Базы данных

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Московский государственный институт электроники и математики

И.П. Карпова

Утверждено Редакционно-издательским советом института в качестве Учебного пособия

УДК 004.65
ББК 32.973
К26

Москва 2009

Рецензенты: докт. техн. наук И.П. Беляев (НИИ ИТ)
канд. физ.-мат. наук Ю.А. Семёнов (МФТИ)

Карпова И.П.
Базы данных. Учебное пособие. – Московский государственный институт электроники и математики (Технический университет). – М., 2009.

ОСНОВЫ БАЗ ДАННЫХ

Рассматриваются основные модели данных и технологии организации баз данных.

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

Содержание

"Цивилизация развивается за счёт расширения числа важных операций, которые можно

выполнять, не думая о них".

Альфред Норт Уайтхед,
британский математик, логик,
философ

1. ВВЕДЕНИЕ

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

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 в рамках данного курса наше внимание будет сосредоточено именно на системах, которые основаны на данных.

Существуют две основные предпосылки создания таких систем:

1. Разработка методов конструирования и эксплуатации систем, предназначенных для коллективного использования.
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 системах, основанных на данных, называется базой данных. База данных (БД) является моделью предметной области информационной системы.

На заре развития вычислительной техники обрабатываемые данные являлись частью программ: они располагались сразу за кодом программы в так называемом сегменте данных (рис. 1.1,а). Следующим шагом стало хранение данных в отдельных файлах (рис. 1.1,б). Недостатком этих двух подходов являлась зависимость программ от данных: сведения о структуре данных включались в код программы. При изменении структуры данных необходимо было вносить изменения в программу.

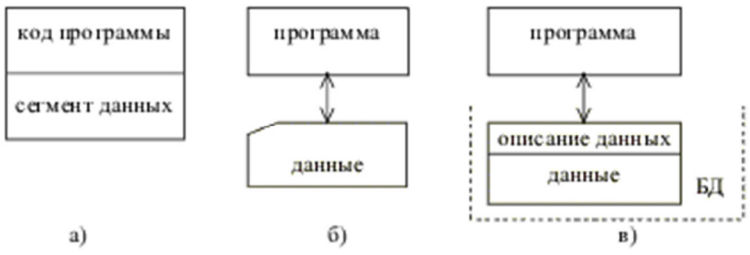

Рис.1.1. Развитие принципов обработки данных

Логичным продолжением этой эволюции является перенос описания данных в массив данных (рис. 1.1,в). Это позволило обеспечить независимость данных от программ.

Основным принципом организации баз данных является совместное хранение данных и их описания.

Описание данных называют *метаданными*. Метаданные хранятся в части базы данных, которая называется **каталогом** или **словарём-справочником** данных (ССД). Зная формат метаданных, можно запрашивать и изменять данные без написания дополнительных программ.

Одна и та же база данных может быть использована для решения многих прикладных задач. Наличие метаданных и возможность информационной поддержки решения многих задач – это принципиальные отличия базы данных от любой другой совокупности данных, расположенных во 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, находя-щаяся 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)	
Управление данными	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(ПО)	часть реального мира, подлежащая изучению с целью организации управления и, в конечном итоге, автоматизации.
База данных (БД)	совокупность данных, организованных по определённым 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 изменению структуры базы данных с целью обеспечения её целостности, сохранности и эффективности использования [6].
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 [6].
Автоматизированная информационная система (АИС)	представляет собой совокупность данных, 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. Банк данных должен: 1. Обеспечивать информационные потребности внешних пользователей. 2. Обеспечивать возможность хранения и модификации больших объёмов многоаспектных данных.

3. Обеспечивать заданный уровень достоверности хранимых данных и их непротиворечивость.

4. Обеспечивать доступ к данным только пользователям с соответствующими полномочиями.

5. Обеспечивать поиск данных по произвольной группе признаков.

6. Удовлетворять заданным требованиям по производительности при обработке запросов.

7. Иметь возможность реорганизации при изменении границ ПО.

8. Обеспечивать выдачу пользователям данных в различной форме.

9. Обеспечивать простоту и удобство обращения внешних пользователей к данным.

1)В данном учебном пособии сокращение ПО всегда означает "предметная область"

1.2. Автоматизированная информационная система

.Под **автоматизированной информационной системой** (АИС) будем понимать совокупность программно-аппаратных средств, предназначенных для автоматизации деятельности, связанной с хранением, передачей и обработкой информации.

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 любая 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.

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.). Фактографические АИС обычно принадлежат к классу систем обработки данных.

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 данных не важно, какие сведения хранятся в документах. Обычно (но не всегда) документальные АИС реализуются в виде информационно-поисковых систем (ИПС).

Основные компоненты документальной ИПС:

- 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 ссы-лок 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 ИПС предназначен для того, чтобы пользова-тель мог запросить данные у системы. Системными средствами пользовательский запрос преобразуется в формальный запрос, понятный системе. Информационный язык ИПС 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. 1.2 приведён пример поиска через экранную форму в справочнике библиотек. В форме указаны два условия – округ и фрагмент названия библиотеки, и в результате поиска система выдала три записи, удовлетворяющие этим условиям.

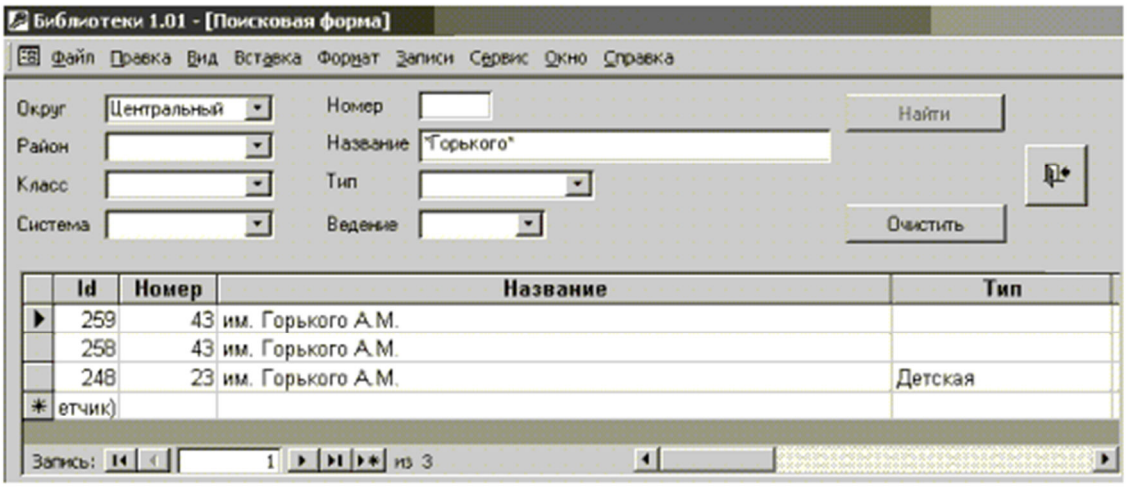

Рис.1.2. Пример поиска данных через экранную форму

Мы будем основное внимание уделять фактографическим АИС, имея в виду, что ИПС и документальные АИС создаются с помощью тех же про-граммных 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 АИС начинается с определения предметной области.

1.3. Предметная область информационной системы

Предметная область (ПО) информационной системы рассматривается как совокупность реальных процессов и объектов (**сущностей**), пред-ставляющих

интерес для её пользователей [7]. Каждая из сущностей 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.

Примечание. В данном учебном пособии наименования сущностей, атрибутов и связей вы-деляются курсивом и подчёркиванием. Кроме того:

1. Сущность записывается прописными буквами (*ОТДЕЛ*)
2. Атрибут сущности начинается с прописной буквы (*Название*). Ключевой атрибут выделяется полужирным шрифтом (***Табельный номер***).
3. Связь между сущностями определяется глаголом (*работает*).

Для упрощения процедуры формализации ПО в большинстве случаев прибегают к определению типов сущностей. Тип позволяет выделить из всего множества сущностей ПО группу сущностей, однородных по структуре и поведению (относительно рамок рассматриваемой ПО). Например, для ПО "Институт" в качестве типов сущностей могут рассматриваться студенты, преподаватели, дисциплины и т.п. Данные предметной области представляются экземплярами сущностей (студент Иванов, преподаватель Сидоров, дисциплина "Базы данных"). Экземпляры сущностей одного типа обладают одинаковыми наборами атрибутов, но должны отличаться значением хотя бы одного атрибута для того, чтобы быть узнаваемыми (например, студенты могут иметь одинаковые ФИО, но должны иметь разные номера зачётных книжек).

Между сущностями ПО могут существовать **связи**, имеющие различный содержательный смысл (семантику). Например, студент *учится* в группе, врач *лечит* пациента, клиент *имеет* вклад в банке. Связи могут быть **факультативными** или **обязательными**. Если вновь порождённая сущность одного из типов оказывается по необходимости связанной с сущностью другого типа, то между этими типами сущностей есть обязательная связь. Иначе связь является факультативной. Примеры обязательной и факультативной связей приведены на рис. 1.3. Здесь связь *замещает*является обязательной (изображается двойной линией), потому что каждый сотрудник *должен*работать на определённой должности, а связь *замещается* является факультативной, т.к. должность *может быть* вакантна.

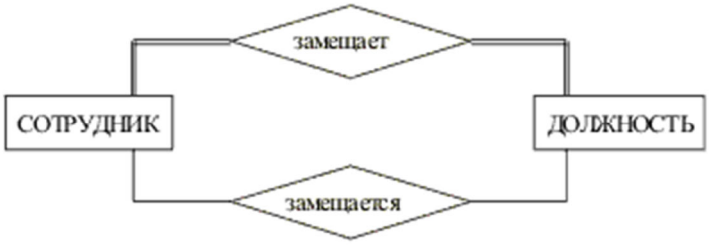

Рис.1.3. Примеры обязательной и факультативной связей

Для удобства каждую связь между сущностями можно изображать одним ромбом (рис. 1.4). Выделяют также показатель **кардинальности связи**: "один к одному" (1:1), "один ко многим" (1:n) и "многие ко многим" (m:n) (рис. 1.4).

Рис.1.4. Примеры различной кардинальности связей

Связи, приведённые на рис. 1.4, с учётом семантики означают следующее:

- пациент–койка (1:1) – каждый пациент занимает одну койку, каждая койка в каждый момент времени 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.

Степень связи – это количество сущностей, которые входят в связь. Различают унарные (рис. 1.5,а), бинарные (рис. 1.5,б) и тернарные (рис.1.5,в) связи. (На практике связи с большей степенью редко используют-ся). Унарная связь означает, что одни экземпляры сущности связаны с другими экземплярами этой же сущности (например, одни сотрудники руководят другими, а деталь может являться частью механизма)

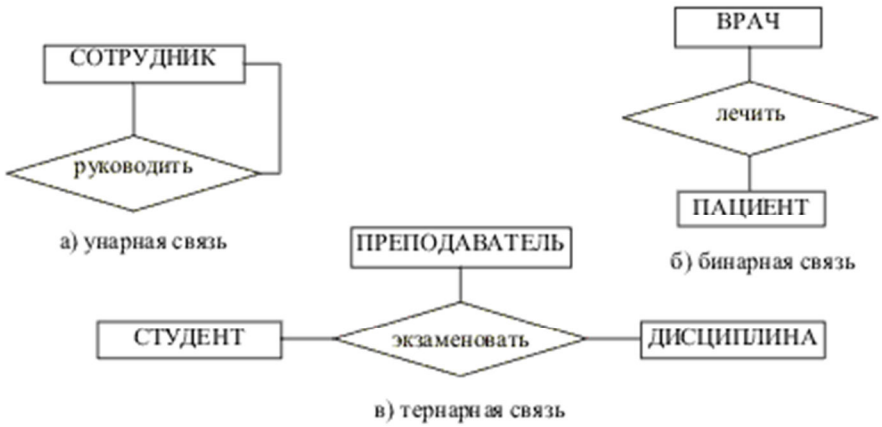

Рис.1.5. Примеры связей различной степени

Различают тип *связи* и *экземпляр связи*. Тип связи определяется её име-нем, обязательностью, степенью и кардинальностью, например, бинарная связь *учится* между сущностями *ГРУППА* и*СТУДЕНТ*, обязательная для студента, кардинальностью 1:n. А экземпляр связи – это конкретная связь между студентом Сидоровым и группой Н-11, в которой он учится.

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

Множества экземпляров сущностей, значения атрибутов сущностей и экземпляры связей 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 сущностей, типы сущностей, типы связей и/или их экземпляры. Фактически, 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 определённому 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 (например, запись сведений о выдаче абоненту книги в библиотеке);
- автоматизировано, если изменения инициируются пользователем, но выполняются программно (например, обновление списка должников в библиотеке – читателей, которые просрочили дату возврата книг);
- автоматически, если данные поступают в электронном виде и обрабатываются программой без участия человека (это касается, например, автоматизированных систем управления производством).

Правильность обновлений может контролироваться программно, но правильнее контролировать их автоматически с помощью ограничений целостности БД.

База данных является информационной моделью внешнего мира, некоторой предметной области. Во внешнем мире сущности 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 и хранить их раздельно.

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. 1.6). Большинство СОД включают также программы обработки данных (прикладное программное обеспечение, ППО), которые обращаются к дан-ным через СУБД.

Рис.1.6. Компоненты системы баз данных

В соответствии с рис. 1.6 СУБД обеспечивает выполнение двух групп функций:

- предоставление доступа к базе данных прикладному программному обеспечению (или квалифицированным пользователям);
- управление хранением и обработкой данных в БД.

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

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 три группы пользователей: читатели, которым нужно осуществлять поиск книг по различным признакам; сотрудники, выдающие и принимающие у читателей книги (библиотекари) и сотрудники отдела комплектации, осуществляющие приём новых книг и списание старых.

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. 1.7).

Рис.1.7 Уровни представления данных

Концептуальный уровень архитектуры ANSI/SPARC служит для под-держки 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 или схемой БД.

Схема базы данных – это описание базы данных в терминах конкретной модели данных.

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 данных. Описание БД на внутреннем уровне называется*внутренней схемой* или *схемой хранения*.

Внешний уровень архитектуры БД предназначен для групп пользователей. Описание представления данных для группы пользователей называется внешней схемой. Наличие внешнего уровня позволяет поддерживать разное представление одних и тех же данных для различных групп пользователей или задач [6].

Каждый из этих уровней может считаться управляемым, если он об-ладает внешним интерфейсом, который обеспечивает 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, или изменять представление данных для пользователей без изменения концептуальной схемы. **Физическая независимость данных** подразумевает возможность вносить изменения в схему хранения, не меняя концептуальную схему БД.

Основной характеристикой баз данных является совместное использование данных многими пользователями АИС. Должно существовать какое-то общее понимание информации, представленной данными. Общее понимание должно относиться к чему-либо внешнему по отношению к пользователям, и оно должно быть зафиксировано. Для этого необходима некоторая предварительно определённая грамматика, которую принято называть моделью данных.

> "Границы моего языка означают границы моего мира".
>
> Людвиг Витгенштейн, англо-австрийский философ и логик

2. ОСНОВНЫЕ МОДЕЛИ ДАННЫХ

Модель данных является инструментом моделирования произвольной предметной области.

2.1. Понятие модели данных

Модель данных – это совокупность правил порождения структур данных в базе данных, операций над ними, а также ограничений целостности, определяющих допустимые связи и значения данных, последовательность их изменения [6]. Итак, модель данных состоит из трёх частей:

1. Набор типов структур данных.

   Здесь можно провести аналогию с языками программирования, в которых тоже есть предопределённые типы структур данных, такие как скалярные данные, вектора, массивы, структуры (например, тип *struct* в языке Си) и т.д.

2. Набор 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. Набор 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 наборы, составляющие модель данных.

2.1.1. Типы структур данных

Структуризация данных базируется на использовании концепций "агрегации" и "обобщения". Один из первых вариантов 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).

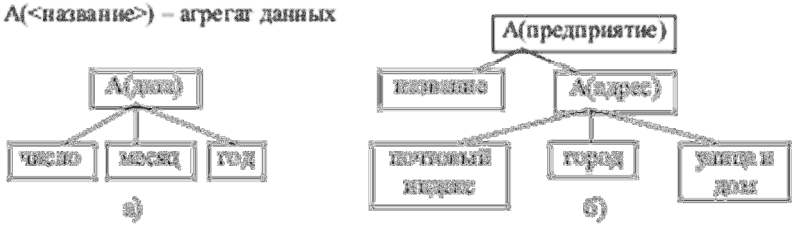

Рис.2.2 Примеры агрегатов: а) простой и б) составной агрегат

Запись – поименованная совокупность элементов данных или эле-ментов 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 одной сущности ПО (экземпляра). Иногда термин "запись" за-меняют термином "группа".

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

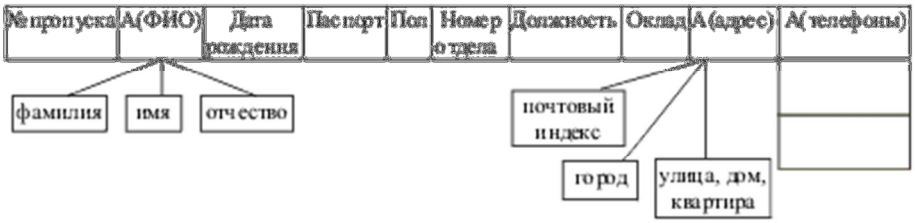

Рис.2.3 Пример записи типа *СОТРУДНИК*

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 записи) выделяются одно или несколько ключевых *полей*. Значения ключевых полей позволяют классифицировать сущность, к которой относится конкретная запись. Ключи с уникальными значениями называются *потенциальными*. Каждый ключ может представлять собой агрегат данных. Один из ключей назначается первичным, остальные являются вторичными. **Первичный ключ** идентифицирует экземпляр записи, его значение должно быть уникальным и обязательным 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 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).

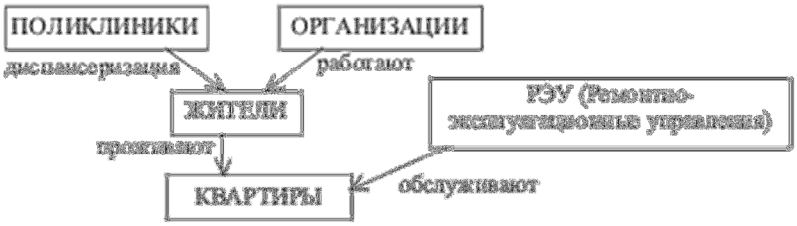

Рис. 2.4 Пример диаграммы Бахмана для фрагмента БД "Город"

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. Это самый высокий уровень структуризации данных.

Примечание: структуризация данных по версии CODASYL используется в сетевой и иерар-хической моделях данных. В реляционной модели принята другая структуризация данных, основанная на теории множеств.

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.**Транзакцией** называют упорядоченное множество операций, переводящих БД из одного согласованного состояния в другое. Транзакция либо выполняется полностью, т.е. выполняются все входящие в неё операции, либо не выполняется совсем, если в процессе её выполнения возникает ошибка.

