции управления данными были выделены в отдельный компонент – **сервер**. В этой модели взаимодействие между сервером и пользователем следовало парадигме «один к одному» (рис. 37), где сервер обслуживал только одного клиента. Для обслуживания нескольких клиентов нужно было запускать столько же серверных процессов, сколько клиентов подключалось.

Это разделение серверных функций на отдельную программу стало важным шагом вперёд, так как позволило разместить сервер на одной машине, а интерфейс взаимодействия с пользователем – на другой. Между этими компонентами устанавливалась сеть для обмена данными. Однако одно из главных ограничений этой модели заключалось в том, что для работы с большим количеством клиентов необходимо было запускать множество серверных процессов, что значительно повышало требования к вычислительным ресурсам.

В этой модели каждый серверный процесс был независимым, и, даже если один запрос уже был выполнен для другого клиента, запрос от нового клиента всё равно выполнялся повторно, что было неэффективным. Также в ней возникали проблемы с взаимодействием серверных процессов, поскольку каждый процесс работал автономно.

Эта модель была самой простой и первой в истории, но её ограниченные возможности по обслуживанию множества клиентов и высокая нагрузка на ресурсы ЭВМ стали основными её недостатками.

Проблемы, связанные с моделью «один к одному», решаются в архитектуре **систем с выделенным сервером**, которая позволяет серверу обрабатывать запросы от множества клиентов. В этой архитектуре сервер становится единственным центром управления данными и может одновременно взаимодействовать с несколькими клиентами (рис. 38).

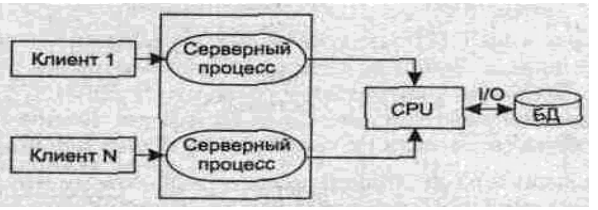


Рис. 37. *Взаимодействие пользовательских и клиентских процессов в модели «один к одному»*

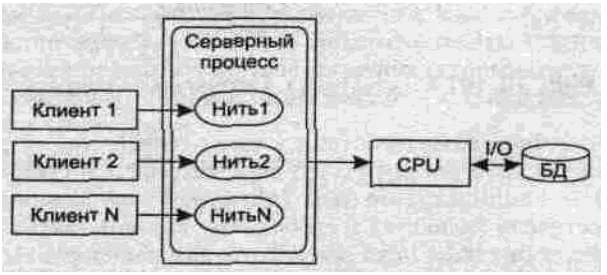


Рис. 38. *Многопотоковая односерверная архитектура*

Каждому клиенту соответствует отдельный поток («thread»), по которому передаются его запросы. Такая архитектура называется **многопотоковой односерверной** («multi-threaded»).

Основное преимущество этой модели – значительное снижение нагрузки на операционную систему, возникающей при большом количестве пользователей, известной как «trashing» (сильная перегрузка системы).

Кроме того, возможность подключения нескольких клиентов к одному серверу позволяет эффективно использовать разделяемые объекты, такие как открытые файлы и данные из системных каталогов. Это значительно снижает потребности в памяти и уменьшает количество процессов в операционной системе. Например, в системе с архитектурой «один к одному» для 100 пользователей создастся 100 копий процессов СУБД, в то время как в системе с многопотоковой архитектурой потребуется всего один серверный процесс (см. рис. 38).

Однако у этой архитектуры есть и свои недостатки. Поскольку сервер работает только на одном процессоре, возникает естественное ограничение на использование СУБД на многопроцессорных платформах. Если, например, сервер работает на компьютере с четырьмя процессорами, то СУБД будет использовать только один из них, оставляя три неиспользуемыми.

Для решения этой проблемы в некоторых системах вводится промежуточный диспетчер, что приводит к архитектуре виртуального сервера (рис. 39). В этой архитектуре клиенты подключаются не к реальному серверу, а к диспетчеру, который выполняет функции маршрутизации запросов к актуальным серверам. Это позволяет устранить ограничения на использование многопроцессорных платформ, так как количество серверов может быть скорректировано в зависимости от количества процессоров в системе.

Тем не менее, такая архитектура также имеет свои недостатки. Во-первых, добавление промежуточного слоя увеличивает нагрузку на систему, так

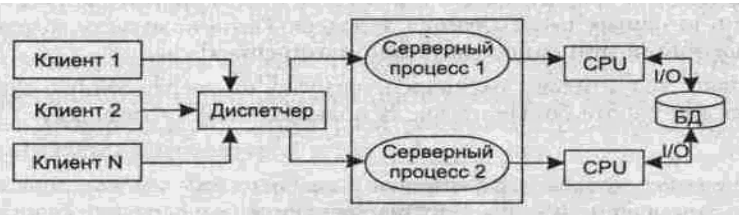


Рис. 39. *Архитектура с виртуальным сервером*

как необходимо поддерживать баланс загрузки серверов («load balancing») и управлять распределением запросов. Во-вторых, она ограничивает возможности управления взаимодействием между клиентом и сервером, так как становится невозможным направить запрос от конкретного клиента к конкретному серверу, а все серверы становятся равноправными, что исключает возможность установления приоритетов для обслуживания запросов.

