**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Московский государственный институт электроники и математики**
**(Технический университет)**

**И.П. Карпова**

# ОСНОВЫ БАЗ ДАННЫХ

**Утверждено Редакционно-издательским советом института**
**в качестве Учебного пособия**

**Москва 2007**

УДК 681.3.07
ББК 32.973
К26

Рецензенты: докт. техн. наук И.П. Беляев
канд. ф.-м. наук

Карпова И.П.
К26 Основы баз данных. Учебное пособие. – Московский Государственный институт электроники и математики. – М.: 2007. – 75 с.

Рассматриваются основные модели данных, технологии организации баз данных, методы оптимизации запросов и функциональной поддержки баз данных.

Для студентов дневных и вечерних факультетов технических вузов, изучающих автоматизированные информационные системы и системы управления базами данных.

ISBN

## Содержание

# 1. ВВЕДЕНИЕ

Развитие средств вычислительной техники и информационных технологий обеспечило возможности для создания и широкого использования автоматизированных информационных систем (АИС) разнообразного назначения. Разрабатываются информационные системы управления хозяйственными и техническими объектами, модельные комплексы для научных исследований, системы автоматизации проектирования и производства, всевозможные тренажеры и обучающие системы.

Различают АИС, основанные на знаниях, и АИС, основанные на данных. К первым можно отнести, например, экспертные системы (ЭС), интеллектуальные системы поддержки принятия решений (СППР) и т.п. Ко вторым – всевозможные прикладные системы, которые сейчас активно используются и на предприятиях, и в учреждениях. Такие прикладные системы применяются очень широко, и основное внимание в рамках данного курса мы уделим именно системам, основанным на данных.

Существуют две основные предпосылки создания таких систем:

1. Разработка методов конструирования и эксплуатации систем, предназначенных для коллективного пользования.
2. Возможность собирать, хранить и обрабатывать большое количество данных о реальных объектах и явлениях, то есть оснащение этих систем "памятью".

Массив данных общего пользования в таких системах называется *базой данных*.

**Основным принципом организации базы данных является совместное хранение данных и описания их структуры.**

Это описание обычно называют *метаданными*. В системах обработки данных, не использующих базы данных, интерпретация данных возлагается на программы обработки. А с помощью принципа хранения данных вместе с их описанием обеспечивается независимость данных от программ обработки. Зная формат описания данных, можно запрашивать и модифицировать данные без написания дополнительных программ.

Одни и те же данные БД могут быть использованы для решения многих прикладных задач. Наличие метаданных и возможность информационной поддержки решения многих задач – это принципиальные отличия базы данных от любой другой совокупности данных внешней памяти ЭВМ.

## 1.1. Информация, данные, знания. Терминология

**Информация** – любые сведения о каком-либо событии, сущности, процессе и т.п., являющиеся объектом некоторых операций: восприятия, передачи, преобразования, хранения или использования.

**Данные** – это информация, зафиксированная в некоторой форме, пригодной для последующей обработки, передачи и хранения, например, находящаяся в памяти ЭВМ или подготовленная для ввода в ЭВМ.

**Подготовка информации** состоит в её формализации, сборе и переносе на машинные носители.

**Обработка данных** – это совокупность задач, осуществляющих преобразование массивов данных. Обработка данных включает в себя ввод данных в

ЭВМ, отбор данных по каким-либо критериям, преобразование структуры данных, перемещение данных на внешней памяти ЭВМ, вывод данных, являющихся результатом решения задач, в табличном или в каком-либо ином удобном для пользователя виде.

**Система обработки данных** (СОД) – это набор аппаратных и программных средств, осуществляющих выполнение задач по управлению данными.

**Управление данными** – совокупность функций обеспечения требуемого представления данных, их накопления и хранения, обновления, удаления, поиска по заданному критерию и выдачи данных. [6]

**Предметная область** – часть реального мира, подлежащая изучению с целью организации управления и, в конечном итоге, автоматизации.

**База данных** (БД) – совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ [6]. Эти данные относятся к определённой предметной области и организованы таким образом, что могут быть использованы для решения многих задач многими пользователями.

**Ведение базы данных** – деятельность по обновлению, восстановлению и перестройке структуры базы данных с целью обеспечения ее целостности, сохранности и эффективности использования.

**Система управления базами данных** (СУБД) – это совокупность программ и языковых средств, предназначенных для управления данными в базе данных, ведения базы данных и обеспечения взаимодействия её с прикладными программами.

**Автоматизированная информационная система** (АИС) представляет собой совокупность информации, экономико-математических методов и моделей, технических, программных средств и специалистов, предназначенную для обработки информации и принятия управленческих решений.

**Банк данных** (БнД) – это автоматизированная информационная система, включающая в свой состав комплекс специальных методов и средств (математических, информационных, программных, языковых, организационных и технических) для поддержания динамической информационной модели предметной области с целью обеспечения информационных запросов пользователей.

## 1.2. Автоматизированная информационная система

Автоматизированная информационная система (АИС), основанная на базе данных, служит для сбора, накопления, хранения информации, а также её эффективного использования для различных целей. Информация представляется в виде данных, хранимых в памяти ЭВМ. При проектировании АИС, с одной стороны, решается вопрос о том, какие сведения и для каких целей будут содержаться в системе, с другой – как соответствующие данные будут организованы в памяти ЭВМ и как они будут обрабатываться при эксплуатации АИС.

По сферам применения и правилам организации различают два основных класса АИС, основанных на базе данных: **информационно-поисковые** (ИПС)

и **системы обработки данных** (СОД). ИПС ориентированы, как правило, на извлечение подмножества хранимых данных, удовлетворяющих некоторому поисковому критерию. Пользователя ИПС интересует, в основном, сами извлекаемые из базы данных сведения, а не результаты их обработки. Примером ИПС является справочная служба: к ней обращаются с запросом и получают в результате те данные, которые удовлетворяют этому запросу.

Обращения пользователя к СОД чаще всего приводят к обновлению данных. Вывод данных может вовсе отсутствовать или представлять собой результат программной обработки хранимых сведений. Пример СОД – банковские системы, осуществляющие открытие/закрытие счетов, пересчёт вкладов в зависимости от процентов, приём/снятие сумм и т.п.

В зависимости от характера информационных ресурсов, с которыми имеют дело АИС, их подразделяют на **документальные** и **фактографические**. На практике используются также системы комбинированного типа.

В документальной системе объект хранения – документ, который содержит информацию, относящуюся к определённой предметной области. Это могут быть графические изображения (например, географические карты); информация на естественном языке (монографии, тексты законодательных актов, научные отчёты и т.п.); звуковая информация (например, мелодии для системы, хранящей фонотеку) и т.д. Для обработки информации не важно, какие сведения хранятся в документах. Обычно документальные АИС реализуются в виде информационно-поисковых систем.

Основные компоненты документальной ИПС – это программные средства, поисковый массив документов, средства поддержки информационного языка системы. Программные средства ИПС служат для организации управления данными (ввода, хранения, защиты, поиска и выдачи). Поисковый массив документов в ИПС обычно называется базой данных. Он представляет собой набор ссылок на документы (или их описаний), хранящий основную информацию о документах и организованный так, чтобы обеспечить быстрый поиск документов. Описание документа зависит от предметной области и состоит из значений атрибутов, характеризующих содержание документа. Например, для БД географических карт это могут быть координаты и масштаб, а для БД законодательных актов – тип документа (закон, постановление и др.), дата принятия, область действия и т.п.

Информационный язык системы предназначен для того, чтобы пользователь мог сформулировать запрос к системе. Системными средствами запрос преобразуется в формализованное поисковое предписание – *поисковый образ запроса*. Этот образ сопоставляется с поисковыми образами документов, хранимыми в системе, по критерию смыслового соответствия. Информационный язык системы может быть основан на подмножестве естественного языка, которое относится к обслуживаемой ПО. Но чаще поиск документа осуществляется с помощью шаблонов – экранных форм, включающих поля описания документа. В эти поля вносятся конкретные значения, которые и определяют условия поиска документов. На рис. 1.1 приведен пример такого поиска через экранную

форму. В форме указаны два условия – округ и фрагмент названия, и в результате поиска система выдала три записи, удовлетворяющие этим условиям.

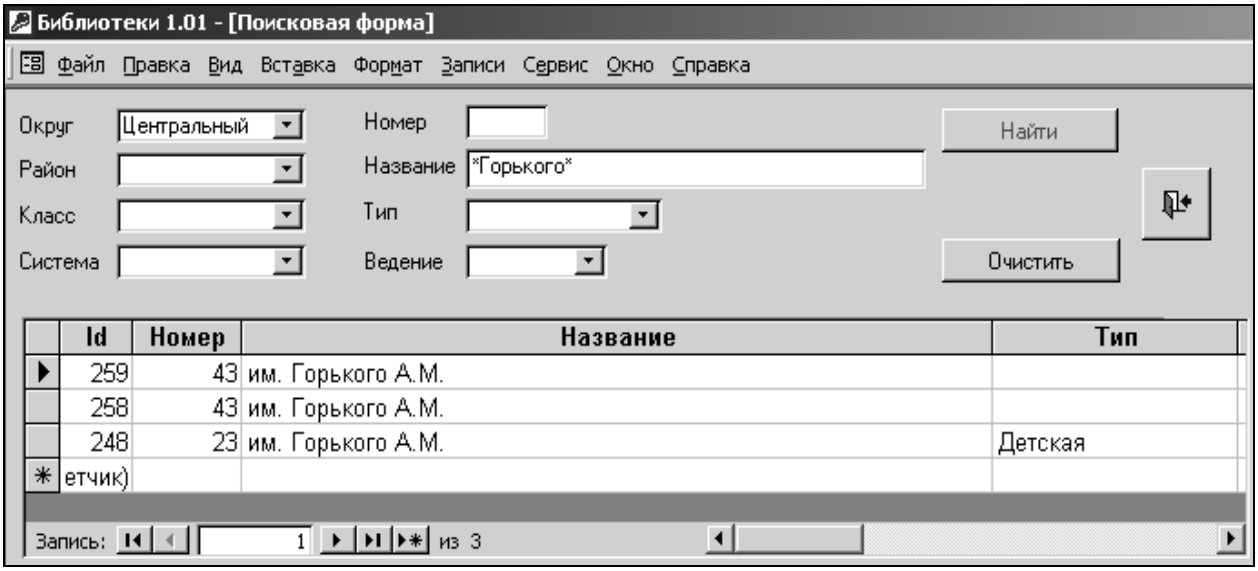


Рис.1.1. Пример поиска данных через экранную форму

Фактографические АИС хранят сведения об объектах предметной области, их свойствах и взаимосвязях. Сведения о каждом объекте могут поступать в систему из множества различных источников. Кроме поиска и модификации данных, фактографические системы поддерживают статистические функции (нахождение суммы, минимума, максимума и т.п.).

Мы будем основное внимание обращать на фактографические АИС, имея в виду, что ИПС и документальные АИС создаются с помощью те же программных средств и на тех же принципах, что и СОД, а специфические моменты обработки данных реализуются через приложения (программы, внешние по отношению к ядру СОД).

Разработка любой ИС начинается с определения предметной области.

## 1.3. Предметная область информационной системы

Предметная область (ПО) информационной системы рассматривается как совокупность реальных процессов и объектов (сущностей), представляющих интерес для её пользователей. Каждый из объектов ПО обладает определённым набором свойств (атрибутов), среди которых можно выделить существенные и малозначительные. Признание какого-либо свойства существенным носит относительный характер. Например, атрибут *Должность* для сотрудника является существенным, а для читателя библиотеки – малозначительным.

Для упрощения процедуры формализации ПО в большинстве случаев прибегают к разбиению всего множества объектов ПО на группы объектов, однородных по структуре и поведению (относительно рамок рассматриваемой ПО), называемых *типами объектов*. Данные предметной области представлены *экземплярами объектов*. Экземпляры объектов одного типа обладают одинаковыми наборами атрибутов, но должны отличаться значением хотя бы одного атрибута для того, чтобы быть узнаваемыми.

Между объектами ПО могут существовать связи, имеющие различный содержательный смысл (семантику). Эти связи могут быть **факультативными** или **обязательными**. Если вновь порождённый объект одного из типов оказывается по необходимости связанным с объектом другого типа, то между этими типами объектов существует обязательная связь. Иначе связь является факультативной. Примеры обязательной и факультативной связей приведены на рис. 1.2. Здесь связь *замещает* является обязательной, потому что каждый сотрудник должен работать на определенной должности, а связь *замещается* является факультативной, т.к. должность может быть вакантна.

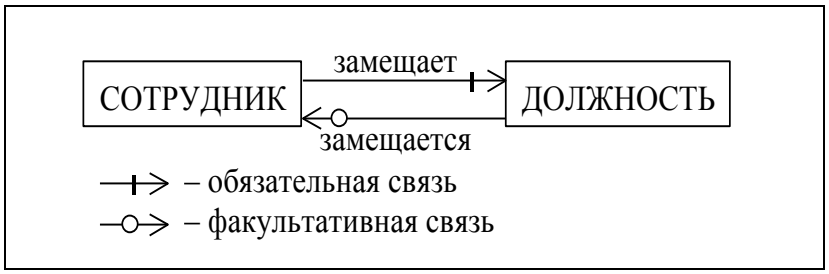


Рис.1.2. Примеры обязательной и факультативной связей

Для удобства связи между объектами двух типов изображают одной двунаправленной стрелкой. Различают также типы множественных связей: "один к одному" (1:1), "один ко многим" (1:n) и "многие ко многим" (m:n) (рис. 1.3).

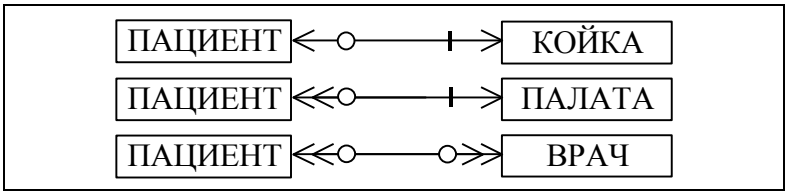


Рис.1.3. Примеры типов множественных связей

Связи, приведённые на рис. 1.3, с учётом семантики означают следующее:

- пациент–койка (1:1) – каждый пациент занимает одну койку, каждая койка может быть занята только одним пациентом;
- палата–пациент (1:n) – каждый пациент находится в одной палате, в каждой палате могут находиться несколько пациентов;
- пациент–врач (n:m) – каждый пациент может лечиться у нескольких врачей, каждый врач может лечить несколько пациентов.

Обратите внимание: необязательная связь имеет модификатор "может", а у обязательной связи его нет.

Совокупность типов сущностей и типов связей между ними характеризует структуру предметной области. Собственно данные представлены экземплярами объектов и связей между ними. Данные экземпляров объектов и связей хранятся в базе данных информационной системы.

Множества экземпляров объектов, значения атрибутов объектов и связи между ними могут изменяться во времени. Поэтому каждому моменту времени можно сопоставить некоторое **состояние предметной области**. Состояния ПО должны подчиняться совокупности правил, которые характеризуют семантику предметной области. В базе данных эти правила могут быть заданы с помощью

так называемых **ограничений целостности**, которые накладываются на атрибуты объектов, типы объектов, типы связей и/или их экземпляры. Ограничения целостности – это правила, которым должны удовлетворять значения данных в БД. Например, для библиотеки можно привести такие ограничения целостности: количество экземпляров книги не может быть отрицательным; номер паспорта читателя должен быть уникальным; каждая книга относится к определенному разделу рубрикатора ББК – библиотечно-библиографической классификации и т.д.

Для того чтобы обеспечить соответствие базы данных текущему состоянию предметной области, база данных *динамически обновляется* (периодически или в режиме реального времени). Это обновление называется **актуализацией данных.** Актуализация может проводиться:

- вручную, если изменения в данные вносит пользователь;
- автоматизировано, если изменения инициируются пользователем, но выполняются программно;
- автоматически, если данные поступают в электронном виде и обрабатываются программой без участия человека.

Правильность обновлений может контролироваться как программно, так и автоматически с помощью ограничений целостности.

База данных является информационной моделью внешнего мира, некоторой предметной области. Во внешнем мире объекты ПО взаимосвязаны, поэтому в БД эти связи должны быть отражены. Если связи между данными в БД отсутствуют, то имеет смысл говорить о нескольких независимых БД, имеющих раздельное хранение.

## 1.4. Назначение и основные компоненты системы баз данных

Система БД включает два основных компонента: собственно базу данных и систему управления базами данных – СУБД (рис. 1.4). Большинство СОД включают также программы обработки данных (прикладное программное обеспечение, ППО), которые обращаются к данным через СУБД. В соответствии с рис. 1.4 СУБД обеспечивает выполнение двух групп функций:

- предоставление доступа к базе данных пользователям (или ППО);
- управление хранением и обработкой данных в БД.

Таким образом, обращение к базе данных возможно только через СУБД.

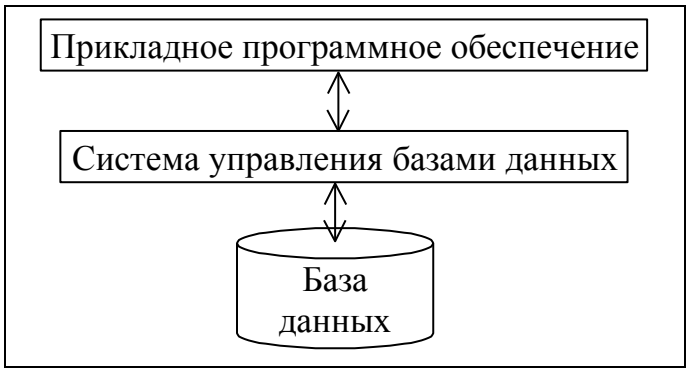


Рис.1.4. Компоненты системы баз данных

БД предназначена для хранения данных информационной системы. Пользователи обращаются к базе данных обычно не напрямую через средства СУБД, а с помощью внешнего интерфейса – приложения, входящего в состав АИС. Если пользователей можно разделить на группы по характеру решаемых задач, то приложений может быть несколько (по количеству задач или групп пользователей). Например, для библиотеки можно выделить 3 группы пользователей: читатели, которым нужно осуществлять поиск книг по автору, названию или теме; сотрудники, выдающие и принимающие у читателей книги (библиотекари) и сотрудники отдела комплектации, осуществляющие приём новых книг.

## 1.5. Уровни представления данных

Современная технология баз данных основана на концепции многоуровневой архитектуры СУБД. Эти идеи впервые были сформулированы в отчёте рабочей группы по базам данных Комитета по планированию стандартов Американского национального института стандартов (ANSI/X3/SPARC). Этот отчёт был опубликован в 1975 г. В нём была предложена обобщенная трёхуровневая модель архитектуры СУБД, включающая концептуальный, внешний и внутренний уровни (рис. 1.5).

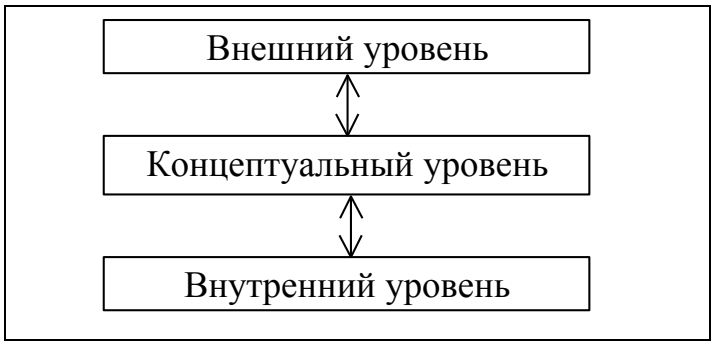


Рис.1.5. Уровни представления данных

**Концептуальный уровень** архитектуры ANSI/SPARC служит для поддержки единого взгляда на базу данных, общего для всех её приложений и независимого от них и от среды хранения [6]. Концептуальный уровень представляет собой формализованную информационно-логическую модель ПО. Описание этого представления называется *концептуальной схемой*.

**Внутренний уровень** архитектуры поддерживает представление данных в среде хранения и пути доступа к ним [6]. На этом архитектурном уровне БД представлена в полностью “материализованном” виде, тогда как на других уровнях идёт работа на уровне отдельных экземпляров или множества экземпляров записей. Описание БД на внутреннем уровне называется *внутренней схемой* или *схемой хранения.*

**Внешний уровень** архитектуры БД предназначен для различных групп пользователей. Описания таких представлений называются *внешними схемами.* В системе баз данных могут одновременно поддерживаться несколько внешних схем для различных групп пользователей или задач [6].

**Схема базы данных** – это описание базы данных в контексте конкретной модели данных.

Каждый из этих уровней может считаться управляемым, если он обладает внешним интерфейсом, который поддерживает возможности определения данных. В этом случае становится возможными формирование и системная поддержка независимого взгляда на БД для какой-либо группы персонала или пользователей, взаимодействующих с БД через интерфейс данного уровня.

В архитектурной модели ANSI/SPARC предполагается наличие в СУБД механизмов, обеспечивающих междууровневое отображение данных “внешний – концептуальный” и “концептуальный – внутренний”. Функциональные возможности этих механизмов определяют степень независимости данных на всех уровнях. На переходе “внешний – концептуальный” обеспечивается логическая независимость данных, на переходе “концептуальный – внутренний” – физическая независимость. Под логической независимостью подразумевается возможность вносить изменения в концептуальный уровень, не меняя представление БД для пользователей. Физическая независимость – это возможность вносить изменения в схему хранения, не меняя концептуальную схему БД.

# 2. ОСНОВНЫЕ МОДЕЛИ ДАННЫХ

## 2.1. Понятие модели данных

**Модель данных** – это комбинация трех составляющих:

1. Набора типов структур данных.
   Здесь можно провести аналогию с языками программирования, в которых тоже есть предопределенные типы структур данных, такие как скалярные данные, вектора, массивы, структуры (например, тип struct в языке Си) и т.д.
2. Набора операторов или правил вывода, которые могут быть применены к любым правильным примерам типов данных, перечисленных в (1), чтобы находить, выводить или преобразовывать информацию, содержащуюся в любых частях этих структур в любых комбинациях.

   Такими операциями являются: создание и модификация структур данных, внесение новых данных, удаление и модификация существующих данных, поиск данных по различным условиям.
3. Набора общих правил целостности, которые прямо или косвенно определяют множество непротиворечивых состояний базы данных и/или множество изменений её состояния.

   Правила целостности определяются типом данных и предметной областью. Например, значение атрибута *Счётчик* является целым числом, т.е. может состоять только из цифр. А ограничения предметной области таковы, что это число не может быть меньше нуля.

Теперь рассмотрим подробнее составляющие модели данных.

### 2.1.1. Типы структур данных

Структуризация данных базируется на использовании концепций "агрегации" и "обобщения". Первый вариант структуризации данных был предложен

Ассоциацией по языкам обработки данных (Conference on Data Systems Languages, CODASYL) (рис. 2.1).

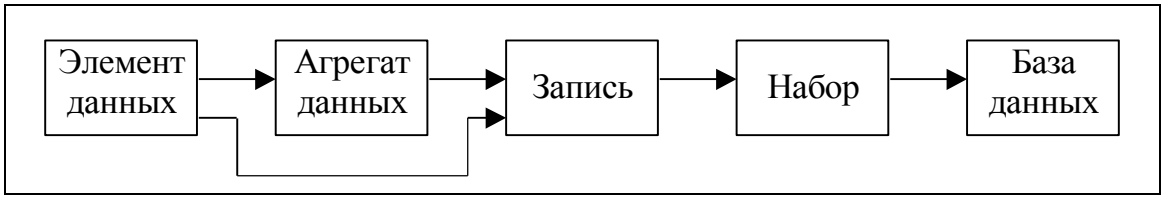


Рис.2.1. Композиция структур данных по версии CODASYL

**Элемент данных** – наименьшая поименованная единица данных, к которой СУБД может обращаться непосредственно и с помощью которой выполняется построение всех остальных структур. Для каждого элемента данных должен быть определён его тип.

**Агрегат данных** – поименованная совокупность элементов данных внутри записи, которую можно рассматривать как единое целое. Агрегат может быть простым (включающим только элементы данных, рис. 2.2,а) и составным (включающим наряду с элементами данных и другие агрегаты, рис. 2.2,б).