2.1.3. Ограничения целостности

Ограничения целостности – это правила, которым должны удовлетворять значения элементов данных. Ограничения целостности делятся на **явные** и **неявные**.

Неявные ограничения определяются самой структурой данных. Например, тот факт, что запись типа *СОТРУДНИК* имеет поле *Дата рождения*, служит, по существу, ограничением целостности, означающим, что каждый сотрудник организации имеет дату рождения, причём только одну.

Явные ограничения включаются в структуру базы данных с помощью средств языка контроля данных (DCL, Data Control Language). В качестве явных ограничений чаще всего выступают условия, накладываемые на значения данных. Например, номер паспорта является уникальным, заработная плата не может быть отрицательной, а дата приёма сотрудника на работу обязательно будет меньше, чем дата его перевода на другую работу.

Также различают **статические** и **динамические** ограничения целостно-сти. 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 индивидуального номера налогоплательщика (ИНН) или задание ограниченного множества значений атрибута "Пол" ('м' и 'ж'). В качестве примера динамического ограничения целостности можно привести правило, которое распространяется на поля-счётчики: значение счётчика не может уменьшаться.

За выполнением ограничений целостности следит СУБД в процессе своего функционирования. Она проверяет ограничения целостности каждый раз, когда они могут быть нарушены (например, при добавлении данных, при удалении данных и т.п.), и гарантирует их соблюдение. Если какая-либо команда нарушает ограничение целостности, она не будет выполнена и система выдаст соответствующее сообщение об ошибке. Например, если задать в качестве ограничения правило «Остаток денежных средств на счёте не может быть отрицательным», то при попытке снять со счёта денег больше, чем там есть, система выдаст сообщение об ошибке и не позволит выполнить эту операцию.

Таким образом, ограничения целостности обеспечивают логическую непротиворечивость данных при переводе БД из одного состояния в другое.

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), характеризующих экземпляр сущности. Для каждого типа записи выделяется первичный ключ – атрибут, значение которого позволяет однозначно идентифицировать запись среди экземпляров записей данного типа.

Связи между записями в СМД выполняются в виде указателей, т.е. каждая запись хранит ссылку на другую однотипную запись (или признак конца списка) и ссылки на списки подчинённых записей, связанных 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 характеризуются следующими признаками:

1. Способ упорядочения подчинённых записей

   Поддерживаются три способа упорядочения:

   - Очередь – добавление в конец списка (FIFO – first input, first output).
   - Очередь – добавление в конец списка (FIFO – first input, first output).
   - Сортировка по значению ключа. При этом задаётся ключев Очередь – добавление в конец списка (FIFO – first input, first output).
   - ое поле (группа полей), 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 подчинённая 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 подчинённой записи.)

При ручном режиме включения подчинённая 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**:/p>

- 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 удалён.
- 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*.

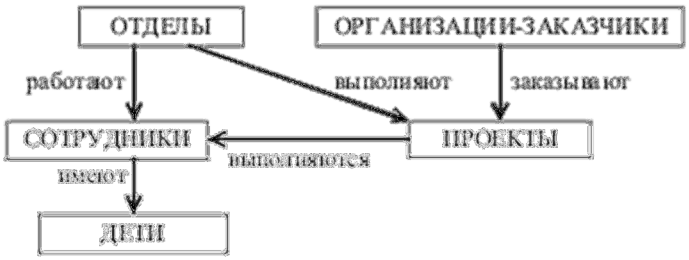

Рис. 2.5. Пример фрагмента сетевой БД "Предприятие"

Групповые отношения чаще всего описывают связь "один-ко-многим": один владелец, много подчинённых. 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, каждый проект могут выполнять несколько человек. Что касается

классов членства подчинённых 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 связи 1:n между разными сущностями реализуются с помощью групповых отношений, а связи 1:n между атрибутами сущности – в рамках записи. Для реализации связей типа n:m вводится вспомогательный тип записи и две связи 1:n.

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 навига-ции и манипулирования данными. Аппарат навигации в графовых моделях служит для установления тех записей, к которым будет применяться очередная операция манипулирования данными. Такие записи называются текущими. В СМД возможны переходы:

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

Навигация в СМД может начинаться с любой записи.

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. Примером се-тевой СУБД является система Integrated Database Management System (IDMS).

СМД является наиболее полной с точки зрения реализации различных типов связей и ограничений целостности, но она является достаточно сложной для проектирования и поддержки. В этой модели не обеспечивается физическая независимость данных, т.к. наборы организованы с помощью физических ссылок. Также в СМД не обеспечивается независимость данных от программ. Из-за этих недостатков эта модель 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 (версия CODASYL). Группу в ИМД принято называть сегментом. 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 определяется так:

- 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 древо-видной 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 вершиной (сегментом) иерархическим групповым отношением. Каждая вершина дерева соответствует типу сущности ПО. Тип сущности характеризуется произвольным количеством атрибутов, связанных с ней отношением 1:1. Атрибуты, связанные с сущностью отношением 1:n, образуют отдельную сущность (сегмент) и переносятся на следующий уровень иерархии. Реализация связей типа n:m не поддерживается.

Рис. 2.6. Пример фрагмента иерархической базы данных

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.

По сравнению с СМД иерархическая имеет ограниченный набор режимов включения и исключения подчинённых записей. Это определяется обязательностью связей: в дереве не может быть «висячих» вершин, не связанных с вершиной верхнего уровня (кроме корневой). Поэтому ИМД не поддерживает необязательный класс членства и ручной режим включения записей.

В ИМД предусмотрены специальные способы навигации. Передвижение по дереву всегда начинается с корневой вершины, от которой можно перейти 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. 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).

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. Например, для студента (рис. 2.6) ключ – это
(Шифр_факультета+Номер_курса+Номер_группы+Номер_зачётной_книжки).

Основным недостатком ИМД является дублирование данных. Оно вызвано тем, что каждая сущность (атрибут) может относиться только к одной родительской сущности. Например, если в БД хранятся данные о детях сотрудников, а на предприятии работает и отец, и мать ребёнка, то сведения об этом ребёнке нужно хранить дважды. Аналогичная ситуация возникает, если нужно отразить в БД связь «многие-ко-многим». Дублирование данных может вызвать нарушение логической целостности БД при внесении изменений в эти данные.

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.

В качестве примера типичного представителя иерархических СУБД можно привести систему IMS (Information Management System, IBM).

Сетевая и иерархическая модели данных относятся к базам данных I-го поколения (60-е – начало 70-х гг. XX века). Эти модели не смогли в полной мере реализовать независимость данных от программ. Из-за особенностей их организации структура запросов к данным в таких системах определяется наличием связей между записями.

Следующее, II-е поколение баз данных основано на реляционной модели.

2.4. Реляционная модель данных (РМД)

2.4.1. Понятие отношения

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

Базовой структурой РМД является **отношение**, основанное на декарто-вом произведении доменов. **Домен** – это множество значений, которое может принимать элемент данных (например, множество целых чисел, множество дат, множество комбинаций символов длиной N и т.п.). Домен может задаваться перечислением элементов, указанием диапазона значений, функцией и т.д.

Пусть D1, D2 ,…, Dk – произвольные конечные и не обязательно различ-ные множества (домены). **Декартово произведение** этих множеств определяется следующим образом:

D1×D2×...×Dk={(d1, d2,...,dk | di Î Di, I=1,...,k)}

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

Пример. Для доменов D1 = (1, 2), D2 = (A, B, C) декартово произведение D = D1×D2 будет таким: D = {(1,A), (1,B), (1,C), (2,A), (2,B), (2,C)}

Подмножество декартова произведения доменов называется **отношением**.

Отношение содержит данные о сущностях определённого типа. Поясним это на примере. Если построить произведение трёх доменов *Должности* ('директор',

'бухгалтер', 'водитель', 'продавец'),*Оклады* (x | 20000?x?80000), *Надбавки* (1.1, 1.2, 1.3), то мы получим 4*60001*3=720012 комбинаций. Но реально отношение «Штатное расписание» содержит по одной строке на каждую должность, т.е. является именно подмножеством декартова произведения доменов.

Элементы отношения называют *кортежами* (или *записями*). Каждый кортеж отношения соответствует одному экземпляру сущности определённого типа. Элементы кортежа принято называть *атрибутами* (или *полями*).

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.

Таким образом, в отношении не бывает первого, второго или последнего кортежа: при выводе данных отношения кортежи выводятся в произвольном порядке, если не задано упорядочение по значениям полей.

Отношение удобно представлять как таблицу, где строка является кортежем, а столбец соответствует домену (рис. 2.7, отношение *СТУДЕНТЫ*). Количество строк в таблице (кортежей в отношении) называется **мощностью отношения**, количество столбцов (атрибутов) – **арностью**.

Группа	ФИО студента	Номер зачетной книжки	Год рождения	Размер стипендии
С-72	Волкова Елена Павловна	С-12298	1991	1550.00
С-91	Белов Сергей Юрьевич	С-12299	1990	1400.00
.....				
С-72	Фролов Юрий Вадимович	С-14407	1991	0

Рис.2.7. Пример табличной формы представления отношения

Отношение имеет имя, которое отличает его от имён всех других отно-шений. 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.

Каждый атрибут определён на некотором домене, несколько атрибутов отношения могут быть определены на одном и том же домене (например, номера рабочего и домашнего телефонов). Домен задаётся типом данных, размером и ограничениями целостности: например, пол – это символьное поле

длиной 1, которое может принимать значения из множества ('м', 'ж'). В реляционных базах данных поддерживаются такие типы данных как символьный, числовой, дата и некоторые другие (конкретный перечень типов зависит от СУБД).

Атрибут может быть обязательным и необязательным. Значение обяза-тельного атрибута должно быть определено в момент внесения данных в БД. Если атрибут необязательный, то для таких случаев предусмотрено специальное значение – NULL, которое можно интерпретировать как "неизвестное значение". Значение NULL не привязано к определённому 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 (группа атрибутов), значения которого классифицируют или идентифицируют кортеж. Например, значение атрибута *Группа* отношения *СТУДЕНТЫ* позволяет выделить среди всех студентов института студентов конкретной группы. Если ключ состоит из нескольких атрибутов, он называется *составным*. Если значения ключа уникальны в рамках столбца отношения, то такой ключ называется *потенциальным*. Потенциальных ключей может быть несколько (или не быть ни одного), но для отношения выделяется один основной ключ – первичный. **Первичный ключ** идентифицирует экземпляр сущности, его значение должно быть уникальным (*unique*) и обязательным (*not null*). (На рис. 2.7 первичный ключ выделен полужирным шрифтом). Неуникальные ключи ещё называют *вторичными*.

РМД не поддерживает групповые отношения (по версии CODASYL). Для связей между отношениями используются внешние ключи. Внешний ключ (*foreign key*) – это атрибут подчинённого (дочернего) отношения, который является копией первичного (*primary key*) или уникального (*unique*) ключа родительского отношения. (Пример – отношение *ОЦЕНКИ*, связанное с отношением *СТУДЕНТЫ* внешним ключом *Номер зачётной книжки*, рис. 2.8).

Номер зачетной книжки	Дисциплина	Оценка
С-12298	Программирование	5
С-1229891	Дискретная математика	4
С-14407	Программирование	3
.....		

Рис.2.8. Связь отношений "Оценки" и "Студенты" по внешнему ключу

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

Фактически внешние ключи *логически* связывают экземпляры сущно-стей разных типов (родительской и подчинённой сущностей).

Подмножество декартова произведения доменов называется **отношением**.

Внешний ключ – это ограничение 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 ссылаться на первичный ключ этой же таблицы. Это позволяет описывать унарную связь – 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). Атрибут *Руководитель* 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.	нач. отдела	002
909	2	Серова Т.В.	вед. программист	988

Рис.2.9. Внешний ключ "Руководитель", ссылающийся на первичный ключ этой же таблицы

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

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

Структуризация данных в РМД существенно отличается от структуризации данных по версии CODASYL (см. табл. 2.1).

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

Примечание: в реляционной модели данных набор (групповое отношение) моделируется с помощью внешнего ключа, описывающего связь между двумя таблицами.

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, [3]).

Моделирование предметной области в рамках реляционной модели создаёт некоторые сложности, т.к. в этой модели нет специальных средств для отображения различных типов связей и агрегатов. Отсутствие агрегатов приводит к тому, что при проектировании реляционной БД приходится проводить нормализацию отношений. После нормализации данные об одной сущности предметной области распределяются по нескольким таблицам, что усложняет работу с БД. (Подробнее об этом рассказано в разделе 8.9.7).

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. Но с помощью внешних ключей и дополнительных возможностей СУБД их можно эмулировать. (Подробнее об этом рассказано в [3]).

Итак, реляционная модель данных – это модель данных, основанная на представлении данных в виде набора отношений, каждое из которых является подмножеством декартова произведения определённых множеств. Манипулирование данными в РМД осуществляется с помощью операций реляционной алгебры (РА) или реляционного исчисления [1]. Реляционная алгебра основана на теории множеств, а реляционное исчисление базируется на математической логике (вернее, на исчислении предикатов первого порядка). Изучение реляционного исчисления выходит за рамки данного пособия. Мы рассмотрим только операции реляционной алгебры.

2.4.4. Операции реляционной алгебры

Операндами для операций реляционной алгебры являются реляционные отношения. Результатом выполнения операций РА также является отношение. Таким образом, механизм реляционной алгебры замкнут относительно понятия отношения. Это позволяет применять операции РА каскадно.

Использование операций РА 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 с помощью специальных команд (ключевых слов) 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 pA,C(R) будет такой, как показано на рис. 2.10,б.

Отношение R

A	B	C
a	b	c

Секция pAC(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 sC=d(R) (при условии "значение атрибута C равно d") будет такой (рис. 2.11,б):

Отношение R

A	B	C
a	b	c
c	a	d
c	b	d
а)		

Секция sC=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, соответст-вующая 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
1	2	3
4	5	6

Декартово произведение R ×S

A	B	C	D	E
a	b	1	2	3
a	b	4	5	6
c	a	1	2	3
c	a	4	5	6
b	d	1	2	3
b	d	4	5	6

а)	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** (minus).

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
c	a	d
c	h	c

Отношение S

A	B	C
g	h	a
h	d	d

Разность R–S

A	B	C
c	a	d
c	h	c

а) б)

Рис. 2.13. Пример разности отношений

Следующие три операции являются вспомогательными операциями РА.

6. **Пересечение** (intersection).

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

R∩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 отноше-ний выполняется некоторое условие F. Соединение может быть выражено так:

R><S=sF(R×S))
F

Если условием является равенство атрибутов исходных отношений, такая операция называется*эквисоединением*. **Естественное соединение** – это 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 >< 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

Соединение R><S

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

а) б)

Рис. 2.14. Пример естественного соединения отношений

8. **Деление** (division).

Пусть отношение R содержит атрибуты {r1,r2,...,rk, rk+1,...,rn}, а отношение S – атрибуты {rk+1,...,rn}. Тогда результирующее отношение содержит атрибуты {r1,r2,...,rk}. 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
c	v	g	h
c	f	c	b

Отношение S

C	D
g	h
c	b

Частное R/S

A	B
a	b
c	f

а) б)

Рис. 2.15. Пример операции деления

Языком обработки данных, основанным на реляционной алгебре, является SQL (основы этого языка изложены в [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 [4]. Базы данных, основанные на этих моделях, принято относить к III-у поколению.

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-99, а в 2003 г. вышел второй релиз этого стандарта, получивший название SQL-3, который определяет основные характеристики ОРМД. Но до сих пор объектно-реляционные модели, поддерживаемые различными производителями СУБД, существенно отличаются друг от друга. 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 объектно-реляционной БД 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.

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

В 2000 г. рабочая группа ODMG (Object Database Management Group), образованная фирмами-производителями ООСУБД, выпустила очередной стандарт (ODMG 3.0) для ООСУБД, в котором описана объектная модель, язык определения запросов, язык объектных запросов и связующие языки C++, Smalltalk и Java. Стандарты ODMG не являются официальными. Подход ODMG к стандартизации заключается в том, что после принятия очередной версии стандарта организациями-членами ODMG публикуется книга, в которой содержится текст стандарта.

Теперь рассмотрим, как поддержка моделей данных реализована в реальных системах управления базами данных.

> "Чистая математика делает то, что можно, и так, как нужно. Практическая математика делает то, что нужно, и так, как можно".
>
> «Фантазия или наука»,
> Д.А. Поспелов, профессор,
> специалист по искусственному интеллекту

3. СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Система управления базами данных (СУБД) – это важнейший ком-понент 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 по-зволяет 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 создаёт 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 операций над данными. СУБД ОН имеют в своём 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. Причин может быть несколько:

- не достигается требуемого быстродействия обработки данных;
- необходима работа СУБД в условиях жёстких аппаратных ограничений;
- требуется поддержка специфических функций обработки 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 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.

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

Для реляционных СУБД Э.Ф. Кодд предложил и обосновал 12 правил, которым должна удовлетворять реляционная СУБД данных (РСУБД).

3.2. Правила Кодда для реляционной СУБД (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. Динамический каталог данных, основанный на реляционной модели (Dynamic On-Line Catalog Based on the Relational Model). Каталог (или словарь-справочник) 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 (DDL, Data Definition Language), манипулирование данными (DML, Data Manipulation Language), управление доступом пользователей к данным, управление транзакциями.
6. Поддержка обновляемых представлений (View Updating Rule). Представление (view) – это хранимый запрос к таблицам базы данных. (Подробнее о представлениях рассказано в [3]). Обновляемое представление должно поддерживать все операции манипулирования данными, которые поддерживают реляционные таблицы: операции 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 произвольной таблицы.
8. Физическая независимость данных (Physical Data Independence). Приложения не должны зависеть от используемых способов хранения данных на носителях, от аппаратного обеспечения компьютера, на котором 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 реализуется с помощью ограничений целостности и механизма транзакций (см. разделы 5.2 и 6.1).
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 влияние 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 пользовате-лями. 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. раздел 5.2).
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 (**реорганизация БД**) для повышения производительности системы или повторного использования освободившейся памяти;
- модификацию концептуальной схемы данных (**реструктуризация БД**) при изменении предметной области и/или потребностей пользователей.

Задачи администратора БД (АБД) достаточно важны, поэтому на них следует остановиться несколько подробнее.

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 соот-ветствие 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 фи-зическим носителям с целью распараллеливания доступа к ней;
        - построение кластеров (раздел 4.5);
        - изменение физических параметров среды хранения, например, размера блока данных в пространстве памяти.
    - изменения используемых методов доступа к данным, например, построение индексов или введение хеширования (раздел 4.5).
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 несанкцио-нированного 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). Иногда ССД называют каталогом данных. 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 информационной системы.