Эту организацию взаимодействия между клиентом и сервером можно представить как аналог банка с несколькими кассами, где специальный сотрудник – диспетчер – направляет каждого клиента к свободному кассиру (актуальному серверу). Такая система эффективно работает, пока все клиенты имеют одинаковый приоритет. Однако, если появляются клиенты с более высоким приоритетом, которые должны обслуживаться в специальном окне, то возникают проблемы. Учет приоритета клиентов особенно важен в системах, работающих с транзакциями в реальном времени, но архитектура с диспетчеризацией не может предоставить такую возможность.

Современное решение проблемы для СУБД на многопроцессорных платформах заключается в возможности запуска нескольких серверов базы данных, причем каждый из этих серверов должен быть многопотоковым. Если оба условия выполнены, можно говорить о многопотоковой архитектуре с несколькими серверами, как это показано на рис. 40.

Эта архитектура также известна многонитевой мультисерверной архитектурой, и она ориентирована на распараллеливание выполнения одного пользовательского запроса несколькими серверными процессами.

Для распараллеливания запроса существует несколько подходов. Один из них заключается в разбиении пользовательского запроса на несколько подзапросов, которые могут выполняться одновременно. После выполнения этих подзапросов результаты объединяются, чтобы предоставить общий результат запроса. В таком случае, чтобы ускорить выполнение запросов, подзапросы могут быть направлены различным серверным процессам, а затем их результаты комбинируются в единый итог (рис. 41).

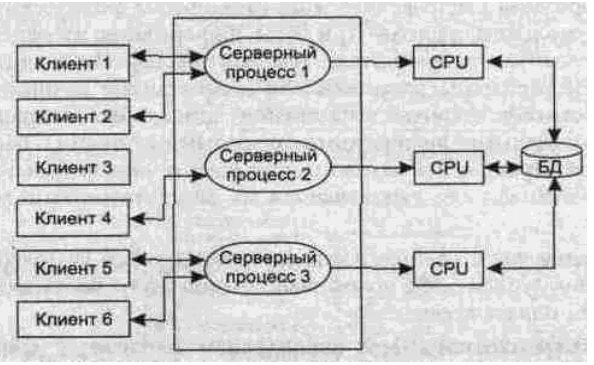


Рис. 40. *Многопотоковая мультисерверная архитектура*

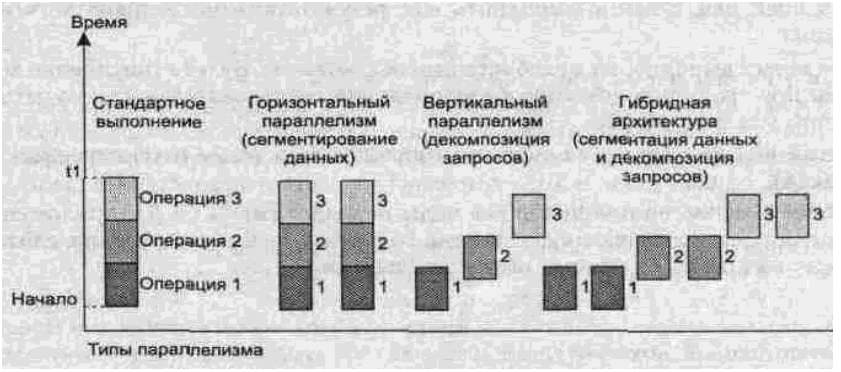


Рис. 41. *Многонитевая мультисерверная архитектура*

Здесь серверные процессы не являются полностью независимыми, как это было в предыдущих моделях. Эти процессы называют **нитями (threads)**. Управление нитями, которые обрабатывают запросы различных пользователей, требует дополнительных затрат ресурсов от СУБД, однако этот подход оказывается особенно эффективным при оперативной обработке данных в хранилищах.

Существует несколько способов распараллеливания запросов в СУБД.

**Горизонтальный параллелизм**. Этот тип параллелизма применяется, когда данные в базе данных распределяются на несколько физических устройств хранения, таких как разные диски. В рамках этого подхода данные одного отношения разбиваются на части по горизонтали. Такой способ распараллеливания также называется **сегментацией данных**. Параллельность достигается путем выполнения одинаковых операций, например,

фильтрации, над различными частями данных. Эти операции могут выполняться одновременно разными процессами, и они не зависят друг от друга. Конечный результат запроса получается путем объединения результатов от каждого из подзапросов.

Время выполнения запроса в таком случае значительно сокращается по сравнению с традиционным методом, при котором запрос выполняется одним процессом.

**Вертикальный параллелизм**. Этот тип параллелизма основан на выполнении операций запроса поэтапно в виде конвейера. Такой подход требует значительных изменений в модели выполнения реляционных операций ядром СУБД. В нем ядро СУБД декомпозирует запрос на его функциональные компоненты, и при этом несколько подзапросов могут выполняться одновременно, связь между различными этапами минимальна.

**Гибридный параллелизм** представляет собой сочетание горизонтального и вертикального параллелизма, сочетая оба метода для более эффективного выполнения запросов.