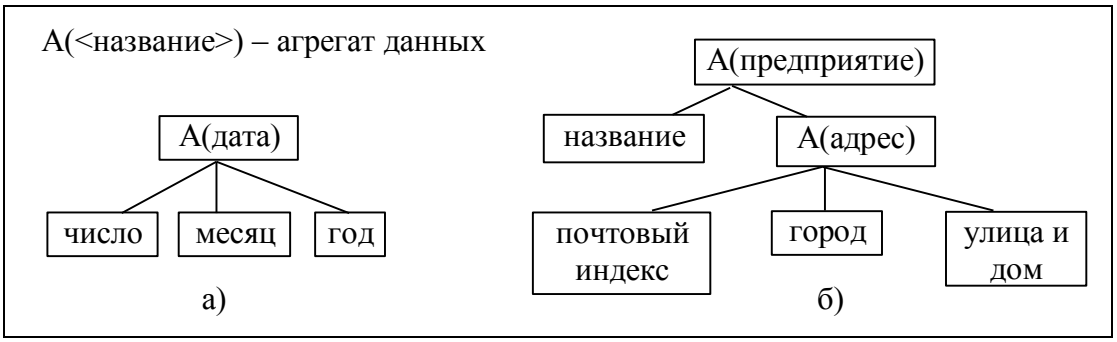


Рис.2.2. Примеры агрегатов: а) простой и б) составной агрегат

**Запись** – поименованная совокупность элементов данных или элементов данных и агрегатов. Запись – это агрегат, не входящий в состав никакого другого агрегата; она может иметь сложную иерархическую структуру, поскольку допускается многократное применение агрегации. Различают тип записи (её структуру) и экземпляр записи, т.е. запись с конкретными значениями элементов данных. Одна запись описывает свойства одного объекта ПО (экземпляра).

Иногда термин "запись" заменяют термином "группа".

Пример записи, содержащей сведения о сотруднике, приведен на рис. 2.3.

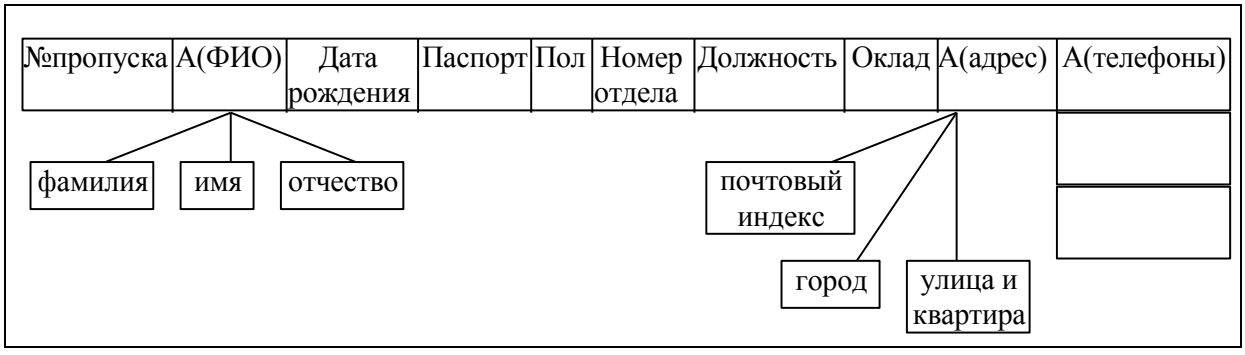


Рис.2.3. Пример записи типа *СОТРУДНИК*

Эта запись имеет несколько элементов данных (*Номер пропуска*, *Должность*, *Пол* и т.д.) и три агрегата: простые агрегаты *ФИО* и *Адрес* и повторяющийся агрегат *Телефоны*. (Повторяющийся агрегат может включаться в запись произвольное число раз).

Среди элементов данных (полей) выделяются одно или несколько ключевых полей. Значения ключевых полей позволяют классифицировать объект, к которому относится конкретная запись. Ключи с уникальными значениями называются *потенциальными*. Каждый ключ может представлять собой агрегат данных. Один из ключей является первичным, остальные – вторичными. **Первичный ключ** идентифицирует экземпляр записи, его значение должно быть уникальным для записей одного типа. Для примера на рис. 2.3 потенциальными ключами являются поля *№ пропуска* и *Паспорт*, а первичным ключом целесообразнее выбрать поле *№ пропуска*, т.к. оно явно занимает меньше памяти, чем паспортные данные.

**Набор** (или групповое отношение) – поименованная совокупность записей, образующих двухуровневую иерархическую структуру. Каждый тип набора представляет собой отношение (связь) между двумя или несколькими типами записей. Для каждого типа набора один тип записи объявляется владельцем набора, остальные типы записи объявляются членами набора. Каждый экземпляр набора должен содержать только один экземпляр записи типа владельца и столько экземпляров записей типа членов набора, сколько их связано с владельцем. Для группового отношения также различают тип и экземпляр.

Групповые отношения удобно изображать с помощью диаграммы Бахмана, которая названа так по имени одного из разработчиков сетевой модели данных. Диаграмма Бахмана – это ориентированный граф, вершины которого соответствуют группам (типам записей), а дуги – групповым отношениям (рис. 2.4).

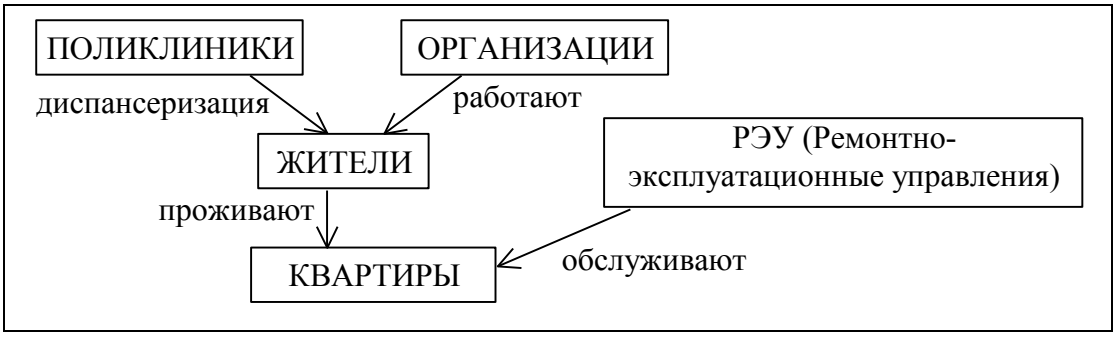


Рис. 2.4. Пример диаграммы Бахмана для фрагмента БД "Город"

Здесь запись типа *ПОЛИКЛИНИКА* является владельцем записей типа *ЖИТЕЛЬ* и они связаны групповым отношением *диспансеризация*. Запись типа *ОРГАНИЗАЦИЯ* также является владельцем записей типа *ЖИТЕЛЬ* и они связаны групповым отношением *работают*. Записи типа *РЭУ* и типа *ЖИТЕЛЬ* являются владельцами записей типа *КВАРТИРА* с отношениями соответственно *обслуживают* и *проживают*. Таким образом, запись одного и того же типа может быть членом одного отношения и владельцем другого.

**База данных** – поименованная совокупность экземпляров групп и групповых отношений. База данных – это самый высокий уровень структуризации данных.

### 2.1.2. Операции над данными

Модель данных определяет множество действий, которые допустимо производить над некоторой реализацией БД для её перевода из одного состояния в другое. Это множество соотносят с языком манипулирования данными (Data Manipulation Language, DML).

Любая операция над данными включает в себя селекцию данных (select), то есть выделение из всей совокупности именно тех данных, над которыми должна быть выполнена требуемая операция, и действие над выбранными данными, которое определяет характер операции. Условие селекции – это некоторый критерий отбора данных, в котором могут быть использованы логическая позиция элемента данных, его значение и связи между данными.

По типу производимых действий различают следующие операции:

- идентификация данных и нахождение их позиции в БД;
- выборка (чтение) данных из БД;
- включение (запись) данных в БД;
- удаление данных из БД;
- модификация (изменение) данных БД.

Обработка данных в БД осуществляется с помощью процедур базы данных – **транзакций**. Транзакция – это последовательность операций над данными, которая является логически неделимой, то есть рассматривается как единая макрооперация. Транзакция либо выполняется полностью, т.е. выполняются се входящие в неё операции, либо не выполняется совсем.

### 2.1.3. Ограничения целостности

Ограничения целостности обеспечивают непротиворечивость данных при переводе системы баз данных из одного состояния в другое и позволяют адекватно отражать предметную область данными, хранимыми в БД. Ограничения целостности делятся на **явные** и **неявные.**

Неявные ограничения определяются самой структурой данных. Например, тот факт, что записи типа *СОТРУДНИК* являются обязательными членами какого-либо экземпляра набора данных *ПОДРАЗДЕЛЕНИЕ*, служит, по существу, ограничением целостности, означающим, что каждый сотрудник организации непременно должен быть в штате некоторого подразделения.

Явные ограничения включаются в структуру базы данных с помощью средств языка описания данных (DDL, Data Definition Language). В качестве явных ограничений чаще всего выступают условия, накладываемые на значения данных. Например, заработная плата не может быть отрицательной, а дата приёма сотрудника на работу обязательно будет меньше, чем дата его перевода на другую работу. За выполнением этих ограничений следит СУБД в процессе своего функционирования.

Также различают **статические** и **динамические** ограничения целостности. Статические ограничения присущи всем состояниям ПО, а динамические определяют возможность перехода ПО из одного состояния в другое. Примерами статических ограничений целостности могут служить требование уникальности номера паспорта или задание ограниченного множества значений поля "Пол" ('м' и 'ж'). В качестве примера динамического ограничения целостности можно привести правило, которое распространяется на значения любых счётчиков: значение счётчика не может уменьшаться.

В настоящее время разработано много различных моделей данных. Основные – это сетевая, иерархическая и реляционная модели.

## 2.2. Сетевая модель данных (СМД)

Сетевая модель позволяет организовывать БД, структура которых представляется графом общего вида (пример СМД – на рис. 2.4). Организация данных в сетевой модели соответствует структуризации данных по версии CODASYL. Каждая вершина графа хранит экземпляры сущностей (записи одного типа) и сведения о групповых отношениях с сущностями других типов. Каждая запись может хранить произвольное количество значений атрибутов (элементов данных и агрегатов), соответствующих экземпляру сущности.

Связи между записями в СМД выполняются в виде указателей, т.е. каждая запись хранит ссылки на другие однотипные записи и записи, связанные с ней групповыми отношениями. Таким образом, в каждой вершине записи хранятся в виде связного списка. Если список организован как однонаправленный, запись имеет ссылку на следующую однотипную запись в списке; если список двунаправленный – то на следующую и предыдущую однотипные записи.

Групповые отношения характеризуют следующие признаки:

1. Способ упорядочения подчинённых записей.

Поддерживаются три способа упорядочения:

- Очередь – добавление в конец списка (FIFO – first input, first output).
- Стек – добавление в начало списка (LIFO – last input, first output).
- Сортировка по значению ключа. При этом задаётся ключевое поле (поля), и вновь поступившая запись добавляется в упорядоченный список в соответствии со значением этого поля (значением ключа).

2. Режим включения подчинённых записей.

Режим включения бывает **автоматический** и **ручной.**

При автоматическом режиме подчиненная запись связана с записью-владельцем обязательной связью, поэтому она включается в групповое отношение и прикрепляется к записи-владельцу в момент внесения в БД. (Из этого следует, что запись-владелец должна быть внесена в базу данных ДО внесения первого экземпляра подчиненной записи.)

При ручном режиме включения подчиненная запись может находиться в БД и не быть прикрепленной к записи-владельцу. Она вручную включается в групповое отношение тогда, когда это отношение (связь) возникает.

3. Режим исключения подчинённых записей.

Режим исключения определяется **классом членства**. Различают три класса членства – **фиксированный, обязательный** и **необязательный:**

- Записи с обязательным членством должны быть удалены до удаления записи–владельца: владелец, к которому прикреплена хотя бы одна запись с обязательным членством, не может быть удален.
- Записи с фиксированным членством удаляются вместе с записью–владельцем.
- Записи с необязательным членством при удалении записи–владельца останутся в БД.

Рассмотрим фрагмент БД "Предприятие" (рис. 2.5). Здесь записи типов *ОТДЕЛЫ* и *ОРГАНИЗАЦИИ-ЗАКАЗЧИКИ* являются владельцами записей типа *ПРОЕКТЫ* и они связаны групповыми отношениями соответственно *выполняют* и *заказывают*. Записи типов *ОТДЕЛЫ* и *ПРОЕКТЫ* являются владельцами записей типа *СОТРУДНИКИ* и они связаны групповыми отношениями *работают* и *выполняются*. Записи типа *СОТРУДНИКИ* являются владельцами записей типа *ДЕТИ*.

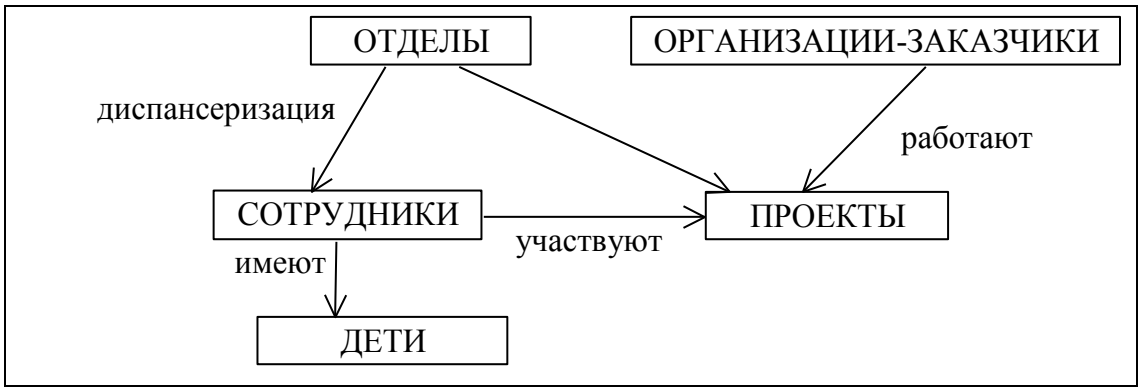


Рис.2.5. Пример фрагмента сетевой БД "Предприятие"

Групповые отношения чаще всего описывают связь "один-ко-многим": один владелец, много подчиненных. Например, отношение *работают* подразумевает, что каждый сотрудник работает в одном отделе, но в каждом отделе могут работать несколько сотрудников. С другой стороны, групповое отношение *выполняются* отражает связь "многие-ко-многим": каждый сотрудник может участвовать в выполнении нескольких проектов, и каждый проект могут выполнять несколько человек. Что касается классов членства подчиненных записей, то связь сотрудники-дети относится к фиксированному классу членства, связь сотрудники-проекты – к необязательному, а все остальные – к обязательному классу членства. Режим включения для связи сотрудники-проекты ручной, для всех остальных – автоматический.

В СМД применяются следующие операции над данными:

- *запомнить*: внесение информации в БД;
- *включить в групповое отношение:* установление связей между данными;
- *переключить*: переход члена набора к другому владельцу;
- *обновить*: модификация данных;
- *извлечь*: чтение данных;

- *удалить*: физическое или логическое удаление данных;
- *исключить из группового отношения*: разрыв связей между данными.

В сетевой модели данных предусмотрены специальные способы навигации и манипулирования данными. Аппарат навигации в графовых моделях служит для установления тех объектов данных, к которым будет применяться очередная операция манипулирования данными. Такие объекты называются *текущими*. В СМД возможны переходы:

- от текущего экземпляра записи определённого типа к другим экземплярам записи этого же типа;
- из текущей вершины в любую вершину, с которой текущая связана групповым отношением.

Наиболее распространенной и стандартизованной из реализаций СМД является модель CODASYL. В соответствии с ней описание схемы БД осуществляется на языке COBOL, а манипулирование данными – с помощью включающего языка программирования высокого уровня.

Сетевая модель данных является наиболее полной с точки зрения реализации различных типов связей и ограничений целостности. Но СМД является достаточно сложной для проектирования и поддержки, поэтому она не получила широкого распространения.

## 2.3. Иерархическая модель данных (ИМД)

Иерархическая модель позволяет строить БД с иерархической древовидной структурой. Структура ИМД описывается в терминах, аналогичных терминам сетевой модели данных. В основе ИМД лежит понятие дерева.

**Дерево** – это связный неориентированный граф, который не содержит циклов. При работе с деревом выделяют какую-то конкретную вершину, определяют её как корень дерева и рассматривают особо – в эту вершину не заходит ни одно ребро. В этом случае дерево становится ориентированным, ориентация определяется от корня. Дерево как ориентированный граф можно определить следующим образом:

- имеется единственная особая вершина, называемая корнем, в которую не заходит ни одно ребро;
- во все остальные вершины заходит только одно ребро, а исходит произвольное количество ребер;
- граф не содержит циклов.

Конечные вершины, то есть вершины, из которых не выходит ни одной дуги, называются *листьями* дерева. Количество вершин на пути от корня к листьям в разных ветвях дерева может быть различным.

В иерархических моделях данных используется ориентация древовидной структуры от корня к листьям. Графическая диаграмма концептуальной схемы базы данных называется *деревом определения*. Пример иерархической базы данных приведён на рис. 2.6. Каждая некорневая вершина в ИМД связана с родительской записью иерархическим групповым отношением. Каждая вершина дерева соответствует сущности ПО. Эта сущность характеризуется произволь-

ным количеством атрибутов, связанных с ней отношением 1:1. Атрибуты, связанные с сущностью отношением 1:n, образуют отдельную сущность и переносятся на следующий уровень иерархии.

Тип вершины определяется типом сущности и набором её атрибутов. Каждая вершина дерева хранит экземпляры сущностей – записи. Следствием внутренних ограничений иерархической модели является то, что каждому экземпляру зависимой группы в БД соответствует уникальное множество экземпляров родительских записей – по одному экземпляру (записи) каждого типа вершин вышестоящих уровней.

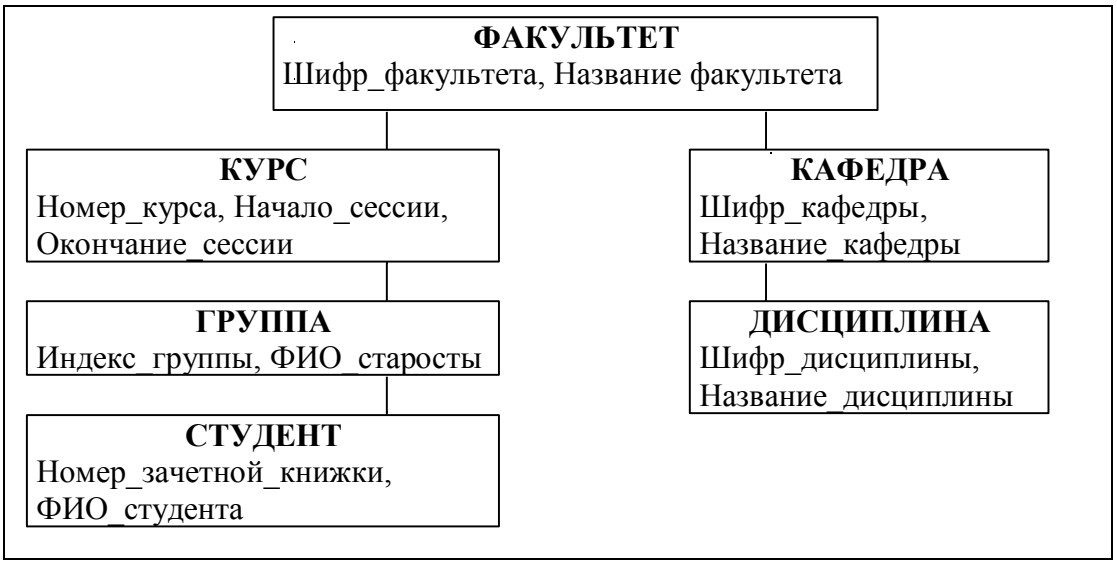


Рис.2.6. Пример иерархической базы данных

В ИМД также предусмотрены специальные способы навигации. Передвижение по дереву всегда начинается с корневой вершины, от которой можно прейти на конкретный экземпляр записи любой вершины следующего уровня. Эта вершина становится текущей вершиной, а экземпляр – текущей записью. От этой записи можно перейти к другой записи данной вершины, к экземпляру записи родительской вершины или к экземпляру записи подчинённой вершины.

Корневая запись дерева должна содержать ключ с уникальным значением. Ключи некорневых записей должны иметь уникальные значения только в экземплярах групповых отношений, т.е. на одном и том же уровне иерархии в разных ветвях дерева могут быть экземпляры записей с одинаковыми ключами. Это объясняется тем, что каждая запись идентифицируется **полным сцепленным ключом**, который образуется путём конкатенации всех ключей экземпляров родительских записей. Т.о., попасть в любую вершину можно, только проделав полный путь по дереву от корня. Связи между записями в ИМД обычно выполнены в виде ссылок (т.е. хранятся адреса связанных записей).

Основным недостатком ИМД является дублирование данных. Оно вызвано тем, что каждая сущность (атрибут) может относиться только одной к родительской сущности. Таким образом, если надо сохранить, например, данные о детях сотрудника, а на предприятии трудится и отец, и мать ребенка, то информацию о детях придётся хранить дважды. Это может вызвать нарушение логической целостности БД при внесении изменений в данные о детях.

Если данные имеют естественную древовидную структуризацию, то использование иерархической модели данных не вызывает проблем. Но на практике часто требуется реализовать структуры данных, отличные от иерархической. Для решения этих задач конкретные СУБД, основанные на ИМД, включают дополнительные средства, облегчающие представление произвольно организованных данных.

## 2.4. Реляционная модель данных (РМД)

### 2.4.1. Понятие отношения

Реляционная модель данных была предложена математиком Э.Ф. Коддом (Codd E.F.) в 1970 г. РМД является наиболее широко распространенной моделью данных и единственной из трех основных моделей данных, для которой разработан теоретический базис с использованием теории множеств.

В основе РМД лежит понятие отношения, основанного на декартовом произведении доменов. **Домен** – это множество значений, которое может принимать элемент (например, множество целых чисел, множество комбинаций символов длиной N и т.п.).

Пусть $D_1, D_2, \ldots, D_k$ – произвольные конечные и не обязательно различные множества (домены). **Декартово произведение** этих множеств определяется следующим образом:

$$D_1 \times D_2 \times \ldots \times D_k = \{(d_1, d_2, \ldots, d_k) \mid d_i \in D_i, i=1,\ldots,k\}.$$

Таким образом, декартово произведение позволяет получить все возможные комбинации элементов исходных множеств.

**Пример**. Для доменов $D_1 = (1,2)$, $D_2 = (A,B,C)$ декартово произведение $D = D_1 \times D_2$ будет таким: $D = \{(1,A), (1,B), (1,C), (2,A), (2,B), (2,C)\}$.

Подмножество декартова произведения доменов называется **отношением.**

Элементы отношения называют *кортежами*. Элементы кортежа принято называть *атрибутами*.

Отношение содержит информацию о сущностях одного типа. Каждый кортеж отношения соответствует одному экземпляру сущности.

### 2.4.2. Свойства отношений

Отношение обладает двумя основными свойствами:

1. В отношении не должно быть одинаковых кортежей, т.к. это множество.
2. Порядок кортежей в отношении несущественен.

Отношение удобно представлять как таблицу, где строка является кортежем, столбец соответствует домену (рис. 2.7, отношение *СТУДЕНТЫ*).

Отношение имеет имя, которое отличает его от имён всех других отношений. Атрибутам реляционного отношения назначаются имена, уникальные в рамках отношения. Обращение к отношению происходит по его имени, а обращение к атрибуту – по имени отношения и имени атрибута.

| домен 1 | домен 2 | домен 3 (ключ) | домен 4 | домен 5 |
|---|---|---|---|---|

| *Группа* | *ФИО студента* | ***Номер зачётной книжки*** | *Год рождения* | *Размер стипендии* |
|---|---|---|---|---|
| С–72 | Волкова Елена Павловна | С-12298 | 1981 | 566.40 |
| С–91 | Белов Сергей Юрьевич | С-12299 | 1980 | 400.00 |
| | | . . . | | |
| С–72 | Фролов Юрий Вадимович | С-14407 | 1981 | 0 |

Рис.2.7. Пример табличной формы представления отношения

Каждый атрибут определён на некотором домене, несколько атрибутов отношения могут быть определены на одном и том же домене (например, номера рабочего и домашнего телефона). Домен задаётся типом данных и ограничениями целостности, например, оклад – это число больше нуля.