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

Словарь БД служит для поддержки функционирования компонентов программного обеспечения – СУБД и прикладных программ, работающих с БД. Словарь содержит сведения об организации БД, её составе и структуре, описание данных: форматы представления, структуру, методы доступа, способы размещения данных в памяти и т.п. Информация в словаре представлена в виде, удобном для программного использования.

Справочник БД содержит сведения о семантике данных, способах их идентификации, источниках данных и т.п. Справочник предназначен глав-ным образом для документирования разработки БД и справочного обслуживания её пользователей. Информация в справочнике представлена в виде, удобном для восприятия человеком.

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

Далее мы познакомимся с теми механизмами, с помощью которых в СУБД организуется хранение данных и доступ к ним.

> "В действительности всё выглядит иначе, чем на самом деле".
>
> Станислав Ежи Лец, польский поэт, участник Сопротивления

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.

С точки зрения пользователя работа с данными происходит на уровне записей концептуального уровня и заключается в добавлении, поиске, изменении и удалении записей. При этом механизмы среды хранения делают следующее:

1. При запоминании новой записи:
    - определение места размещения новой записи в пространстве памяти;
    - выделение необходимого ресурса памяти;
    - запоминание этой записи (сохранение в памяти);
    - формирование связей с другими записями (конкретный механизм зависит от модели данных).

    Примечание: в реляционных базах данных формирование связей осуществляется на логиче-ском уровне (т.е. по значениям атрибутов), а в иерархических и сетевых БД – на фи-зическом уровне (по адресам записей).

2. При поиске записи:
    - поиск места размещения записи в пространстве памяти по заданным значениям атрибутов;
    - выборка записи для обработки в оперативную память (в буфер данных).
3. При изменении атрибутов записи:

- поиск записи и считывание её в ОП;
- изменение значений атрибута (атрибутов) записи;
- сохранение записи на диск.

Запись помещается на прежнее место, если она не увеличилась в объёме или на прежнем месте достаточно памяти для неё. Если запись увеличи-лась в объёме и не помещается на прежнем месте, то она либо записывается на новое место, либо разбивается на части, и первая часть хранится на прежнем месте, а продолжение – на новом, на которое указывается ссылка из первой части.

- При удалении записи:

- удаление записи с освобождением памяти (*физическое удаление*) или без освобождения (*логическое удаление*);
- разрушение связей с другими записями (конкретный механизм зависит от модели данных).

В случае логического удаления 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, которую автоматически запускает СУБД или вручную АБД. При физическом удалении записи ранее занятый участок освобождается и становится доступным для повторного использования.

Физическую организацию БД мы будем рассматривать только для РСУБД. Ознакомиться со способами организации СУБД, основанных на других моделях данных, можно в [1].

Все операции на физическом уровне выполняются по запросам механизмов концептуального уровня СУБД. На физическом 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.

В трехуровневой модели архитектуры СУБД декларируется независи-мость архитектурных уровней. Но 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. Структура хранимых данных

Единицей хранения данных в БД является **хранимая запись**. Она может представлять собой как полную запись концептуального уровня, так и некоторую её часть. Если запись разбивается на части, то эти части представляются экземплярами хранимых записей каких-либо типов. Все части записи связываются указателями (ссылками) или размещаются по специальному закону так, чтобы механизмы междууровневого 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.

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 и проч. *Никакие пользовательские программы не имеют доступа к служебной части хранимой записи*.
2. *Информационная часть*. Содержит значения элементов данных.

Поля хранимой записи могут иметь фиксированную или переменную длину. При этом желательно поля фиксированной длины размещать в начале записи, а необязательные поля – в конце. Хранение полей переменной длины осуществляется одним из двух способов: размещение полей через разделитель или хранение размера значения поля. Наличие полей переменной длины позволяет не хранить незначащие символы и снижает затраты памяти на хранение данных; но при этом увеличивается время на из-влечение записи.

Каждой хранимой записи БД система присваивает внутренний идентификатор, называемый (по стандарту CODASYL) **ключом базы данных** (КБД). (Иногда используется термин*идентификатор строки*, RowID). Значение КБД формируется системой при размещении записи и содержит информацию, позволяющую однозначно определить место размещения записи (преобразовать значение КБД в адрес записи). В качестве КБД может выступать, например, последовательный номер записи в файле или совокупность адреса страницы памяти 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.

4.3. Управление пространством памяти и размещением 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, сегменты и др.). (Во многих системах область соответствует файлу.) В каждой *области памяти*, как правило, хранятся данные одного объекта БД (одной таблицы). Сведения о месте расположения данных таблицы (ссылка на область хранения) СУБД

хранит в словаре-справочнике данных (ССД). Области разбиваются на пронумерованные*страницы (блоки)* 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]. Некоторые СУБД позволяют управлять размером страницы (блока) для базы данных. В таких системах размер страницы определяется на основе компромисса между производительностью системы и требуемым объёмом оперативной памяти.

Страница имеет **заголовок** со служебной информацией, вслед за ко-торым располагаются собственно данные. В большинстве случаев в качестве единицы хранения данных принимается хранимая запись. На странице размещается, как правило, несколько хранимых 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.

Система автоматически управляет свободным пространством памяти на страницах. Как правило, это обеспечивается одним из двух способов:

- ведение списков свободных участков;
- динамическая реорганизация страниц.

При **динамической реорганизации страниц** записи БД плотно разме-щаются 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. При увеличении размера существующей записи она записывается по прежнему адресу, а вслед идущие записи сдвигаются.

Достоинство такого подхода – отсутствие фрагментации. Недостатки:

- Адрес записи может быть определён с точностью до адреса страницы, т.к. внутри страницы запись может перемещаться.
- Поиск места размещения новой записи может занять много времени. Система будет читать страницы одну за другой до тех пор, пока не найдёт странницу, на которой достаточно места для размещения новой записи.

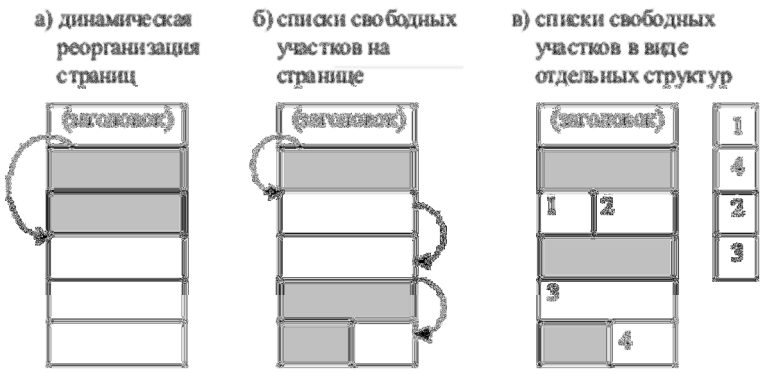

Рис. 4.1. Управление свободным простанством памяти на страницах

Для того чтобы уменьшить время поиска места для размещения записей, при динамической реорганизации страниц могут создаваться так называемые инвентарные страницы, на которых хранятся размеры свободных участков для каждой страницы. Поиск свободного места для размещения новых записей осуществляется через *инвентарные страницы*, которые загружаются в оперативную память. При каждом удалении/размещении данных содержимое инвентарных страниц обновляется. Таким образом, обеспечение актуальности содержимого инвентарных страниц занимает дополнительное время, но оно меньше, чем время поиска свободного участка на страницах.

Некоторые СУБД управляют памятью по-другому: они ведут **список свободных участков**. Здесь можно рассмотреть два варианта:

1. Ссылка на первый свободный участок на странице хранится 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.
2. Списки свободных участков реализуются в виде отдельных структур (рис. 4.1,в). Эти структуры также 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.

Ведение списков свободных участков не приводит к перемещению записи, и адрес записи можно определить с точностью до смещения на странице. Это ускоряет поиск данных, т.к. не нужно просматривать все записи на странице для поиска каждой конкретной записи.

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 формат переменной длины, возможны ситуации, когда запись не помещается на прежнее место. Тогда запись разбивается на фрагменты, которые могут размещаться на разных страницах. Эти фрагменты связаны друг с другом ссылками, что позволяет системе "собирать" запись из отдельных фрагментов.

Основным недостатком, возникающим при использовании списков сво-бодных участков, является *фрагментация* пространства памяти, т.е. появление разрозненных незаполненных участков памяти. Для того чтобы уменьшить фрагментацию, в подобных СУБД предусмотрены фоновые процедуры, которые периодически проводят слияние смежных свободных участков в один (например, участки 1 и 2 на рис. 4.1,в).

Структура и представление хранимых данных, их размещение в про-странстве памяти и используемые методы доступа называются *схемой хранения*. Схема хранения оперирует в терминах типов объектов.

4.4. Виды адресации хранимых записей

В общем случае адреса записей БД нигде не хранятся. При поиске данных СУБД из словаря-справочника данных берёт информацию о том, в какой области памяти (например, в каком файле и/или на каких страницах памяти) расположены данные указанной таблицы. Но при этом для поиска конкретной записи (по значениям ключевых полей) система вынуждена будет прочитать всю таблицу. В РСУБД для ускорения поиска данных применяются индексы – специальные структуры, устанавливающие соответствие значений ключевых полей записи и "адреса" этой записи (КБД). Таким образом, вид адресации хранимых записей оказывает влияние на производительность, а также на переносимость БД с одного носителя на другой.

Рассмотрим три вида адресации: прямую, косвенную и относительную.

Прямая адресация предусматривает указание непосредственного местоположения записи в пространстве памяти. Прямая адресация используется, например, в системе ADABAS. Недостатком такой адресации является большой размер адреса, обусловленный большим размером пространства памяти. Кроме того, прямая адресация не позволяет перемещать записи в памяти без изменения КБД. Такие изменения привели бы к необходимости коррекции различных указателей на записи в среде хранения (например, в индексах, см. раздел 4.5.2), что было бы чрезвычайно трудоёмкой

процедурой. Отсутствие возможности перемещать запись ведёт к фрагментации памяти.

Указанные недостатки можно преодолеть, используя **косвенную адресацию**. Общий принцип косвенной адресации заключается в том, что в качестве КБД выступает не сам "адрес записи", а адрес места хранения "адреса записи".

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. В качестве КБД записи выступает совокупность номера нужной страницы и номера требуемого слота в индексе этой страницы (значения **N, i** на рис. 4.2). В **i**-м слоте на **N**-й странице хранится собственно адрес записи (смещение от начала страницы).

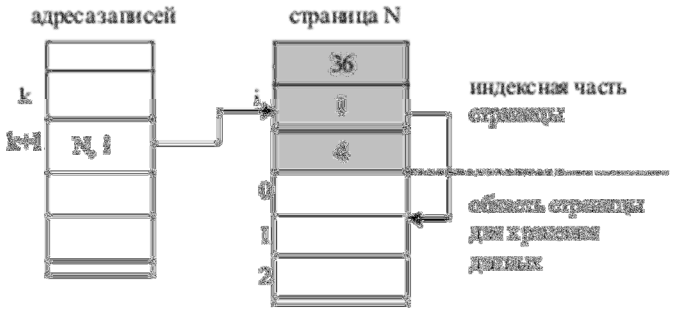

Рис. 4.2. Косвенная адресация с использованием индексируемых страниц

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 освободившуюся память для повторного использования.

Третий способ адресации – **относительная адресация**. Простейший ва-риант относительной адресации может использоваться, например, в ситуации, когда данные одного объекта БД (таблицы) хранятся в отдельном файле и хранимая запись имеет формат фиксированной длины. Тогда в качестве значения КБД берётся порядковый номер записи, по которому можно вычислить смещение от начала файла. (Пример такой адресация – системы dBaseIII, dBaseIV).

Общий принцип относительной адресации заключается в том, что адрес отсчитывается от начала той области памяти, которую занимают данные объекта БД. Если память разбита на страницы (блоки), то адресом может выступать номер страницы (блока) и номер записи на странице (или смещение

от начала страницы). В случае относительной адресации перемещение записи приведёт к изменению КБД и необходимости корректировки индексов, если они есть.

Примечание: некоторые СУБД, использующие относительную адресацию, при необходимости перемещения отдельной записи оставляют КБД прежним. Т.е. физически запись хранится на новом месте, а по старому адресу хранится новый адрес записи. Это по-зволяет не менять КБД и не перестраивать индексы, но приводит к увеличению вре-мени доступа к записи (2 физических чтения вместо одного).

4.5. Способы размещения данных и доступа к данным в РБД

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 размещения данных отметим хеширование и кластеризацию.

Хеширование заключается в том, что специально подобранная хеш-функция преобразует значение ключа записи в адрес блока (страницы) памяти, в котором эта запись будет размещаться. Под ключом записи здесь подразумевается поле или набор полей, позволяющие классифицировать запись. Например, для таблицы *СОТРУДНИКИ* в качестве ключа записи может выступать поле*Номер паспорта* или набор полей (*Фамилия, Имя, Дата рождения*).

Кластеризация – это способ хранения в одной области памяти таблиц, связанных внешними ключами (одна родительская таблица, одна или несколько подчинённых таблиц). Для размещения записей используется значение внешнего ключа таким образом, чтобы все данные, имеющие одинаковое значение внешнего ключа, размещались в одном блоке данных. Например, для таблиц*СОТРУДНИКИ, ДЕТИ СОТРУДНИКОВ, ТРУДОВАЯ КНИЖКА, ОТПУСКА* в качестве внешнего ключа подчинённых таблиц выступает первичный ключ *Идентификатор сотрудника* таблицы*СОТРУДНИКИ*, и тогда при кластеризации все данные о каждом сотруднике будут храниться в одном блоке данных.

4.5.1. Способы доступа к данным

Рассмотрим основные способы доступа к данным:

- **Последовательная обработка области БД**. Областью БД может быть файл или другое множество страниц (блоков) памяти. 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 ключу (в частности, первичному)**. Если система обеспечивает доступ по ключу, то этот ключ также может использоваться при запомина-нии записи (для определения места размещения записи в памяти). В базах данных применяются такие способы доступа по ключу, как индексирование, хеширование и кластеризация.

Примечание: в иерархических и сетевых СУБД есть ещё доступ по структуре. Эта разно-видность доступа применяется для групповых отношений и позволяет перейти к пре-дыдущему или следующему экземпляру группового отношения, к экземпляру-владельцу группового отношения или к списку подчинённых экземпляров./p>

4.5.2. Индексирование данных

Определим индексирование как способ доступа к данным в реляци-онной таблице с помощью специальной структуры – индекса.

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

Рис. 4.3. Пример индекса

Индекс обычно хранится в отдельном файле или отдельной области па-мяти. Пустые значения атрибутов (NULL) не индексируются.

Индексирование используется для ускорения доступа к записям по значению ключа и не влияет на размещение данных этой таблицы. Ускорение поиска данных через индекс обеспечивается за счёт:

1. упорядочивания значений индексируемого атрибута. Это позволяет просматривать в среднем половину индекса при линейном поиске;
2. индекс занимает меньше страниц памяти, чем сама таблица, поэтому система тратит меньше времени на чтение индекса, чем на чтение таблицы.

Индексы поддерживаются динамически, т.е. после обновления таблицы – добавления или удаления записей, а также модификации индексируемых полей, – индекс приводится в соответствие с последней версией данных таблицы. Обновление индекса, естественно, занимает некоторое время (иногда, очень большое), поэтому существование многих индексов может замедлить работу БД.

Примечание: в реальных СУБД существуют методы оптимизации переиндексации. Напри-мер, при выполнении пакетной операции модификации БД обновление индексов мо-жет происходить один раз после внесения всех изменений в данные.

Обращение к записи таблицы через индексы осуществляется в два этапа: сначала СУБД считывает индекс в оперативную память (ОП) и находит в нём требуемое значение атрибута и соответствующий адрес записи (КБД), затем по этому адресу происходит обращение к внешнему запоминающему устройству. Индекс загружается в ОП целиком или хранится в ней постоянно во время работы с таблицей БД, если хватает объ-ёма ОП.

Индекс называется *первичным*, если каждому значению индекса соответствует уникальное значение ключа. Индекс по ключу, допускающему дубликаты значений, называется *вторичным*. Большинство СУБД автоматически строят индекс по первичному ключу и по уникальным столбцам. Эти индексы используются для проверки ограничения целостности *unique*(уникальность).

Для каждой таблицы можно одновременно иметь 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. Из примера на рис. 4.4 видно, что данные в индексе отсортированы по первому столбцу (ID), внутри группы с одинаковыми значениями ID – отсортированы по второму столбцу (EDATE), а внутри группы с одинаковыми значениями ID и EDATE – по третьему столбцу (CODE).

Таблица				
ID	**EDATE**	**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	01.12.95	A1	Amos ltd.	52.8

Индекс		
ID	EDATE	CODE
100	01.12.95	A4
100	02.12.95	A2
110	10.12.95	A4
200	01.12.95	A2
2000	01.12.95	A4
200	02.12.95	A1

Рис. 4.4. Пример составного индекса

4.5.2.1. 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 хранятся записи, отсортированные по значениям индексируемого атрибута. Тогда для каждой страницы в индексе задаётся 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 записи индекса можно хранить не полное значение ключа, а лишь информацию, позволяющую восстановить его из известного предыдущего значения. Такой индекс называется **сжатым**.
3. **Одноуровневый индекс** представляет собой линейную совокупность значений одного или нескольких полей записи. На практике он используется редко. В развитых СУБД применяются более сложные методы организации индексов. Особенно эффективными являются **многоуровневые индексы** в виде сбалансированных деревьев (B-деревьев, balance trees).

4.5.2.2. 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. Пример построения B-дерева порядка 3 приведён на рис. 4.5.