# Глава 11. ЗАЩИТА ИНФОРМАЦИИ В БАЗАХ ДАННЫХ

В современных СУБД используются два основных подхода для обеспечения безопасности данных: избирательный и обязательный. В обоих случаях объектами данных, для которых необходимо создать систему безопасности, могут быть как вся база данных целиком, так и отдельные объекты внутри базы данных.

Эти подходы различаются следующими характеристиками.

**Избирательный подход** предполагает, что пользователи имеют различные права (привилегии или полномочия) для работы с данными объектами. Разные пользователи могут иметь разные уровни доступа к одному и тому же объекту. Такой подход отличается гибкостью в управлении правами доступа.

**Обязательный подход** заключается в том, что каждому объекту данных присваивается определённый классификационный уровень, и доступ к этим объектам имеют только пользователи, обладающие соответствующим уровнем допуска.

Для реализации избирательного подхода используется следующий метод: в базе данных вводится новый тип объектов – пользователи. Каждому пользователю в системе присваивается уникальный идентификатор. Для повышения безопасности каждому пользователю, помимо идентификатора, назначается уникальный пароль. При этом, если идентификаторы пользователей доступны системному администратору, пароли обычно хранятся в зашифрованном виде и известны только самим пользователям.

Эта архитектура также называется многонитевой мультисерверной архитектурой. Она связана с распараллеливанием выполнения одного пользовательского запроса несколькими серверными процессами.

Пользователи могут быть объединены в специальные группы. Один пользователь может быть частью нескольких групп. В стандарте определяется группа **PUBLIC**, для которой должен быть задан минимальный набор прав. По умолчанию предполагается, что каждый новый пользователь относится к группе **PUBLIC**, если не указано иное.

Привилегии или полномочия пользователей и групп – это набор действий (операций), которые они могут выполнять с объектами базы данных.

В последних версиях некоторых коммерческих СУБД введено понятие **роли**. Роль представляет собой набор полномочий с именем. Во время установки базы данных создаются стандартные роли, и также возможно создавать новые роли, группируя в них различные полномочия. Роли позволяют упростить управление привилегиями, структурировать процесс назначения прав. Важно, что роли не привязаны к конкретным пользователям, их можно определить и настроить до создания пользователей.

Пользователю может быть назначена одна или несколько ролей.

Объектами базы данных, которые должны быть защищены, являются все элементы, хранимые в базе данных: таблицы, представления, хранимые процедуры и триггеры. Для каждого типа объекта могут быть определены различные права доступа, так как для разных объектов существуют разные действия.

На базовом уровне концепции обеспечения безопасности баз данных достаточно просты. Нужно поддерживать два ключевых принципа: **проверку полномочий** и **проверку подлинности** (аутентификацию).

**Проверка полномочий** основывается на том, что для каждого пользователя или процесса информационной системы существует набор действий, которые он может выполнить с определёнными объектами.

**Проверка подлинности** означает достоверное подтверждение того, что пользователь или процесс, пытающийся выполнить действие, действительно тот, за кого себя выдает.

Система назначения полномочий имеет иерархический характер. Самые высокие полномочия обычно принадлежат системному администратору или администратору сервера базы данных. Традиционно только эти пользователи могут создавать новых пользователей и наделять их правами.

Системы управления базами данных хранят информацию как о пользователях, так и о их привилегиях в отношении различных объектов базы данных.

Процесс предоставления полномочий в СУБД организован следующим образом: каждый объект в базе данных имеет своего владельца – пользователя, который создал этот объект. Владелец объекта обладает полными правами на данный объект, включая возможность предоставлять или забирать права на доступ к объекту другим пользователям.

В некоторых СУБД существует дополнительный уровень иерархии пользователей – **администратор базы данных**. В таких системах один сервер может управлять множеством баз данных, как это реализовано в MS SQL Server или Sybase.

В **Oracle** используется однобазовая архитектура, поэтому здесь вводится понятие **подсхемы** – части общей схемы базы данных. В этой архитектуре пользователи получают доступ только к определённой подсхеме.

Стандарт SQL не предусматривает команду для создания пользователей, однако почти все коммерческие СУБД позволяют создавать пользователей не только через интерфейс, но и программно, с помощью специальных хранимых процедур. Чтобы выполнить эту операцию, пользователь должен иметь соответствующие права для запуска нужной системной процедуры.

В SQL стандарте предусмотрены два ключевых оператора для работы с привилегиями: **GRANT** – для предоставления прав, и **REVOKE** – для отмены прав.

## 11.1. Реализация системы защиты в MS SQL Server

SQL Server 6.5 предоставляет три режима аутентификации пользователей:

1. Стандартный режим (Standard).
2. Интегрированный режим (Integrated Security).
3. Смешанный режим (Mixed).

**Стандартный режим** требует, чтобы каждый пользователь имел учетную запись в домене NT Server. Учетная запись домена включает имя пользователя и его уникальный пароль. Пользователи могут быть объединены в группы, и как часть домена, они получают доступ к определенным ресурсам, включая SQL Server. Однако для доступа к SQL Server необходима дополнительная учетная запись, которая также должна содержать имя пользователя и пароль для этого сервера. При подключении к операционной системе пользователь вводит имя и пароль домена, а при подключении к серверу базы данных – имя пользователя и пароль для SQL Server.