Атрибут может быть обязательным и необязательным. Значение обязательного атрибута должно быть определено в момент внесения записи в БД. Если атрибут необязательный, то для таких случаев предусмотрено специальное значение – null, которое можно интерпретировать как "неизвестное значение". Значение null не привязано к определенному типу данных, т.е. может назначаться данным любых типов.

Перечень атрибутов отношения с их типами данных и размерами определяют **схему отношения**. Отношения, построенные по одинаковой схеме, называют **односхемными**; по различным схемам – **разносхемными.**

**Ключ отношения** – это атрибут, значения которого идентифицируют кортеж. Таким образом, ключ имеет уникальные в рамках отношения значения. (На рис. 2.7 ключ выделен полужирным шрифтом). Если ключ состоит из нескольких атрибутов, он называется составным. Ключей может быть несколько; основной ключ – первичный, его значения не могут обновляться. Другие ключи называются *возможными* или *потенциальными* ключами.

РМД не поддерживает групповые отношения. Для связей между отношениями используются внешние ключи. Внешний ключ – это атрибут подчиненного отношения, который является копией первичного (или уникального) ключа родительского отношения. (Пример – отношение *ДЕТИ*, связанное с отношением *СТУДЕНТЫ* внешним ключом *Номер зачётной книжки*, рис. 2.8).

внешний ключ

| ***Номер зачётной книжки*** | ***Имя, отчество ребенка*** | *Дата рождения* |
|---|---|---|
| С-12298 | Антон Павлович | 01.12.01 |
| С-12298 | Юлия Павловна | 01.12.01 |
| С-12299 | Ольга Сергеевна | 16.04.02 |

Рис.2.8. Отношение "Дети", связанное с отношением "Студенты" по внешнему ключу

Если связь необязательная, то значение внешнего ключа может быть неопределённым (null).

Фактически, внешние ключи логически связывают экземпляры сущностей разных типов (связывают родительскую и дочерние сущности).

**Внешний ключ** является ограничением целостности, в соответствии с которым множество значений внешнего ключа является подмножеством значений первичного или уникального ключа родительской таблицы.

Ограничение целостности по внешнему ключу проверяется в двух случаях:

- при добавлении записи в подчинённую таблицу СУБД проверяет, что в родительской таблице есть запись с таким же значением первичного ключа;
- при удалении записи из родительской таблицы СУБД проверяет, что в подчинённой таблице нет записей с таким же значением внешнего ключа.

**Примечание:** внешний ключ может быть определен на той же таблице, что и первичный. Это позволяет описывать иерархию однотипных сущностей. Например, если в таблицу *СОТРУДНИКИ* добавить поле *Руководитель*, и описать его как внешний ключ на эту же таблицу, то в этом поле будет храниться идентификатор руководителя данного сотрудника (рис. 2.9). Поле *Руководитель*, естественно, является необязательным.

| ***Табельный номер*** | *Номер отдела* | *ФИО* | *Должность* | *Руководитель* |
|---|---|---|---|---|
| 002 | 1 | Сухов К.А. | директор | |
| 034 | 1 | Петрова К.В. | секретарь | 002 |
| 988 | 2 | Рюмин В.П. | начальник отдела | 002 |
| 909 | 2 | Серова Т.В. | вед. программист | 988 |
| 100 | 2 | Волков Л.Д. | программист | 988 |
| 110 | 3 | Буров Г.О. | гл. бухгалтер | 002 |

Рис.2.9. Внешний ключ "Руководитель", определенный на первичном ключе этой же таблицы *СОТРУДНИКИ*

Все операции над данными в РМД выполняются над отношением и требуют задания имени отношения. Если операция применяется к части отношения, то может потребоваться идентификация кортежа или группы кортежей и задание имен атрибутов. В РМД используются следующие операции:

- *запомнить*: внесение информации в БД (требует формирования значений уникального ключа и обязательных атрибутов кортежа);
- *обновить*: модификация данных – изменение значений отдельных атрибутов кортежей;
- *извлечь*: чтение данных;
- *удалить*: физическое или логическое удаление данных (кортежей).

Структуризация данных в РМД существенно отличается от структуризации данных по версии CODASYL. В таблице 2.1. приведено соответствие этих двух вариантов структуризации.

Таблица 2.1. Сравнение структуризации данных в РМД и по версии CODASYL

| Термины версии CODASYL | Термины (и синонимы) РМД |
|---|---|
| Элемент данных | Атрибут (поле) |
| Агрегат | — |
| Запись (группа) | Кортеж (запись, строка) |
| Совокупность записей одного типа | Отношение (таблица) |
| Набор (групповое отношение) | — |
| База данных | База данных |

### 2.4.3. Достоинства и недостатки РМД

Широкое распространение реляционной модели объясняется в первую очередь простотой представления и формирования базы данных, универсальностью и удобством обработки данных, которая осуществляется с помощью декларативного языка запросов SQL (Structured Query Language). Подробнее об SQL рассказано в [3].

Моделирование предметной области в рамках реляционной модели создаёт некоторые сложности, т.к. в этой модели нет специальных средств для отображения различных типов связей. Отсутствие множественных связей (n:m) вызывает увеличение объёма хранимой (дублируемой) информации. Отсутствие агрегатов приводит к тому, что при проектировании РБД приходится проводить нормализацию отношений. После нормализации данные об одной сущности ПО распределяются по нескольким таблицам, что усложняет работу с БД. (Подробнее об этом рассказано в [2]).

Отсутствие специальных механизмов навигации (как в иерархической или сетевой моделях), с одной стороны, ведёт к упрощению модели, а с другой – к многократному увеличению времени на извлечение данных, т.к. во многих случаях требуется просмотреть всё отношение для поиска нужных данных.

В РМД нет понятий режим включения и класс членства, поэтому набор средств описания ограничений целостности очень мал. (Подробнее об этом рассказано в разделе 8.1."Обеспечение целостности данных").

Обработка данных в реляционных базах данных (РБД) осуществляется с помощью операций реляционной алгебры (РА).

### 2.4.4. Операции реляционной алгебры

Операндами для операций реляционной алгебры являются реляционные отношения. Результатом выполнения операций РА также является отношение. Использование реляционной алгебры накладывает на отношения два ограничения: порядок столбцов (полей) в отношении фиксирован; отношения конечны.

Существует пять основных операций реляционной алгебры – проекция, селекция, декартово произведение, разность, объединение – и три вспомогательных: соединение, пересечение и деление. Вспомогательные операции могут быть выражены через основные, но в некоторых системах реализуются отдельно для удобства пользователей.

1. **Проекция** (projection).

Это унарная операция (выполняемая над одним отношением), служащая для выбора подмножества атрибутов из отношения R. Она уменьшает арность отношения и может уменьшить мощность отношения за счёт исключения одинаковых кортежей.

Пример 1. Пусть имеется отношение R(A,B,C) (рис. 2.10,а).

Тогда проекция $\pi_{A,C}(R)$ будет такой, как показано на рис. 2.10,б.

**Отношение R**

| A | B | C |
|---|---|---|
| a | b | c |

**Проекция $\pi_{A,C}$(R)**

| A | C |
|---|---|
| a | c |

| | | |
|---|---|---|
| c | a | d |
| c | b | d |

а)

| |
|---|
| c | d |

б)

Рис.2.10. Пример проекции отношения

2. **Селекция** (selection).

Это унарная операция, результатом которой является подмножество кортежей исходного отношения, соответствующих условиям, которые накладываются на значения определённых атрибутов.

Пример 2. Для отношения R(A,B,C) (рис. 2.11,а) селекция $\sigma_{C=d}(R)$ (при условии "значение атрибута C равно d") будет такой (рис. 2.11,б):

**Отношение R**

| A | B | C |
|---|---|---|
| a | | c |
| c | a | d |
| c | b | d |

а)

**Селекция $\sigma_{C=d}(R)$**

| A | B | C |
|---|---|---|
| c | a | d |
| c | b | d |

б)

Рис.2.11. Пример селекции отношения

3. **Декартово произведение** (Cartesian product).

Это бинарная операция над разносхемными отношениями, соответствующая определению декартова произведения для РМД.

Пример 3. Пусть имеются отношение R(A,B) и отношение S(C,D,E) (рис. 2.12,а). Тогда декартово произведение R×S будет таким (рис. 2.12,б).

**Отношение R**

| A | B |
|---|---|
| a | b |
| c | a |
| b | d |

**Отношение S**

| C | D | E |
|---|---|---|
| g | h | a |
| a | b | c |

а)

**Декартово произведение R×S**

| A | B | C | D | E |
|---|---|---|---|---|
| a | b | g | h | a |
| a | b | a | b | c |
| c | a | g | h | a |
| c | a | a | b | c |
| b | d | g | h | a |
| b | d | a | b | c |

б)

Рис.2.12. Пример декартова произведения отношений

4. **Объединение** (union).

Объединением двух односхемных отношений R и S называется отношение T = R ∪ S, которое включает в себя все кортежи обоих отношений без повторов.

5. **Разность** (set difference).

Разностью односхемных отношений R и S называется множество кортежей R, не входящих в S.

Пример 4. Пусть имеются отношение R(A,B,C) и отношение S(A,B,C) (рис. 2.13,а). Тогда разность R–S будет такой (рис. 2.13,б):

**Отношение R**

| A | B | C |
|---|---|---|
| a | b | c |

**Отношение S**

| A | B | C |
|---|---|---|
| g | h | a |

**Разность R–S**

| A | B | C |
|---|---|---|
| c | a | d |

| | | |
|---|---|---|
| c | a | d |
| c | h | c |

| | | |
|---|---|---|
| a | b | c |
| h | d | d |

а)

| | | |
|---|---|---|
| c | h | c |

б)

Рис.2.13. Пример разности отношений

Следующие три операции являются вспомогательными операциями реляционной алгебры.

6. **Пересечение** (intersection).

Пересечение двух односхемных отношений R и S есть подмножество кортежей, принадлежащих обоим отношениям. Это можно выразить через разность:

$R \cap S = R - (R - S)$.

7. **Соединение** (join).

Эта операция определяет подмножество декартова произведения двух разносхемных отношений. Кортеж декартова произведения входит в результирующее отношение, если для атрибутов разных исходных отношений выполняется некоторое условие F. Соединение может быть выражено так:

$$R \underset{F}{\triangleright\!\triangleleft} S = \sigma_F (R \times S)$$

Если условием является равенство атрибутов исходных отношений, такая операция называется **эквисоединением.** *Естественным* называется эквисоединение по одинаковым атрибутам исходных отношений.

Пример 5. Пусть имеются отношения R(A,B,C) и S(A,D,E) (рис. 2.14,а). Тогда естественное соединение $R \triangleright\!\triangleleft S$ будет таким, как показано на рис. 2.14,б.

**Отношение R**

| A | B | C |
|---|---|---|
| a | b | c |
| c | a | d |
| c | h | c |
| g | b | d |

**Отношение S**

| A | D | E |
|---|---|---|
| g | h | a |
| c | b | c |
| h | d | d |

а)

**Соединение $R \triangleright\!\triangleleft S$**

| A | B | C | D | E |
|---|---|---|---|---|
| c | a | d | b | C |
| c | h | c | b | C |
| g | b | d | h | A |

б)

Рис.2.14. Пример естественного соединения отношений

8. **Деление** (division).

Пусть отношение R содержит атрибуты $\{r_1,r_2,...,r_k, r_{k+1},...,r_n\}$, а отношение S – атрибуты $\{r_{k+1},...,r_n\}$. Тогда результирующее отношение содержит атрибуты $\{r_1,r_2,...,r_k\}$. Кортеж отношения R включается в результирующее отношение, если его декартово произведение с отношением S входит в R.

Пример 6. Пусть имеются отношения R(A,B,C) и S(A,B) (рис. 2.15,а). Тогда частное R/S будет таким как показано на рис. 2.15,б.

**Отношение R**

| A | B | C | D |
|---|---|---|---|
| a | b | c | b |
| c | f | g | h |
| a | v | c | b |
| a | b | g | h |

**Отношение S**

| C | D |
|---|---|
| g | h |
| c | b |

**Частное R/S**

| A | B |
|---|---|
| a | b |
| c | f |

| | | | |
|---|---|---|---|
| c | v | g | h |
| c | f | c | b |

а) б)

Рис.2.15. Пример операции деления

**Примечание**: в РБД операции реляционной алгебры реализованы в языке SQL [3].

## 2.5. Другие модели данных

Всё возрастающая сложность приложений баз данных и ограниченность реляционной модели привели к развитию модели Кодда, которое сначала получило название *расширенной реляционной модели*, а позже получило свое развитие в объектно-реляционной модели данных (ОРМД) [4].

### 2.5.1. Объектно-реляционная модель данных

Объектно-реляционная модель данных (ОРМД) реализована с помощью реляционных таблиц, но включает объекты, аналогичного понятию объекта в объектно-ориентированном программировании. В ОРМД используются такие объектно-ориентированные компоненты, как пользовательские типы данных, инкапсуляция, полиморфизм, наследование, переопределение методов и т.п.

К сожалению, до настоящего времени разработчики не пришли к единому мнению о том, что должна обеспечивать ОРМД. В 1999 г. был принят стандарт SQL-3, который определяет основные характеристики ОРМД, но до сих пор модели, поддерживаемые различными производителями СУБД, существенно отличаются по своим функциональным возможностям. О перспективах этого направления свидетельствует тот факт, что ведущие фирмы–производители СУБД, в числе которых Oracle, Informix, INGRES и др., расширили возможности своих продуктов до объектно-реляционной СУБД (ОРСУБД).

В большинстве реализаций ОРМД объектами признаются агрегат и таблица (отношение), которая может входить в состав другой таблицы. Методы обработки данных представлены в виде хранимых процедур и триггеров, которые являются процедурными объектами базы данных, и связаны с таблицами. На концептуальном уровне все данные ОРБД представлены в виде отношений, и ОРСУБД поддерживают язык SQL.

### 2.5.2. Объектно-ориентированная модель данных

Ещё один подход к построению БД – использование объектно-ориентированной модели данных (ООМД) [5]. Моделирование данных в ООМД базируется на понятии объекта. ООМД обычно применяется в сложных предметных областях, для моделирования которых не хватает функциональности реляционной модели (например, для систем автоматизации проектирования (САПР), издательских систем и т.п.).

При создании объектно-ориентированных СУБД (ООСУБД) используются разные методы, а именно:

- встраивание в объектно-ориентированный язык средств, предназначенных для работы с БД;

- расширение существующего языка работы с базами данных объектно-ориентированными функциями;
- создание объектно-ориентированных библиотек функций для работы с БД;
- создание нового языка и новой объектно-ориентированной модели данных.

К достоинствам ООМД можно отнести широкие возможности моделирования предметной области, выразительный язык запросов и высокую производительность. Каждый объект в ООМД имеет уникальный идентификатор (OID – object identifier). Обращение по OID происходит существенно быстрее, чем поиск в реляционной таблице напрямую или через индекс.

Среди недостатков ООМД следует отметить отсутствие общепринятой модели, недостаток опыта создания и эксплуатации ООБД, сложность использования и недостаточность средств защиты данных.

В 1997 г. рабочая группа ODMG (Object Database Management Group), образованная фирмами-производителями ООСУБД, выпустила стандарт ODMG 2.0 для ООСУБД, в котором описана объектная модель, язык определения запросов, язык объектных запросов и связующие языки C++, Smalltalk и Java.

## 3. СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Система управления базами данных (СУБД) – это важнейший компонент АИС, основанной на базе данных. СУБД необходима для создания и поддержки базы данных информационной системы в той же степени, как для разработки программы на алгоритмическом языке – транслятор. Программные составляющие СУБД включают в себя ядро и сервисные средства (утилиты).

**Ядро СУБД** – это набор программных модулей, необходимый и достаточный для создания и поддержания БД, то есть универсальная часть, решающая стандартные задачи по информационному обслуживанию пользователей. **Сервисные программы** предоставляют пользователям ряд дополнительных возможностей и услуг, зависящих от описываемой предметной области и потребностей конкретного пользователя.

**Системой управления базами данных** называют программную систему, предназначенную для создания на ЭВМ общей базы данных для множества приложений, поддержания её в актуальном состоянии и обеспечения эффективного доступа пользователей к содержащимся в ней данным в рамках предоставленных им полномочий.

Принципиально важное свойство СУБД заключается в том, что она позволяет различать и поддерживать два независимых взгляда на БД: "взгляд" пользователя, воплощаемый в "логическом" представлении данных, и "взгляд" системы – "физическое" представление (организация хранимых данных).

Для инициализации базы данных разработчик средствами конкретной СУБД описывает логическую структуру БД, её организацию в среде хранения и пользовательские представления данных (соответственно концептуальную схему БД, схему хранения и внешние схемы). Обрабатывая эти схемы, СУБД со-

здаёт пустую БД требуемой структуры и предоставляет средства для наполнения её данными предметной области и дальнейшей эксплуатации.

## 3.1. Классификация СУБД

По степени универсальности СУБД делят на два класса: СУБД **общего назначения** (СУБД ОН) и **специализированные** СУБД (СпСУБД).

СУБД ОН не ориентированы на какую-либо предметную область или на конкретные информационные потребности пользователей. Каждая система такого рода является универсальной и реализует функционально избыточное множество операция над данными. СУБД ОН имеют в своем составе средства настройки на конкретную предметную область, условия эксплуатации и требования пользователей. Производство этих систем поставлено на широкую коммерческую основу.

Специализированные СУБД создаются в тех случаях, когда ни одна из существующих СУБД общего назначения не может удовлетворительно решить задачи, стоящие перед разработчиками, например, не достигается требуемое быстродействие обработки или не обеспечивается поддержка необходимого объёма данных. СпСУБД предназначены для решения конкретной задачи, а приемлемые параметры этого решения достигаются следующим образом:

1) за счёт знания особенностей конкретной предметной области,
2) путём сокращения функциональной полноты системы.

Создание СпСУБД – дело весьма трудоёмкое, поэтому для того, чтобы выбрать этот путь, надо иметь действительно веские основания. В дальнейшем будут рассматриваться только СУБД общего назначения.

По модели данных различают **иерархические, сетевые, реляционные** и **объектно-ориентированные** СУБД.

По методам организации хранения и обработки данных СУБД делят на **централизованные** и **распределённые**. Первые работают с БД, которая физически хранится в одном месте (на одном компьютере). Это не означает, что пользователь может работать с БД только за этим же компьютером: доступ может быть удалённым (в режиме клиент–сервер). Большинство централизованных СУБД перекладывает задачу организации удалённого доступа к данным на сетевое обеспечение, выполняя только свои стандартные функции, которые усложняются за счёт одновременности доступа многих пользователей к данным.

## 3.2. Правила Кодда для реляционной СУБД

Э.Ф. Кодд преложил и обосновал 12 правил, которым должна удовлетворять реляционная система управления базами данных (РСУБД):

1. Явное представление данных (The Information Rule). Информация должна быть представлена в виде данных, хранящихся в ячейках. Данные, хранящиеся в ячейках, должны быть атомарны. Порядок строк в реляционной таблице не должен влиять на смысл данных.

2. Гарантированный доступ к данным (Guaranteed Access Rule). К каждому элементу данных должен быть гарантирован доступ с помощью комбинации имени таблицы, первичного ключа строки и имени столбца.
3. Обработка неизвестных значений (Systematic Treatment of Null Values). Неизвестные значения NULL, отличные от любого известного значения, должны поддерживаться для всех типов данных при выполнении любых операций. Например, для числовых данных неизвестные значения не должны рассматриваться как нули, а для символьных данных – как пустые строки.
4. Доступ к словарю данных в терминах реляционной модели (Dynamic On-Line Catalog Based on the Relational Model). Словарь данных должен сохраняться в форме реляционных таблиц, и РСУБД должна поддерживать доступ к нему при помощи стандартных языковых средств, тех же самых, которые используются для работы с реляционными таблицами, содержащими пользовательские данные.
5. Полнота подмножества языка (Comprehensive Data Sublanguage Rule). РСУБД должна поддерживать единственный язык, который позволяет выполнять все операции над данными: определение данных, манипулирование данными, управление доступом к данным, управление транзакциями.
6. Поддержка обновляемых представлений (View Updating Rule). Обновляемое представление должно поддерживать все операции манипулирования данными, которые поддерживают реляционные таблицы: операции выборки, вставки, модификации и удаления данных.
7. Наличие высокоуровневых операций управления данными (High-Level Insert, Update, and Delete). Операции вставки, модификации и удаления данных должны поддерживаться не только по отношению к одной строке таблицы, но по отношению к любому множеству строк.
8. Физическая независимость данных (Physical Data Independence). Приложения не должны зависеть от используемых способов хранения данных на носителях, от аппаратного обеспечения компьютеров, на которых находится БД. РСУБД должна предоставлять некоторую свободу модификации способов организации базы данных в среде хранения, не вызывая необходимости внесения изменений в логическое представление данных. Это позволяет оптимизировать среду хранения данных с целью повышения эффективности системы, не затрагивая созданных прикладных программ, работающих с БД.
9. Логическая независимость данных (Logical Data Independence). Это свойство позволяет сконструировать несколько различных логических взглядов (представлений) на одни и те же данные для разных групп пользователей. При этом пользовательское представление данных может сильно отличаться не только от физической структуры их хранения, но и от концептуальной (логической) схемы данных. Оно может синтезироваться динамически на основе хранимых объектов БД в процессе обработки запросов.
10. Независимость контроля целостности (Integrity Independence). Вся информация, необходимая для поддержания целостности, должна находиться в словаре данных. Язык для работы с данными должен выполнять проверку входных данных и автоматически поддерживать целостность данных. Это реали-

зуется с помощью ограничений целостности и механизма транзакций (см. раздел 6.2).

11. Независимость от распределённости (Distribution Independence). База данных может быть распределённой (может находиться на нескольких компьютерах), и это не должно оказывать влияние на приложения. Перенос базы данных на другой компьютер не должен оказывать влияния на приложения.
12. Согласование языковых уровней (Non-Subversion Rule). Не должно быть иного средства доступа к данным, отличного от стандартного языка для работы с данными. Если используется низкоуровневый язык доступа к данным, он не должен игнорировать правила безопасности и целостности, которые поддерживаются языком более высокого уровня.

## 3.3. Основные функции реляционной СУБД

Основные функции реляционной СУБД определяются правилами Кодда. Но потребности пользователей обуславливают также следующие функции:

1. **Поддержка многопользовательского режима доступа.**
   База данных создаётся для решения многих задач многими пользователями. Это подразумевает возможность одновременного доступа многих пользователей к данным. Данные в БД являются разделяемым ресурсом, и РСУБД должна обеспечивать разграничение доступа к ним.
2. **Обеспечение физической целостности данных.**
   Проблема обеспечения физической целостности данных обусловлена возможностью разрушения данных в результате сбоев и отказов в работе вычислительной системы или в результате ошибок пользователей. Развитые РСУБД позволяют в большинстве случаев восстановить потерянные данные. Восстановление данных чаще всего основано на периодическом создании резервных копий БД и ведении журнала регистрации изменений (журнала транзакций) (см. разделы 6,7).
3. **Управление доступом.**
   Для многопользовательских систем актуальна проблема защиты данных от несанкционированного доступа. Каждый пользователь этой системы в соответствии со своим уровнем (приоритетом) имеет доступ либо ко всей совокупности данных, либо только к её части. Управление доступом также подразумевает предоставление прав на проведение отдельных операций над отношениями или другими объектами БД.
4. **Настройка РСУБД.**
   Настройка РСУБД обычно выполняется администратором БД, отвечающим за функционирование системы в целом. В частности, она может включать в себя следующие операции:
   - подключение внешних приложений к БД;
   - модификация параметров организации среды хранения данных с целью повышения эффективности системы;

- изменение структуры хранимых данных или их размещения в среде хранения (реорганизацию БД) для повышения производительности системы или повторного использования освободившейся памяти;
- модификацию концептуальной схемы данных (реструктуризацию БД) при изменении предметной области и/или потребностей пользователей.