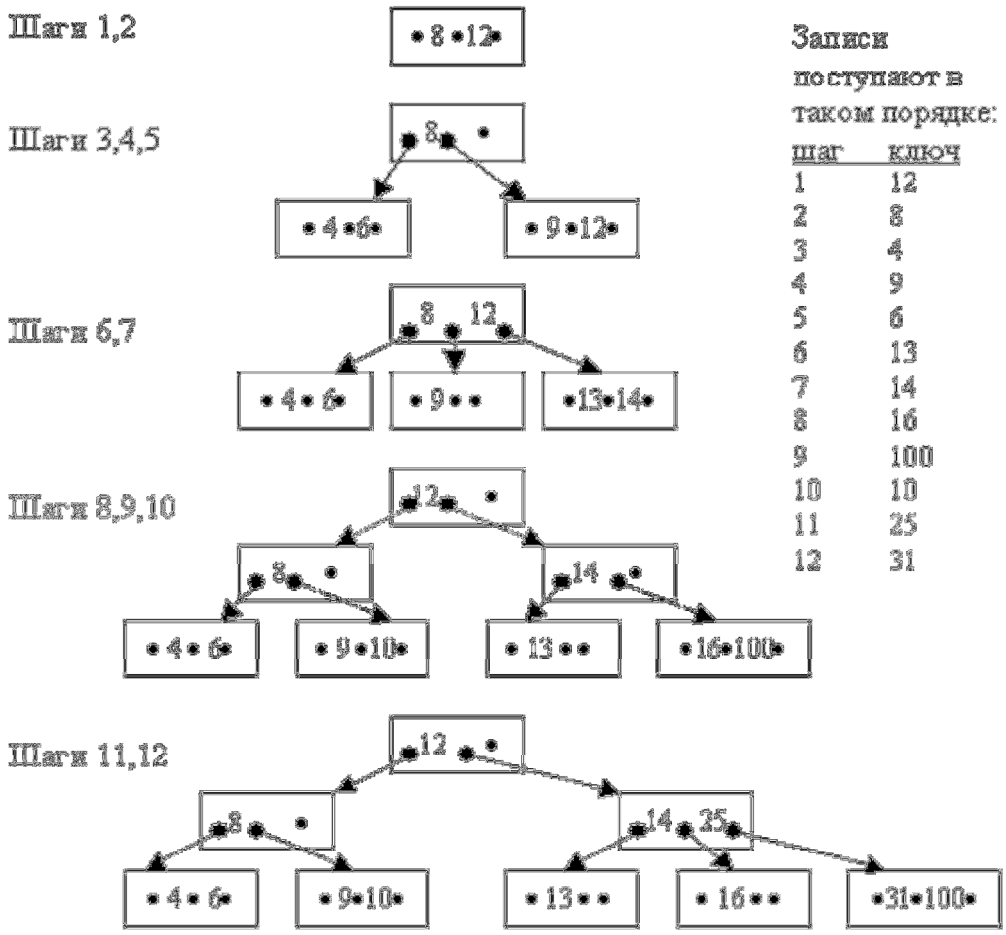

Рис.4.5. Пример построения B-дерева порядка 3

Каждое B-дерево должно удовлетворять следующим условиям:

1. Все конечные вершины расположены на одном уровне, т.е. длина пути от корня к любой конечной вершине одинакова.
2. Каждая вершина может содержать n адресных ссылок и (n-1) ключей. Ссылка влево от ключа обеспечивает переход к вершине дерева с меньшими значениями ключей, а вправо – к вершине с большими значениями.
3. Любая неконечная вершина имеет не менее n/2 подчинённых вершин. (Для деревьев нечётного порядка значение n/2 округляется в большую сторону).
4. Если неконечная вершина содержит k (k<n) ключей, то ей подчинена (k+1) вершина на следующем уровне иерархии.

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.

В качестве конкретного примера рассмотрим индексирование в виде B-дерева, которое используется в СУБД Oracle (рис. 4.6).

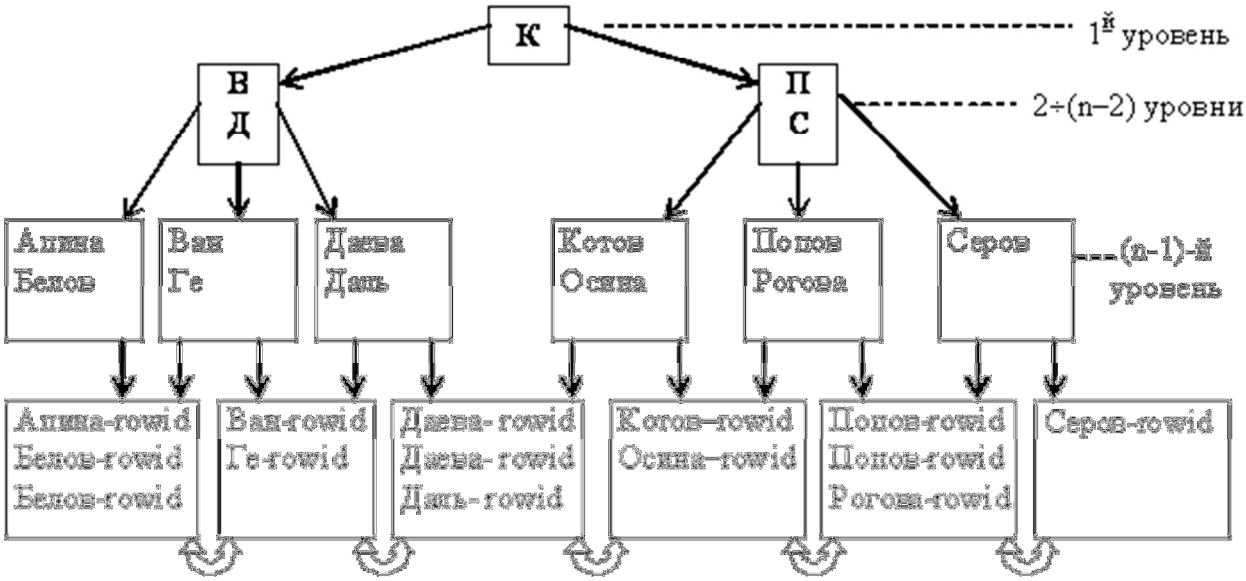

Рис.4.6. Пример индексного блока СУБД Oracle

Организация индексов в СУБД Oracle несколько отличается от рассмот-ренной выше классической организации B-дерева, но принцип остаётся тот же: одинаковое количество уровней на любом пути и автоматическая сбалансированность. Верхние блоки индекса содержат автоматически вычисляемые значения, которые позволяют осуществлять поиск данных. Предпоследний (n-1)-й уровень содержит значения индексируемого поля (атрибута) без повторов (т.е. каждое значение один раз). Самый нижний n-й уровень – блоки-листья, которые содержат индексируемые значения и соответствующие идентификаторы записей RowID (row identification, КБД), используемые для нахождения самих записей. Для неуникальных индексов значения идентификаторов строк (RowID) в блоках-листьях индекса также отсортированы по возрастанию. Блоки-листья связаны между собой двунаправленными ссылками.

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 X0 и конечное значения Xк ключа, а также три указателя. Если значение искомого ключа меньше, чем X0, то происходит обращение по левому указателю; если оно больше, чем Xк, то происходит обращение по правому указателю; если оно попадает в диапазон X0÷Xк – по среднему указателю. Вершины, которые делят следующий уровень дерева на три поддерева, могут занимать несколько уровней. Это зависит от количества индексируемых записей и среднего размера индексируемых значений.

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.

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.

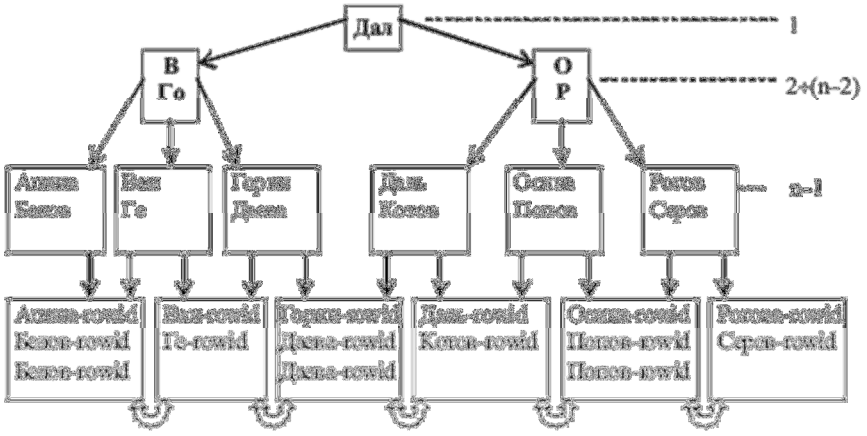

Рис.4.7. Пример перераспределения данных индексного блока СУБД Oracle

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

Структура B-дерева имеет следующие преимущества:

- B-дерево автоматически поддерживается в сбалансированном виде.
- Все блоки-листья в дереве расположены на одном уровне, следовательно, поиск любой записи в индексе занимает примерно одно и то же время.
- B-деревья обеспечивают хорошую производительность для широкого спектра запросов, включая поиск по конкретному значению и поиск в открытом и закрытом интервалах (благодаря ссылкам между блоками-листьями).
- Модификация данных таблицы выполняется достаточно эффективно, т.к. в блоках индекса обычно есть свободное место для размещения новых значений, а полная перестройка дерева выполняется достаточно редко.
- Производительность B-дерева одинаково хороша для маленьких и больших таблиц, и не меняется существенно при росте таблицы.

4.5.2.3. Использование индексов

В системах, поддерживающих язык SQL, индекс создаётся командой *create index*. Синтаксис этой команды следующий:

```
create index <имя_индекса>     on <имя_таблицы>(<поле1> [, <поле2>,...])
        [<параметры>];
```

Имя индекса должно быть уникальным среди имён объектов БД. Если индекс составной, то входящие в него поля перечисляются через запятую. Необязательные <параметры> зависят от используемой СУБД.

Например, с помощью следующей команды можно создать составной индекс для таблицы*СОТРУДНИКИ* (EMP) по полям *Фамилия* (fam) и *Имя* (name):

create index ind_emp_name on emp(fam, name);

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. Например, если запрос выбирает больше половины записей отноше-ния, то извлечение данных через индекс потребует больше времени, чем последовательное чтение данных. Это следует из того, что данные через индекс выбираются не в той последовательности, в которой они хранятся в памяти. Для подобных запросов построение 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 условиях поиска.

- Стоит индексировать столбцы, которые используются для соединения таб-лиц 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 (подробнее о блокировках – раздел 5.4).
- 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 выражений или функций: как правило, это не позволяет 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.

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 определения адреса записи в области хранения к значению ключевого поля этой записи применяется так называемая *хеш-функция* ***h(K)***. Она преобразует значение ключа ***K*** в адрес участка памяти (это называется свёрткой ключа). Новая запись будет размещаться по тому адресу, который выдаст хеш-функция для ключа этой записи. При поиске записи по значению ключа K хеш-функция выдаст адрес, указывающий на начало того участка памяти, в котором надо искать эту запись.

Хеш-функция *h(K)* должна обладать двумя основными свойствами:

1. выдавать такие значения адресов, чтобы обеспечить равномерное распределение записей в памяти, в частности, для близких значений ключа значения адресов должны сильно отличаться, чтобы избегать перекосов в размещении данных:

$K1 \approx K2 \Rightarrow h(K1) >> h(K2)$ V $K2 >> h(K1)$

2. для разных значений ключа выдавать разные адреса:

$K1 \neq K2 \Rightarrow h(K1) \neq h(K2)$

Второе требования является сложно выполнимым. Трудно подобрать такую хеш-функцию, которая для любого распределения значений ключа всегда выдавала бы разные адреса для разных значений. Для реальных функций хеширования допускается совпадение значений функции *h(K)*для различных ключей. Для разрешения неопределённости при совпадении адресов после вычисления *h(K)* используются специальные методы (см. раздел 4.5.3.2).

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 обычно неупорядочены по значению ключа, что приводит к дополнительным затратам, например, при выполнении сортировки. К преимуществам хеширования относится то, что ускоряется доступ к данным по значению ключа. Обращение к данным происходит за одну операцию ввода/вывода, т.к. значение ключа с помощью хеш-функции непосредственно преобразуется в адрес соответствующей записи (или адрес блока памяти, в котором хранится эта запись). При этом не нужно создавать никаких дополнительных структур (типа индекса) и тратить память на их хранение.

4.5.3.1. Методы хеширования

Многочисленные эксперименты с реальными данными 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:

$h(K) = K \bmod M$ (4.1)

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:

$M \neq rk \pm a$,

где *k* и *a* – небольшие числа, а *r* – "основание системы счисления" для большинства используемых литер (как правило, 128 или 256), 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:

$$h(K) = M\left(\left(\frac{A}{w}K\right) \bmod 1\right) \qquad (4.2)$$

где ***w*** – размер машинного слова (обычно, 231); ***A*** – целое число простое по отношению к ***w***; а ***M*** – некоторая степень основания системы счисления ЭВМ (2m). Таким образом, в качестве значения функции берутся ***M*** правых значащих цифр дробной части произведения значения ключа и константы ***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:

$$A(K) = h(K) \bmod N \qquad (4.3)$$

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 каждый раз при размещении новой записи или при поиске существующей, если возникла коллизия. В системах баз данных рехеширование выполняется одним из следующих способов:

1. **Открытая адресация**: новая запись размещается вслед за последней запи-сью 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: значения ключа хеширования не обязательно должны быть уникальными. В реальных базах данных в качестве адреса записи может выступать адрес блока (стра-ницы памяти), в котором размещается несколько записей, возможно, с одинаковым значением ключа. Коллизией в этом случае является ситуация переполнения блока, адрес которого получен в результате применения функции хеширования к значению ключа новой записи. Тогда система выполнит для этой записи рехеширование.

4.5.3.3. Использование хеширования

Хеширование таблицы полезно в следующих случаях:

- В таблице есть уникальный ключ, и большинство запросов обращаются к записям по значению этого ключа, например:

  ```
  SELECT <список выбора>  FROM <таблица>  WHERE unique_key = <значение>;
  ```

  Значение, указанное в условии, хешируется; по этому хеш-значению происходит прямой доступ к соответствующему блоку данных (обычно, одно физическое чтение, если нет коллизий и запись помещается в одном блоке).

- Для неуникального хеш-ключа все записи с таким значением ключа помещаются в одном блоке, который также можно прочитать за один раз.
- Таблица практически статична (редко обновляется). Число записей и их средний размер можно определить заранее и сразу выделить под таблицу требуемое физическое пространство.

  Хеширование не рекомендуется в следующих случаях:

- Нельзя сразу выделить столько памяти, сколько требуется таблице. Если потребуется выделять таблице дополнительную память, эта память будет отведена под коллизионные страницы, что сильно ухудшит производительность (это следует из формулы (4.3), по которой рассчитывается адрес записи)./li>
- Большинство запросов выбирают записи в некотором интервале значений ключа. Хеширование не даёт здесь преимуществ, т.к. записи обычно не упорядочены, и система использует последовательное чтение.

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 хеш-функция 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 (тогда данные будут размещаться в порядке увеличения значений хеш-ключа).

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).

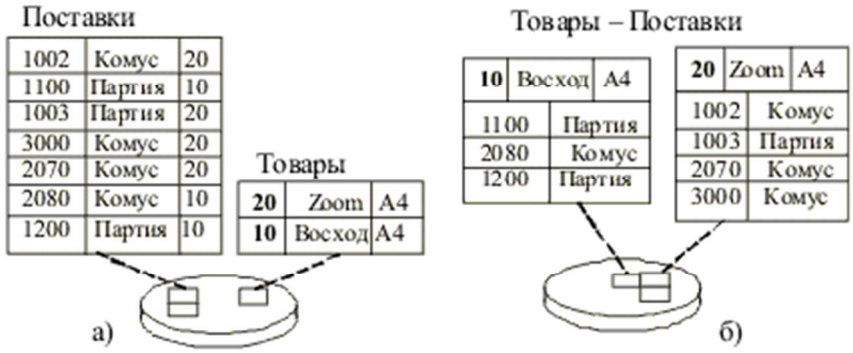

Рис.4.8. Некластеризованные (а) и кластеризованные (б) данные

Кластерный ключ (КК) – это поле или набор полей, общих для всех кластеризуемых таблиц. Каждая таблица, хранимая в кластере, должна иметь поля, соответствующие типам и размерам полей кластерного ключа. Количество полей в кластерном ключе ограничено (например, для СУБД Oracle8 это ограничение равно 16).

Совместное хранение данных 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. Поэтому соединение кластеризованных таблиц по сравнению с раздельно хранимыми таблицами выполняется в 3-6 раз быстрее.

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 из-менять размер блока (в частности, СУБД Oracle), при создании кластера желательно установить размер блока исходя из оценки среднего объёма записей с одинаковыми значениями кластерного ключа. Если же записи с одинаковым значением КК занимают только часть блока (например, в среднем 1К при размере блока 4К), то при создании таблицы кластера можно указать количество значений КК на один блок.

Значения кластерного ключа таблицы могут обновляться. Но это обновление может вызвать физическое перемещение записи, т.к. 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 время соединения таблиц по значению кластерного ключа.
- Каждое значение кластерного ключа хранится только один раз, за счёт чего достигается экономия памяти.

С другой стороны, наличие кластеров обычно увеличивает время выполнения операции добавления записи (INSERT): система тратит дополнительное время на поиск блока, в который нужно поместить новую запись.

4.5.4.2. Использование кластеризации

Кластер создаётся с помощью команды CREATE CLUSTER:

```
create cluster <имя_кластера> (<имя_поля1> <тип_поля1>  [,<имя_поля2>
<тип_поля2> ,…] );
```

Здесь в скобках перечисляются поля кластерного ключа. Затем создаются таблицы в кластере:

```
create table <имя_таблицы> (<список полей таблицы>)  cluster <имя_кластера>
(<список полей КК>);
```

Количество и типы полей кластерного ключа таблицы должны совпадать с количеством и типами полей КК в определении кластера, а имена полей могут быть другими. Типы данных в <списке полей КК> для таблицы не указываются. Перед занесением данных в таблицы кластера необходимо создать кластерный *индекс* – индекс по кластерному ключу:

```
create index <имя_индекса> on cluster <имя_кластера>;
```

Поля для индексирования не указываются, потому что кластерный индекс создаётся по полям кластерного ключа. В отличие от обычного индекса в кластерном индексе null-значения индексируются.

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

- Если данные в кластерном ключе этих таблиц часто обновляются. 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 перемещение строк.
- Если часто требуется полный просмотр отдельной таблицы. Полный просмотр индивидуальных таблиц кластера требует больше времени, чем просмотр раздельно хранящихся 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.
- Если суммарные данные таблиц с одним и тем же значением кластерного ключа занимают больше одного блока данных. Второй и последующие блоки для одного и того же значения кластерного ключа выделяются не подряд, что вызывает частые перемещения считы-вающей головки диска и увеличение времени доступа к данным.

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

Обратите внимание:

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 данных и методах доступа к данным хранится в словаре-справочнике данных и используется системой при выполнении запросов.

Для кластеризованных и хешированных таблиц можно строить дополнительные индексы по полям, не входящим в кластерный ключ и не являющихся ключом хеширования. Это также относится к преимуществам кластеризации и хеширования и позволяет устранить некоторые присущие им недостатки.

"Кто хочет работать – ищет
средства, кто не хочет – причины".
С.П. Королёв, советский ученый и

конструктор в области
космонавтики

5. МНОГОПОЛЬЗОВАТЕЛЬСКИЙ ДОСТУП К ДАННЫМ

Данные в БД являются разделяемым ресурсом. 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. Для каждого запроса выделяется некоторое количество процессорного времени (квант времени), по истечении которого выполнение запроса приостанавливается, он ставится в очередь запросов, а на выполнение запускается следующий по очереди запрос. Т.о., процессорное время делится между запросами, и создаётся иллюзия, что запросы выполняются одновременно.

При параллельном доступе к данным запросы на чтение не мешают друг другу. Наоборот, если один запрос считал данные в оперативную память (в буфер данных), то другой запрос не будет тратить время на обращение к диску за этими данными, а получит их из буфера данных. Проблемы возникают в том случае, если доступ подразумевает внесение изменений. Для того чтобы исключить нарушения логической целостности данных при многопользовательском доступе, используется механизм транзакций.