**Интегрированный режим** предполагает, что для пользователя существует лишь одна учетная запись в операционной системе (как пользователь домена). В этом случае SQL Server идентифицирует пользователя по данным этой учетной записи. Пользователь вводит только одно имя и пароль для подключения как в операционную систему, так и в SQL Server.

**Смешанный режим** позволяет одновременно использовать как стандартный, так и интегрированный режимы. В этом случае часть пользователей подключается с использованием стандартного режима, а другие – через интегрированный. Когда пользователь пытается подключиться, система сначала проверяет, какой метод аутентификации применяется к этому пользователю. Если используется **Windows NT Authentication Mode**, то проверяется, имеет ли пользователь домена доступ к SQL Server. Если доступ есть, то осуществляется попытка подключения с использованием учетных данных пользователя домена. Если доступ отсутствует, пользователю выводится сообщение об ошибке. При смешанном режиме аутентификации SQL Server

проверяет имя пользователя и пароль. Если они правильные, подключение проходит успешно, в противном случае выводится сообщение о невозможности подключения.

Для **СУБД Oracle** всегда используется дополнительная проверка, помимо имени пользователя и пароля в операционной системе, также требуется ввод имени пользователя и пароля для работы с сервером базы данных.

## 11.2. Проверка полномочий

Вторая задача, связанная с работой с базой данных, – это проверка полномочий пользователей. Эти полномочия хранятся в специальных системных таблицах, и их проверка осуществляется ядром СУБД при выполнении каждой операции. Логически для каждого пользователя и каждого объекта базы данных создается условная матрица, в которой по одному измерению расположены объекты, а по другому – пользователи. На пересечении строк и столбцов этой матрицы указываются разрешенные операции для каждого пользователя над каждым объектом. На первый взгляд, эта модель проверки выглядит достаточно устойчиво, но возникают сложности, когда используется косвенное обращение к объектам. Например, пользователь **user_N** не имеет доступа к таблице **Tab1**, но у него есть разрешение на запуск хранимой процедуры **SP_N**, которая делает выборку из этой таблицы. По умолчанию все хранимые процедуры выполняются под именем их владельца.

Подобные проблемы требуют организационных методов для их решения. При предоставлении доступа пользователям необходимо учитывать возможность косвенного доступа.

В любом случае проблема защиты данных не является исключительно технической задачей. Это комплекс организационно-технических мероприятий, направленных на обеспечение максимальной конфиденциальности информации, хранимой в базе данных.

Кроме того, в условиях работы в сети возникает еще одна проблема – проверка подлинности полномочий. Проблема заключается в следующем: например, процессу 1 даны полномочия для работы с базой данных, а процессу 2 они не предоставлены. В таком случае процесс 2 не может напрямую обратиться к базе данных, но может обратиться к процессу 1 и через него получить доступ к данным. Поэтому в безопасной среде должна существовать модель проверки подлинности, которая подтверждает заявленные идентификаторы пользователей или процессов.

Проблемы проверки подлинности особенно актуальны в условиях массового распространения распределенных вычислений. С высокой степенью связности вычислительных систем необходимо контролировать все обраще-

ния к системе. Проблемы проверки подлинности часто относятся к сфере безопасности сетей и коммуникаций, поэтому их подробно рассматривать не будем. Однако стоит отметить, что в целостной системе безопасности компьютеров проверка подлинности имеет прямое отношение к безопасности баз данных.

Стоит также подчеркнуть, что модель безопасности, основанная на проверке полномочий и подлинности, не решает таких проблем, как украденные идентификаторы пользователей и пароли, или злонамеренные действия пользователей, обладающих полномочиями. Например, программист, имеющий полный доступ к учетной базе данных, может встроить в программу «Троянского коня» с целью хищения или изменения информации, хранимой в базе данных. Поэтому программа обеспечения безопасности информации должна охватывать не только технические области (защита сетей, баз данных и операционных систем), но и вопросы физической защиты, надежности персонала (например, скрытые проверки), аудита и различных процедур поддержки безопасности, которые выполняются вручную или частично автоматизированы.

# Глава 12. СИСТЕМЫ ОБРАБОТКИ ТРАНЗАКЦИЙ

## 12.1. Системы OLTP и OLAP

Фактографические системы можно разделить на два ключевых класса: системы операционной обработки данных и системы, предназначенные для анализа данных и поддержки принятия решений.

Первый класс ориентирован на быстрое выполнение сравнительно простых запросов, поступающих от большого количества пользователей. Такие системы должны обеспечивать защиту данных от несанкционированного доступа, нарушений целостности, а также возможных сбоев в аппаратном или программном обеспечении. Время обработки типичных запросов в этих системах, как правило, не превышает нескольких секунд. Основные сферы их применения включают платежные системы, бронирование мест в поездах, самолетах и гостиницах, банковские и биржевые платформы. Основной логической единицей таких систем является транзакция, представляющая собой завершенное действие с точки зрения пользователя. В современной литературе подобные системы часто обозначаются термином OLTP (On-Line Transaction Processing), что переводится как оперативная обработка транзакций или выполнение транзакций в реальном времени. В последующих разделах будут рассмотрены основные понятия транзакций, процесс их выполнения в OLTP-системах, способы обеспечения целостности базы данных, а также методы эффективного управления ресурсами в распределенных системах операционной обработки.