## 3.4. Администрирование базы данных

Основные задачи администрирования базы данных – обеспечение надежного и эффективного функционирования системы БД, адекватности содержания БД информационным потребностям пользователей, отображения в БД актуального состояния ПО.

Администрирование БД возлагается на администратора (или персонал администрирования, если система БД велика). В задачи администратора входит выполнение нескольких групп функций:

1. Администрирование предметной области: поддержка представления БД на концептуальном уровне архитектуры СУБД (общем для всех приложений); адекватное отображение в БД изменений, происходящих в ПО. Последнее требование может подразумевать реструктуризацию (изменение схемы) БД и последующее приведение содержимого БД в соответствие с новой схемой.
2. Администрирование БД: поддержка представления БД в среде хранения, эффективная и надежная эксплуатация системы БД. Если на этом уровне проводится реорганизация БД (с целью повышения эффективности работы), то она заключается в следующем:
   - изменения в структуре хранимых данных, например, выведение в отдельную таблицу редко используемых данных;
   - изменения способов размещения данных в памяти, например:
     – разбиение таблицы на части для распределения её по различным физическим носителям с целью распараллеливания доступа к ней;
     – построение кластеров;
     – изменение физических параметров среды хранения, например, размера блока данных.
   - изменения используемых методов доступа к данным, например, построение индексов или введение хеширования.
3. Администрирование приложений: поддержка представлений БД для различных групп пользователей механизмами внешнего уровня СУБД. При изменении концептуальной схемы БД или схемы хранения может потребоваться внесение соответствующих изменений в приложения.
4. Администрирование безопасности данных: предоставление пользователям прав на доступ к БД и настройка системных средств защиты от несанкционированного доступа.

В состав СУБД обычно включаются вспомогательные средства (различные утилиты), упрощающие администрирование БД.

## 3.5. Словарь-справочник данных

**Словарь-справочник данных** (ССД) – это программная система, предназначенная для централизованного хранения и использования описания объектов БД (метаданных). Эта система содержит сведения:

- о владельцах объектов данных, пользователях ресурсов данных и полномочиях их доступа;
- о составе и структуре базы данных;
- об ограничениях целостности;
- о вспомогательных объектах и компонентах ИС.

ССД обеспечивает непротиворечивость метаданных, единую точку зрения на базу данных всего персонала разработчиков, администраторов и пользователей системы. Метаданные в словаре–справочнике реляционной СУБД обычно организованы в виде набора таблиц.

**Словарь БД** содержит сведения об организации БД, её составе и структуре, о семантике данных, способах их идентификации, источниках данных и т.п. Словарь предназначен главным образом для документирования разработки БД и справочного обслуживания её пользователей. Информация в словаре представлена в виде, удобном для восприятия человеком.

**Справочник БД** служит для поддержки функционирования компонентов программного обеспечения – СУБД и прикладных программ, работающих с БД. Справочник содержит описания данных: форматы представления, структуру, методы доступа, способы размещения данных в памяти и т.п. Информация в словаре представлена в виде, удобном для программного использования.

Множества метаданных словаря и справочника в значительной мере пересекаются. Более того, они могут реализовываться совместно: во многих РСУБД справочник состоит из таблиц, содержащих описание объектов БД, а словарь реализуется с помощью представлений над таблицами справочника.

# 4. ФИЗИЧЕСКАЯ ОРГАНИЗАЦИЯ ДАННЫХ

## 4.1. Механизмы среды хранения и архитектура СУБД

Механизмы среды хранения БД служат для управления двумя группами ресурсов – ресурсами **хранимых данных** и ресурсами **пространства памяти.** В задачу этого механизма входит отображение структуры хранимых данных в пространство памяти, позволяющее эффективно использовать память и определить место размещения данных при запоминании и при поиске данных.

Механизмы среды хранения выполняют следующие операции:

1. При запоминании нового объекта:
   - определение места размещения нового объекта "физической" БД в пространстве памяти;
   - выделение необходимого ресурса памяти;
   - запоминание этого объекта (запись в память);
   - формирование связей с другими объектами.
2. При поиске объекта:

- поиск места размещения объекта в пространстве памяти по заданным атрибутам или "адресу";
- выборка объектов для обработки в оперативную память.

3. При изменении данных объекта:
   - поиск объекта и считывание его в оперативную память;
   - изменение значений атрибута (атрибутов) объекта;
   - запись объекта на прежнее место (или на новое, если объект увеличился в объеме и на прежнем месте недостаточно памяти для его размещения).
4. При удалении объекта:
   - удаление объекта с освобождением памяти (*физическое удаление*) или без освобождения (*логическое удаление*);
   - разрушение связей с другими объектами.

**Примечание:** в реляционных базах данных формирование связей осуществляется на логическом уровне (т.е. по значениям атрибутов), а в иерархических и сетевых БД – на физическом уровне (по адресам записей).

Все операции выполняются по запросам механизмов концептуального уровня СУБД. На этом уровне никаких операций непосредственного обновления пользовательских данных или преобразований представления хранимых данных не происходит, это задача более высоких архитектурных уровней. Управление памятью выполняется операционной системой по запросам СУБД или непосредственно самой СУБД.

Несмотря на декларируемую независимость архитектурных уровней, для достижения более высокой производительности на уровне организации среды хранения часто приходится учитывать специфику концептуальной модели. Аналогично, организация файловой системы не может не оказывать влияния на среду хранения.

## 4.2. Пространство памяти и размещение хранимых данных

Ресурсам пространства памяти соответствуют объекты внешней памяти ЭВМ, управляемые средствами операционной системы или СУБД.

Для обеспечения естественной структуризации хранимых данных, более эффективного управления ресурсами и/или для технологического удобства всё пространство памяти БД обычно разделяется на части (области, разделы и др.). (Во многих системах область соответствует файлу.) *Области памяти* используются для размещения хранимых записей одного или нескольких типов и разбиваются на пронумерованные *страницы* фиксированного размера. В большинстве систем обработку данных на уровне страниц ведёт операционная система (ОС), а обработку записей внутри страницы обеспечивает только СУБД.

Страницы представляются в среде ОС блоками внешней памяти или секторами, доступ к которым осуществляется за одно обращение [6]. В системах, которые позволяют управлять размером страницы (блока), приходится искать компромисс между производительностью системы и требуемым объёмом оперативной памяти.

Страница имеет **заголовок** со служебной информацией, вслед за которым располагаются собственно данные. В большинстве случаев в качестве единицы хранения данных принимается хранимая запись. На странице размещается, как правило, несколько записей, и есть свободный участок для размещения новых записей. Если запись не помещается на одной странице, она разбивается на фрагменты, которые хранятся на разных страницах и ссылаются друг на друга.

Существуют различные механизмы, позволяющие решать проблемы, которые возникают при модификации данных в БД. Рассмотрим их.

Удаление записей может быть *логическим* или *физическим*. В первом случае запись помечается как удаленная, но фактически она остаётся на прежнем месте. Фактическое удаление этой записи будет произведено либо при реорганизации БД, либо специальной сервисной программой, которая запускается администратором БД. При физическом удалении записи ранее занятый участок освобождается и становится доступным для повторного использования. Система автоматически управляет свободным пространством памяти на страницах. Как правило, это обеспечивается одним из двух способов: ведением списков свободных участков или динамической реорганизацией страниц.

При **динамической реорганизации страниц** записи БД плотно размещаются вслед за заголовком страницы, а после них расположен свободный участок (рис. 4.1,а). Смещение начала свободного участка хранится в заголовке страницы. При удалении записи оставшиеся записи переписываются подряд в начало страницы и изменяется смещение начала свободного участка.

Если система ведёт **список свободных участков**, то возможны разные варианты. Ссылка на первый свободный участок на странице может храниться в заголовке страницы, и каждый свободный участок хранит ссылку на следующий (или признак конца списка) (рис. 4.1,б). Каждый освобождаемый участок включается в список свободных участков на странице.

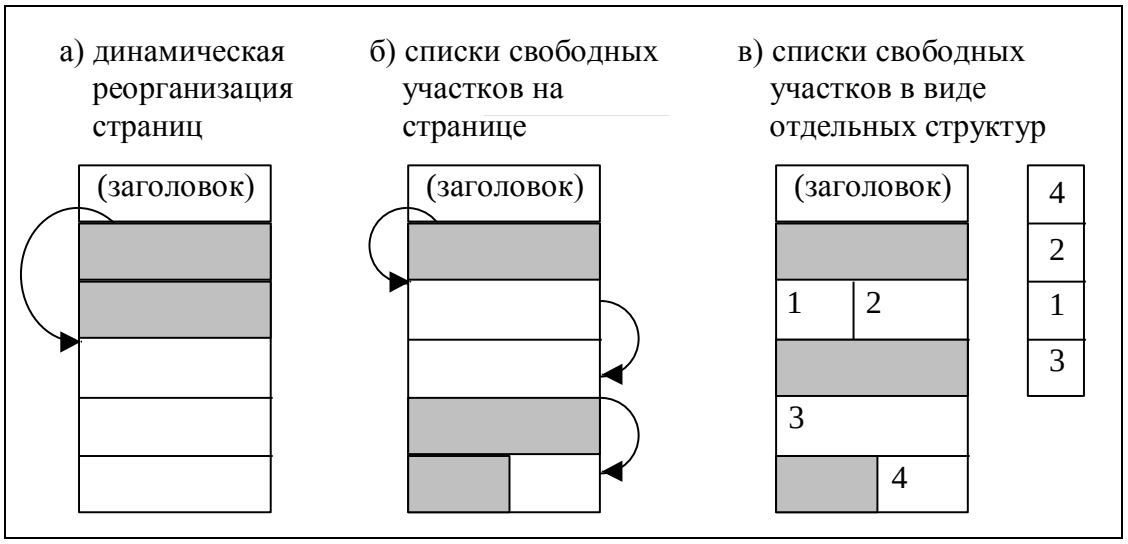


Рис. 4.1. Управление свободным простанством памяти на страницах

Другой способ ведения списков заключается в том, что списки свободных участков реализуются в виде отдельных структур (рис. 4.1,в). Эти структуры хранятся на отдельных **инвентарных страницах**. Каждая инвентарная страница относится к области (или группе страниц памяти) и содержит информацию о

свободных участках в этой области. Список ведётся как стек, очередь или упорядоченный список. В последнем случае упорядочение осуществляется по размеру свободного участка, что позволяет при размещении новой записи выбирать для неё наиболее подходящий по размеру участок.

При запоминании новой записи система через инвентарные страницы ищет свободный участок, достаточный для размещения этой записи. (Обычно выбирается первый подходящий участок, размер которого не меньше требуемого.) Если выбранный участок больше, чем запись, то остаток оформляется в виде свободного участка. (При динамической реорганизации страниц запись просто размещается вслед за последней записью на данной странице.) После этого система корректирует содержимое инвентарных страниц (если они есть).

При изменении записи, имеющей фиксированный формат, она просто перезаписывается на прежнее место. Если же запись имеет плавающий формат, возможны ситуации, когда запись не помещается на прежнее место. Тогда процедуры корректировки приводят либо к реорганизации страницы, либо к перемещению записи на другой участок памяти, что, в свою очередь, приведёт к обновлению нескольких страниц.

Использование списков свободных участков ведёт к *фрагментации* пространства памяти, т.е. появлению разрозненных незаполненных участков памяти. Для того чтобы уменьшить фрагментацию, в подобных системах предусмотрены фоновые процедуры СУБД, которые периодически проводят слияние смежных свободных участков в один.

Структура и представление хранимых данных, их размещение в пространстве памяти и используемые методы доступа определяются схемой хранения. Схема хранения оперирует в терминах типов объектов.

## 4.3. Структура хранимых данных

Единицей хранения данных в БД является **хранимая запись**. Она может представлять как полную запись концептуального уровня, так и некоторую её часть. Если запись разбивается на части, то её фрагменты представляются экземплярами хранимых записей каких-либо типов. Все части записи связываются указателями (ссылками) или размещаются по специальному закону так, чтобы механизмы междуровневого отображения могли опознать все компоненты и осуществить сборку полной записи концептуальной БД по запросу механизмов концептуального уровня.

Хранимые записи одного типа состоят из фиксированной совокупности полей и могут иметь формат фиксированной или переменной длины.

Записи переменной длины возникают, если допускается использование повторяющихся групп полей (агрегатов) с переменным числом повторов или полей переменной длины. Работа с хранимыми записями переменной длины существенно усложняет управление пространством памяти, но может быть продиктована желанием уменьшить объём требуемой памяти или характером модели данных концептуального уровня. В последнем случае для повышения производительности системы записи могут разбиваться на фрагменты фиксированной длины, возможно, различных типов.

Хранимая запись состоит из двух частей:

1. *Служебная часть*. Используется для идентификации записи, задания её типа, хранения признака логического удаления, для кодирования значений элементов записи, для установления структурных ассоциаций между записями. Никакие пользовательские программы не имеют доступа к служебной части хранимой записи.
2. *Информационная часть*. Содержит значения элементов данных.

Элементы хранимой записи могут иметь фиксированную или переменную длину. При этом обычно элементы фиксированной длины хранятся в начале записи и размещаются в памяти с заранее определённых позиций, а за ними размещаются элементы переменной длины. Хранение элементов переменной длины осуществляется одним из двух способов: размещение через разделитель или хранение размера элемента данных. Наличие полей переменной длины позволяет не хранить незначащие символы и снижает затраты памяти на хранение данных; но при этом увеличивается время на извлечение элементов записи.

Каждой записи БД система присваивает внутренний идентификатор, называемый (по стандарту CODASYL) **ключом базы данных** (КБД). Значение КБД формируется системой при размещении записи и содержит информацию, позволяющую однозначно определить место размещения записи (её адрес). В качестве КБД может выступать, например, последовательный номер записи в файле или совокупность адреса страницы и смещения от начала страницы.

Конкретные составляющие КБД зависят от операционной системы и от СУБД, точнее, от вида используемой адресации и от структуризации памяти, принятой в данной СУБД.

## 4.4. Виды адресации хранимых записей

Рассмотрим два вида адресации: прямую и косвенную.

**Прямая адресация** предусматривает указание непосредственного местоположения записи в пространстве памяти. Простейший вариант адресации используется в том случае, если в памяти хранится один вид записи фиксированной длины. Тогда в качестве адреса записи может использоваться её порядковый номер. (Прямая адресация используется, например, в системах ADABAS и dBaseIII PLUS).

Прямая адресация не позволяет перемещать записи в памяти без изменения ключа базы данных. Такие изменения привели бы к необходимости коррекции различных указателей на записи в среде хранения, что было бы чрезвычайно трудоемкой процедурой. В связи с отсутствием возможности перемещения при этом возникает фрагментация памяти.

Указанные недостатки можно преодолеть, используя **косвенную адресацию**. Существует множество способов косвенной адресации. Один из них состоит в том, что часть адресного пространства страницы выделяется под индекс страницы (рис. 4.2). Число статей (слотов) в нём одинаково для всех страниц.

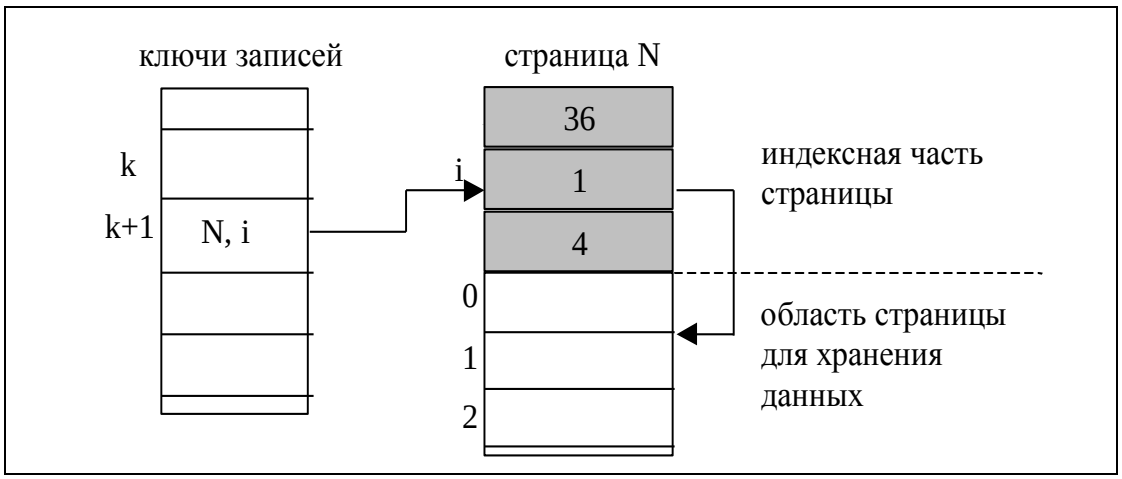


Рис.4.2. Косвенная адресация с использованием индексируемых страниц

Ключом БД по-прежнему служит номер этой записи в области. С помощью простых арифметических действий можно получить по номеру записи номер нужной страницы и номер требуемого слота в индексе этой страницы. Найденный слот укажет местоположение записи на этой странице, где N, n – это соответственно номер страницы памяти и номер слота на этой странице, в котором хранится адрес записи (смещение от начала страницы).

При перемещении записи она остаётся на той же странице, и слот по-прежнему указывает на неё (меняется его содержимое, но не сам слот). Если запись не вмещается на страницу, она помещается на специально отведённые в данной области *страницы переполнения*, и соответствующий слот продолжает указывать на место её размещения.

Этот подход позволяет перемещать записи на странице, исключать фрагментацию, возвращать освободившееся пространство для повторного использования. При этом приложения БД остаются нечувствительными к таким операциям. Таким образом, косвенная адресация входит в группу методов, обеспечивающих физическую независимость данных.

## 4.5. Способы размещения и доступа к данным

При создании новой записи во многих случаях существенно размещение этой записи в памяти, т.к. это оказывает огромное влияние на время выборки. Простейшая стратегия размещения данных заключается в том, что новая запись размещается на первом свободном участке (если ведется учёт свободного пространства) или вслед за последней из ранее размещённых записей. Среди более сложных методов можно отметить хеширование и кластеризацию.

### 4.5.1. Способы доступа к записям

Рассмотрим основные способы доступа к данным.

- **Последовательная обработка области БД**. Областью БД может быть файл или другое множество страниц. Последовательная обработка предполагает, что система последовательно просматривает страницы, пропускает пустые участки и выдаёт записи в физической последовательности их хранения.

- **Доступ по ключу базы данных** (КБД). КБД определяет местоположение записи в памяти ЭВМ. Зная его, система может извлечь нужную запись за одно обращение к памяти.
- **Доступ по структуре.** Эта разновидность доступа применяется для групповых отношений и позволяет перейти к предыдущему или следующему экземпляру группового отношения, к экземпляру-владельцу группового отношения или к списку подчинённых экземпляров.
- **Доступ по первичному ключу.** Первичный ключ идентифицирует записи внутри типа. Если система обеспечивает доступ по первичному ключу, то он (ключ) используется также при запоминании записи и, более того, его значение в этом случае может использоваться при размещении записи в памяти. Наиболее распространённые механизмы доступа по первичному ключу – индексирование и хеширование.

### 4.5.2. Индексирование данных

Определим индексирование как способ доступа к данным в реляционной таблице. Индексирование используется для ускорения доступа к записям по значению ключа и не влияет на размещение данных этой таблицы.

**Индекс** – это специальная структура, которая определяет соответствие значения ключа (атрибута или группы атрибутов) и местоположения записи (рис. 4.3). Индекс связан с определённой таблицей, но является внешним по отношению к таблице и хранится отдельно от неё.

Индекс

| *Значение атрибута* | *КБД* |
|---|---|
| Белова | FA:00 |
| Волков | F6:1E |
| Волкова | F6:00 |
| Осипов | FA:2B |
| Поспелов | F6:31 |
| Фридман | FA:1D |

Пространство памяти

| | | |
|---|---|---|
| F6:00 | Волкова | … |
| F6:1E | Волков | … |
| F6:31 | Поспелов | … |
| | **…** | |
| FA:00 | Белова | … |
| FA:1D | Фридман | … |
| FA:2B | Осипов | … |

Рис. 4.3. Пример индекса

Значения индексируемого атрибута упорядочиваются (чаще всего, по возрастанию). Индекс обычно хранится в отдельном файле или отдельной области памяти. Пустые значения атрибутов (null) не индексируются.

Индексы поддерживаются динамически, т.е. после обновления БД – добавлении или удалении записей, а также при модификации полей записи, входящих в ключ, – индекс приводится в соответствие с обновленной версией БД. Обновление индекса, естественно, занимает некоторое время (иногда, очень большое), поэтому существование многих индексов может замедлить работу БД. В реальных СУБД существуют методы оптимизации переиндексации. Например, при выполнении пакетной операции модификации БД обновление индексов может происходить один раз после внесения всех изменений в записи.

Обращение к записи через индексы осуществляется в два этапа: сначала в индексной структуре находится требуемое значение атрибута и соответствующий адрес записи (КБД), затем по этому адресу происходит обращение к внеш-

нему запоминающему устройству. Индекс загружается в оперативную память (ОП) целиком или хранится в ней постоянно во время работы с БД, если хватает объёма ОП.

Индекс называется *первичным,* если каждому значению индекса соответствует уникальное значение ключа. Если же индекс строится по ключу, допускающему дубликаты значений, такой индекс называется *вторичным*. Для каждой таблицы можно одновременно поддерживать несколько первичных и вторичных индексов, что также относится к достоинствам индексирования.

Различают индексы по одному полю и по нескольким (составные). **Составной индекс** включает два или более столбца одной таблицы (рис. 4.4). Последовательность вхождения столбцов в индекс определяется при его создании.

Таблица

| **ID** | **DATA** | **CODE** | FIRM | PRICE |
|---|---|---|---|---|
| 100 | 01.12.95 | A4 | Комус | 312.0 |
| 200 | 01.12.95 | A4 | Партия | 321.5 |
| 100 | 02.12.95 | A2 | ОАО "Заря" | 110.6 |
| 110 | 10.12.95 | A4 | Фирма "Б+" | 314.0 |
| 200 | 01.12.95 | A2 | Партия | 114.0 |
| 200 | 02.12.95 | A1 | Amos ltd. | 52.8 |

Индекс

| ID | DATA | CODE |
|---|---|---|
| 100 | 01.12.95 | A4 |
| 100 | 02.12.95 | A2 |
| 110 | 10.12.95 | A4 |
| 200 | 01.12.95 | A2 |
| 200 | 01.12.95 | A4 |
| 200 | 02.12.95 | A1 |

Рис. 4.4. Пример составного индекса

### 4.6.2.1. Способы организации индексов

Существует множество способов организации индексов:

1. В **плотных индексах** для каждого значения ключа имеется отдельная запись индекса, указывающая место размещения конкретной записи. **Неплотные (разреженные) индексы** строятся в предположении, что на каждой странице памяти (или в блоке) хранятся записи, отсортированные по значениям ключа индексирования. Тогда для каждой страницы индекс задаёт диапазон значений ключей хранимых в ней записей, и поиск записи осуществляется среди записей на указанной странице.
2. Для больших индексов актуальна проблема **сжатия ключа**. Наиболее распространенный метод сжатия основан на устранении избыточности хранимых данных. Последовательно идущие значения ключа обычно имеют одинаковые начальные части, поэтому в каждой статье индекса можно хранить не полное значение ключа, а лишь информацию, позволяющую его восстановить из известного предыдущего значения.
3. **Одноуровневый индекс** представляет собой линейную совокупность значений одного или нескольких полей записи. На практике он используется только в простейших случаях, когда количество индексируемых записей невелико. В более сложных случаях индекс занимает много памяти (иногда – несколько страниц), и возникает задача минимизации доступа к нему. Тогда индекс разбивается на иерархические уровни, что позволяет ускорить поиск требуемого значения. Особенно эффективной является организация **многоуровневых индексов** в виде сбалансированных деревьев (balance trees, B-деревьев), в которых все пути от корня к листьям имеют одинаковую длину.