5.1. Механизм транзакций

Транзакция – это упорядоченная последовательность операторов обработки данных, которая переводит базу данных из одного согласованного состояния в другое.

Все команды работы с данными выполняются в рамках транзакций. Для каждого сеанса связи с БД в каждый момент времени может существовать единственная транзакция или не быть ни одной транзакции.

Транзакция обладает следующими свойствами:

1. Логическая неделимость (**атомарность**, Atomicity) означает, что выполняются либо все операции (команды), входящие в транзакцию, либо ни одной. Система гарантирует невозможность запоминания части изменений, произведённых транзакцией. До тех пор, пока транзакция не завершена, её можно "откатить", т.е. отменить все сделанные командами транзакции изменения. Успешное выполнение

транзакции (фиксация) означает, что все команды транзакции проанализированы, интерпретированы как правильные и безошибочно исполнены.

2. **Согласованность** (Consistency): транзакция начинается на согласованном множестве данных и после её завершения множество данных согласовано. Состояние БД является согласованным, если данные удовлетворяют всем установленным ограничениям целостности и относятся к одному моменту в состоянии предметной области.
3. **Изолированность** (Isolation), т.е. отсутствие влияния транзакций друг на друга. (На самом деле это влияние существует и регламентируется стандартом: см. раздел 5.3. "Уровни изоляции транзакций").
4. **Устойчивость** (Durability): результаты завершённой 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 (по первым буквам названий свойств).

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

- **фиксация** транзакции (запоминание изменений): COMMIT [WORK];
- **откат** транзакции (отмена изменений): ROLLBACK [WORK];
- создание точки сохранения: SAVEPOINT <имя_точки_сохранения>;

(Ключевое слово WORK необязательно). Для фиксации или отката транзакции система создаёт неявные точки фиксации и отката (рис. 5.1).

Рис.5.1. 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 последнейзавершённой команде в транзак-ции. 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.

Для обеспечения целостности транзакции СУБД может откладывать за-пись изменений в БД до момента успешного выполнения всех операций, вхо-дящих в транзакцию, и получения команды подтверждения транзакции (commit). Но

чаще используется другой подход: система записывает изменения в БД, не дожидаясь завершения транзакции, а старые значения данных сохраняет на время выполнения транзакции в сегментах отката.

Сегмент отката (rollback segment, RBS) – это специальная область памяти на диске, в которую записывается информация обо всех текущих (незавершённых) изменениях. Обычно записывается "старое" и "новое" содержимое изменённых записей, чтобы можно было восстановить прежнее состояние БД при откате транзакции (по команде rollback) или при откате текущей операции (в случае возникновения ошибки). Данные в RBS хранятся до тех пор, пока транзакция, изменяющая эти данные, не будет завершена. Потом они могут быть перезаписаны данными более поздних транзакций.

Команда 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>). На одну транзакцию может быть несколько точек сохранения (ограничение на их количество зависит от СУБД).

Для сохранения сведений о транзакциях СУБД ведёт журнал транзакций. **Журнал транзакций** – это часть БД, в которую поступают данные 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 фиксацией или откатом, не завершена, находится в состоянии ожидания);
- точки сохранения (явные и неявные);
- 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 (exit, quit), в рамках которой выполнялась транзакция. В этом случае транзакция фиксируется автоматически.
- Программа, выполняющая транзакцию, завершена аварийно (abort). При этом транзакция автоматически откатывается.

Примечания:

1. Возможна работа в режиме AUTOCOMMIT, когда каждая команда воспринимается системой как транзакция. В этом режиме пользователи меньше задерживают друг друга, требуется меньше памяти для сегмента отката, зато результаты ошибочно вы-полненной операции нельзя отменить командой rollback.
2. 2. В некоторых СУБД реализованы расширенные модели транзакций, в которых суще-ствуют дополнительные ситуации фиксации транзакций. Например, в СУБД Oracle команды DDL выполняются в режиме AUTOCOMMIT, т.е. не могут быть откачены.

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 СУБД Oracle КТ формируется:

- 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, помечаются как постоянные.
2. Уничтожаются все точки сохранения для данной транзакции.
3. Если выполнение транзакций осуществляется с помощью блокировок, то освобождаются объекты, заблокированные транзакцией (см. раздел 5.5).
4. В журнале транзакций транзакция помечается как завершенная, уничтожаются системные записи о транзакции в оперативной памяти.

А при откате транзакции вместо п.1 обычно выполняется считывание из сегмента отката прежних значений данных и переписывание их обратно в БД (остальные пункты сохранятся без изменений). Поэтому откат транзакции практически всегда занимает больше времени, чем фиксация.

5.2. 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 обра-щаются к одному и тому же набору данных и хотя бы одна из транзакций изменяет данные.

В общем случае взаимовлияние транзакций может проявляться в виде:

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

Потеря изменений могла бы произойти при одновременном обнов-лении 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 = 39200  WHERE   Номер = 1123;
транзакция 2 – UPDATE  СОТРУДНИКИ  SET Должность = "старший экономист"  WHERE
       Номер = 1123;
```

Обе транзакции считали одну и ту же запись (1123, "Рудин В.П.", "эконо-мист", 28300) и внесли каждая свои изменения: в бухгалтерии изменили оклад (транзакция 1), в отделе кадров – должность (транзакция 2). Результаты транзакции 1 будут потеряны (рис. 5.2).

Отношение "Сотрудники"

Номер	ФИО	Должность	Оклад	
1123	Рудин В.П.	экономист	28300	
1123	Рудин В.П.	экономист	**39200**	Транзакция 1
1123	Рудин В.П.	**старший экономист**	28300	Транзакция 2

Рис.5.2. 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 общий 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.

Для разграничения двух пишущих транзакций и предотвращения потери изменений СУБД используют механизмы блокировок или временных отметок, а для разграничения пишущей и читающих транзакций – специальные правила поведения транзакций, которые называются уровнями изоляции транзакций.

5.3. Уровни изоляции транзакций

Стандарт ANSI/ISO для SQL устанавливает различные уровни изоляции для операций, выполняемых над БД, 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. 5.1). Использование уровней изоляции обеспечивает предсказуемость работы приложений.

Таблица 5.1. 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 в СУБД обычно установлен 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.

5.4. Блокировки

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

Блокировка относится к пессимистическим алгоритмам, т.к. предполагается, что существует высокая вероятность одновременного обращения нескольких пишущих транзакций к одним и тем же данным. 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. Исключающая (монопольная) блокировка таблицы накладывается, например, в случае выполнения операции ALTER TABLE, т.е. изменения структуры таблицы. На отдельные записи (блоки) монопольная блокировка накладывается тогда, когда эти записи (блоки) подвергаются модификации.

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 TABLE 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.

Явную блокировку также можно наложить с помощью ключевых слов *for update*, например:

```
for update, например:  SELECT *  FROM <имя_таблицы>  WHERE <условие>  for update;
```

При этом блокировка будет накладываться на те записи, которые удовлетворяют <условию>.

И явные, и неявные блокировки снимаются при завершении транзакции.

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 (first input – first output): "первый пришел – первый ушел".

Тупиковые ситуации (deadlocks) возникают при взаимных блоки-ровках транзакций, которые выполняются на пересекающихся множествах данных (рис. 5.3). Здесь приведён пример взаимной блокировки трех транзакций Ti на отношениях Rj. Транзакция T1 заблокировала данные B1 в от-ношении R1 и ждёт освобождения данных B2 в отношении R2, которые заблокированы транзакцией T2, ожидающей освобождения данных B3 в отношении R3, заблокированных транзакцией T3, которая не может продолжить выполнение

из-за транзакции T1. Если не предпринимать никаких дополнительных действий, то эти транзакции никогда не завершатся, т.к. они вечно будут ждать друг друга.

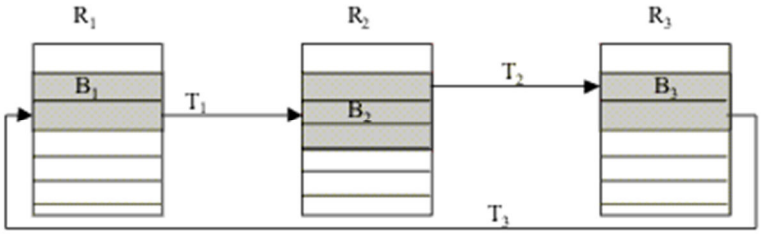

Рис.5.3. Взаимная блокировка трех транзакций

Существует много стратегий разрешения проблемы взаимной блокировки, в частности:

1. Транзакция запрашивает сразу все требуемые блокировки. Такой метод снижает степень параллелизма в работе системы. Также 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 с последующим рестартом через случайный промежуток времени. 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.

5.5. Временные отметки

Использование временных отметок относится к оптимистическим алго-ритмам разграничения транзакций. Для их эффективного функционирования необходимо, чтобы вероятность одновременного обращения нескольких пишущих транзакций к одним и тем же данным была невелика.

Временная отметка – это уникальный идентификатор, который СУБД создаёт для обозначения относительного момента запуска транзакции. Временная отметка может быть создана с помощью системных часов или путём присвоения каждой следующей транзакции очередного номера (SCN – system change number). Каждая транзакция *Ti* имеет временную отметку *ti*, и каждый элемент данных в БД (запись или блок) имеет две отметки: *tread(x)* – временная отметка транзакции, которая последней считала элемент *x*, и *twrite(x)* – временная отметка транзакции, которая последней записала элемент x.

При выполнении транзакции *Ti* система сравнивает отметку *ti* и отметки *tread(x)* и *twrite(x)*элемента ***x*** для обнаружения конфликтов:

1. для читающей транзакции ***Ti***: если ***ti <twrite(x)***, то элемент данных ***x*** перезаписан более поздней транзакцией, и его значение может оказаться несогласованным с теми данными, которые эта транзакция уже успела прочитать.
2. для пишущей транзакции:

- если ***ti < tread(x)***, то элемент данных x считан более поздней транзакцией. Если транзакция ***T***изменит значение элемента ***x***, то в другой транзакции может возникнуть ошибка.
- если *ti < twrite(x)*, то элемент x перезаписан более поздней транзакцией, и транзакция ***T***пытается поместить в БД устаревшее значение элемента ***x***.

Во всех случаях обнаружения конфликта система перезапускает текущую транзакцию ***Ti*** с более поздней временной отметкой. Если конфликта нет, то транзакция выполняется. Очевидно следующее: если разные транзакции часто обращаются к одним и тем же данным одновременно, то транзакции часто будут перезапускаться, и эффективность такого механизма будет невелика.

5.6. Многовариантность

Для увеличения эффективности выполнения запросов некоторые СУБД используют алгоритм многовариантности. Этот алгоритм позволяет обеспечивать согласованность данных при чтении, не блокируя эти данные.

Согласованность данных для операции чтения заключается в том, что все значения данных должны относиться к тому моменту, когда начиналась эта операция. Для этого можно предварительно запретить другим транзакциям изменять эти данные до окончания операции чтения, но это снижает степень параллельности работы системы.

При использовании алгоритма многовариантности каждый блок данных хранит номер последней транзакции, которая модифицировала данные, хранящиеся в этом блоке (SCN – system change number). И каждая транзакция имеет свой SCN. При чтении данных СУБД сравнивает номер транзакции и номер считываемого блока данных:

- если блок данных не модифицировался с момента начала чтения, то данные считываются из этого блока;
- если данные успели измениться, то система обратится к сегменту отката и считает оттуда значения данных, относящиеся к моменту начала чтения.

Недостатком этого метода является возможность возникновения ошибки при чтении данных, если старые значения данных в сегменте отката будут перезаписаны. При этом будет выдано сообщение об ошибке и операцию чтения

придётся перезапускать вручную. Для устранения подобных проблем можно увеличить размер сегмента отката или разбить одну большую операцию чтения на несколько (но при этом согласованность данных обеспечиваться не будет).

"Стыдно не уметь защищать себя рукою, но ещё более стыдно не уметь защищать себя словом".

Аристотель, древнегреческий философ

6. 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 (физической целостности данных);
- обеспечение секретности данных.

6.1. 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 в языке SQL:

1. Уникальность значения первичного ключа (PRIMARY KEY).
2. Уникальность ключевого поля или комбинации значений ключевых полей:

   ```
   UNIQUE(A),
   ```

   где A – один или несколько атрибутов, указанных через запятую.

   (1,2 – явные структурные ограничения целостности.)

3. Обязательность/необязательность значения (NOT NULL/NULL).
4. Задание диапазона значений атрибута Field:

```
CHECK(field  BETWEEN  min_value  AND  max_value)
```

5. Задание взаимоотношений между значениями атрибутов Field1 и Field2:

```
CHECK (field1 @ field2),  где @ – оператор отношения (например, знак
">").
```

6. Задание списка возможных значений (констант) для атрибута Field:

```
CHECK (field IN (value1, value2,…, valueN)).
```

7. Определение формата атрибута Field (даты, числа и др.). Например:

```
CHECK (field LIKE '_ _ _-_ _-_ _') -- формат телефонного номера
```

8. Определение домена атрибута на основе значений другого атри
   Определение формата атрибута Field (даты, числа и др.). Например:
9. бута: множество значений некоторого атрибута отношения является подмножеством значений другого атрибута этого или другого отношения (внешний ключ, FOREIGN KEY)./li>

   (3.-8. – явные ограничения целостности на значения данных.)

10. Ограничения на обновление данных (например, каждое следующее значение атрибута должно быть больше предыдущего). В SQL напрямую не реализуется, требует использования специальных возможностей СУБД (триггеров).
11. 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.

СУБД проводит проверку выполнения ограничений целостности для команд DDL до выполнения команды, а для команд DML либо сразу после выполнения команды, либо после выполнения всей транзакции. (По стан-дарту ISO этим можно управлять; по умолчанию проверка проводится после каждой операции DML).

6.2. Обеспечение безопасности данных

Под функцией безопасности (или физической защиты) данных подразу-мевается предотвращение разрушения или искажения данных в результате программного или аппаратного сбоя. Обеспечение безопасности является внутренней задачей СУБД, поскольку связано с её нормальным функционированием, и решается на

уровне СУБД. Цель **восстановления базы данных** после сбоя – обеспечить, чтобы результаты всех подтверждённых транзакций были отражены в восстановленной БД, и вернуться к нормальному продолжению работы как можно быстрее, изолируя пользователей от проблем, вызванных сбоем.

6.2.1. Виды сбоев

В СУБД предусмотрены специальные механизмы, призванные нивелировать последствия сбоев в работе базы данных. Рассмотрим наиболее типичные сбои и способы защиты от них:

1. Сбой предложения.

   Сбой происходит при логической ошибке предложения во время его обработки (например, предложение нарушает ограничение целостности таблицы). Когда возникает сбой предложения, СУБД автоматически откатывает результаты этого предложения, генерирует сообщение об ошибке и возвращает управление пользователю (приложению пользователя).

2. Сбой пользовательского процесса.

   Это ошибка в процессе (приложении), работающем с БД, например, аварийное разъединение или прекращение процесса. Сбившийся процесс пользователя не может продолжать работу, тогда как СУБД и процессы других пользователей могут. Система автоматически откатывает неподтверждённые транзакции сбившегося пользовательского процесса и освобождает все ресурсы, занятые этим процессом.

3. Сбой процесса сервера.

   Такой сбой вызван проблемой, препятствующей продолжению работы сервера. Это может быть аппаратная проблема, такая как отказ питания, или программная проблема, такая как сбой операционной системы. Восстановление после сбоя процесса сервера может потребовать перезагрузки БД, при этом автоматически происходит откат всех незавершённых транзакций.

4. Сбой носителя (диска).

   Эта ошибка может возникнуть при попытке записи или чтения файла, необходимого для работы базы данных (файла БД, файла журнала транзакций и проч.). Типичным примером является отказ дисковой головки, который приводит к потере всех файлов на данном устройстве. В этой ситуации сервер БД не может продолжать работу, и для восстановления базы данных требуется участие человека (обычно, администратора базы данных, АБД).

5. Ошибка пользователя.

Например, пользователь может случайно удалить нужные записи или таблицы. Ошибки пользователей могут потребовать участия человека (АБД) для восстановления базы данных в состояние на момент возникновения ошибки.

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

6.2.2. Средства физической защиты данных

В качестве средств физической защиты данных чаще всего приме-няются 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 (РК) не должна создаваться на том же диске, на котором находится сама БД, т.к. при аварии диска базу невозможно будет восстановить. В случае сбоя (или аварии диска) БД восстанавливается на основе последней копии.

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. Резервная копия может быть **инкрементной**: она 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 создания полной резервной копии.

Создание частичной и инкрементной РК выполняется средствами СУБД, а создание полной РК – средствами СУБД или ОС (например, с помощью команды copy). В резервную копию, созданную средствами СУБД, обычно включаются только те блоки памяти, которые реально содержат данные (т.е. пустые блоки, выделенные под объекты БД, в резервную копию не входят).

Периодичность резервного копирования определяется администратором системы и зависит от многих факторов: объём БД, интенсивность запросов к БД, интенсивность обновления данных и др. Как правило, технология проведения резервного копирования такова:

- раз в неделю (день, месяц) осуществляется полное копирование;
- раз в день (час, неделю) – частичное или инкрементное копирование.

Все изменения, произведённые в данных после последнего резервного копирования, утрачиваются; но при наличии **архива журнала транзакций** их можно выполнить ещё раз, обеспечив полное восстановление БД на момент

возникновения сбоя. Дело в том, что журнал транзакций содержит сведения только о текущих транзакциях. После завершения транзакции информация о ней может быть перезаписана. Для того чтобы в случае сбоя обеспечить возможность полного восстановления БД, необходимо вести архив журнала транзакций, т.е. сохранять копии файлов журнала транзакций вместе с резервной копией базы данных.

6.2.3. Восстановление базы данных

В том случае, если нельзя восстановить БД после сбоя автоматически, восстановление БД выполняется в два этапа:

1. перенос на рабочий диск резервной копии базы данных (или той её части, которая была повреждена);
2. перезапуск сервера БД с повторным проведением всех транзакций, зафиксированных после создания резервной копии и до момента возникновения сбоя.

Если в системе есть архив транзакций, то повторное проведение транзакций может проходить автоматически или под управлением пользователя.

Если произошёл сбой процесса сервера, то требуется перезагрузка сервера для восстановления БД. При перезагрузке СУБД может по содержимому системных файлов узнать, что произошёл сбой, и выполнить восстановление автоматически (если это возможно). Восстановление БД в этой ситуации означает приведение всех данных в БД в согласованное состояние, т.е. откат незавершённых транзакций и проверку того, что все изменения, внесённые завершёнными транзакциями, попали на диск.

Для оптимизации регистрации изменений некоторые СУБД могут запи-сывать в журнал информацию о незавершённых транзакциях, предвидя их завершение. Более того, не дожидаясь подтверждения транзакции, СУБД переписывает на диск модифицированные блоки (при формировании контрольной точки). Поэтому в каждый момент времени в журнале тран-закций и в БД может находиться небольшое число записей, модифицированных незавершёнными транзакциями. Эти записи помечаются соответствующим образом. С другой стороны, т.к. изменения 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 в результате сбоя могут возникнуть две потенциальные ситуации:

- Блоки, содержащие подтверждённые модификации, не были записаны в файлы данных, так что эти изменения отражены лишь в журнале тран-закций. Следовательно, журнал транзакций содержит подтверждённые данные, которые должны быть переписаны в файлы данных.
- Журнал транзакций и блоки данных содержат изменения, которые не были подтверждены. Изменения, внесенные неподтверждёнными

транзакциями, во время восстановления БД должны быть удалены из файлов данных.

Для того чтобы разрешить эти ситуации, СУБД автоматически выполняет два этапа при восстановлении после сбоев: прокрутку вперед и прокрутку назад.

1. *Прокрутка вперед* заключается в применении к файлам данных всех изме-нений, зарегистрированных в журнале транзакций. После прокрутки вперед файлы данных содержат все как подтверждённые, так и неподтверждённые изменения, которые были зарегистрированы в журнале транзакций.
2. *Прокрутка назад* заключается в отмене всех изменений, которые не были подтверждены. Для этого используются журнал транзакций и сегменты отката, информация из которых позволяет определить и отменить те транзакции, которые не были подтверждены, хотя и попали на диск в файлы БД.

После выполнения этих этапов восстановления БД находится в согласованном состоянии и с ней можно работать.

6.3. Защита от несанкционированного доступа

Под функцией секретности данных понимается защита данных от пред-намеренного искажения и/или доступа пользователей или посторонних лиц. Для этого вся информация делится на общедоступные данные и конфиденци-альные, 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 от несанкционированного доступа. Общий принцип управления доступом к базе данных такой: 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:

- назначение отдельным группам пользователей прав доступа (привилегий) к отдельным группам данных в соответствии с правилами ПО;
- организация системы контроля доступа к данным;
- тестирование вновь создаваемых средств защиты данных;
- периодическое проведение проверок правильности работы системы за-щиты, исследование и предотвращение сбоев в её работе.

Примечание: администратор БД обычно назначает права доступа в соответствии с проектом БД, который должен включать перечень групп пользователей и их привилегии.

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 преду-сматривается возможность очень ограниченного доступа к данным постороннего человека, не требующая идентификации (доступ типа "гость").

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 подтвер-ждения того, что это имя ввёл 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 зна-ков, пароль не может содержать пробелы;
- пароли должны часто меняться.

Для контроля выполнения этих требований обычно применяются специаль-ные программы.

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 высоких требований к конфиденциальности данных.

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

1. **GRANT** – предоставление одной или нескольких привилегий пользова-телю (или группе пользователей):