Второй класс информационных систем включает системы поддержки принятия решений, также называемые аналитическими системами. Они предназначены для обработки сложных запросов, требующих анализа исторических данных, собранных за определенный период. Такие системы позволяют моделировать процессы предметной области, прогнозировать будущие тенденции и применять методы искусственного интеллекта для обработки информации. Они используют значительные объемы накопленных данных, чтобы извлекать из них полезную информацию и формировать новые знания.

Современные потребности в скорости и качестве анализа привели к появлению систем оперативной аналитической обработки данных (OLAP – On-Line Analysis Processing). Высокая производительность таких систем

обеспечивается за счет мощных вычислительных ресурсов, в том числе многопроцессорных систем, продвинутых аналитических алгоритмов и специализированных хранилищ данных, в которых собирается информация из различных источников за длительный период времени и обеспечивается быстрый доступ к ней.

Оба класса систем основаны на системах управления базами данных (СУБД), однако характер их запросов существенно различается. Например, в OLTP-системе, предназначенной для продажи железнодорожных билетов, запрос может выглядеть следующим образом: «Есть ли свободные места в купе поезда Москва-Сочи, отправляющегося 20 августа в 23:15?». В то же время в аналитической системе возможен запрос такого типа: «Какой прогнозируемый объем продаж железнодорожных билетов в денежном выражении на ближайшие три месяца с учетом сезонных колебаний?» Различия в структуре баз данных OLTP и OLAP-систем также являются принципиальными.

Эти отличия, а также особенности обработки данных в OLTP и OLAP системах будут рассмотрены далее.

## 12.2. Обработка транзакций в OLTP-системах

Транзакция представляет собой неделимую последовательность операций с данными в базе данных. В ее состав могут входить чтение, удаление, вставка и изменение информации. В системах OLTP транзакция выполняет определенное значимое действие с точки зрения пользователя, например, перевод средств между банковскими счетами или бронирование билетов на поезд.

Ранее обработка транзакций применялась преимущественно в масштабных системах управления данными, таких как международные банковские платформы. Однако со временем информационные системы становятся все более распределенными и разнородными, что усиливает необходимость в механизмах защиты целостности данных и управления доступом. Одним из эффективных решений этой задачи является использование транзакционных механизмов в информационных системах.

Чтобы гарантировать корректность работы с данными и их защиту от сбоев, транзакции должны обладать четырьмя ключевыми свойствами: атомарностью (atomicity), согласованностью (consistency), изолированностью (isolation) и долговечностью (durability). Транзакции, соответствующие этим требованиям, называют ACID-транзакциями (по первым буквам английских терминов).

Атомарность означает, что транзакция должна выполняться как единое целое: либо все ее операции завершаются успешно, либо ни одна из них не выполняется. В случае возникновения ошибки или сбоя система должна отменить все изменения, внесенные в ходе выполнения транзакции, чтобы данные оставались в непротиворечивом состоянии. Это свойство часто выражают фразой: «все или ничего».

Свойство согласованности обеспечивает сохранение целостности данных, гарантируя соблюдение всех установленных ограничений после завершения транзакции. Важно отметить, что во время выполнения транзакции могут временно нарушаться ограничения целостности, однако к моменту ее завершения база данных должна быть приведена в согласованное состояние.

В многопользовательских системах, где одновременно с одной базой данных работают несколько пользователей или приложений, транзакции могут изменять одни и те же данные. Вследствие этого данные могут находиться в несогласованном состоянии. Свойство изолированности обеспечивает выполнение транзакций независимо друг от друга, запрещая доступ к измененным данным другим транзакциям до их завершения.

Свойство долговечности означает, что после успешного выполнения транзакции все внесенные изменения сохраняются и не теряются, даже в случае сбоя системы.

После выполнения транзакции возможны два исхода: фиксация или откат. Фиксация (коммит) транзакции подтверждает и записывает все изменения в базу данных, делая их доступными для других транзакций. До момента фиксации можно отменить все произведенные изменения, вернув базу данных в состояние, предшествующее началу транзакции.

Если транзакция не может завершиться корректно, например, из-за нарушения ограничений целостности или отмены пользователем, выполняется ее откат (роллбэк). Это означает, что база данных возвращается в исходное состояние, а все изменения аннулируются.

Процесс управления фиксацией и откатом транзакций реализуется с использованием журнала транзакций. Этот механизм позволяет отслеживать все изменения, внесенные в базу данных. Вместо сохранения полной копии базы данных система записывает в журнал только измененные строки – их состояние до и после операции. При фиксации транзакции в журнале делается соответствующая отметка. В случае отката система использует журнал для восстановления данных в исходное состояние, отменяя все произведенные изменения.

Для того чтобы оперировать транзакцией как единой логической единицей, СУБД должна уметь определять ее границы, то есть первую и последнюю входящую в нее операции. Стандарт языка SQL предусматривает

следующий принцип выделения транзакции как некоторой законченной последовательности действий. Предполагается, что транзакция начинается с первого SQL-оператора, вводимого пользователем или содержащегося в прикладной программе. Все следующие далее операторы составляют тело транзакции. Тело транзакции завершается SQL-операторами COMMIT WORK или ROLLBACK WORK. Выполнение транзакции заканчивается также при завершении программы, которая сгенерировала транзакцию. Транзакция фиксируется, если ее тело оканчивается оператором COMMIT WORK, либо при успешном завершении программы, сформировавшей транзакцию. Откат транзакции производится при достижении оператора ROLLBACK WORK, либо в случае, когда приложение, сгенерировавшее транзакцию, завершилось с ошибкой. Возможные варианты завершения выполнения транзакции представлены на рис. 42.