### 4.6.2.2. Многоуровневые индексы на основе B-дерева

B-дерево строится динамически по мере заполнения базы данными. Оно растёт вверх, и корневая вершина может меняться. Параметрами B-дерева являются порядок *n* и количество уровней. Порядок – это количество ссылок из вершины i-го уровня на вершины (i+1)-го уровня. Каждое B-дерево должно удовлетворять следующим условиям:

1. Каждая вершина может содержать *n* адресных ссылок и (*n-1*) ключей. Ссылка влево от ключа обеспечивает переход к вершине дерева с меньшими по значению ключами, а вправо – к вершине с большими ключами.
2. Любая неконечная вершина имеет не менее *n*/*2* подчинённых вершин.
3. Если неконечная вершина содержит *k* (*k*≤*n*) ключей, то ей подчинена (*k+1*) вершина на следующем уровне иерархии.
4. Все конечные вершины расположены на одном уровне.

Алгоритм формирования B-дерева порядка *n* предполагает, что сначала заполняется корневая вершина. Затем при появлении новой записи корневая вершина делится, образуются подчинённые ей вершины. При запоминании каждой новой записи поиск места для неё начинается с корневой вершины. Если в существующем на данный момент B-дереве нет места для размещения нового ключа, происходит сдвиг ключей вправо или влево, если это невозможно – осуществляется перестройка дерева. Пример построения B-дерева порядка 3 приведён на рис. 4.5.

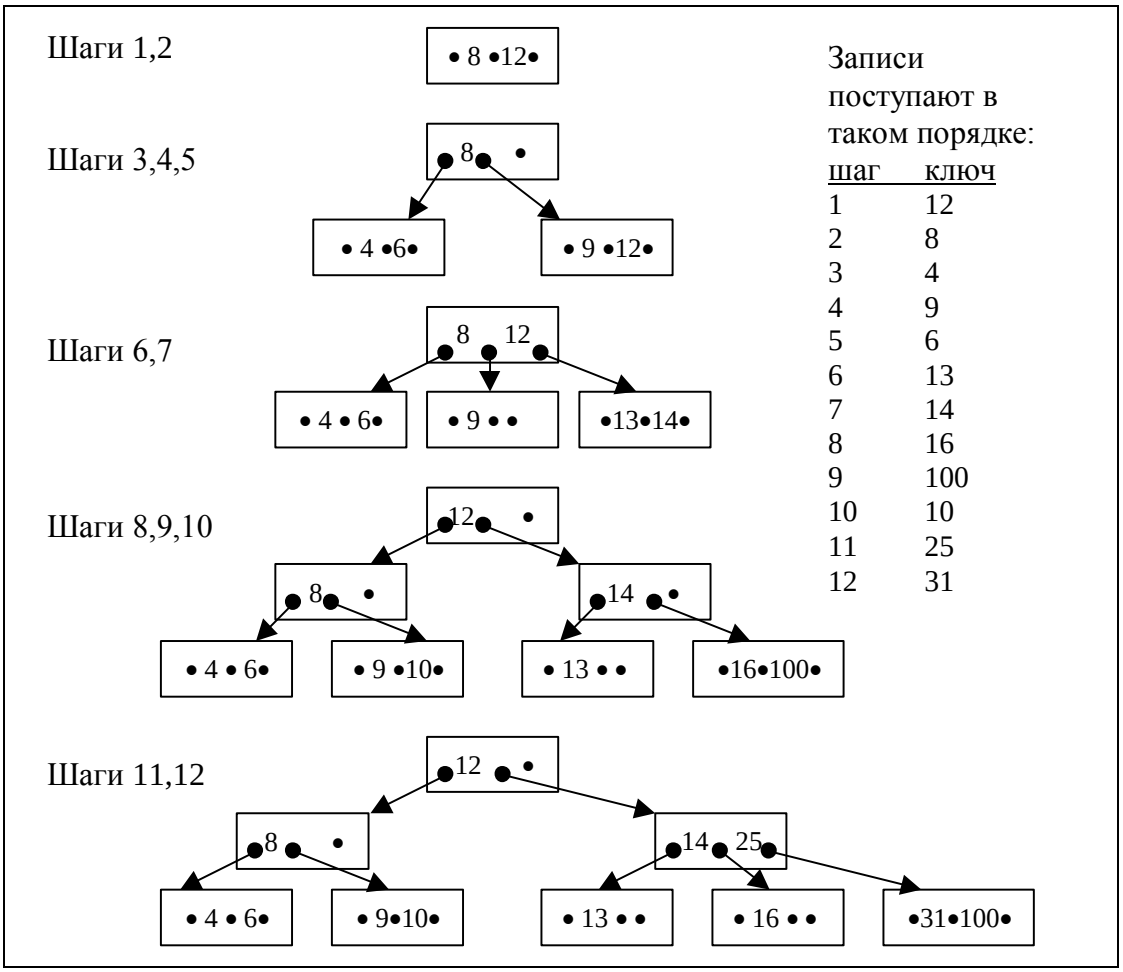


Рис. 4.5. Пример построения B-дерева порядка 3

Индексирование в виде B-дерева используется, например, в СУБД Oracle (рис. 4.6).

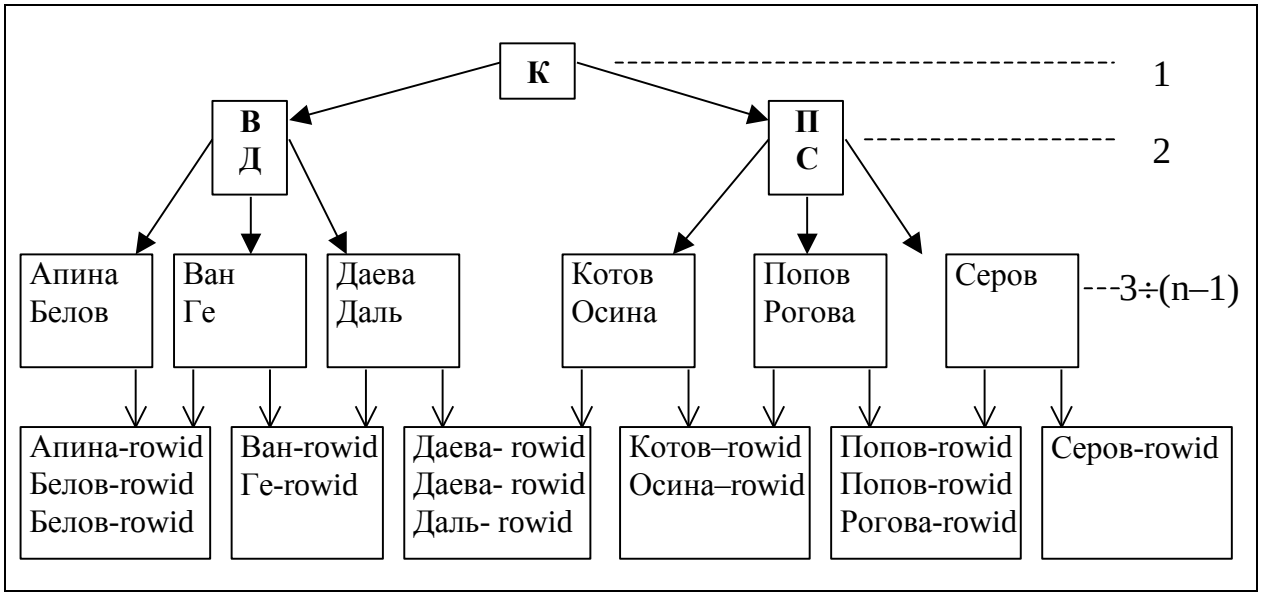


Рис. 4.6. Пример индексного блока СУБД Oracle

Организация индексов в СУБД Oracle несколько отличается от рассмотренной выше классической организации B-дерева, но принцип остаётся тот же: одинаковое количество уровней на любом пути и автоматическая сбалансированность. Верхние блоки содержат данные индекса, которые ссылаются на блоки индекса нижних уровней. Самый нижний n–й уровень содержит блоки индекса (блоки-листья), которые содержат непосредственно данные индекса (ключи) и соответствующие идентификаторы строк ROWID (row identification, КБД), используемые для нахождения самих строк. Блоки-листья связаны между собой указателями.

Поиск по ключу осуществляется следующим образом. Блок верхнего уровня (уровень 1) содержит некоторое значение X и указатели на верхнюю и нижнюю части индекса. Если значение искомого ключа больше X, то происходит переход к верхней части индекса (по левому указателю), иначе – к нижней части. Блоки второго и последующих уровней (кроме двух последних) хранят начальное $X_0$ и конечное значения $X_к$ ключа, а также три указателя. Если значение искомого ключа больше, чем $X_0$, то происходит обращение по левому указателю; если оно меньше, чем $X_к$, то происходит обращение по правому указателю; если оно попадает в диапазон $X_0 \div X_к$ – по среднему указателю.

Предпоследний уровень содержит значения ключей индекса и указатели на блоки последнего уровня, последний – значения ключей индекса и идентификаторы строк (ROWID). Различие между двумя последними уровнями в том, что в случае неуникальных индексов значение ключа индекса в предпоследнем уровне содержится один раз, а в последнем – столько раз, сколько оно встречается в записях файла данных. При обнаружении значения искомого ключа в блоке индекса происходит обращение к диску по ROWID и извлечение требуемой записи (записей). Если же значение не обнаружено, результат поиска пуст.

Уникальные ключи для каждого значения имеют только один соответствующий ROWID. Для неуникальных индексов значения идентификаторов строк в индексе также отсортированы по возрастанию.

Индекс в виде B-дерева автоматически поддерживается в сбалансированном виде. Это означает, что при переполнении какого-либо из блоков индекса происходит перераспределение значений ключей индекса (без физического перемещения записей данных). Например, если при добавлении новой записи с ключом "Горин" возникает переполнение соответствующего блока индекса (рис. 4.6), система может перестроить индекс так, как показано на рис. 4.7.

Если все блоки-листья индекса заполнены приблизительно на три четверти, то при добавлении новой записи осуществляется полная перестройка B-дерева путём введения дополнительного уровня. Всё это скрыто от пользователя и происходит автоматически.

Структура B-дерева имеет следующие преимущества:

- Все блоки-листья в дереве одной и той же глубины, следовательно, поиск любой записи в индексе занимает примерно одно и то же время.
- B-дерево автоматически поддерживается в сбалансированном виде.

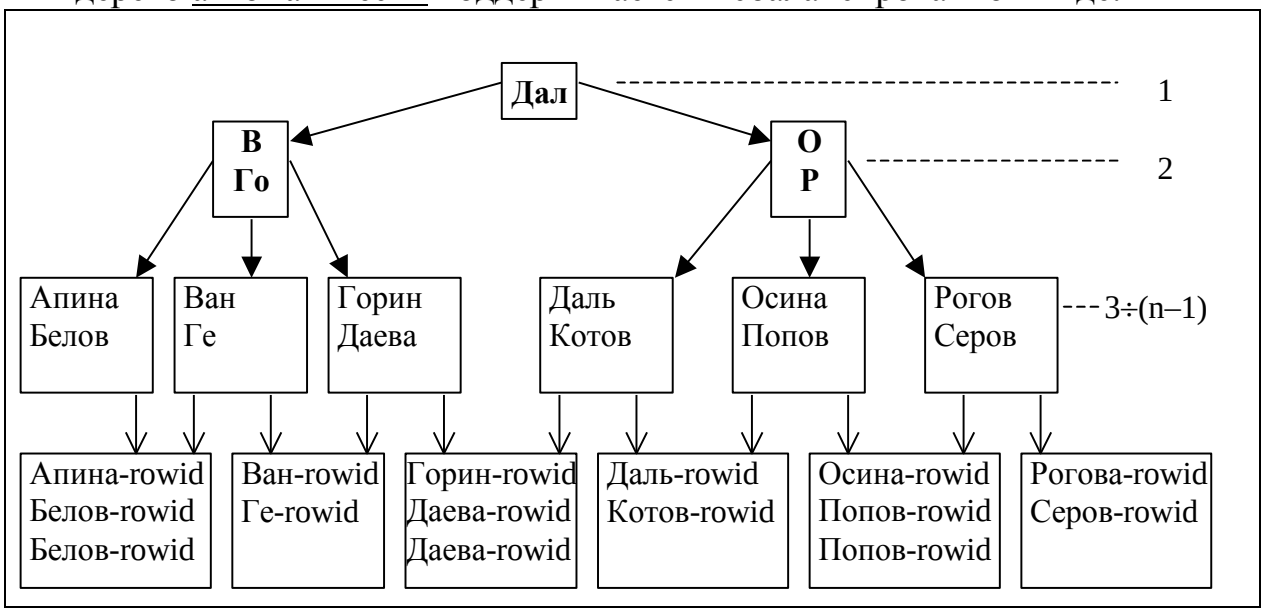


Рис.4.7. Пример перераспределения данных индексного блока СУБД Oracle

- B-деревья обеспечивают хорошую производительность для широкого спектра запросов, включая поиск по конкретному значению и поиск в открытом и закрытом интервалах.
- Модификация (т.е. добавление, обновление и удаление) строк выполняется достаточно эффективно.
- Производительность B-дерева одинаково хороша для маленьких и больших таблиц, и не меняется существенно при росте таблицы.

### 4.6.2.3. Использование индексов

В системах, поддерживающих язык SQL, индекс создаётся командой CREATE INDEX. Индексы повышают производительность запросов, которые выбирают относительно небольшое число строк из таблицы. Для определения

целесообразности создания индекса нужно проанализировать запросы, обращённые к таблице, и распределение данных в индексируемых столбцах.

Система может воспользоваться индексом по определённому полю, если в запросе на значение этого поля накладывается условие, например:

```
SELECT * FROM emp WHERE name = 'Даль';
```

Но даже при наличии такой возможности система не всегда обращается к индексу. Очевидно, что если запрос выбирает больше половины записей отношения, то извлечение данных через индекс потребует больше времени, чем последовательная обработка данных. В подобных случаях использование индекса нецелесообразно.

Обращение к составному индексу возможно только в том случае, если в условиях выбора участвуют столбцы, представляющие собой лидирующую часть составного индекса. Если индекс, например, включает поля (X, Y, Z), то обращение к индексу будет происходить в тех случаях, когда в условии запроса участвуют поля XYZ, XY или X, причём именно в таком порядке.

При создании индекса большое значение имеет понятие селективности. **Селективность** определяется процентом строк, имеющих одинаковое значение индексируемого столбца: чем выше этот процент, тем меньше селективность.

Выбор столбцов для индекса определяется следующими соображениями:

- В первую очередь выбираются столбцы, которые часто встречаются в критериях поиска.
- Стоит индексировать столбцы, которые используются для соединения таблиц или являются внешними ключами. В последнем случае наличие индекса позволяет обновлять строки подчинённой таблицы без блокировки основной таблицы, когда происходит интенсивное конкурентное обновление связанных между собою таблиц.
- Нецелесообразно индексировать столбцы с низкой селективностью. Исключения для низкой селективности составляют случаи, при которых выборка чаще производится по редко встречающимся значениям.
- Не индексируются столбцы, которые часто обновляются, т.к. команды обновления ведут к потере времени на обновление индекса.
- Не индексируются столбцы, которые часто используются как аргументы функций или выражений: как правило, такие функции не позволяют использовать индекс.

В некоторых случаях использование составного индекса предпочтительнее, чем одиночного, а именно:

- Несколько столбцов с низкой селективностью в комбинации друг с другом могут дать гораздо более высокую селективность.
- Если в запросах часто используются только столбцы, участвующие в индексе, система может вообще не обращаться к таблице для поиска данных.

**Примечание:** большинство СУБД автоматически строят индекс по первичному ключу и по уникальным столбцам.

## 4.5.3. Хеширование

При ассоциативном доступе к хранимым записям, предполагающем определение местоположения записи по значениям содержащихся в ней данных, используются более сложные механизмы размещения. Для этой цели используются различные методы отображения значения ключа в адрес, например, методы хеширования (перемешивания).

Принцип хеширования заключается в том, что для ускорения поиска информации область хранения данных разбивается на участки, каждому из которых ставится в соответствие некоторое значение (указатель или адрес). Для определения, в какой участок будет помещена вновь добавляемая запись, к значению ключевого поля этой записи применяется так называемая *хеш-функция h(K)*. Она преобразует значение ключа *K* в адрес произвольного участка памяти (это называется *свёрткой ключа*). При поиске записи по известному значению ключа *K* хеш-функция выдаёт значение адреса, указывающее, где начинается тот участок памяти, в котором надо искать эту запись.

Хеш-функция *h(K)* должна выдавать такие значения адресов, чтобы обеспечить равномерное распределение записей в памяти. При этом для близких значений ключа значения адресов должны сильно отличаться, чтобы избегать перекосов в размещении данных. Хорошая хеш-функция для каждого значения ключа выдаёт свой адрес, таким образом, извлечение записи производится за одно обращение к памяти. Для реальных функций хеширования допускается совпадение значений функции *h(K)* для различных ключей и для разрешения неопределённости после вычисления *h(K)* используются специальные методы.

Недостаток методов подбора хеш-функций заключается в том, что количество данных и распределение значений ключа должны быть известны заранее. Также методы хеширования неудобны тем, что записи неупорядочены по значению ключа, что приводит к дополнительным затратам, например, при выполнении сортировки. К преимуществам хеширования относится то, что обращение к данным происходит за одну операцию ввода/вывода, т.к. значение ключа непосредственно преобразуется в адрес соответствующей записи.

### 4.5.3.1. Методы хеширования

Многочисленные эксперименты с реальными файлами выявили удовлетворительную работу двух основных типов хеш-функций. Один из них основан на делении, другой – на умножении. Все рассуждения ведутся в предположении, что хеш-функция $h(K): 0 \le h(K) \le N$ для всех ключей $K$, где $N$ – размер памяти (количество ячеек).

Метод деления использует остаток от деления на M:

$$h(K) = K \bmod M.$$

Если $M$ – чётное число, то при чётных $K$ значение $h(K)$ будет чётным, и наоборот, что даёт значительные смещения значений функции для близких значений $K$. Нельзя брать $M$ кратным основанию системы счисления машины, а также кратным 3. Вообще, $M$ должно удовлетворять условию:

$$M \neq r^k \pm a ,$$

где $k$ и $a$ – небольшие числа, а $r$ – "основание системы счисления" для большинства используемых литер (как правило, 64, 256 или 100), т.к. остаток от деления на такое число оказывается обычно простой суперпозицией цифр ключа. Чаще всего в качестве $M$ берут простое число, например, вполне удовлетворительные результаты даёт $M = 1009$.

Мультипликативный метод также легко реализовать. В соответствии с ним хеш-функция определяется так:

$$h(K) = M\left(\left(\frac{A}{w}K\right) \bmod 1\right),$$

где $w$ – размер машинного слова (обычно, $2^{31}$), $A$ – целое число простое по отношению к $w$, а $M$ – некоторая степень основания системы счисления ЭВМ ($2^m$). Таким образом, в качестве значения функции берутся $m$ правых значащих цифр дробной части произведения значения ключа и числа $A/w$. Преимущество второго метода перед первым обусловлено тем, что произведение обычно вычисляется быстрее, чем деление.

При использовании любых методов хеширования для размещения записей должен быть выделен участок памяти размером $N$. Для того чтобы полученное в результате значение $h(K)$ не вышло за границы отведённого участка памяти, окончательно адрес записи вычисляется так:

$$A(K) = h(K) \bmod N.$$

### 4.5.3.2. Разрешение коллизий

Случай, когда для двух и более ключей выдаётся одинаковый адрес, называется **коллизией**. Наличие коллизий снижает эффективность хеширования.

Разрешение коллизий достигается путём **рехеширования** – специального алгоритма, который используется при размещении новой записи или при поиске существующей. В системах баз данных рехеширование выполняется одним из следующих способов:

1. **Открытая адресация**: новая запись размещается вслед за последней записью на данной странице или на следующей, если страница заполнена. (Для последней страницы памяти следующей является первая страница). Поиск записи осуществляется также последовательно, откуда следует, что записи нельзя удалять физически (с освобождением памяти), иначе цепочка рехешированных записей прервётся, и часть записей может быть "потеряна".
2. Использование **коллизионных страниц**: новая запись размещается на одной из коллизионных страниц, относящихся к таблице (в *области переполнения*). Для ускорения поиска рехешированных записей может использоваться связанная область переполнения, для которой на странице хранится ссылка на коллизионную страницу. Нулевое значение такой ссылки говорит об отсутствии коллизий для данных, размещённых на этой странице.
3. **Многократное хеширование**. Заключается в том, что при возникновении коллизии для поиска другого адреса (возможно, на коллизионных страницах) применяется другая функция хеширования.

**Примечание:** существуют и более сложные стратегии рехеширования; но их рассмотрение выходит за рамки данного пособия.

#### 4.5.3.2. Использование хеширования

Хеширование таблицы полезно в следующих случаях:

- Большинство запросов обращаются по значению уникального ключа, например:

  SELECT … WHERE unique_key = …;

  Значение, указанное в предикате, хешируется; по этому хеш-значению происходит прямой доступ к соответствующему блоку данных (обычно, одно физическое чтение). В случае обыкновенной индексированной таблицы происходит сначала обращение к индексу (несколько физических операций чтения), затем уже считывается сама строка.
- Таблица практически статична (редко обновляется) Число строк и требуемое физическое пространство можно определить заранее и зафиксировать. Если впоследствии таблица вырастет и придётся отводить ей дополнительные блоки, это может сильно ухудшить производительность.

Хеширование не рекомендуется в следующих случаях:

- Нельзя заранее выделить столько пространства памяти, сколько потребуется таблице в будущем.
- Большинство запросов выбирают строки в некотором интервале. Хеширование не даёт здесь преимуществ, т.к. строки не упорядочены (в отличие от индекса).
- Таблица быстро меняется и постоянно растёт.

Эффективность использования хеширования не в последней степени определяется качеством хеш-функции. Системы, поддерживающие возможность хеширования данных, обычно имеют встроенную хеш-функцию, но и позволяют пользователю задавать свою. Это может понадобиться тогда, когда встроенная хеш-функция не даёт хороших результатов, а пользовательская может учесть особенности распределения значений конкретного ключа. Если же ключ является уникальным и распределение его значений равномерно, то сами значения могут быть использованы в качестве хеш-значений.

### 4.5.4. Кластеризация данных

#### 4.5.4.1. Принцип организации кластеров

Кластеризация является методом совместного хранения родственных данных (таблиц). **Кластер –** это структура памяти, в которой хранится набор таблиц (в одних и тех же блоках памяти). Кластеризуемые таблицы должны иметь общие столбцы, используемые для соединения (например, первичный ключ таблицы *ТОВАРЫ* и внешний ключ таблицы *ПОСТАВКИ*, рис. 4.8,б).

**Кластерный ключ** – это столбец или набор столбцов (полей записи), общих для всех кластеризуемых таблиц. Каждая таблица, созданная в кластере, должна иметь столбцы, соответствующие типам и размерам столбцов кластер-

ного ключа. Количество столбцов в кластерном ключе ограничено (например, для Oracle8 это ограничение равно 16).

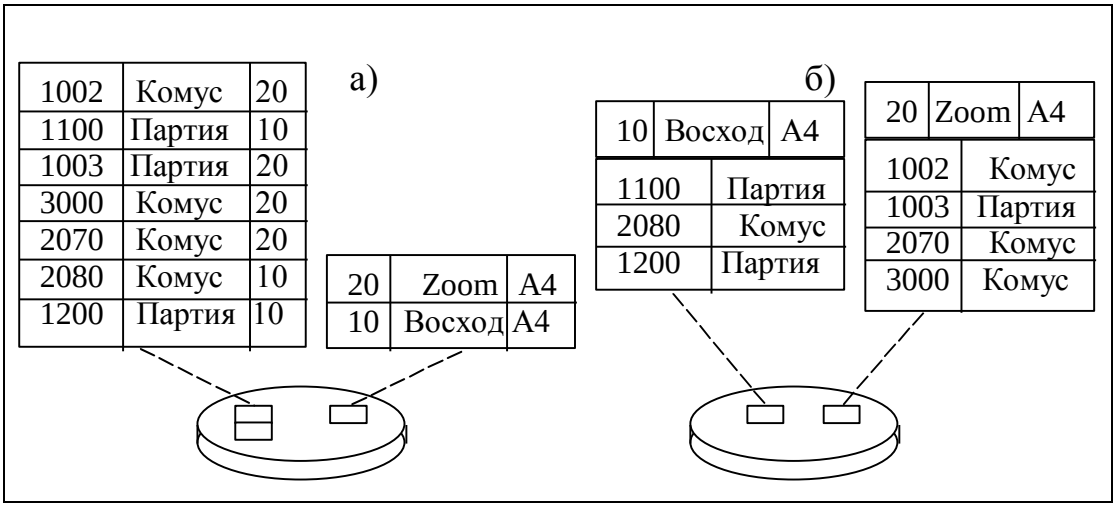


Рис. 4.8. Некластеризованные (а) и кластеризованные (б) данные

Совместное хранение означает, что на одной странице или в одном блоке памяти хранятся данные из всех кластеризованных таблиц, имеющие одинаковое значение кластерного ключа. Физически это обычно реализуется так: в начале страницы (блока) хранится запись из таблицы, для которой кластерный ключ является первичным (или уникальным), а вслед за ней располагаются записи из другой таблицы (таблиц), имеющие те же значения кластерного ключа. Фактически, данные хранятся в виде соединения таблиц по значениям кластерного ключа. В этом случае выигрыш по времени для выполнения соединения таблиц по сравнению с раздельно хранимыми таблицами составляет 3-6 раз.