```
GRANT { <список привилегий> | ALL PRIVILEGES }  ON <имя объекта>  TO
{<список пользователей> | PUBLIC}  [WITH GRANT OPTION];
```

где <список привилегий> – набор прав, которые необходимо предоставить, или ALL PRIVILEGES – все права на данный объект; <имя объекта> – имя объекта БД, к которому предоставляется доступ; <список пользователей> – перечень пользователей (или ролей, см. дальше), которым будут предоставлены указанные права; PUBLIC – предопределённый пользователь, привилегии которого доступны всем пользователям БД. WITH GRANT OPTION – ключевые слова, дающие возможность пользова-телям из списка пользователей предоставлять назначенные права другим пользователям (т.е. передавать эти права).

Права, подразумеваемые под словами ALL PRIVILEGES, зависят от типа объекта. Примерный перечень прав в зависимости от типа объекта БД приведён в табл. 6.1.

Таблица 6.1. Использование объектных привилегий

Привилегия	Операции	Таблицы	Представления	Процедурные объекты
ALTER	изменение определения объекта	+	+	+
DELETE	удаление данных	+	+	...
EXECUTE	выполнение объекта	...	...	+
INSERT	добавление данных	+	+	...
SELECT	чтение данных	+	+	...
UPDATE	изменение данных	+	+	...

Примечание: процедурные объекты – это хранимые процедуры и функции.

2. **REVOKE** – отмена привилегий:

```
REVOKE [GRANT OPTION FOR]  { <список привилегий> | ALL PRIVILEGES }  ON <имя
объекта>  FROM {<список пользователей> | PUBLIC}  { RESTRICT | CASCADE };
```

где [GRANT OPTION FOR] – отмена права передачи привилегий; CASCADE – при отмене привилегий у пользователя отменяются все приви-легии, которые он передавал другим пользователям; RESTRICT – если при отмене привилегий у пользователя необходимо отменить переданные другим пользователям привилегии, то операция завершается с ошибкой.

Другие ключевые слова имеют то же значение, что и в команде GRANT.

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 отнести его к определённой группе пользователей, обладающей набором прав в соответствии с задачами, которые на неё возложены.

Кроме привилегий на доступ к объектам СУБД ещё может поддерживать так называемые*системные привилегии*: это права пользователя на создание/изменение/удаление (create/alter/drop) объектов различных типов. В некоторых системах такими привилегиями обладают только пользователи, включённые в группу АБД. Другие СУБД предоставляют возможность назначения дифференцированных системных привилегий любому пользователю в случае такой необходимости. Например, в СУБД Oracle права на создание таблиц и представлений пользователю manager можно предоставить с помощью той же команды GRANT, только без указания объекта:

```
GRANT create table, create view  TO manager;
```

"Плох тот план, который не допускает изменений".

Публиус Сирус, древнеримский мыслитель

7. ОПТИМИЗАЦИЯ РЕЛЯЦИОННЫХ ЗАПРОСОВ

В оптимизацию реляционных запросов входят два различных аспекта. Во-первых, это внутренняя задача СУБД, которая заключается в опре-делении 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.

7.1. Этапы оптимизации запросов в реляционных СУБД

Язык SQL является декларативным языком. В его командах отсутствует информация о том, как выполнить запрос, какие методы доступа к данным использовать. А почти каждая команда SQL из подмножества языка манипулирования данными (DML) может быть выполнена разными способами. Рассмотрим следующий запрос:

Пример 7.1. Список сотрудников 4-го отдела не старше 25 лет с «чистой» зарплатой не менее 30000 рублей" («чистой» называется зарплата после уплаты подоходного налога, в настоящее время – 13%).

```
        SELECT *  FROM Emp  WHERE depNo=4 AND born>'1985/01/01'  AND
salary*0.87>=30000;
```

В этом запросе к таблице Emp (Сотрудники) указаны условия на поля depNo (Номер отдела), born (Дата рождения) и salary (Зарплата). Если по этим полям есть индексы, то способы выполнения этого запроса могут быть такими:

1. Найти по индексу INDEX(depNo) записи, удовлетворяющие первому условию, и проверить для найденных записей второе условие.
2. Найти по индексу INDEX(born) записи, удовлетворяющие второму условию, и проверить для найденных записей первое условие.
3. Последовательно считать все записи таблицы Emp и проверить для каждой записи оба условия.

Индексом по полю salary система воспользоваться не может, т.к. это поле находится внутри выражения.

Цель СУБД – выполнить запрос, причём сделать это как можно более эффективным способом. Итак, под **оптимизацией**понимается построение квазиоптимального процедурного плана выполнения декларативного запроса.

Примечание: квазиоптимальным план является потому, что система не гарантирует, что она для любого запроса выберет оптимальный план. Для гарантированного выбора опти-мального плана необходимо рассмотреть все возможные планы и сравнить их, а это может потребовать больше времени, чем выполнение самого запроса. Поэтому СУБД выбирает и анализирует лишь несколько планов выполнения каждого запроса, и сре-ди них может не оказаться оптимального плана.

План выполнения запроса состоит из последовательности шагов, каж-дый из которых либо физически извлекает данные из памяти, либо делает подготовительную работу. Построением этого плана занимается оптимизатор – специальная компонента СУБД.

Обработка запроса, поступившего в РСУБД и представленного на декларативном языке запросов, состоит из этапов (фаз), представленных на рис. 7.1.

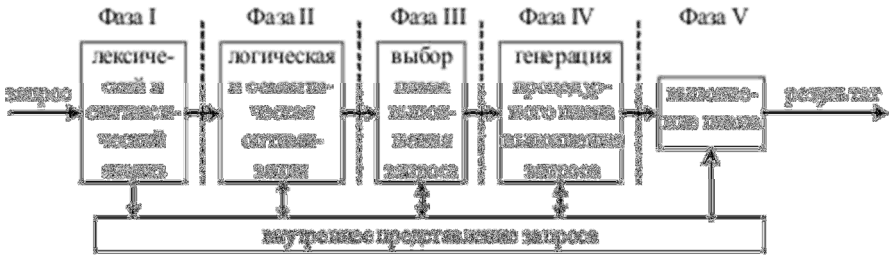

Рис. 7.1. Последовательность выполнения запросов в реляционных СУБД

На первой фазе запрос, представленный на языке запросов, подвер-гается 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 7.2). После проведения эквивалентных преобразований получается внутреннее представление, семантически эквивалентное начальному запросу.

Преобразования могут также использовать информацию об ограни-чениях 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 запросу. Но система гарантирует, что результат выполнения преобразованного запроса совпадает с результатом запроса в начальной форме при соблюдении ограничений целостности, существующих в базе данных. Например, если для таблицы Emp определено такое ограничение целостности:

```
(CHECK (salary>9500 AND salary<80000),
```

то система к запросу из примера 7.1 могла бы добавить эти условия в часть WHERE, чтобы иметь возможность использовать индекс по полю salary.

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 процедурных планов выполнения данного запроса в соответствии с его внутренним 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 за-проса.

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 альтернативных планов выбирается наиболее оптимальный с точки зрения некоторого (заранее выбранного или заданного) критерия.

На четвертом этапе по внутреннему представлению наиболее оптималь-ного 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 выполнения за-проса. Результат помещается в специальную область ОП (т.н. *курсор*).

7.2. Преобразования операций реляционной алгебры

Операндами операций реляционной алгебры (РА) являются отношения, т.е. неупорядоченные множества кортежей. Для выполнения операций необходимо просмотреть все кортежи исходного отношения (или отношений). Следствием этого является большая размерность операций РА. Уменьшения размерности 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:

sF(R1 **U** 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:

sF(R1)**U** sF(R2)

то после селекции останется по 10 записей из каждого отношения, объеди-нение 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 объёму 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 Ri и константы. Каждое выражение реляционной алгебры определяет отображение кортежей переменных-отношений Ri (i=1,…,n) в кортежи единственного отношения, которое получается в результате подстановки кортежей каждого Ri 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×R2=R2×R1

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

   R1><FR2= R2><FR1

3. Закон ассоциативности для декартовых произведений:

   (R1×R2)×R3=R1×(R2×R3)

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

   (R1><F1R2) ><F2R3= R1><F1(R2><F2R3)

5. Комбинация селекций (каскад селекций):

   sF1(sF2(R))= sF1vF2(R)

6. Комбинация проекций (каскад проекций):

   pA1,A2,...Am(pB1,B2,...Bn(R))=pA1,A2,...Am(R), где {Am}d{Bn}

7. Перестановка селекции и проекции:

   sFpA1,A2,...,Am(R))= pA1,A2,...,Am(sF(R))

8. Перестановка селекции с объединением:

   sF(R1**U**R2)=sF(R1)**U**sF(R2)

9. Перестановка селекции с декартовым произведением:
   - sF(R1×R2)= (sF1(R1))×(sF2(R2))

     (если F = F1vF2, где F1 содержит атрибуты, присутствующие только в R1, а F2 содержит атрибуты, присутствующие только в R2);

   - sF(R1×R2)=(sF(R1))×R2

     (если F содержит атрибуты, присутствующие только в R1);

   - sF(R1×R2)=R1×(sF(R2))

     (если F содержит атрибуты, присутствующие только в R2);

   - sF(R1×R2)= sF2(sF1(R1)×R2)

     (если F = F1vF2, где F1 содержит атрибуты, присутствующие только в R1, а F2 содержит атрибуты, присутствующие и в R1, и в R2).

10. Перестановка селекции с разностью:

    sF(R1-R2)=sF(R1)-sF(R2)

11. Перестановка проекции с декартовым произведением:

    pA1,A2,...,Am(R1×R2)= (pB1,B2,...,Bn(R1))×(font face=symbol>pC1,C2,...,Cr(R2))

    где атрибуты {Bn} d{Am}, {Cr}d{Am} и атрибуты B1,B2,...,Bn представлены в отношении R1, а атрибуты C1,C2,...,Cr – в R2.

12. Перестановка селекции с пересечением:

    sF(R1 ∪ R2)=sF(R1) ∪ sF(R2)

7.3. Методы оптимизации

Существуют два принципиально разных подхода к оптимизации запро-сов. 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.

7.3.1. 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. 7.1 в качестве примера приведены ранги путей доступа для СУБД Oracle8.

Таблица 7.1. Ранги путей доступа для СУБД Oracle8

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, КБД) – значение, которое может быть однозначно преоб-разовано в физический адрес записи.

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 значений (ранг 10).

Метод оптимизации по синтаксису учитывает ранги путей доступа. Если для какой-либо таблицы существует более одного пути доступа, то выбирается тот путь, чей ранг выше, т.к. при прочих равных условиях он выполняется быстрее, чем путь с более низким рангом (рис. 7.2).

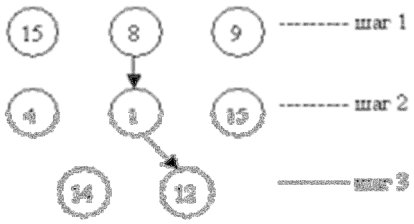

Рис.7.2. Построение плана выполнения запроса в методе оптимизации по синтаксису

План выполнения запроса в методе оптимизации по синтаксису формируется из выбранных путей доступа с максимальными рангами.

7.3.2. Метод оптимизации, основанный на стоимости

При использовании этого метода оптимизатор сначала строит несколько возможных планов выполнения запроса (обычно, 2-3 плана). Для выбора наиболее перспективных планов 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. Эвристики часто основаны на законах преобразования операций реляционной алгебры.

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 оптимизатор в соответствии с критерием выбирает лучший план (рис. 7.3).

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.

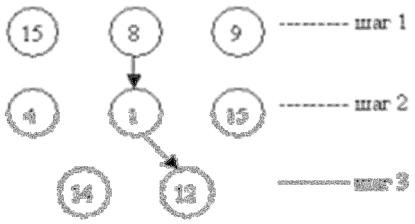

Рис.7.3. Построение плана выполнения запроса в методе оптимизации по стоимости

- Минимальное время реакции – время, необходимое для обработки и выдачи первой строки. Этот критерий предназначен для работы в интерактивном режиме, но может использоваться 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 при работе в пакетном режиме или в ситуации, когда невозможно выдавать результат по частям (например, при использовании агрегирующих функций).

Стоимость плана выполнения запроса определяется на основании сведений о распределении данных в таблицах, к которым обращается команда, и связанных с ними кластеров и индексов. Эти сведения о распределении значений данных называются **статистикой** и хранятся в словаре-справочнике данных. Для таблицы статистика может включать в се-бя:

- общее количество блоков данных (страниц памяти), выделенных таблице;
- количество пустых блоков данных (страниц памяти);
- количество записей в таблице;
- среднюю длину записи в таблице;
- среднее количество записей на блок (страницу) памяти.

Для каждого индекса статистика может содержать такие данные, как:

- общее количество проиндексированных записей (оно может быть меньше, чем количество записей в таблице, т.к. null-значения не индексируются);
- минимальное и максимальное индексированные значения;
- количество различных индексированных значений.

Распределение значений в столбце может быть отражено с помощью гистограммы, которая также входит в статистику. Для этого всё множество значений столбца упорядочивается и разбивается на N интервалов. На рис. 7.4 приведено разбиение на N =10 интервалов множества значений некоторого столбца F. Для равномерного распределения это означает, что в первых 10% записей это поле имеет значение от 1 до 10, в следующих 10% записей – от 11 до 20 и т.д.

Рис.7.4. Примеры равномерного (а) и неравномерного (б) распределения значений

Гистограмма помогает оценить объём данных, удовлетворяющих усло-вию запроса. На рис. 7.4,б представлено неравномерное распределение дан-ных 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 30% записей таблицы.

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 эквива-лентности.

Существуют различные подходы к порядку сбора статистики. Неко-торые 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.

7.3.3. Примеры использования методов оптимизации запросов

Рассмотрим оптимизацию по синтаксису следующего запроса SQL:

Пример 7.2. Запрос, выбирающих всех сотрудников по фамилии 'Иванов' с зарплатой менее 40000 рублей:

```
SELECT *  FROM Emp  WHERE name LIKE 'Иванов%' AND salary<40000;
```

Пусть таблица Emp имеет следующее правило целостности и индексы:

- столбец tabNo определен как PRIMARY KEY, ему соответствует индекс PK_TABNO;
- существует индекс NAME_IND для столбца name;
- существует индекс SALARY_IND для столбца salary.

Возможны следующие пути доступа:

- полный просмотр таблицы (ранг 15);
- доступ по одиночному индексу NAME_IND. Этот путь становится доступным по условию name LIKE 'Иванов%' (ранг 9);
- доступ с помощью открытого интервала salary<40000, используя индекс SALARY_IND (ранг 11).

Индекс PK_TABNO недоступен, т.к. в запросе нет условий на значение поля tabNo. Оптимизатор выберет доступ по индексу с рангом 9.

Теперь рассмотрим примеры оптимизации по стоимости.

Пример 7.3. Запрос, выбирающий сотрудников с номерами больше 7500:

```
SELECT *          FROM Emp  WHERE tabNo > 7500;
```

Статистика для столбца tabNo, в частности, включает значения HIGH_VALUE и LOW_VALUE (максимальное и минимальное значения). Если нет гистограммы, то оптимизатор предполагает, что значения равно-мерно 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 памяти.

Пример 7.4. Запрос, выбирающий название отделов (name из таблицы De-part) и всех сотрудников с максимальной зарплатой в своём отделе (name, salary из таблицы Emp):

```
SELECT d.name, e.name, salary  FROM depart AS d, emp AS e  WHERE
d.depNo=e.depNo AND  e.salary=(SELECT max(salary) FROM emp AS p   WHERE
p.depNo=e.depNo);
```

Для таблиц есть индексы по первичным ключам (Depart.depNo и Emp.tabNo) и по внешнему ключу (Emp.depNo).

Этот запрос можно выполнить по разным планам, например:

1. Выбрать все записи из таблиц Emp и Depart, соединить их по условию d.depNo=e.depNo, 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.depNo=e.depNo в таблице Depart через индекс по первичному ключу (на поле depNo), затем для каждой полученной строки посчитать подзапрос и проверить второе условие.
3. Выбрать все записи из таблицы Emp, каждую запись соединить по условию d.depNo=e.depNo с таблицей Depart через индекс по первично-му ключу (на поле depNo). Предварительно посчитать подзапрос, добавив в него условие группировки по номерам отделов GROUP BY depNo. 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, при оптимизации по стоимости оптимизатор вероятнее всего выберет третий план как самый оптимальный с точки зрения времени выполнения запроса.

7.4. Настройка приложений

Цель настройки приложений – повышение эффективности работы с БД. В настройку приложений входит:

- создание индексов;
- настройка команд SQL;
- выбор метода оптимизации SQL-запросов;
- использование средств сбора статистики.

Первый пункт мы уже обсуждали (см. разделы 4.5.2).

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

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. 7.5. В ОП для каждого сеанса связи с БД выделяется специальная область – курсор, куда помещается результат выполнения последнего (текущего) запроса пользователя.

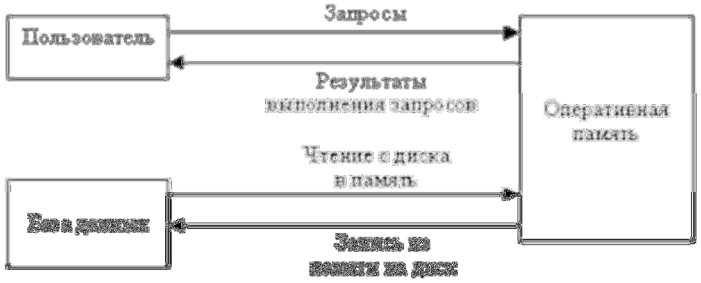

Рис.7.5. Информационные потоки в БД

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 второго 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 depNo=2 AND sex='ж';
   ```

2. В запросе, который реализует соединение двух и более таблиц, эти таблицы должны стоять в списке FROM в порядке уменьшения количества записей в них, а в части WHERE первым должно стоять условие на основную (родительскую) таблицу.