Некоторые диалекты языка SQL, например, диалект, принятый в СУБД Sybase, включают специальные операторы, позволяющие производить промежуточную фиксацию транзакции. В теле транзакции могут быть определены точки, в которых сохраняется состояние базы данных. Откат в этом случае может производиться как к одной из точек промежуточной фиксации, так и к состоянию до начала выполнения транзакции. Точки промежуточной фиксации применяются в «длинных» транзакциях. Они позволяют разделить ее на несколько отдельных фрагментов.

Применение транзакций – эффективный механизм организации многопользовательского доступа к БД. Однако при реализации этого механизма СУБД приходится сталкиваться с целым рядом проблем.

Во-первых, необходимо избежать потери изменений БД в ситуации, когда несколько программ читают одни и те же данные, изменяют их и пытаются записать результат на прежнее место. В БД могут быть сохранены изменения, выполненные только одной программой, результаты работы всех остальных программ будут потеряны.

Во-вторых, необходимо предотвратить ситуацию, при которой одна транзакция читает данные, измененные, но еще не зафиксированные другой транзакцией. Это может привести к ошибкам, если, например, первая транзакция внесла изменения в базу данных, вторая их считала, а затем первая транзакция была отменена с помощью оператора ROLLBACK WORK, аннулировав свои изменения.

Для предотвращения подобных проблем применяется специальный механизм согласованного выполнения (сериализации) транзакций. Его основными принципами являются:

1. **Запрет доступа к незавершенным изменениям** – транзакция не может работать с данными, которые были изменены, но еще не зафиксированы.

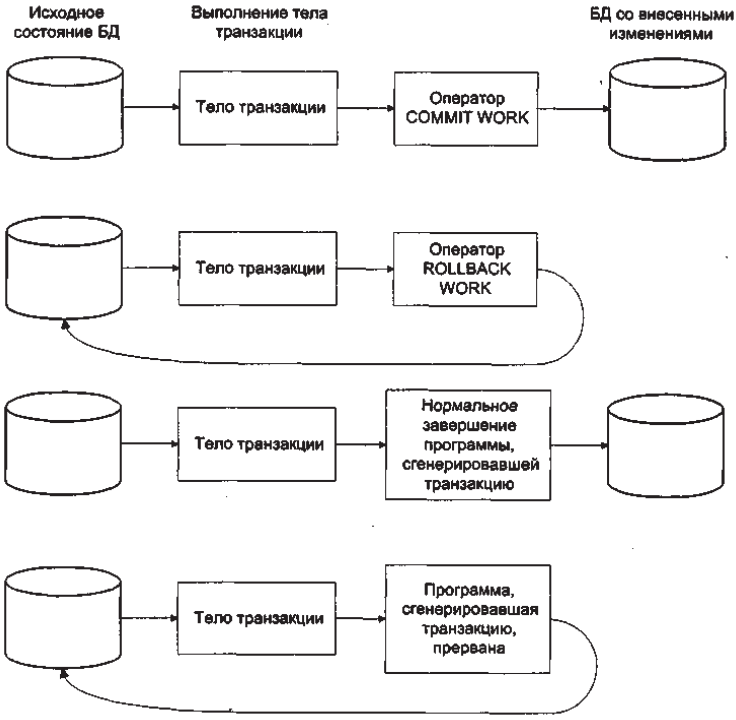


Рис. 42. *Возможные варианты завершения транзакции*

**2. Эквивалентность последовательному выполнению** – результат работы нескольких параллельно выполняемых транзакций должен быть таким же, как если бы они выполнялись строго последовательно: сначала одна, затем другая, либо наоборот.

Современные системы управления базами данных (СУБД) обеспечивают сериализацию транзакций с помощью механизма блокировок. Во время выполнения транзакции СУБД временно ограничивает доступ к определенной части базы данных (будь то отдельная строка, группа строк, таблица или даже вся база). Блокировка сохраняется до завершения транзакции и фиксации ее изменений. Если другая транзакция пытается получить доступ к уже заблокированным данным, ее выполнение приостанавливается до момента освобождения ресурса.

В современных СУБД блокировки могут применяться на разных уровнях: от всей базы данных до конкретных строк. Чем больше область блоки-

ровки, тем выше задержки при обработке транзакций, так как увеличивается время ожидания. Поэтому при работе в режиме оперативного доступа к базе данных чаще всего используется блокировка на уровне отдельных строк. Это позволяет минимизировать время ожидания и повысить производительность системы.

Современные информационные системы часто работают с распределёнными базами данных, поэтому в рамках одной транзакции могут изменяться данные, физически находящиеся на разных вычислительных узлах. Если транзакция вносит изменения сразу на нескольких серверах в сети, её называют распределённой. В случае, когда все операции выполняются в пределах одного узла, транзакция считается локальной. Таким образом, одна распределённая транзакция на логическом уровне состоит из нескольких локальных.