Если все данные, относящиеся к одному значению кластерного ключа, не помещаются в одном блоке, то выделяется новый блок памяти и предыдущий блок хранит ссылку на него. Но если система позволяет изменять размер блока (в частности, СУБД Oracle), при создании кластера желательно установить размер блока исходя из оценки среднего количества записей с одинаковыми значения кластерного ключа.

Значения кластерного ключа таблицы могут обновляться. Но надо иметь в виду, что обновление может вызвать физическое перемещение записи, так как расположение записи зависит от значения кластерного ключа. Поэтому часто обновляющиеся атрибуты не являются хорошими кандидатами на вхождение в кластерный ключ.

Два основных преимущества кластеров:

- Уменьшается обмен с диском, улучшается время доступа к кластеризованным таблицам и их соединение.
- Значение кластерного ключа хранится только один раз для кластера, за счёт чего достигается экономия памяти.

С другой стороны, наличие кластеров обычно увеличивает время выполнения операции добавления записи (INSERT). При добавлении записи система тратит дополнительное время на просмотр блоков данных с целью поиска того блока, куда нужно поместить новую запись.

#### 4.5.4.2. Использование кластеров

Кластеры обычно строятся для таблиц, часто используемых в соединении друг с другом, например, связанных отношением "один-ко-многим". Не стоит создавать кластер:

- если данные в кластерном ключе этих таблиц часто обновляются;
- если часто требуется полный просмотр отдельной таблицы.
- если суммарные данные таблиц с одним и тем же значением кластерного ключа занимают больше одного блока данных.

Изменение столбцов кластерного ключа требует гораздо больше системных ресурсов, чем обновление некластеризованных данных, так что выигрыш от ускорения поиска данных оказывается меньше, чем затраты на физическое перемещение строк.

Полный просмотр индивидуальных таблиц кластера требует больше времени, чем просмотр некластеризованных таблиц, т.к. физически требуется обратиться к большему числу блоков. Если по отдельности некластеризованные таблицы занимают n1 и n2 блока соответственно, то вместе они будут занимать (n1+n2) блоков, и для полного просмотра каждой из них придётся обращаться к диску (n1+n2) раз.

Часто для окончательного определения целесообразности создания кластера в конкретной ситуации ставят эксперименты и измеряют производительность БД на реальных данных и реальных запросах.

Рассмотренные способы размещения и доступа к данным прозрачны для пользователей и приложений. То есть кластеризация, хеширование и индексирование оказывают влияние на время обработки данных, но не требуют изменения программ и запросов. Информация о методах размещения (кластеризация, хеширование) и методах доступа (наличие индексов) хранится в словаре-справочнике данных и используется системой при выполнении запросов.

## 5. ОПТИМИЗАЦИЯ РЕЛЯЦИОННЫХ ЗАПРОСОВ

В оптимизацию реляционных запросов входят два различных аспекта. Во-первых, это внутренняя задача СУБД, которая заключается в определении наиболее оптимального (эффективного) способа выполнения реляционных запросов. Во-вторых, это задача программиста (или квалифицированного пользователя): она заключается в написании таких реляционных запросов, для которых СУБД могла бы использовать более эффективные способы нахождения данных. Сначала рассмотрим первый аспект.

### 5.1. Этапы оптимизации запросов в реляционных СУБД

Каждая команда языка манипулирования данными может быть выполнена разными способами. Определение наиболее оптимального плана выполнения запроса называется **оптимизацией**. Выбором этого плана занимается оптимизатор – специальная компонента СУБД.

Выполнение запроса состоит из последовательности шагов, каждый из которых либо физически извлекает данные из памяти, либо делает подготовительную работу. Последовательность шагов, которую строит оптимизатор, называется **планом выполнения.**

Обработка запроса, поступившего в РСУБД и представленного на некотором языке запросов, состоит из этапов (фаз), представленных на рис. 5.1.

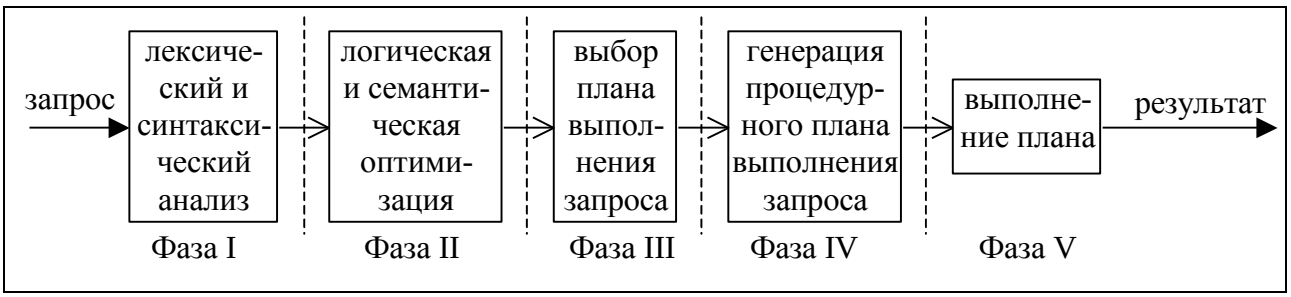


Рис.5.1. Последовательность выполнения запросов в реляционных СУБД

На первой фазе запрос, представленный на языке запросов, подвергается лексическому и синтаксическому анализу. Лексический анализатор разбивает запрос на лексические единицы – лексемы (наименования полей и таблиц, константы, знаки операций и т.д.). Синтаксический анализатор проверяет синтаксическую правильность запроса. В результате вырабатывается внутреннее представление запроса. Оно отражает структуру запроса и содержит информацию, которая характеризует объекты базы данных, упомянутые в запросе (таблицы, поля, константы). Информация об объектах базы данных выбирается из словаря-справочника данных. Внутреннее представление запроса используется и преобразуется на следующих стадиях обработки запроса.

На второй фазе запрос в своём внутреннем представлении подвергается логической оптимизации. При этом могут применяться различные преобразования, "улучшающие" начальное представление запроса. Среди этих преобразований могут быть эквивалентные преобразования (см. раздел 5.2. "Преобразования операций реляционной алгебры"). После проведения эквивалентных преобразований получается внутреннее представление, семантически эквивалентное начальному.

Преобразования могут также использовать информацию об ограничениях целостности, существующих в БД. Такие преобразования являются семантическими, т.е. они основаны на семантике (смысле) предметной области. В этом случае получаемое представление не является семантически эквивалентным начальному. Но система гарантирует, что результат выполнения преобразованного запроса совпадает с результатом запроса в начальной форме при соблюдении ограничений целостности, существующих в базе данных.

В любом случае после выполнения второй фазы обработки запроса его внутреннее представление остается непроцедурным, хотя и является в некотором смысле более эффективным, чем начальное. Это означает, что по этому представлению система сможет построить более эффективный план.

Третий этап обработки запроса состоит в выборе альтернативных проце-

дурных планов выполнения данного запроса в соответствии с его внутреннем представлением, полученным на второй фазе. Для этого оптимизатор использует информацию из словаря-справочника данных, в первую очередь, о существующих путях доступа к данным. Единственный путь доступа, который возможен в любом случае, – это последовательное чтение (FULL). Возможность использования других путей доступа зависит от способов размещения данных в памяти (например, кластеризация или хеширование данных), от наличия индексов и формулировки самого запроса.

Также на третьем этапе для каждого из выбранных планов оценивается предполагаемая стоимость выполнения запроса по этому плану. При оценках используется либо доступная оптимизатору статистическая информация о распределении данных, либо информация о механизмах реализации путей доступа. Из полученных альтернативных планов выбирается наиболее оптимальный с точки зрения некоторого (заранее выбранного или заданного) критерия.

На четвертом этапе по внутреннему представлению наиболее оптимального плана выполнения запроса формируется процедурное представление плана. Выполняемое представление плана может быть программой в машинных кодах, если, как в случае System R, система ориентирована на компиляцию запросов в машинные коды. В других системах, например, в INGRES, представление плана является машинно-независимым, что более удобно для интерпретации запросов. Для нас это непринципиально, поскольку четвертая фаза обработки запроса уже не связана с оптимизацией.

Наконец, на последнем, пятом этапе обработки запроса происходит его реальное выполнение в соответствии с процедурным планом запроса. Это либо выполнение соответствующей подпрограммы, либо вызов интерпретатора с передачей ему для интерпретации выполняемого плана.

## 5.2. Преобразования операций реляционной алгебры

Операндами операций реляционной алгебры являются отношения, поэтому для их выполнения необходимо просмотреть все кортежи исходного отношения (или отношений). Следствием этого является большая размерность реляционных операций. Уменьшения размерности операций можно достичь, изменяя последовательность выполняемых операций.

В качестве примера приведём отношения R1 и R2, содержащие по 1000 кортежей, причём только 10 кортежей в каждом отношении удовлетворяют условию F. Если выполнять следующую последовательность операций:

$$\sigma_F (R1 \cup R2),$$

то после выполнения объединения получится 2000 кортежей (если отношения не содержат одинаковых кортежей), а после селекции останется 20 записей. Если изменить последовательность выполнения операций:

$$\sigma_F (R1) \cup \sigma_F (R2),$$

то после селекции останется по 10 записей из каждого отношения, объединение которых даст 20 требуемых кортежей. Если учитывать, что объединение выполняется путем сортировки данных (для удаления одинаковых кортежей) и

промежуточный результат надо хранить, то выигрыш и по объему памяти, и по времени очевиден: гораздо быстрее отсортировать 20 кортежей, а не 2000.

Оптимизация выполнения запросов реляционной алгебры основана на понятии эквивалентности реляционных выражений. Операндами выражений являются переменные-отношения $R_i$ и константы. Каждое выражение реляционной алгебры определяет отображение кортежей переменных-отношений $R_i$ (i=1,…,n) в кортежи единственного отношения, которое получается в результате подстановки кортежей каждого $R_i$ и выполнения всех определяемых выражением вычислений.

| Два выражения реляционной алгебры считаются **эквивалентными**, если они описывают одно и то же отображение. |
|---|

Существуют законы, которые в соответствии с этим определением позволяют выполнять эквивалентные преобразования выражений реляционной алгебры:

1. Закон коммутативности для декартовых произведений:

   $R1 \times R2 = R2 \times R1$

2. Закон коммутативности для соединений (F – условие соединения):

   $R1 \rhd\lhd_F R2 = R2 \rhd\lhd_F R1$

3. Закон ассоциативности для декартовых произведений:

   $(R1 \times R2) \times R3 = R1 \times (R2 \times R3)$

4. Закон ассоциативности для соединений (F1, F2 – условия):

   $(R1 \rhd\lhd_{F1} R2) \rhd\lhd_{F2} R3 = R1 \rhd\lhd_{F1} (R2 \rhd\lhd_{F2} R3)$

5. Комбинация селекций (каскад селекций):

   $\sigma_{F1} (\sigma_{F2} (R)) = \sigma_{F1 \wedge F2} (R)$

6. Комбинация проекций (каскад проекций):

   $\pi_{A1,A2,...,Am} (\pi_{B1,B2,...,Bn}(R)) = \pi_{A1,A2,...,Am}(R)$

   где $\{A_m\} \subset \{B_n\}$.

7. Перестановка селекции и проекции:

   $\sigma_F (\pi_{A1,A2,...,Am}(R)) = \pi_{A1,A2,...,Am}(\sigma_F (R))$

8. Перестановка селекции с объединением:

   $\sigma_F (R1 \cup R2) = \sigma_F (R1) \cup \sigma_F (R2)$

9. Перестановка селекции с декартовым произведением:

   - $\sigma_F (R1 \times R2) = (\sigma_{F1} (R1)) \times (\sigma_{F2} (R2))$
     (если $F = F1 \wedge F2$, где F1 содержит атрибуты, присутствующие только в R1, а F2 содержит атрибуты, присутствующие только в R2);
   - $\sigma_F (R1 \times R2) = (\sigma_F (R1)) \times R2$
     (если F содержит атрибуты, присутствующие только в R1);
   - $\sigma_F (R1 \times R2) = R1 \times (\sigma_F (R2))$
     (если F содержит атрибуты, присутствующие только в R2);

- $\sigma_F (R1 \times R2) = \sigma_{F2} (\sigma_{F1} (R1) \times R2)$

  (если $F = F1 \wedge F2$, где F1 содержит атрибуты, присутствующие только в R1, а F2 содержит атрибуты, присутствующие и в R1, и в R2).

10. Перестановка селекции с разностью:

    $\sigma_F (R1 - R2) = \sigma_F (R1) - \sigma_F (R2)$

11. Перестановка проекции с декартовым произведением:

    $\pi_{A1,A2,...,Am} (R1 \times R2) = (\pi_{B1,B2,...,Bn}(R1)) \times (\pi_{C1,C2,...,Cr}(R2))$

    где атрибуты $\{B_n\} \subset \{A_m\}$, $\{C_r\} \subset \{A_m\}$ и атрибуты $B_1,B_2,...,B_n$ представлены в отношении R1, а атрибуты $C_1,C_2,...,C_r$ – в R2.

12. Перестановка проекции с объединением:

    $\pi_{A1,A2,...,Am} (R1 \cup R2) = (\pi_{A1,A2,..., Am} (R1)) \cup (\pi_{A1,A2,...,Am} (R2))$

## 5.3. Методы оптимизации

Существуют два принципиально разных подхода к оптимизации запросов. Если оптимизатор основывается только на информации о механизмах реализации путей доступа, то метод оптимизации основан на синтаксисе (на правилах, RULE). Если же помимо этого используется статистическая информация о распределении данных, то это метод оптимизации, основанный на стоимости (на издержках, COST). Рассмотрим эти подходы подробнее.

### 5.3.1. Метод оптимизации, основанный на синтаксисе

При использовании этого метода план составляется на основании существующих путей доступа и их рангов. Все пути доступа ранжируются на основании знаний о правилах и последовательности осуществления этих путей. В табл.5.1 в качестве примера приведены ранги путей доступа для СУБД Oracle8.

Таблица 5.1. Ранги путей доступа для СУБД Oracle

| Ранг | Пути доступа |
|---|---|
| 1 | Одна строка по ROWID* |
| 2 | Одна строка по кластерному соединению |
| 3 | Одна строка по хеш-кластеру с уникальным или первичным ключом |
| 4 | Одна строка по уникальному или первичному ключу |
| 5 | Кластерное соединение |
| 6 | Ключ хеш-кластера |
| 7 | Ключ индексного кластера |
| 8 | Составной индекс |
| 9 | Индекс по одиночному столбцу (по условию равенства) |
| 10 | Индексный поиск по закрытому интервалу |
| 11 | Индексный поиск по открытому интервалу |
| 12 | Сортировка-объединение |
| 13 | MAX и MIN по индексированному столбцу |
| 14 | ORDER BY по индексированному столбцу |
| 15 | Полный просмотр таблицы |

* ROWID (идентификатор строки) – значение, которое может быть однозначно преобразовано в физический адрес записи (КБД)

Ранг пути доступа определяется на основании знаний о последовательности реализации этого пути. Например, самый быстрый способ доступа – это чтение по КБД: если он известен, то это одно физическое чтение. А поиск конкретного значения через индекс (ранг 9) обычно занимает меньше времени, чем поиск в закрытом интервале (ранг 10).

Метод оптимизации по синтаксису учитывает ранги путей доступа. Если для какой-либо операции существует более одного пути доступа, то выбирается тот путь, чей ранг выше, т.к. в большинстве случаев он выполняется быстрее, чем путь с более низким рангом. План выполнения запроса формируется из выбранных путей доступа с максимальными рангами (рис. 5.2).

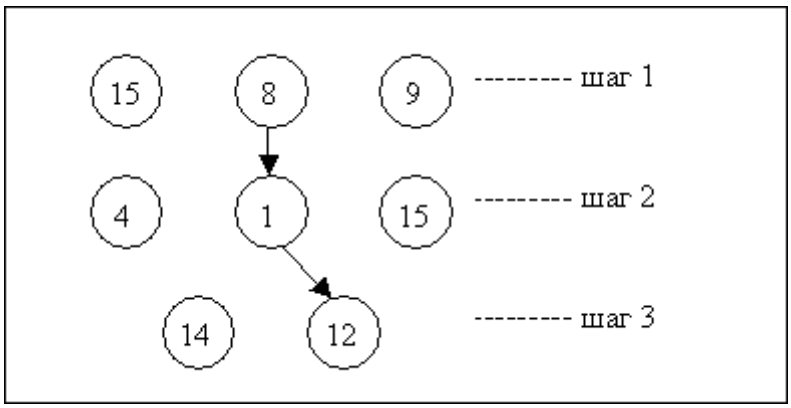


Рис.5.2. Построение плана выполнения запроса в методе оптимизации по синтаксису

### 5.3.2. Метод оптимизации, основанный на стоимости

При использовании этого метода оптимизатор сначала строит несколько возможных планов выполнения запроса. При этом он применяет некоторые **эвристики**, т.е. правила, полученные опытным путем. Эти правила позволяют сузить пространство поиска оптимального плана благодаря тому, что неэффективные планы отбрасываются в самом начале и не рассматриваются. Примером эвристики может послужить такое правило: хорошим является такой план, в котором селекция производится на раннем этапе выполнения запроса.

Для каждого из построенных планов рассчитывается его стоимость. **Стоимость** – это оценка ожидаемого времени выполнения запроса с использованием конкретного плана выполнения. При расчёте стоимости оптимизатор может учитывать такие параметры, как:

- количество необходимых ресурсов памяти,
- время операций дискового ввода-вывода,
- время процессора.

Из множества возможных планов выполнения запроса (обычно, 2-3 плана) оптимизатор в соответствии с критерием выбирает лучший план (рис. 5.3).

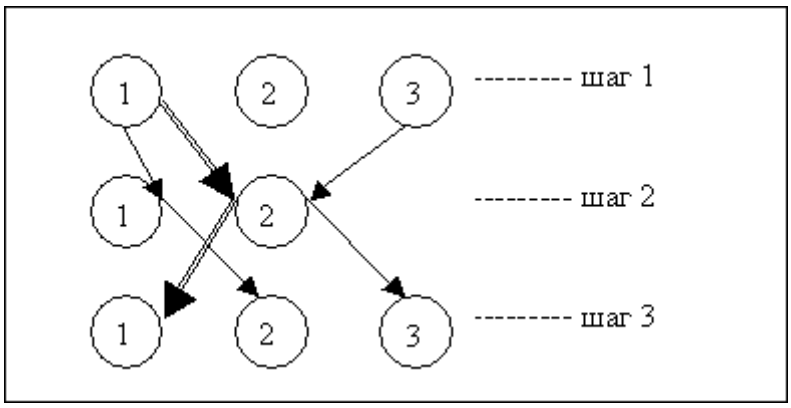


Рис.5.3. Построение плана выполнения запроса в методе оптимизации по стоимости

Естественно, для сравнения планов выполнения оптимизатору нужен критерий. В качестве **критерия оптимизации** может выступать:

- Наилучшая общая производительность системы. Вообще говоря, её можно достичь, если из всех планов выбирать те, которые требуют меньше всего ресурсов (памяти и центрального процессора). Это позволит увеличить степень параллельности работы системы и повысить общую производительность, хотя при этом время выполнения отдельных запросов увеличится.
- Минимальное время реакции – время, необходимое для обработки и выдачи первой строки. Этот критерий чаще применяется при работе в интерактивном режиме, но только для запросов, которые не содержат агрегирующих функций и не требуют сортировки данных результата.
- Минимальные затраты времени на обработку всех строк, к которым обращается данная команда. Этот критерий используется обычно при работе в пакетном режиме или в ситуации, когда невозможно выдавать результат по частям (при использовании агрегирующих функций и др.).

Стоимость плана выполнения запроса определяется на основании сведений о распределении данных в таблицах, к которым обращается команда, и связанными с ними кластерами и индексами. Эти сведения о распределении значений данных называются **статистикой**. Статистика может включать в себя:

– количество блоков данных, занимаемых таблицей;
– количество записей в таблице;
– минимальные и максимальные значения данных по столбцам;
– количество различных значений по столбцам и т.п.

Распределение значений в столбце может быть отражено с помощью гистограммы. Для этого всё множество значений столбца упорядочивается и разбивается на N интервалов (рис. 5.4).


Рис.5.4. Примеры равномерного (а) и неравномерного (б) распределения значений

Гистограмма помогает оценить объём данных, удовлетворяющих условию запроса. Например, на рис. 5.4,б представлено неравномерное распределение данных для некоторого столбца F. В словарь-справочник данных записываются полученные значения (1, 4, 4, 5, 6, 10, 11, 20, 35, 60, 100). При анализе запроса, например, с условием (F<5) система сможет по этой гистограмме определить, что через индекс придётся выбрать не менее 15% записей таблицы.

Существуют различные подходы к порядку сбора статистики. Некоторые СУБД постоянно собирают статистическую информацию, но это может уменьшить быстродействие системы. Другие позволяют осуществлять сбор статистики периодически, например, в период минимальной загрузки системы. Третьи предлагают администратору специальные средства для сбора статистики, которые запускаются интерактивно по его команде. В последнем случае в обязанность администратора (администратора данных или приложений) входит выбор таблиц для анализа и периодическое обновление статистики данных.

Гистограммы полезны только в тех случаях, когда они отражают актуальное распределение данных. Если распределение меняется при загрузке или модификации данных, гистограмму нужно обновлять.

Построение гистограммы бесполезно в следующих случаях:

- значения столбца распределены равномерно;
- столбец не используется в предикатах запросов;
- значения столбца уникальны и используются только в предикатах эквивалентности.

### 5.3.3. Примеры использования методов оптимизации запросов

Рассмотрим оптимизацию по синтаксису следующего запроса SQL:

```
SELECT empNo FROM emp
    WHERE eName = 'Tom' AND Sal > 2000;
```

Пусть таблица Emp имеет следующее правило целостности и индексы:

- столбец empNo определен как PRIMARY KEY, ему соответствует индекс PK_EMPNO;
- существует индекс ENAME_IND для столбца eName;
- существует индекс SAL_IND для столбца Sal.

Возможны следующие пути доступа:

- полный просмотр таблицы (ранг 15);
- доступ по одиночному индексу ENAME_IND. Этот путь становится доступным по условию eName = 'Tom' (ранг 9);
- доступ с помощью открытого интервала SAL>2000, используя индекс SAL_IND (ранг 11).

Индекс PK_EMPNO недоступен, т.к. в запросе не происходит обращения к полю empNo. Оптимизатор выберет доступ по индексу с рангом 9.

При выборе пути доступа в методе оптимизации по стоимости для оценки каждого плана выполнения оптимизатор использует статистику.

Теперь рассмотрим примеры оптимизации по стоимости.

1) Запрос, выбирающий сотрудников с номерами больше 7500:

```
SELECT * FROM Emp
    WHERE empNo < 7500;
```

Статистика для столбца empNo, в частности, включает значения HIGH_VALUE и LOW_VALUE (максимальное и минимальное значения). Если нет гистограммы, то оптимизатор предполагает, что значения равномерно распределены в интервале [LOW_VALUE, HIGH_VALUE], и может определить процент значений, попадающий в интервал до 7500. Доступ будет осуществляться по индексу, если этот процент невысок, например, не более 10, хотя конкретное пороговое значение зависит и от других параметров, например, количества записей в блоке.