   Например, список пациентов по отделениям №№1,2:

   ```
   SELECT *    FROM patients p, depart d  WHERE d.id IN (1,2) AND
   p.depNo=d.id;
   ```

3. Если запрос содержит условие с неопределённой лидирующей частью 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:

   ```
   SELECT *    FROM doctors  WHERE special>'A' AND special like
   '%хирург%';
   ```

Здесь условие special>'A' не исключает из поиска ни одной записи таблицы, но позволяет системе проводить этот поиск по индексу, который занимает гораздо меньше памяти, чем сама таблица.

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. Следует использовать UNION ALL вместо UNION, если в объединяемых отношениях отсутствуют одинаковые записи (или наличие одинаковых записей некритично). Дело в том, что UNION вычисляется путем сортировки, которая может занять много времени, а UNION ALL сортировки не требует.
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 (при наличии индекса по внешнему ключу):

   ```
   SELECT DISTINCT e.*    FROM Emp AS e, Children AS c  WHERE
   c.empNo=e.empNo;
   ```

7. Если оптимизатор плохо оптимизирует операцию "или" (OR), то можно заменять её операцией UNION при наличии индексов. Убедиться в "плохой оптимизации" можно так: выполнить запрос по условию (field=X) и запрос с условием ((field=X) OR (field=Y)) на большой таблице. Если второй запрос выполняется намного дольше, чем первый, то OR не оптимизируется. Например, список "Пациенты палат №3 и пациенты, больные гриппом" в отсутствие индексов можно сформулировать так:

   ```
   SELECT *    FROM Patients  WHERE room=3 OR diagnose LIKE 'грипп%';
   ```

   а если индексы есть, то таким:

   ```
   SELECT *    FROM Patients  WHERE room=3  UNION ALL  SELECT *    FROM
   Patients  WHERE diagnose LIKE 'грипп%';
   ```