С точки зрения пользователя, локальные и распределённые транзакции должны обрабатываться одинаково. Это означает, что система управления базами данных должна обеспечить корректную фиксацию всех локальных транзакций, входящих в состав распределённой, на соответствующих серверах. Важно, что фиксация возможна только в том случае, если все локальные транзакции завершились успешно. Если хотя бы одна из них прерывается, вся распределённая транзакция должна быть отменена.

Для реализации этого механизма в СУБД применяется двухфазный протокол фиксации (two-phase commit). Этот процесс включает два последовательных этапа.

1. **Подготовка к фиксации.** Сервер, управляющий распределённой транзакцией, отправляет всем задействованным узлам команду на подготовку к фиксации. В ответ локальные серверы должны подтвердить свою готовность. Если хотя бы один из них не отвечает (например, из-за сбоя в работе программного обеспечения или оборудования), происходит отмена изменений на всех узлах, включая те, которые уже выразили готовность.
2. **Фиксация транзакции.** Если все локальные серверы подтвердили свою готовность, управляющий сервер отправляет команду на окончательную фиксацию. После этого транзакция считается завершённой, а все внесённые изменения становятся постоянными.

Использование двухфазного протокола фиксации позволяет гарантировать, что данные будут синхронно обновлены на всех узлах сети, обеспечивая их целостность и согласованность.

## 12.3. Тиражирование данных

Предложенный метод выполнения транзакций в распределённых системах не является единственным вариантом. Альтернативой является техноло-

гия тиражирования данных, которая предполагает отказ от распределения данных, требуя, чтобы на каждом узле вычислительной системы находилась своя собственная копия базы данных. В этой технологии специальные инструменты тиражирования автоматически поддерживают синхронизацию данных между несколькими базами данных, копируя изменения, вносимые в любую из них. Каждая транзакция в такой системе выполняется локально, что исключает необходимость в сложных процедурах фиксации.

Основная проблема этого подхода заключается в поддержании согласованности данных между узлами сети. Процесс передачи изменений из исходной базы данных в базы, расположенные на разных узлах распределённой системы, называется тиражированием данных. Эту задачу выполняет специальный компонент СУБД – сервер тиражирования данных (репликатор). При любом изменении данных в реплицируемой базе данных репликатор распространяет их на все остальные узлы системы. Технология тиражирования может быть реализована через полное обновление данных на удалённых серверах (метод с полным обновлением) или обновление только изменённых записей (метод быстрого обновления). Если система не требует постоянной синхронизации данных и базы данных узлов должны быть согласованы лишь время от времени, репликатор собирает изменения и периодически передаёт их на другие узлы. Процесс тиражирования данных скрыт от прикладных программ пользователей, и репликатор автоматически поддерживает согласованность баз данных.

Использование технологии тиражирования позволяет снизить трафик, так как все запросы обрабатываются локально, а на другие узлы передаются только изменения данных, что также ускоряет доступ к информации. Кроме того, даже при потере связи между узлами система продолжает работать. Однако у этой технологии есть и минусы. Например, невозможно полностью избежать конфликтов, возникающих при одновременном изменении одинаковых данных на разных узлах. В результате, при распространении изменений между узлами системы могут возникать несогласованные версии базы данных, из-за чего пользователи на разных узлах могут получать разные результаты на одинаковые запросы.

## 12.4. Средства восстановления после сбоев

Одним из ключевых требований к современным OLTP-системам является высокая надежность хранения данных. Система управления базами данных должна быть способна восстанавливать согласованное состояние базы данных после любых аппаратных или программных сбоев. Для восстановле-

ния данных СУБД использует журнал транзакций, который представляет собой последовательность записей, фиксирующих изменения в базе данных.

Основной принцип восстановления после сбоя заключается в том, что результаты транзакций, которые были зафиксированы до сбоя, должны быть восстановлены в базе данных. В свою очередь, изменения от незавершенных транзакций не должны быть учтены. Таким образом, восстанавливается последнее согласованное состояние базы данных до сбоя. Процесс восстановления включает в себя откат незавершённых транзакций, как это было описано ранее.

Однако журнал транзакций не сможет помочь в случае, если данные на внешнем носителе системы были физически уничтожены, что приведет к полной утрате информации из базы данных. Чтобы предотвратить такие ситуации, используется дублированное хранение данных, например, зеркалирование дискового пространства – когда данные записываются одновременно на несколько устройств. После сбоя из зеркала копируется содержимое базы данных, и затем, как и в предыдущем случае, с помощью журнала выполняется откат незавершённых транзакций.

## 12.5. Мониторы транзакций

С увеличением сложности распределённых вычислительных систем возникают проблемы эффективного использования их ресурсов. Для решения этих проблем в распределённые OLTP-системы добавляется дополнительный компонент – монитор транзакций (TPM – transaction processing monitor). Мониторы транзакций выполняют две ключевые функции: динамическое распределение запросов в системе (выравнивание нагрузки) и оптимизацию числа выполняемых серверных приложений. Рассмотрим эти функции подробнее.