2) Рассмотрим запрос, выбирающий название отделов (dname из таблицы Dept) и всех сотрудников с максимальной зарплатой в своем отделе (ename, sal из таблицы Emp):

```
select dname, ename, sal
  from dept d, emp e
  where d.deptno=e.deptno and
    e.sal=(select max(sal) from emp p
            where p.deptno=e.deptno);
```

Для таблиц есть индексы по первичным ключам (Dept.deptno и Emp.empno) и по внешнему ключу (Emp.deptno).

Этот запрос можно выполнить по разным планам, например:

1. Выбрать все записи из таблиц Emp и Dept, соединить их по условию d.deptno=e.deptno, затем для каждой полученной строки посчитать подзапрос (выбрать максимальную зарплату для данного отдела, обратившись к таблице Emp по индексу) и проверить второе условие.
2. Выбрать все записи из таблицы Emp, для каждой записи найти соответствие по условию d.deptno=e.deptno в таблице Dept через индекс по первичному ключу (на поле deptno), затем для каждой полученной строки посчитать подзапрос и проверить второе условие.
3. Выбрать все записи из таблицы Emp, каждую запись соединить по условию d.deptno=e.deptno с таблицей Dept через индекс по первичному ключу (на поле deptno). Предварительно посчитать подзапрос, добавив в него

условие группировки по номерам отделов. Затем для каждой строки соединения проверить второе условие.

Расчёты здесь достаточно громоздкие, поэтому мы их приводить не будем, а ограничимся качественным анализом предложенных планов. Первый план от второго отличается способом соединения исходных таблиц. Но декартово произведение (т.е. полный просмотр одной таблицы для каждой строки другой таблицы) практически всегда выполняется дольше, чем соединение через индекс, когда не надо просматривать все отношение для поиска соответствия. Поэтому второй план предпочтительнее первого. (За исключением случая маленьких таблиц, когда их размер сопоставим с размером индекса).

Третий план от второго отличается предварительным вычислением подзапроса. Это позволит один раз построить агрегированные значения (по количеству отделов). А во втором запросе агрегированные значения будут строиться для каждого сотрудника. Т.е. чем больше сотрудников в одном отделе, тем больше выигрыш по времени в третьем запросе.

Таким образом, оптимизатор вероятнее всего выберет третий план как самый оптимальный с точки зрения времени выполнения запроса.

## 5.4. Настройка приложений

В настройку приложений входит:

- определение потребности в кластеризации и хешировании данных;
- создание индексов;
- настройка команд SQL;
- выбор метода оптимизации SQL-запросов;
- использование средств сбора статистики.

Первые два пункта мы уже обсуждали (см. разделы 4.6.2, 4.6.3., 4.6.4).

Настройка команд SQL, которые используются в приложениях к БД, – это один из основных способов повышения производительности системы. Эта настройка должна производиться каждым разработчиком.

Для оптимизации приложений необходимо иметь представление о порядке и механизмах реализации запросов в СУБД. Основные информационные потоки между пользователями, оперативной памятью и базой данных приведены на рис. 5.5. В ОП хранятся все результаты ранее выполненных запросов до тех пор, пока не потребуется память для записи результатов следующих запросов.

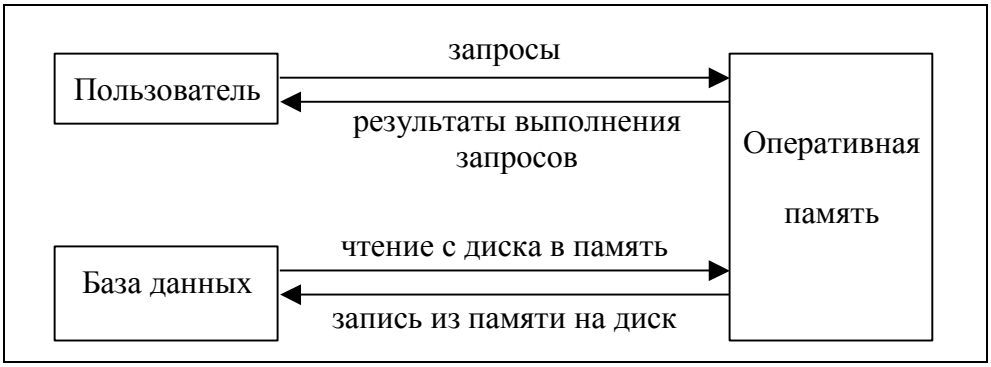


Рис.5.5. Информационные потоки в БД

Подготовленные к исполнению SQL-операторы обычно помещаются в разделяемую SQL-область. Перед началом выполнения запроса система проверяет, есть ли в этой области аналогичный запрос: если есть, то он отправляется на выполнение, минуя стадию предварительной обработки (компиляции). Составляя запросы таким образом, чтобы они совпадали с уже имеющимися в SQL-области, можно исключить предобработку запроса, что является важным моментом оптимизации приложений.

Приведем основные рекомендации по написанию запросов, удобных для оптимизатора и эффективных при выполнении.

1. Если запрос содержит несколько условий, то они должны располагаться в порядке уменьшения селективности.
   Например, для получения списка сотрудниц 2-го отдела при условии, что во втором отделе сотрудников около 5% от общего числа сотрудников, а женщин на предприятии – примерно половина, запрос должен выглядеть так:
   ```
   select * from emp
       where deptNo=2 and sex='ж';
   ```
2. В запросе, который реализует соединение двух и более таблиц, эти таблицы должны стоять в списке в порядке уменьшения количества записей в них.
   Например, список пациентов по отделениям.
   ```
   select * from patients p, depart d
       where p.deptNo=d.id;
   ```
3. Если запрос содержит условие типа (field LIKE '%…') или (field LIKE '_…'), то необходимо дополнять это условие так, чтобы система могла воспользоваться индексом по полю field (если он существует).
   Например, список всех хирургов лучше записать так:
   ```
   select * from doctors
       where special>'A' and special like '%хирург%';
   ```
4. Если запрос содержит условие для проиндексированного поля маленькой таблицы, которая может быть считана за одно обращение к памяти, то запрос нужно сформулировать так, чтобы система игнорировала индекс.
   Например, запрос на выборку названия отделения №3:
   ```
   select name from depart
       where id*1=3;
   ```
   id – это первичный ключ, по нему есть индекс. Но при доступе через индекс потребуется минимум два обращения к диску. Включение индексированного поля в выражение (id*1 вместо id) подавляет использование индекса.
5. Используйте UNION ALL вместо UNION DISTINCT, если уверены в том, что в объединяемых отношениях отсутствуют одинаковые записи (или наличие одинаковых записей некритично). Дело в том, что UNION DISTINCT вычисляется путем сортировки, которая может занять много времени.
6. Следует использовать IN вместо EXISTS, если EXISTS не оптимизируется.
   Например, список сотрудников, у которых есть дети:
   ```
   select * from emp
       where empNo in (select empNo from children);
   ```

Но для подзапросов, выдающих большой список, более оптимальным может оказаться вариант запроса с соединением:

```
select distinct e.* from emp e, children c
    where c.empNo=e.empNo;
```

7. Если оптимизатор плохо оптимизирует операцию "или" (OR), то нужно заменять её операцией UNION при наличии индекса. Убедиться в "плохой оптимизации" можно так: выполнить запрос по условию (field=X) и запрос с условием ((field=X) OR (filed=Y)) на большой таблице. Если второй запрос выполняется намного дольше, чем первый, то OR не оптимизируется.
   Например, список "Пациенты палат №3,8" без индекса должен быть таким:
   ```
   select * from patients
       where room=3 or room=8;
   ```
   а если индекс есть, то таким:
   ```
   select * from patients
       where room=3
   union all
   select * from patients
       where room=8;
   ```
8. Условие "не равно" ('<>') также подавляет использование индекса. Поэтому, если значения индексированного столбца распределены неравномерно, следует заменять его комбинацией условий '<' or '>' и, с учетом предыдущего правила, реализовывать это с помощью UNION.
   Например, список сотрудников всех отделов (10% от общего числа), кроме сотрудников центрального офиса (отдел №3) будет выглядеть так:
   ```
   select * from emp
       where deptNo<3
   union all
   select * from emp
       where or deptNo>3;
   ```

При настройке команд SQL важно помнить, что, настраивая одну из них, можно оказать влияние (и не всегда позитивное) на другие команды. Поэтому во время настройки необходимо периодически осуществлять регрессионное тестирование, т.е. повторный запуск уже протестированных команд для оценки времени их выполнения.

# 6. МНОГОПОЛЬЗОВАТЕЛЬСКИЙ ДОСТУП К ДАННЫМ

## 6.1. Организация многопользовательского доступа к данным

Параллельный (многопользовательский) доступ к данным подразумевает одновременное выполнение двух и более запросов к одним и тем же объектам данных (таблицам, блокам и т.п.). Для организации одновременного доступа не обязательно наличие многопроцессорной системы. На однопроцессорной ЭВМ запросы выполняются не одновременно, а параллельно. Обычно для каждого запроса выделяется некоторое количество процессорного времени (квант вре-

мени), по истечении которого выполнение запроса приостанавливается, он ставится в очередь запросов, а на выполнение запускается следующий (по очереди) запрос. Таким образом, процессорное время делится между запросами, и создаётся иллюзия, что запросы выполняются одновременно.

При параллельном доступе к данным проблемы возникают в том случае, если доступ подразумевает внесение изменений. Для того чтобы исключить нарушения логической целостности данных при многопользовательском доступе, используется механизм транзакций.

## 6.2. Механизм транзакций

**Транзакция** – это последовательность операторов обработки данных, которая рассматривается как логически неделимая единица работы с БД.

Транзакция обладает следующими свойствами:

1. Логическая неделимость (**атомарность,** Atomicity) означает, что выполняются либо все операции, входящие в транзакцию, либо ни одной. (Логическая неделимость не подразумевает физической неделимости).
   Система гарантирует невозможность фиксации части изменений, произведённых транзакцией. До тех пор, пока транзакция не зафиксирована, её можно "откатить", т.е. отменить все сделанные операторами из транзакции изменения в БД. Успешное выполнение транзакции означает, что все операторы транзакции проанализированы, интерпретированы как правильные и безошибочно исполнены.
2. **Согласованность** (Consistency): транзакция начинается на согласованном множестве данных, и после её завершения множество данных согласовано.
3. **Изолированность** (Isolation), т.е. отсутствие влияния транзакций друг на друга. (На самом деле это влияние существует и регламентируется стандартом: см. раздел 6.4. "Уровни изоляции транзакций").
4. **Устойчивость** (Durability): результаты зафиксированной транзакции не могут быть потеряны. Возврат БД в предыдущее состояние может быть достигнут только путём запуска компенсирующей транзакции.

Транзакции, удовлетворяющие этим свойствам, называют ACID-транзакциями.

Для управления транзакциями в системах, поддерживающих механизм транзакций и язык SQL, используются следующие операторы:

– фиксация транзакции: COMMIT [WORK];
– откат транзакции: ROLLBACK [WORK];
– точка сохранения: SAVEPOINT <имя_точки_сохранения>;

(Ключевое слово WORK необязательно). Для фиксации или отката транзакции система создаёт неявные точки фиксации и отката (рис. 6.1).

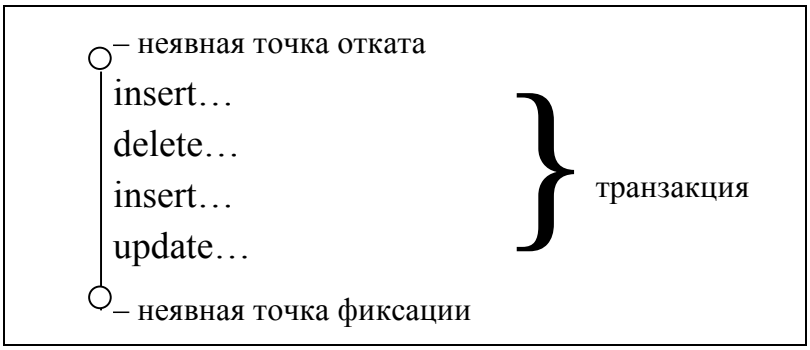


Рис.6.1. Неявные точки фиксации и отката транзакции

По команде rollback система откатит транзакцию на начало (на неявную точку отката), а по команде commit – зафиксирует всё до неявной точки фиксации, которая соответствует последней завершенной команде в транзакции. Если в транзакции из нескольких команд во время выполнения очередной команды возникнет ошибка, то система откатит только эту ошибочную команду, т.е. отменит её результаты и сохранит прежнюю неявную точку фиксации.

Предложение SAVEPOINT запоминает промежуточную "текущую копию" состояния базы данных для того, чтобы при необходимости можно было вернуться к состоянию БД в точке сохранения: откатить работу от текущего момента до точки сохранения (rollback to <имя_точки>) или зафиксировать работу от начала транзакции до точки сохранения (commit to <имя_точки>).

Для сохранения сведения о транзакциях СУБД ведёт журнал транзакций. **Журнал транзакций** – это часть БД, в которую поступают сведения обо всех изменениях всех объектов БД. Журнал недоступен пользователям СУБД и поддерживается особо тщательно (иногда ведутся две копии журнала, хранимые на разных физических носителях). Форма записи в журнал изменений зависит от СУБД. Но обычно там фиксируется следующее:

- номер транзакции (номера присваиваются автоматически по возрастанию);
- состояние транзакции (завершена, не завершена, находится в состоянии ожидания);
- точки сохранения;
- команды, составляющие транзакцию, и проч.

Начало транзакции соответствует появлению первого исполняемого SQL-оператора. При этом в журнале появляется запись об этой транзакции.

По стандарту ANSI/ISO транзакция завершается при наступлении одного из следующих событий:

- Поступила команда COMMIT (результаты транзакции фиксируются).
- Поступила команда ROLLBACK (результаты транзакции откатываются).
- Успешно завершена программа, в рамках которой выполнялась транзакция. В этом случае транзакция фиксируется автоматически.
- Программа, выполняющая транзакцию, завершена аварийно. При этом транзакция автоматически откатывается.

**Примечание: в**о многих СУБД реализованы расширенные модели транзакций, в которых существуют дополнительные ситуации фиксации транзакций.

Для организации отката система во время выполнения транзакции производит запись в сегменты отката всех внесённых изменений. **Сегмент отката** – это специальная область памяти на диске, в которую записывается информация обо всех текущих (незавершенных) изменениях. Обычно записывается "старое" и "новое" содержимое записи, чтобы можно было восстановить прежнее состояние БД при откате транзакции или одной из составляющих её команд.

Все изменения данных выполняются в оперативной памяти в буфере данных, затем фиксируются в журнале транзакций и в сегменте отката, и периодически (при выполнении контрольной точки) переписываются на диск. Процесс формирования **контрольной точки** (КТ) заключается в синхронизации данных, находящихся на диске (т.е. во вторичной памяти) с теми данными, которые находятся в ОП: все модифицированные данные из ОП переписываются во вторичную память. В разных системах процесс формирования контрольной точки запускается по-разному. Например, в Oracle контрольная точка формируется в следующих случаях:

– при поступлении команды *commit,*
– при переполнении буфера данных,
– в момент заполнения очередного файла журнала транзакций,
– через три секунды со времени последней записи на диск.

Внесение изменений в журнал транзакций всегда носит опережающий характер по отношению к записи изменений в основную часть БД (протокол WAL – Write Ahead Log). Эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала транзакций раньше, чем изменённый объект попадёт во внешнюю память основной части БД. Если СУБД корректно соблюдает протокол WAL, то с помощью журнала транзакций можно решить все проблемы восстановления БД после сбоя, если сбой препятствуют дальнейшему функционированию системы (например, после сбоя приложения или фонового процесса СУБД).

Таким образом, при использовании протокола WAL измененные данные почти сразу попадают в базу данных, ещё по поступления команды *commit*. Поэтому фиксация транзакции чаще всего заключается в следующем:

1. Изменения, внесённые транзакцией, делаются постоянными.
2. Уничтожаются все точки сохранения для данной транзакции.
3. Если выполнение транзакций осуществляется с помощью блокировок, то освобождаются объекты, заблокированные транзакцией.
4. В журнале транзакций транзакция помечается как завершенная, уничтожаются системные записи о транзакции в оперативной памяти.

А откат транзакции можно произвести, только считав из сегмента отката прежние значения данных и переписав их обратно в БД. Поэтому откат транзакции практически всегда занимает больше времени, чем фиксация.

## 6.3. Взаимовлияние транзакций

Транзакции в многопользовательской БД должны быть изолированы друг от друга, т.е. в идеале каждая из них должна выполняться так, как будто выполняется только она одна. В реальности транзакции выполняются одновременно и

могут влиять на результаты друг друга, если они обращаются к одному и тому же набору данных и хотя бы одна из транзакций изменяет данные.

Взаимовлияние транзакций может проявляться в виде:

- потери изменений;
- чернового чтения;
- неповторяемого чтения;
- фантомов.

**Потеря изменений** могла бы произойти при одновременном обновлении двумя и более транзакциями одного и того же набора данных. Транзакция, закончившаяся последней, перезаписала бы результаты изменений, внесённых предыдущими транзакциями, и они были бы потеряны.

Представим, что одновременно начали выполняться две транзакции:

транзакция 1 – UPDATE *СОТРУДНИКИ* SET *Оклад* = 9200
WHERE ***Номер*** = 1123

транзакция 2 – UPDATE *СОТРУДНИКИ*
SET *Должность* = "старший экономист", *ЕТС* = 14
WHERE ***Номер*** = 1123

Обе транзакции считали одну и ту же запись (1123, "Рудин В.П.", "экономист", 12, 8300) и внесли каждая свои изменения: в бухгалтерии изменили оклад (транзакция 1), в отделе кадров – должность и ставку по ЕТС (транзакция 2). Результаты транзакции 1 будут потеряны (рис. 6.2).

Отношение "Сотрудники"

| Номер | ФИО | Должность | ЕТС | Оклад | |
|---|---|---|---|---|---|
| 1123 | Рудин В.П. | экономист | 13 | 8300 | |
| 1123 | Рудин В.П. | экономист | 13 | **9200** | Транзакция 1 |
| 1123 | Рудин В.П. | **старший экономист** | **14** | 8300 | Транзакция 2 |

Рис.6.2. Недопустимое взаимовлияние транзакций: потеря изменений

**СУБД не допускает такого взаимовлияния транзакций, при котором возможна потеря изменений.**

Ситуация **чернового чтения** возникает, когда транзакция считывает изменения, вносимые другой (незавершенной) транзакцией. Если эта вторая транзакция не будет зафиксирована, то данные, полученные в результате чернового чтения, будут некорректными. Транзакции, осуществляющие черновое чтение, могут использоваться только при невысоких требованиях к согласованности данных: например, если транзакция считает статистические показатели, когда отклонения отдельных значений данных слабо влияют на результат.

При повторяемом чтении один и тот же запрос, повторно выполняемый одной транзакцией, возвращает один и тот же набор данных (т.е. игнорирует изменения, вносимые другими завершёнными и незавершёнными транзакциями). **Неповторяемое чтение** является противоположностью повторяемого, т.е.

транзакция "видит" изменения, внесённые другими (завершёнными!) транзакциями. Следствием этого может быть несогласованность результатов запроса, когда часть данных запроса соответствует состоянию БД до внесения изменений, а часть – состоянию БД после внесения и фиксации изменений.

**Фантомы** – это особый тип неповторяемого чтения. Возникновение фантомов может происходить в ситуации, когда одна и та же транзакция сначала производит обновление набора данных, а затем считывание этого же набора. Если считывание данных начинается раньше, чем закончится их обновление, то в результате чтения можно получить несогласованный (не обновлённый или частично обновлённый) набор данных. При последующих запросах это явление пропадает, т.к. на самом деле запрошенные данные после завершения обновления будут согласованными в соответствие со свойствами транзакции.

## 6.4. Уровни изоляции транзакций

Стандарт ANSI/ISO для SQL устанавливает различные уровни изоляции для операций, выполняемых над базами данных, которые работают в многопользовательском режиме. **Уровень изоляции** определяет, может ли читающая транзакция считывать ("видеть") результаты работы других одновременно выполняемых завершённых и/или незавершённых пишущих транзакций (табл. 7.1). Использование этих уровней изоляции обеспечивает предсказуемость работы приложений.

Таблица 7.1. Уровни изоляции по стандарту ANSI / ISO

| Уровень изоляции | Черновое чтение | Неповторяемое чтение | Фантомы |
|---|---|---|---|
| Read Uncommited – чтение незавершённых транзакций | да | да | да |
| Read Commited – чтение завершённых транзакций | нет | да | да |
| Repeatable Read – повторяемое чтение | нет | нет | да |
| Serializable – последовательное чтение | нет | нет | нет |

По умолчанию обычно используется уровень Read Commited.

Уровень изоляции позволяет транзакциям в большей или меньшей степени влиять друг на друга: при повышении уровня изоляции повышается согласованность данных, но снижается степень параллельности работы и, следовательно, производительность системы.

Наиболее распространённый механизм разграничения транзакций – использование блокировок.

## 6.5. Блокировки

**Блокировка** – это временное ограничение доступа к данным, участвующим в транзакции, со стороны других транзакций.

Различают следующие типы блокировок:

- по степени доступности данных: разделяемые и исключающие;

- по множеству блокируемых данных: строчные, страничные, табличные;
- по способу установки: автоматические и явные.

**Строчные, страничные** и **табличные блокировки** накладываются соответственно на строку таблицы, страницу (блок) памяти и на всю таблицу целиком. Табличная блокировка приводит к неоправданным задержкам исполнения запросов и сводит на нет параллельность работы. Другие виды блокировки увеличивают параллелизм работы, но требуют накладных расходов на поддержание блокировок: наложение и снятие блокировок требует времени, а для хранения информации о наложенной блокировке нужна дополнительная память (для каждой записи или блока данных).

**Разделяемая блокировка**, установленная на определённый ресурс, предоставляет транзакциям право коллективного доступа к этому ресурсу. Обычно этот вид блокировок используется для того, чтобы запретить другим транзакциям производить необратимые изменения. Например, если на таблицу целиком наложена разделяемая блокировка, то ни одна транзакция не сможет удалить эту таблицу или изменить её структуру до тех пор, пока эта блокировка не будет снята. (При выполнении запросов на чтение обычно накладывается разделяемая блокировка на таблицу.)

**Исключающая блокировка** предоставляет право на монопольный доступ к ресурсу. Монопольная блокировка таблицы накладывается, например, в случае выполнения операции ALTER, т.е. изменения структуры таблицы. На отдельные записи (блоки) монопольная блокировка накладывается тогда, когда эти записи (блоки) подвергаются модификации данных.

Блокировка может быть **автоматической** и **явной**. Если запускается новая транзакция, СУБД сначала проверяет, не заблокирована ли другой транзакцией строка, требуемая этой транзакции: если нет, то строка автоматически блокируется и выполняется операция над данными; если строка заблокирована, транзакция ожидает снятия блокировки. Явная блокировка, накладываемая командой LOCK языка SQL, обычно используется тогда, когда транзакция затрагивает существенную часть отношения. Это позволяет не тратить время на построчную блокировку таблицы. Кроме того, при большом количестве построчных блокировок транзакция может не завершиться (из-за возникновения взаимных блокировок, например), и тогда все сделанные изменения придётся откатить, что снизит производительность системы.

Блокировки могут стать причиной бесконечного ожидания и тупиковых ситуаций. Бесконечное ожидание возможно в том случае, если не соблюдается очерёдность обслуживания транзакций и транзакция, поступившая раньше других, всё время отодвигается в конец очереди. Решение этой проблемы основывается на выполнении правила FIFO: "первый пришел – первый ушел".

**Тупиковые ситуации (deadlocks)** возникают при взаимных блокировках транзакций, которые выполняются на пересекающихся множествах данных (рис. 6.3). Здесь приведён пример взаимной блокировки трех транзакций $T_i$ на отношениях $R_j$. Транзакция $T_1$ заблокировала данные $B_1$ в отношении $R_1$ и ждёт освобождения данных $B_2$ в отношении $R_2$, которые заблокированы транзакцией

$T_2$, ожидающей освобождения данных $B_3$ в отношении $R_3$, заблокированных транзакцией $T_3$, которая не может продолжить выполнение из-за транзакции $T_1$.

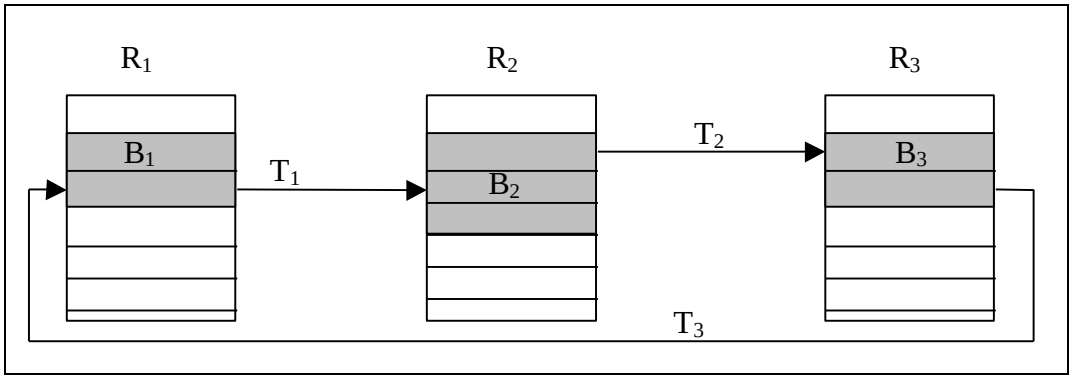


Рис.6.3. Взаимная блокировка трех транзакций

Существует много стратегий разрешения проблемы взаимной блокировки, в частности:

1. Транзакция запрашивает сразу все требуемые блокировки. Такой метод снижает степень параллелизма в работе системы. Кроме того, он не может применяться в тех случаях, когда заранее неизвестно, какие данные потребуются, например, если выборка данных из одной таблицы осуществляется на основании данных из другой таблицы, которые выбираются в том же запросе.
2. СУБД отслеживает возникающие тупики и отменяет одну из транзакций. Этот метод требует дополнительных накладных расходов.
3. Вводится **таймаут (**time-out) – максимальное время, в течение которого транзакция может находиться в состоянии ожидания. Если транзакция находится в состоянии ожидания дольше таймаута, считается, что она находится в состоянии тупика, и СУБД инициирует её откат с последующим рестартом через случайный промежуток времени.

# 7. ЗАЩИТА ДАННЫХ В БАЗАХ ДАННЫХ

**Защита данных** – это организационные, программные и технические методы и средства, направленные на удовлетворение ограничений, установленных для типов данных или экземпляров типов данных в СОД [6].

Защита данных включает предупреждение случайного или несанкционированного доступа к данным, их изменения или разрушения со стороны пользователей или при сбоях аппаратуры. Реализация защиты включает:

- контроль достоверности данных с помощью ограничений целостности;
- обеспечение безопасности данных;
- обеспечение секретности данных.

## 7.1. Обеспечение целостности данных

Обеспечение целостности данных касается защиты от внесения непреднамеренных ошибок и предотвращения последних. Оно достигается за счёт

проверки ограничений целостности – условий, которым должны удовлетворять значения данных.

Рассмотрим различные типы ограничений целостности:

1. Уникальность значения первичного ключа (PRIMARY KEY).
2. Уникальность значения комбинации ключевых полей:
   UNIQUE(A),
   где A – атрибут или комбинация атрибутов.

(1,2 – структурные ограничения.)

3. Задание диапазона значений атрибута:
   field BETWEEN min_value AND max_value
4. Задание взаимоотношений между значениями атрибутов:
   CHECK (field1 @ filed2),
   где @ – оператор отношения (например, знак ">").
5. Задание списка возможных значений (констант):
   CHECK (field1 IN (value1, value2,…, valueN))
6. Определение формата атрибута (например, даты или числа), например.
   CHECK (filed LIKE '_ _ _-_ _-_ _') – формат телефонного номера.
7. Определение домена атрибута на основе значений другого атрибута: множество значений некоторого атрибута отношения является подмножеством значений другого атрибута этого или другого отношения (внешний ключ).

(3.-7. – ограничения на значения данных.)

8. Ограничения на обновление данных (например, каждое следующее значение атрибута должно быть больше предыдущего). В языке SQL напрямую не реализуется, требует использования специфических возможностей СУБД.
9. Ограничения на параллельное выполнение операций (механизм транзакций) и проверка ограничений целостности после окончания внесения взаимосвязанных изменений.

Реализация ограничений целостности возлагается на СУБД или выполняется с помощью специальных программных модулей.

**Если какое-либо ограничение целостности может быть включено в структуру БД (на языке DDL), то его надо реализовать именно так.**

СУБД проверяет выполнение ограничений целостности при каждой операции модификации данных, если эта операция может нарушить целостность БД. Таким образом, при наличии ограничений целостности, включенных в схему базы данных, они проверяются автоматически и нельзя внести в БД ошибочные данные. Если же перенести проверку ограничений целостности в программу, то гарантировать их соблюдение нельзя. Программа, во-первых, может содержать ошибки, во-вторых, её можно "обойти", обратившись к БД напрямую с помощью SQL.

СУБД проводит проверку выполнения ограничений целостности для команд DDL до выполнения команды, а для команд DML либо сразу после выполнения команды DML, либо после выполнения всей транзакции. (По стан-

дарту ISO этим можно управлять; по умолчанию проверка проводится после каждой операции DML).

## 7.2. Обеспечение физической защиты данных

Под функцией безопасности данных подразумевается предотвращение разрушения или искажения данных при случайном доступе или в результате аппаратного сбоя. Обеспечение безопасности является внутренней задачей БД, поскольку связано с её нормальным функционированием, и решается на уровне СУБД. Цель восстановления БД после сбоя – обеспечить, чтобы результаты всех подтверждённых транзакций были отражены в восстановленной базе данных, и вернуться к нормальному продолжению работы как можно быстрее, в то же время изолируя пользователей от проблем, вызванных сбоем.

Наиболее типичными сбоями являются следующие:

1. Пользовательские ошибки.
   Ошибки пользователей могут потребовать восстановления базы данных в состояние на момент возникновения ошибки. Например, пользователь может случайно удалить таблицу.
2. Сбой предложения.
   Сбой происходит при логической ошибке предложения во время его обработки (например, предложение нарушает ограничение целостности таблицы). Когда возникает сбой предложения, результаты этого предложения (если они есть) должны автоматически отменяться СУБД, а управление – возвращаться пользователю.
3. Сбой процесса.
   Это ошибка в пользовательском процессе, обращающемся к БД, например, аварийное разъединение или прекращение процесса. Сбившийся процесс пользователя не может продолжать работу, тогда как СУБД и процессы других пользователей могут. Система должна откатить неподтверждённые транзакции сбившегося пользовательского процесса и освободить все ресурсы, занятые этим процессом.
4. Сбой экземпляра базы данных (сервера).
   Этот сбой происходит при возникновении проблемы, препятствующей продолжению работы сервера. Сбой может быть вызван аппаратной проблемой, такой как отказ питания, или программной проблемой, такой как сбой операционной системы. Восстановление после такого сбоя может потребовать перезагрузки БД с откатом всех незавершённых транзакций.
5. Сбой носителя (диска).
   Эта ошибка может возникнуть при попытке записи или чтения файла, требуемого для работы базы данных. Типичным примером является отказ дисковой головки, который приводит к потере всех файлов на данном устройстве. Этот тип сбоя может касаться различных типов файлов, поддерживаемых СУБД. Кроме того, поскольку сервер не может продолжать работу, данные из буферов оперативной памяти не могут быть записаны в файлы данных.

Таким образом, после некоторых сбоев система может восстановить БД автоматически, а ошибка пользователя или сбой диска требуют участия в восстановлении человека (обычно, администратора).

В качестве средств физической защиты данных чаще всего применяются резервное копирование и журналы транзакций.

**Резервное копирование** означает периодическое сохранение файлов БД на внешнем запоминающем устройстве. Оно выполняется тогда, когда состояние файлов БД является непротиворечивым. Резервная копия не должна создаваться на том же диске, на котором находится сама БД, т.к. при аварии диска базу невозможно будет восстановить. Периодичность резервного копирования определяется администратором системы и зависит от многих факторов: объём БД, интенсивность запросов к БД, интенсивность обновления данных и др. В случае сбоя (или аварии диска) БД восстанавливается на основе последней копии. Все изменения, произведённые в данных после последнего копирования, утрачиваются; но при наличии архива журнала транзакций их можно выполнить ещё раз, обеспечив полное восстановление БД.

Общая стратегия восстановления БД заключается в переносе на рабочий диск резервной копии БД (или той её части, которая была повреждена), и повторном проведении всех транзакций, зафиксированных после выполнения данный резервной копии и до момента возникновения сбоя. В зависимости от типа повреждения данных восстановление может проводиться частично или полностью автоматизировано.

**Полная резервная копия** включает всю базу данных (все файлы БД, в том числе, вспомогательные, состав которых зависит от СУБД). **Частичная резервная копия** включает часть БД, определённую пользователем. В резервную копию включаются только те блоки памяти, которые реально содержат данные (т.е. пустые блоки, выделенные под объекты БД, в резервную копию не входят).

Резервное копирование может быть **полным** и **инкрементным**. Полная копия (level 0 backup) включает всю базу данных. Инкрементная копия состоит только из тех блоков (страниц памяти), которые изменились со времени последнего резервного копирования. Создание инкрементной копии происходит быстрее, чем полной, но оно возможно только после проведения полного резервного копирования.

Обычно технология проведения резервного копирования такова:

- раз в неделю (день, месяц) осуществляется полное копирование;
- раз в день (час, неделю) проводится обновление копии (инкрементное копирование).

Для оптимизации регистрации изменений некоторые СУБД (в том числе, например, Oracle) может записывать в журнал информацию о незавершённых транзакциях, предвидя из завершение. Более того, не дожидаясь подтверждения транзакции, СУБД переписывает в БД модифицированные блоки (при формировании контрольной точки). Поэтому в каждый момент времени в журнале транзакций и в БД может находиться небольшое число записей, модифицированных незавершёнными транзакциями. Эти записи помечаются соответствующим образом и не участвуют в восстановлении БД. С другой стороны, т.к. из-

менения сначала попадают в журнал транзакций и только потом в файл базы данных, в любой момент времени БД может не содержать блоков данных, модифицированных подтверждёнными транзакциями. Поэтому после сбоя могут возникнуть две потенциальных ситуации:

- Блоки, содержащие подтверждённые модификации, не были записаны в файлы данных, так что эти изменения отражены лишь в журнале транзакций. Следовательно, журнал транзакций содержит подтверждённые данные, которые должны быть переписаны в файлы данных.
- Журнал транзакций и блоки данных, возможно, содержат изменения, которые не были подтверждены. Изменения неподтверждённых транзакций во время восстановления должны быть удалены из файлов данных.

Для того чтобы разрешить эту ситуацию, СУБД применяет два этапа при восстановлении после сбоев: прокрутку вперед и прокрутку назад.

1. Прокрутка вперед.
   Это первый этап восстановления. Он заключается в применении к файлам данных всех изменений, зарегистрированных в журнале транзакций. Прокрутка вперед обрабатывает столько файлов журнала транзакций, сколько необходимо, чтобы привести БД в состояние на требуемый момент времени. Если вся необходимая информация повторения находится в активном журнале транзакций, СУБД выполняет этот шаг восстановления автоматически при запуске базы данных. После прокрутки вперед файлы данных содержат все как подтверждённые, так и неподтверждённые изменения, которые были зарегистрированы в журнале транзакций.
2. Прокрутка назад.
   Этот этап заключается в отмене всех изменений, которые не были подтверждены. Для этого используются сегменты отката, информация из которых позволяет идентифицировать и отменить те транзакции, которые не были подтверждены, хотя и попали в журнал.

## 7.3. Защита от несанкционированного доступа

Под функцией секретности данных понимается защита данных от преднамеренного искажения и/или доступа пользователей или посторонних лиц. Для этого вся информация делится на общедоступные и конфиденциальные данные, доступ к которым разрешен только для отдельных групп лиц. Решение этого вопроса относится к компетенции юридических органов или администрации предприятия, для которого создаётся БД, и является внешней функцией по отношению к БД.

Рассмотрим техническую сторону обеспечения защиты данных в БД. Будем считать, что СУБД не должна разрешать пользователю выполнение какой-либо операции над данными, если он не получил на это права. Санкционирование доступа к данным осуществляется администратором БД. В обязанности администратора БД входит:

- классификация данных в соответствии с их использованием;
- определение прав доступа отдельных групп пользователей к отдельным группам данных;

- организация системы контроля доступа к данным;
- тестирование вновь создаваемых средств защиты данных;
- периодическое проведение проверок правильности работы системы защиты, исследование и предотвращение сбоев в её работе.

Для каждого пользователя система поддерживает паспорт пользователя, содержащий его идентификатор, имя процедуры подтверждения подлинности и перечень разрешённых операций. Подтверждение подлинности заключается в доказательстве того, что пользователь является именно тем человеком, за которого себя выдаёт. Чаще всего подтверждение подлинности выполняется путём парольной идентификации. Перечень операций обычно определяется той группой, к которой принадлежит пользователь. В реальных системах иногда предусматривается возможность очень ограниченного доступа постороннего человека к данным (доступ типа "гость"), не требующая идентификации.

**Парольная идентификация** заключается в присвоении каждому пользователю двух параметров: имени (login) и пароля (password). При входе в систему она запрашивает у пользователя его имя, а для подтверждения того, что это имя ввел его владелец, система запрашивает пароль. Имя выдаётся пользователю при регистрации администратором, пароль пользователь устанавливает сам.

При задании пароля желательно соблюдать следующие требования:

- длина пароля должна быть не менее 6-и символов;
- пароль должен содержать комбинацию букв и цифр или специальных знаков, пароль не может содержать пробелы;
- пароли должны часто меняться.

Для обеспечения выполнения этих требований обычно применяются специальные программы.

В общем случае вопрос о доступе к БД разрешает специальная привилегированная программа на основании паспорта пользователя, набора прав и конкретных значений данных в базе.

Управление доступом к данным осуществляется через СУБД, которая и обеспечивает защиту данных. Но такие данные вне СУБД становятся общедоступны. Если известен формат БД, можно осуществить к ней доступ с помощью другой программы (СУБД), и никакие ограничения при этом не помешают.

Для таких случаев предусмотрено кодирование данных. Используются различные методы кодирования: перекомпоновка символов в кортеже, замена одних символов (групп символов) другими символами (группами символов) и т.д. Кодирование может быть применено не ко всему кортежу, а только к ключевым полям. Декодирование производится непосредственно в процессе обработки, что, естественно, увеличивает время доступа к данным. Поэтому к кодированию прибегают только в случае исключительной важности конфиденциальности данных.

Предоставление прав доступа (привилегий) в системах, поддерживающих язык SQL, осуществляется с помощью двух команд:

1. GRANT – предоставление одной или нескольких привилегий пользователю (или группе пользователей).

2. REVOKE – отмена привилегий.

Синтаксис этих команд зависит от СУБД.

Для того чтобы упростить процесс управления доступом, многие СУБД предоставляют возможность объединять пользователей в группы или определять роли. *Роль* – это совокупность привилегий, предоставляемых пользователю и/или другим ролям. Такой подход позволяет предоставить конкретному пользователю определённую роль или соотнести его определённой группе пользователей, обладающей набором прав в соответствии с задачами, которые на неё возложены.

## 8. ПЕРСПЕКТИВЫ РАЗВИТИЯ ТЕХНОЛОГИИ БАЗ ДАННЫХ

Вот уже более 30-и лет базы данных являются одной из одной из наиболее широко востребованных информационных технологий. Некоторые авторы утверждают [1], что появление баз данных стало самым важным достижением в области программного обеспечения. Системы баз данных коренным образом изменили работу многих организаций, и практически нет такой области деятельности, которую они не затронули. Ежегодный рост объёмов продаж СУБД и вспомогательного программного обеспечения с 1995 г. составляет более 20%.

К числу наиболее важных и перспективных направлений развития БД следует отнести следующие:

1. **Хранилища данных и OLAP-обработка**. Хранилище данных (ХД) – это предметно-ориентированный, интегрированный, привязанный ко времени и неизменяемый набор данных, предназначенный для поддержки принятия решений. ХД позволяют сохранять исторические данные с целью анализа и прогнозирования развития ситуаций. При правильном проектировании ХД даёт высокую отдачу за счёт более качественного управления работой организации (предприятия). Данные в ХД обрабатываются с помощью OLAP (online analytical processing) – инструментов оперативной аналитической обработки данных. OLAP позволяет быстро производить расчёты над огромными объёмами данных, в том числе, с целью выявления динамики изменения различных параметров (параметры задаются аналитиком).
2. **Работа с неточными данными**. Информация в базах данных часто содержит ошибки или является неполной. Результаты запроса по такой БД могут сильно отличаться от реального положения дел. Процессор запросов, работающий с вероятностями, коэффициентами доверия, коэффициентами полноты и т.д. позволил бы учитывать степень достоверности данных при принятии решений на основе этих данных.

3. **Новые пользовательские интерфейсы.** Это одно из наиболее актуальных направлений современных информационных технологий. Конечные пользователи не знают язык запросов (SQL), и для получения информации из БД вынуждены пользоваться интерфейсами, которые для них создают программисты. В приложения обычно включают некоторый набор готовых запросов и возможность сформулировать произвольный запрос с помощью некоего

конструктора. Но для того, чтобы воспользоваться конструктором, пользователь должен знать структуру базы данных и хорошо разбираться в преложенном ему формализме ПО.

Наиболее естественным видом является запрос к БД, сформулированный на естественном языке (ЕЯ). Но для таких запросов характерны неточности и неоднозначность. Решение этой задачи невозможно без использования знаний о предметной области и о структуре языка.

Одним из вариантов решения этой проблемы являются онтологии. Под онтологией понимается определённым образом формализованная система знаний о предметной области, описывающая, классифицирующая и увязывающая между собой понятия этой ПО. Интеграция онтологий и баз данных позволит пользователям задавать запросы в собственной терминологии с использованием ограниченного естественного языка. Это упростит создание и сопровождение приложений и повысит эффективность использования БД.

4. Проблемы **оптимизации запросов**. Помимо остающейся актуальной задачи поиска новых способов оптимизации, можно выделить ещё две серьёзные проблемы оптимизации: обработка неструктурированных запросов (возможно, на ограниченном естественном языке), и оптимизация группы запросов. Работа с неструктурированными запросами особенно актуальна в свете использования баз данных в поисковых системах (в том числе, при поиске в Internet). А оптимизация группы одновременно выполняющихся запросов позволит улучшить характеристики СУБД с точки зрения быстродействия.
5. **Интеграция разнородных и слабо формализованных данных**. Изначально базы данных предназначались для хранения и обработки фактографических хорошо структурированных данных. Но огромное количество данных представлено в различных графических и мультимедийных форматах. Включение в СУБД способов обработки подобных данных позволяет использовать технологии баз данных в таких сферах, как, например, ГИС (геоинформационные системы), издательские системы (с поддержкой вёрстки номеров издания), САПР (системы автоматизации проектирования) и т.д.
6. **Организация доступа к базам данных через Internet.** Многие web-сайты содержат динамическую информацию, например, о товарах и ценах в Internet-магазинах. В локальных системах такая информация традиционно хранится в базах данных. Интеграция СУБД в web-среду позволяет сохранить все преимущества баз данных для использования в web-приложениях.

   Основные вопросы, связанные с интеграцией СУБД в web-среду, следующие: 1) организация эффективного интерфейса, рассчитанного на неподготовленного пользователя; 2) оптимизация запросов, направленная на уменьшение сетевого трафика; 3) высокая производительность СУБД в многопользовательском режиме работы.
7. **Самоадаптация**. Современные СУБД имеют широкие возможности по настройке баз данных под конкретную предметную область и аппаратные средства. Но использование этих возможностей – достаточно сложная задача, которая требует наличия высококвалифицированного администратора БД. Для упрощения настройки и сопровождения баз данных СУБД должна

брать на себя большинство функций настройки и выполнять их в автоматическом или автоматизированном режиме.

8. **Использование GRID**. GRID – это концепция объединения вычислительных ресурсов в единую сеть. В качестве аналогии здесь можно привести электрические сети: при возникновении потребности пользователь просто подключается к сети и получает электричество. Точно также при возникновении потребности в вычислениях пользователь должен просто подключаться к GRID и получать вычислительные ресурсы. Преимущества этого подхода очевидны: возможность решать более ресурсоёмкие задачи и перераспределять нагрузку на узлы сети. Но и нерешённых проблем здесь тоже достаточно, поэтому это задача будущего.

   Тем не менее, первая промышленная GRID-система уже существует, но поддерживает она только базы данных: это система Oracle 10G. Она динамически выделяет ресурсы для выполнения задач пользователя по доступу к БД Oracle и перераспределяет нагрузку на узлы сети с целью оптимизации использования вычислительных ресурсов и повышения общей производительности системы.
9. **Сохранность данных**. Количество накопленных цифровых данных в мире огромно. Но со временем устаревают и форматы хранения данных, и средства доступа к ним. Происходит также старение носителей: размагничиваются магнитные ленты и диски, изменяются оптические и физические свойства носителя. Поэтому даже архивированные данные могут стать недоступными, особенно если нет устройства для чтения устаревшего носителя или отсутствует возможность запустить приложение, которое может читать устаревший формат. Решить эту проблему могут средства, обеспечивающие миграцию данных в новые форматы с сохранением их описания (метаданных).
10. **Технологии разработки данных и знаний** (data mining и knowledge mining). Технологии разработки данных предназначены для поиска неочевидных тенденций и скрытых закономерностей в больших объёмах данных. А knowledge mining – это извлечение знаний из баз данных (или из ХД). Здесь используются как формальные методы (регрессионный, корреляционный и другие виды статистического анализа), так и методы интеллектуальной обработки данных, основанные на моделировании познавательных механизмов – индукции, дедукции, абдукции.

## Список используемых сокращений

АИС – автоматизированная информационная система
БД – база данных
ИМД – иерархическая модель данных
ИПС – информационно–поисковая система
ИС – информационная система
КБД – ключ базы данных
ОП – оперативная память
ОС – операционная система
ПО – предметная область
ППО – прикладное программное обоеспечение
РБД – реляционная база данных
РМД – реляционная модель данных
РСУБД– реляционная система управления базами данных
СМД – сетевая модель данных
СОД – системы обработки данных
СУБД – система управления базами данных
ХД – хранилище данных
DDL – data definition language, язык определения данных
DML – data manipulation language, язык модификации данных

## Библиографический список

1. Коннолли Т., Бегг К. Базы данных: проектирование, реализация, сопровождение. Теория и практика, 3-е изд. : Пер. с англ. : Уч. пос. – М.: Изд. дом "Вильямс", 2003. – 1440 с.
2. Проектирование реляционной базы данных: Метод. указания к курсовому проектированию по курсу "Базы данных" / Московский государственный институт электроники и математики; Сост.: Карпова И.П. – М., 2003. – 28 с.
3. Изучение языка SQL: Метод. указания к лабораторным работам по курсу "Базы данных" / Московский государственный институт электроники и математики; Сост.: И. П. Карпова. М., 2003. – 32 с.
4. Манифест "Системы баз данных третьего поколения". – Журнал «СУБД»,·1995, № 2. – с. 143-159. http://rema.44.ru/resurs/study/ddb/manifest.html
5. Манифест «Системы объектно-ориентированных баз данных». – Журнал «СУБД»,·1995, № 4. – с. 142-155. http://rema.44.ru/resurs/study/ddb/manif_oo.html
6. ГОСТ 20886-85. Организация данных в системах обработки данных. Термины и определения.

КАРПОВА Ирина Петровна

Основы баз данных

Редактор Е. С. Резникова
Технический редактор О. Г. Завьялова

Лиц. № 020304 от 28.11.96. Подписано в печать.
Формат 60×84/16.
Бумага офсетная № 2. Ризография. Усл. печ. л.4,6. Уч.-изд.л.4,0.
Изд. № 52. Тираж 75 экз. Заказ .
Московский государственный институт электроники и математики.
109028, Москва, Б. Трехсвятительский пер., д.1-3/12, стр.8.
Отдел оперативной полиграфии
Московского государственного института электроники и математики.
113054, Москва, ул. М. Пионерская, д.12-18/4-6, стр.1.