Когда в системе работают несколько серверов, предоставляющих одинаковые услуги, например, доступ к базе данных, важно оптимизировать пропускную способность системы (количество обработанных запросов за единицу времени). Для этого нужно обеспечить равномерное распределение запросов между серверами, чтобы каждый сервер обрабатывал примерно одинаковое количество запросов. При распределении запросов могут также учитываться такие факторы, как удалённость серверов, их текущая доступность и содержание запроса. Функция выравнивания нагрузки реализуется по схеме (рис. 43).

Когда запрос от клиентского приложения поступает в монитор транзакций, он действует от имени клиента и определяет, на какой сервер направить запрос. Для этого монитор обращается к динамической маршрутной

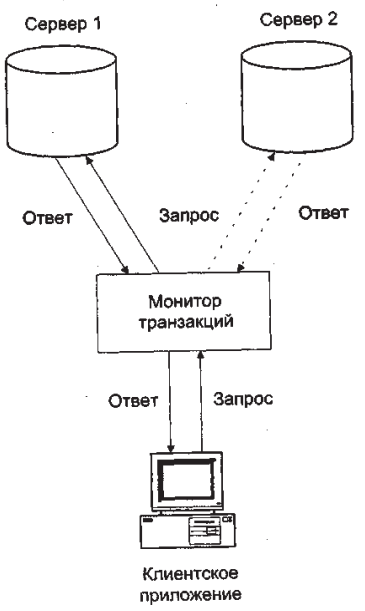


Рис. 43. *Упрощенная схема работы монитора транзакций*

таблице, по которой выбирает систему, предоставляющую требуемую услугу. Если нужный сервис доступен на нескольких системах, то в зависимости от алгоритма маршрутизации выбирается один из серверов, и запрос перенаправляется на него. Алгоритм маршрутизации может быть разным: он может быть случайным, когда система выбирается произвольно, циклическим, когда запросы передаются по очереди, или зависеть от содержания запроса, если, например, серверы базы данных обслуживают разные подмножества данных. После выполнения запроса его результат через монитор транзакций возвращается обратно в клиентское приложение. Клиенты не знают, на какой сервер будет направлен их запрос, предлагают ли несколько серверов нужную услугу, расположен ли сервер локально или удалённо, или обслуживают ли они запросы одновременно локально и удалённо. В любом случае запрос будет обработан оптимальным способом. Такая схема обработки запросов называется «прозрачностью местоположения серверов» (service location transparency).

Скорость обработки транзакций напрямую зависит от количества запущенных серверных приложений. Чем больше приложений одновременно обрабатывают запросы, тем выше пропускная способность вычислительной

системы. Это увеличение особенно заметно на многопроцессорных системах, где каждое приложение может работать на отдельном процессоре. Для эффективного использования системных ресурсов необходимо динамически увеличивать или уменьшать количество серверных приложений в зависимости от объема обрабатываемых запросов. Чтобы решить эту задачу, мониторы транзакций периодически измеряют соотношение числа запросов в очереди к числу активных серверных приложений. Если это соотношение превышает установленный максимальный порог (maximum watermark), запускается дополнительная копия серверного приложения. Если же соотношение падает ниже минимального порогового значения (minimum watermark), одна из копий приложения завершает свою работу.

На рынке существует множество продуктов для мониторов транзакций. Среди наиболее известных можно выделить: TUXEDO компании USL, TOP END от NCR, CICS от IBM, ENCINA от Transarc и ACMS от DEC.

# Список использованной литературы

1. Голицина, О. Л. Базы данных: учебное пособие / О. Л. Голицина. – Москва: ФОРУМ: ИНФА – М, 2006. – 352 с. – Текст : непосредственный.

2. Голицина, О. Л. Системы управления базами данных: учебное пособие / О. Л. Голицина. – Москва: ФОРУМ: ИНФА–М, 2011. – 432 с. – Текст : непосредственный.

3. Золотова, С. И. Практикум по Access / С. И. Золотова. – Москва: Финансы и статистика, 2006. – Текст : непосредственный.

4. Корнеев, В. В. Базы данных. Интеллектуальная обработка информации / В. В. Корнеев, А. Гареев. – Москва: Нолидж, 2000. – 352 с. – Текст : непосредственный.

5. Пушников, А. Ю. Введение в системы управления базами данных : учебное пособие / А. Ю. Пушников. – Уфа: Башкирский гос. ун-т, 1999. – URL : http://www.citforum.ru/database/dblearn/. – Текст : электронный.

6. Форта, Б. Освой самостоятельно SQL. 10 минут на урок, 3-е издание : Перевод с английского / Б. Форта. – Москва: Издательский дом «Вильямс», 2005 – Текст : непосредственный.

### *Интернет-ресурсы*

Центр информационных технологий : [сайт]. – URL : http:// www.citforum.ru//

Учебное издание

**БАЗЫ ДАННЫХ**
*Учебное пособие*

*Составители:*
**Ольга Михайловна Фурдуй**
**Оксана Валерьевна Комарова**

Редактор : *Е.Ю. Кривошеева.* Компьютерная верстка : *Маракуца А.А.*
ИЛ № 06150. Сер. АЮ от 21.02.02.
Подписано в печать 10.06.2025. Формат 60х90/16.
Уч. печ. л. 8,25. Электронное издание. Заказ № 584.
Изд-во Приднестр. ун-та. 3300, г. Тирасполь, ул. Мира, 18.
Опубликовано на образовательном портале moodle.spsu.ru