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 deptNo>3;
   ```

9. 9. Некоторые оптимизаторы будут использовать индексное сканирование, если запрос содержит раздел ORDER BY с указанием индексированного столбца. Для выполнения следующего запроса будет использован индекс на столбце tabNo, даже если этот столбец не используется в условиях раздела WHERE:

   ```
   SELECT *    FROM emp  WHERE depNo<3    ORDER BY tabNo;
   ```

10. Условие <выражение1> op <выражение2>, где op – операция, также не позволяют использовать индекс. Из выражений надо по возможности вынести в левую часть поле, по которому есть индекс. Например, условие (salary*0.87>30000) лучше записать так: salary>30000/0.87.

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.

Многие СУБД позволяют просмотреть план выполнения запроса средствами администрирования. Так можно убедиться в том, что система использует поостренные индексы для выполнения запросов.

> "Сложная система, спроектированная наспех, никогда не работа-ет, и исправить её, чтобы заставить работать, невозможно".
>
> Законы Мерфи. 16-й закон системантики

8. ЭЛЕМЕНТЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

Проектирование базы данных (БД) – одна из наиболее сложных и ответственных задач, связанных с созданием автоматизированных информационных систем (АИС).

В первую очередь АИС должна обеспечивать ведение БД: запись, чтение, модификацию данных, удаление неактуальных данных (возможно, в архив) и защиту данных. Взаимодействие конечных пользователей с БД обычно осуществляется с помощью интерфейсного приложения, входящего в состав АИС. Если пользователей АИС можно разделить на группы по характеру решаемых задач, то приложений может быть несколько (по количеству задач или групп пользователей).

В результате проектирования БД должны быть определены состав базы данных, эффективный для всех её будущих пользователей способ организации данных и инструментальные средства управления данными.

8.1. Требования к проекту базы данных

Основные требования, которым должен удовлетворять проект БД:

1. Корректность схемы БД.

   База данных должна быть гомоморфным образом моделируемой предметной области, т.е. каждой сущности ПО должны соответствовать данные в памяти ЭВМ, а каждому процессу – адекватные процедуры обработки данных. Корректность подразумевает также логическую непротиворечивость базы данных, которая поддерживается автоматически с помощью средств СУБД.

2. Обеспечение ограничений на ресурсы вычислительной системы.

   В первую очередь имеются в виду ограничения на объёмы внешней и оперативной памяти, которые потребуются для функционирования БД.

3. Эффективность функционирования.

   База данных должна быть спроектирована таким образом, чтобы при её эксплуатации соблюдались ограничения на время реакции системы на запросы и модификацию данных.

4. Защита данных.

   Проект БД должен включать описание защиты данных от несанкционированного доступа. Защита от сбоев является внутренней функцией СУБД, но требования к настройке механизмов защиты также выдвигаются на этапе проектирования БД, т.к. определяются предметной областью.

5. Гибкость

   Под этим подразумевается возможность развития и адаптации БД к изменениям предметной области и/или требований пользователей.

Конечно, нельзя предусмотреть все возможные варианты использования и изменения базы данных. Но в большинстве предметных областей основные сущности и их взаимосвязи относительно стабильны. Меняются только информационные требования, т.е. способы использования данных для решения задач.

6. Простота и удобство эксплуатации.

   Под этим подразумевается соблюдение привычного для пользователя алгоритма работы с данными. От этого не в последнюю очередь зависит количество ошибок пользователя.

Удовлетворение первых 4-х требований обязательно для принятия проекта.

8.2. Этапы проектирования базы данных

В создании автоматизированной информационной системы, вклю-чающей базу данных, можно выделить три этапа:

I. Предпроектная подготовка.
II. Проектирование БД.
III. Реализация (создание БД и ППО).

Проектирование начинается обычно с планирования, что позволяет:

- разбить задачу на небольшие, независимые, управляемые шаги;
- поставить краткосрочные и долговременные цели, которые служат для оценки фактических результатов проектирования и сравнения их с планом;
- определить временные зависимости между задачами, т.е. определить, какие задачи должны быть решены раньше других (составить сетевой план-график работ);
- выявить узкие места, т.е. ресурсы, от которых план зависит сильнее всего;
- спрогнозировать потребности в кадрах для проекта.

  Работа по созданию БД начинается с подбора кадров. Требуется определить, какие специалисты необходимы для выполнения этой работы. В общем случае это должны быть следующие категории:

- Аналитики. Это специалисты исследуемой предметной области, которые в идеале должны быть знакомы с основами создания баз данных. В их задачу входит постановка задачи проектирования: анализ ПО, выявление бизнес-процессов и бизнес-правил, определение требований к БД.
- Пользователи. Наличие этой категории работников особенно важно тогда, когда проект БД создаётся в развитие существующей информационной системы, т.к. в этом случае есть определённый опыт работы (традиции, привычки и вместе с тем пожелания). Всё это желательно учитывать для

того, чтобы обеспечить преемственность и не вызвать негативного отношения к системе, например, из-за непривычного интерфейса.
- Проектировщики. Это сотрудники, которые будут заниматься собственно разработкой проекта БД.
- Администраторы. В том случае, если система небольшая, администратор БД может быть один. Если же система большая и территориально распределенная, то помимо АБД потребуется ещё администратор системы, и, возможно, не один. АБД должен появиться не тогда, когда система уже спроектирована, а на этапе проектирования БД. Это необходимо хотя бы потому, что при проектировании для отладки и тестирования обязательно создаётся рабочий прототип БД, и желательно, чтобы за общее обеспечение функционирования этого прототипа отвечал отдельный специалист.
- Разработчики программного обеспечения. Любая БД требует помимо СУБД создания некоторого прикладного программного обеспечения (ППО). Если сложность этого ППО невелика, то обычно его созданием занимаются сами проектировщики. В противном случае, необходимо набрать программистов (или выделить из имеющихся), которые будут этим заниматься.

При определении потребности в кадрах может возникнуть ситуация, когда уже на этом этапе станет очевидна невозможность подбора нужного количества специалистов определённого профиля и квалификации. Тогда это может привести к пересмотру объёма задач, стоящих перед базой данных.

Общая схема жизненного цикла приложения баз данных приведена на рис. 8.1. Как видно из этого рисунка, проектирование носит итерационный характер. Например, если на каком-либо этапе выясняется, что ранее сформулированные требования или решения не могут быть реализованы, то разработчики должны вернуться на более ранний этап и внести соответствующие изменения. Эти изменения, естественно, могут потребовать корректировки ранее выполненных этапов.

Перечислим более подробно задачи, решение которых должно предшествовать созданию проекта БД.

1. Предварительный анализ ПО.

   Включает в себя сбор документов, характеризующих ПО, укрупнённое описание ПО (не детализированное) и общую постановку задачи.

   В процессе анализа и проектирования желательно ранжировать планируемые функции системы по степени важности. Один из возможных вариантов классификации – MoSCoW-анализ (терминология Клегга и Баркера, [8]):

   Must have – необходимые функции;

Should have – желательные функции;

Could have – возможные функции;

Won't have – отсутствующие функции

Необходимые функции обеспечивают возможности, которые являются критическими для успешной работы системы. Реализация *желательных* и возможных функций системы ограничена временными и/или финансовыми рамками. *Отсутствующие* функции – это те функции, которые реально существуют, но не будут реализованы в этом проекте по различным причинам.

Примечание: если применяются средства автоматизации проектирования (CASE-средства), то задачу последовательного внесения изменений берёт на себя это CASE-средство.

2. Рассмотрение и принятие результатов анализа.

   Эта задача обычно решается итеративно во взаимодействии проектировщиков и заказчиков (или аналитиков). На этом этапе очень важно определить, что проектировщики правильно понимают описание предметной области и задачи, поставленные перед ними аналитиками. Для этого обычно проводятся совместные семинары, на которых проверяется адекватность модели и предметной области.

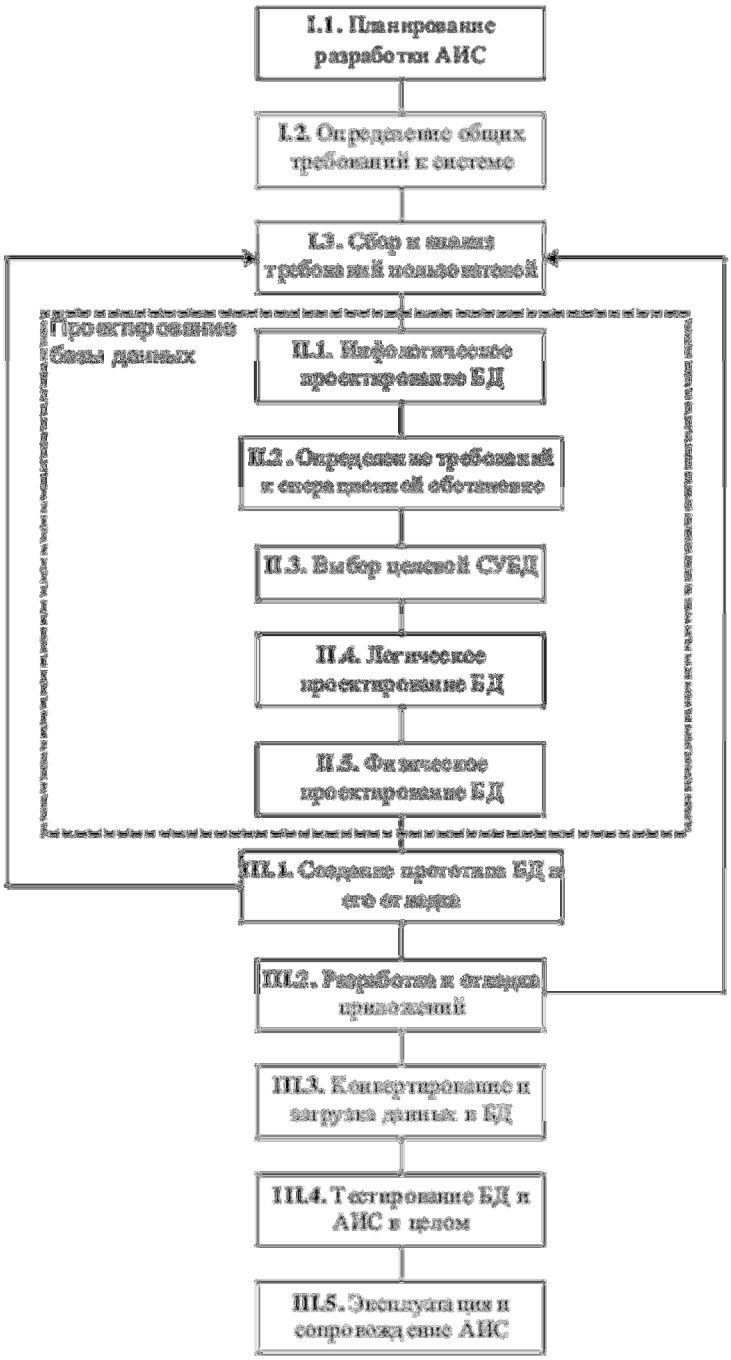

Рис. 8.1. Жизненный цикл приложения баз данных

3. Определение критических факторов успеха.

   В данном случае под термином критические факторы подразумеваются как "жизненно важные для приёмки и успешной реализации проекта", так и "критические с точки зрения функционирования системы". Очевидно, что если не учесть хотя бы один из таких факторов, то существование и успешное функционирование проекта будет поставлено под вопрос.

4. Оценка системных ограничений.

В качестве часто встречающихся ограничений можно отметить следующие:

- финансовые
- временные
- технические (например, выбор определённой аппаратуры);
- программные (например, выбор определённого программного обеспечения);

5. Определение целевой архитектуры.

   Под целевой архитектурой в данном случае понимается архитектура с точки зрения СУБД (однозадачная или многозадачная, архитектура клиент-сервер, параллельный сервер). Выбор архитектуры повлияет в дальнейшем на перечень требуемых аппаратных и программных средств.

6. Определение требований к производительности.

   Необходимо примерно оценить количество транзакций в единицу времени и объём обрабатываемых этими транзакциями данных. Требования к производительности зависят от режима, в котором будет функционировать система:

   a. Интерактивный режим. Для этого режима устанавливается время, в течение которого пользователь должен получить ответ на свой запрос. Обычно время реакции системы не должно превышать нескольких секунд.
   b. Пакетный режим. Здесь требования к производительности обычно не такие жёсткие, как для интерактивного режима, и выражаются в минутах или часах, требующихся на получение конечного результата вычислений.
   c. Режим реального времени. Этот режим является самым сложно реализуемым. В настоящее время (2009 г.) существует только одна СУБД, которая в полной мере отвечает требованиям режима реального времени: СУБД ЛИНТЕР – единственная СУБД отечественного производства (компания РЕЛЭКС, г. Воронеж).

7. Согласование стандартов проектирования, в частности:
   - правил именования объектов;
   - стандарта проектной документации;
   - правил введения общих типов и т.п.
8. Выбор программных средств для проектирования и реализации системы (имеется в виду вспомогательные средства типа CASE и др.).

Собственно процесс проектирования БД включает в себя следующие основные этапы:

I. Информационно-логическое (инфологическое) проектирование

II. Определение требований к операционной обстановке, в которой будет функционировать информационная система.
III. Выбор СУБД и других инструментальных программных средств
IV. Логическое проектирование БД. (Иногда этот этап называется даталогическим проектированием).
V. Физическое проектирование БД.

Эти этапы подробно рассмотрены в следующем разделе.

После того, как проект базы данных создан, наступает этап реализации проекта. Он разбивается на следующие шаги:

A. Создание прототипа БД и его отладка. Отладка подразумевает проверку правильности функционирования процедурных объектов БД (триггеры, процедуры, функции). Прототип позволяет определить жизнеспособность проекта БД и выявить его недостатки, что может потребовать внесения изменений в проект. Прототип также нужен как база для разработчиков приложений. Для этого БД наполняется реальными или тестовыми данными.
B. Разработка и отладка приложений. Выполняется разработчиками программного обеспечения на основе функциональных требований, которые были выявлены на этапах I.2, I.3, и спецификации БД (схемы БД).
C. Конвертирование и загрузка данных в БД. Этот этап выполняется в том случае, если данные в БД загружаются из ранее существовавшей системы.
D. Тестирование работы базы данных и АИС в целом. Различают такие виды тестов, как:
    - *автономные* – тесты отдельных модулей;
    - *тесты связей* – тесты между модулями;
    - *регрессивные* – тесты на проверку уже *автономные* – тесты отдельных модулей;
    - протестированных модулей в связи с подключением новых модулей (функций), которые могут нарушить работу ранее созданных модулей;
    - *нагрузочные* – тесты на проверку времени реакции системы в рабочем режиме или определение производительности системы;
    - *системные* – тесты на проверку функционирования системы в целом;
    - *приёмо-сдаточные* – тесты, которые проводятся при сдаче системы (АИС) в эксплуатацию.

    На этапе III.4 обычно выполняются нагрузочные, системные и приёмо-сдаточные тесты.

E. Эксплуатация и сопровождение созданной АИС. Здесь можно выделить ряд задач:
    - В процессе эксплуатации АИС может возникнуть необходимость внесе-ния изменений в систему. Это может быть вызвано

изменениями пред-метной области, появлением новых задач или выявлением существенных недостатков в АИС. Нельзя забывать о том, что все вносимые изменения должны быть документированы.
- Необходимо выполнять резервное копирование данных, чтобы предот-вратить их потерю в случае серьёзного сбоя или ошибки пользователя.
- Сопровождение АИС обычно включает периодические проверки выполнения системных ограничений (на объём данных и время реакции системы). В результате этих проверок удаляются устаревшие данные (если не предусмотрено автоматическое архивирование данных). Улучшение показателей производительности системы может быть достигнуто за счёт настройки СУБД, которая выполняется администратором базы данных.

Теперь перейдём к более подробному обсуждению этапов проектирования БД.

8.3. Инфологическое проектирование

Инфологический подход не содержит формальных способов моделирования реальности, но он закладывает основы методологии проектирования БД.

Первой задачей инфологического проектирования является определение *предметной области*(ПО) системы, позволяющее изучить информационные потребности будущих пользователей. Другая задача этого этапа – анализ ПО, который призван сформировать взгляд на неё с позиций сообщества будущих пользователей БД, т.е. *инфологической модели* ПО.

Анализ ПО выполняется проектировщиком БД с помощью специалистов в данной ПО. В основе анализа лежат документы, используемые в работе предприятия (организации), и технология работы с данными.

Инфологическая модель ПО включает описание структуры и динамики ПО, характера информационных потребностей пользователей системы. Описание выполняется в терминах, понятных пользователю и независимых от реализации системы. Обратите внимание: инфологическая модель ПО не должна зависеть от модели данных, которая будет использована при создании БД.

Обычно описание ПО выражается в терминах не отдельных сущностей и связей между ними, а их типов, связанных с ними ограничений целостности и тех процессов, которые приводят к переходу ПО из одного состояния в другое. Такое описание может быть представлено любым способом, допускающим однозначную интерпретацию.

В простых случаях описание ПО представляется на естественном языке. В более сложных случаях используется также математический аппарат: таблицы,

диаграммы, графы и т.п. Если анализ ПО выполняется несколькими специалистами, то они должны принять соглашения, которые касаются:

- используемых методов анализа предметной области;
- правил именования и обозначения сущностей ПО, атрибутов и связей;
- содержания и формата создаваемых ими документов.

Этап инфологического проектирования начинается с моделирования ПО. Проектировщик разбивает ПО на ряд локальных областей (**локальных представлений**), каждая из которых (в идеале) включает в себя информацию, достаточную для обеспечения информационных потребностей одной группы будущих пользователей или решения отдельной задачи. Каждое локальное представление моделируется отдельно, а затем выполняется их объединение.

Выбор локального представления зависит от масштабов ПО. Обычно ПО разбивается на локальные области так, чтобы каждая из них соответствовала отдельному внешнему приложению и содержала 6-7 сущностей (т.е. объектов, о которых в системе будет накапливаться информация). Таким образом, если ПО небольшая, то разбиение на локальные представления не требуется и моделирование выполняется для ПО в целом.

Существуют разные подходы к инфологическому проектированию. Рассмотрим основные из них.

1. **Функциональный подход к проектированию БД.**

   Этот метод реализует принцип "от задач" и применяется в том случае, когда известны функции некоторой группы лиц и/или комплекса задач, для обслуживания информационных потребностей которых создаётся рассматриваемая БД.

2. **Предметный подход к проектированию БД.**

   Предметный подход применяется в тех случаях, когда у разработчиков есть чёткое представление о самой ПО и о том, какую именно информацию они хотели бы хранить в БД, а структура запросов не определена или определена не полностью. Тогда основное внимание уделяется исследованию ПО и наиболее адекватному её отображению в БД с учётом самого широкого спектра информационных запросов к ней.

3. **Проектирование с использованием метода "сущность–связь".**

   Метод "сущность–связь" (Entity–Relation, ER–method) был разработан в 1976 г. П.Ченом (Chen P.P.). Он является комбинацией двух предыдущих и обладает достоинствами обоих.

ER-метод является наиболее распространённым методом проектирования БД, поэтому мы рассмотрим его подробно.

- **8.3.1. Метод "сущность-связь"**

  В предметной области необходимо выделить *сущности, атрибуты и связи*. Сущности, существование которых не зависит от существования других сущностей, называются *базовыми*, остальные сущности – *зависимыми*. Например, сущность *ЛЕКЦИЯ* зависит от базовых сущностей*ГРУППА, ПРЕПОДАВАТЕЛЬ, ДИСЦИПЛИНА*.

  Для каждой сущности определяются атрибуты (свойства), которые можно условно классифицировать следующим образом:

  1. *Идентифицирующие и описательные атрибуты*. Идентифицирующие атрибуты имеют уникальное значение для сущностей данного типа и являются *потенциальными ключами*. Они позволяют однозначно распознавать экземпляры сущности. Из потенциальных ключей выбирается один первичный ключ (ПК). В качестве ПК обычно выбирается потенциальный ключ, по которому чаще происходит обращение к экземплярам сущности. Кроме того, ПК должен включать в свой состав минимально необходимое для идентификации количество атрибутов. Остальные атрибуты называются описательными и заключают в себе интересующие свойства сущности.
  2. *Составные и простые атрибуты*. Простой атрибут состоит из одного компонента, его значение неделимо. Составной атрибут является комбинацией нескольких компонентов, возможно, принадлежащих разным типам данных (например, ФИО или адрес). Решение о том, использовать составной атрибут или разбивать его на компоненты, зависит от характера его обработки и формата пользовательского представления этого атрибута.
  3. *Однозначные и многозначные атрибуты* (могут иметь соответственно одно или много значений для каждого экземпляра сущности).
  4. *Основные и производные атрибуты*. Значение основного атрибута не зависит от других атрибутов. Значение производного атрибута вычисляется на основе значений других атрибутов (например, возраст человека вычисляется на основе даты его рождения и текущей даты).

  Спецификация атрибута состоит из его названия, типа данных, размера и описания ограничений целостности – множества значений, которые может принимать данный атрибут.

  Далее осуществляется спецификация связей. Под связью понимается осмысленная ассоциация между сущностями, например, *СТУДЕНТ учится* в *ГРУППЕ*, *ВОДИТЕЛЬ выполняет РЕЙС* и т.п. Выявляются все связи между сущностями внутри локального представления. Каждая связь

именуется, для неё определяются степень, кардинальность и обязательность.

Кроме спецификации связей типа "сущность – сущность", выполняется спецификация связей типа "сущность – атрибут" и "атрибут – атрибут" внутри одной сущности. Для этого надо определить зависимости между экземплярами сущностями и атрибутами, а также между атрибутами, относящимися к одному экземпляру сущности. Например, атрибут *Телефоны* сущности*СОТРУДНИК* может быть многозначным и необязательным, т.е. связь *СОТРУДНИК –>Телефоны* имеет тип 1:n и является необязательной для сотрудника. А если рассмотреть атрибуты*Маршрут* и *Стоимость* сущности *БИЛЕТ*, то между ними есть связь 1:1, т.к. стоимость билета зависит от маршрута (пункт отправления – пункт назначения).

После выявления сущностей и связей ПО строят ER-диаграмму, которая является наглядным отображением модели ПО. (Пример ER-диаграммы приведён на рис. 1.4).

8.3.2. Объединение локальных представлений

При небольшом количестве локальных областей (не более пяти) объединение выполняется за один шаг. В противном случае обычно выполняют бинарное объединение. При этом проектировщик может формировать конструкции, производные по отношению к тем, которые были использованы в локальных представлениях. Цель введения подобных абст-ракций:

- объединение в единое целое фрагментарных представлений о различных свойствах одной и той же сущности;
- введение абстрактных понятий, удобных для решения задач системы, установление их связи с более конкретными понятиями модели;
- образование классов и подклассов подобных сущностей (например, класс "изделие" и подклассы типов изделий, производимых на предприятии).

При объединении локальных представлений используют три основополагающие концепции:

8. **Идентичность.** Два или более элементов модели идентичны, если они имеют одинаковое семантическое значение.
   Например, *СОТРУДНИК* для отдела кадров и *СОТРУДНИК* для отдела закупок – это один и тот же тип сущности, возможно, с разным набором атрибутов. (Наборы атрибутов исходных сущностей при этом объединяются).

9. **Агрегация.** Позволяет рассматривать связь между элементами как новый элемент. Например, связь *экзаменовать* между сущностями *ДИСЦИПЛИНА, ПРЕПОДАВАТЕЛЬ,СТУДЕНТ* может быть представлена агрегированной сущностью *ЭКЗАМЕН* с атрибутами*Название дисциплины, Фамилия преподавателя, Фамилия студента, Оценка*.
10. **Обобщение.** Позволяет образовывать многоуровневую иерархию обобщений. Например, в объединяемых представлениях присутствуют следующие сущности:

ДЕТАЛИ СОБСТВЕННОГО ПРОИЗВОДСТВА

ДЕТАЛИ ПОКУПНЫЕ

СБОРОЧНЫЕ ЕДИНИЦЫ ПОКУПНЫЕ

СБОРОЧНЫЕ ЕДИНИЦЫ СОБСТВЕННОГО ПРОИЗВОДСТВА

Их можно объединить так, как показано на рис. 8.2.

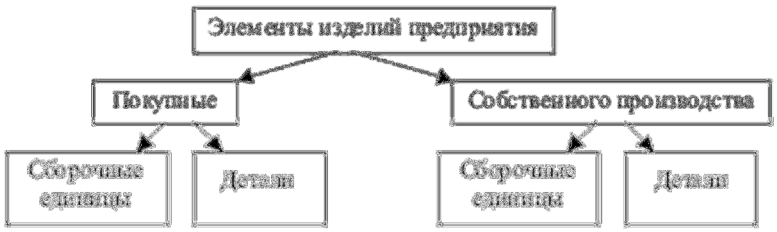

Рис. 8.2. Использование обобщений при объединениих

Это позволит упростить формализацию процессов обработки данных. Например, оформление заказа на покупные элементы изделий в данном примере может быть описано один раз (для второго уровня иерархии).

На этапе объединения необходимо выявить и устранить все противоречия. Например, изменить одинаковые названия семантически различных сущностей или связей или несогласованные ограничения целостности на одни и те же атрибуты в разных приложениях. Устранение противоречий вызывает необходимость возврата к этапу моделирования локальных представлений с целью внесения в них соответствующих изменений.

По завершении объединения результаты проектирования представляют собой концептуальную инфологическую модель ПО. Она фиксируется в виде общей ER-диаграммы предметной области. Модели локальных представлений – это внешние инфологические модели (внешние схемы).

На этапе анализа ПО также решаются следующие задачи:

11. Определение правил (ограничений целостности), которым должны удовлетворять сущности ПО, атрибуты сущностей и связи между ними. Часть этих правил реализуется в схеме базы данных. Возможности реализации ограничений целостности в схеме БД определяются моделью данных той СУБД, которая будет выбрана для реализации проекта. Остальные правила реализуются с помощью программного обеспечения.
12. Выделение групп пользователей системы. Каждая группа выполняет определённые задачи и обладает разными правами доступа к системе.
13. Создание внешней спецификации тех функций (процессов), которые эта система будет выполнять. Например, для той же библиотечной системы это задачи поиска книг (по определённым критериям), выдачи/приёма книг, определение списка должников и т.д. Эта спецификация является основой для разработки приложений и выдаётся программистам в качестве задания.

8.4. Определение требований к операционной обстановке

На этом этапе производится оценка требований к вычислительным ресурсам, необходимым для функционирования системы, выбор типа и конфигурации ЭВМ, типа и версии операционной системы (ОС).

Выбор зависит от таких показателей, как:

- примерный объём данных в БД;
- динамика роста объёма данных;
- характер запросов к данным (извлечение и обновление отдельных записей, обработка групп записей, обработка отдельных отношений или соединение отношений);
- интенсивность запросов к данным по типам запросов;
- требования ко времени отклика системы по типам запросов;
- режим работы (интерактивный, пакетный или режим реального времени).

Эта информация позволяет определить системные требования к объёму оперативной и дисковой памяти, а также функциональным возможностям ОС.

8.5. Выбор СУБД и инструментальных программных средств

Выбор СУБД является одним из важнейших моментов в разработке проекта БД, так как он принципиальным образом влияет на процесс проектирования БД и реализации информационной системы.

Теоретически при осуществлении этого выбора нужно принимать во внимание десятки факторов. Но на практике разработчики руководствуются лишь собственной интуицией и несколькими наиболее важными критериями, к которым относятся:

- тип модели данных, которую поддерживает данная СУБД, адекватность модели данных структуре рассматриваемой ПО;
- характеристики производительности СУБД;
- запас функциональных возможностей для дальнейшего развития информационной системы;
- степень оснащённости СУБД инструментарием для персонала администрирования данными;
- удобство и надежность СУБД в эксплуатации;
- наличие специалистов по работе с конкретной СУБД;
- стоимость СУБД и дополнительного программного обеспечения.

По результатам предыдущего этапа определены основные характеристики БД, такие как объём памяти и необходимая производительность. В зависимости от этого выбираются 2-3 СУБД, которые соответствуют выявленным требованиям. Например, если объём БД не превысит 100M, большинство запросов выбирает от 1 до 20 записей и время реакции системы не должно превышать 10 секунд, то следует остановить выбор на системах среднего класса, таких как Firebird, PostgreSQL, FoxPro. Для меньших по объёму БД можно выбрать Access или MySQL, а такие серьёзные СУБД как Oracle, DB/2 или Informix следует рассматривать в тех случаях, когда велик объём данных или имеются высокие требования к производительности системы.

Выбранные СУБД оцениваются по степени соответствия выявленным требованиям к БД, и выбирается та система, которая лучше им соответствует.

8.6. Логическое проектирование БД

На этапе логического проектирования инфологическая модель ПО, представленная в виде ER-диаграммы, преобразуется в логическую (концептуальную) схему БД. Решение этой задачи существенно зависит от модели данных, поддерживаемой выбранной СУБД.

Результатом выполнения этапа логического проектирования являются схемы БД концептуального и внешнего уровней архитектуры, составленные на языке определения данных (DDL, Data Definition Language) выбранной СУБД.

Более подробно этот этап мы рассмотрим для РМД (п. 8.9).

8.7. Физическое проектирование БД

Основой для физического проектирования является схема БД, полученная на предыдущем этапе. Физическое проектирование заключается в увязке логической структуры БД и физической среды хранения с целью наиболее эффективного размещения данных. Решается вопрос размещения хранимых данных в пространстве памяти и выбора эффективных методов доступа к различным компонентам "физической" БД. Результаты этого этапа документируются в форме схемы хранения на языке определения данных. Принятые на этом этапе решения оказывают определяющее влияние на производительность системы.

Для реляционной БД на этом этапе определяются параметры распределения памяти для объектов БД, строятся индексы, определяется целесообразность использования хеширования и кластеризации.

Фактически проектирование БД имеет итерационный характер. В процессе функционирования системы становится возможным измерение её реальных характеристик, выявление "узких" мест. И если система не отвечает предъявляемым к ней требованиям, то обычно она подвергается реорганизации, т.е. модификации первоначально созданного проекта.

8.8. Автоматизация проектирования БД

Функциональное ядро систем автоматизированного проектирования (САПР) БД строится как совокупность взаимосвязанных модулей инфологического моделирования, проектирования схем и физической организации БД.

Существующие в настоящее время САПР БД строятся как человеко-машинные экспертные системы. В первую очередь это определяется слабо поддающимся формализации процессом синтеза инфологического описания ПО, т.е. преобразования неформальных представлений реального мира в формальные категории. Этот процесс выполняется экспертом – специалистом в той или иной ПО. Поэтому все проблемы, которые характерны для формирования базы знаний экспертной системы, возникают и в случае САПР БД.

Характерной особенностью САПР БД является её ориентация на коллективное творчество и продолжительность самого процесса проектирования, предполагающего множество итераций. Это находит свое отражение в наличии журнала проектирования и других средств, обеспечивающих ведение и коллективное использование исходных данных, промежуточных и окончательных результатов проектирования. Общая структура САПР БД приведена на рис. 8.3.

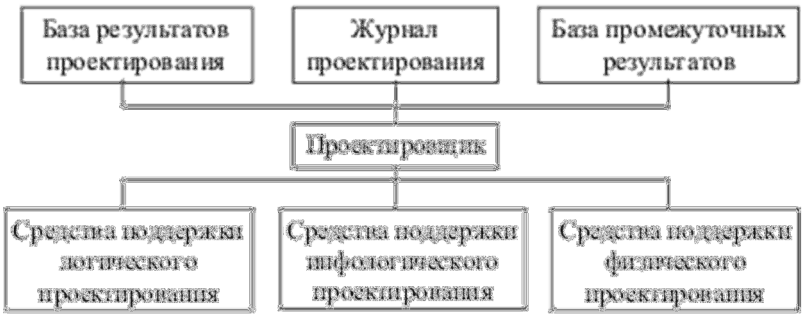

Рис.8.3. Общая структура САПР БД

В настоящее время создан ряд САПР БД, которые называются CASE–средствами. В качестве примеров таких систем можно привести ERWin, BPWin, Designer (Oracle) и др. Подробный обзор современных можно найти в [9].

8.9. Особенности проектирования реляционных БД

Проектирование реляционной базы данных проходит в том же порядке, что и проектирование БД других моделей данных, но имеет свои особенности, которые в первую очередь касаются этапа логического проектирования.

На этапе логического проектирования реляционной базы данных также необходимо решить следующие задачи:

27. Преобразовать ER-диаграмму в схему БД.
28. Выявить нереализуемые и необычные конструкции данных.
29. Определить все первичные ключи (ПК).
30. Определить типы данных для полей таблиц.
31. Описать все ограничения целостности.

8.9.1. Преобразование ER-диаграммы в схему БД

Правила преобразование ER-диаграммы в схему БД следующие:

32. Каждый тип сущности преобразуется в таблицу БД. В таблицу вносятся все атрибуты, относящиеся к данному типу сущности.
33. Бинарная связь 1:n (между сущностями **разных типов**) реализуется с помощью внешнего ключа между двумя таблицами (рис. 8.4). Например, *ОТДЕЛЫ и СОТРУДНИКИ, ГРУППЫи СТУДЕНТЫ* и т.п. *Номер группы* в таблице *ГРУППЫ* является первичным ключом, а*Номер группы* в таблице *СТУДЕНТЫ* – внешним ключом. Это самый часто встречающийся вид связи.

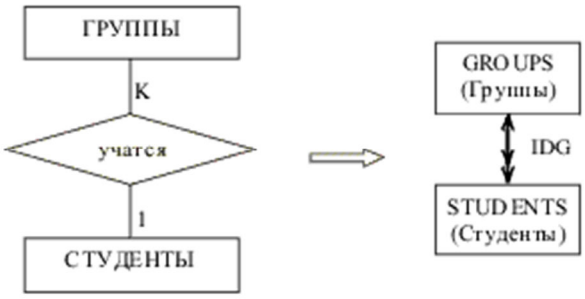

Рис. 8.4. Преобразование бинарной связи 1:n между сущностями разных типовх

34. Каждая связь со степенью больше двух и связь, имеющая атрибуты, преобразуется в таблицу БД (рис. 8.5).

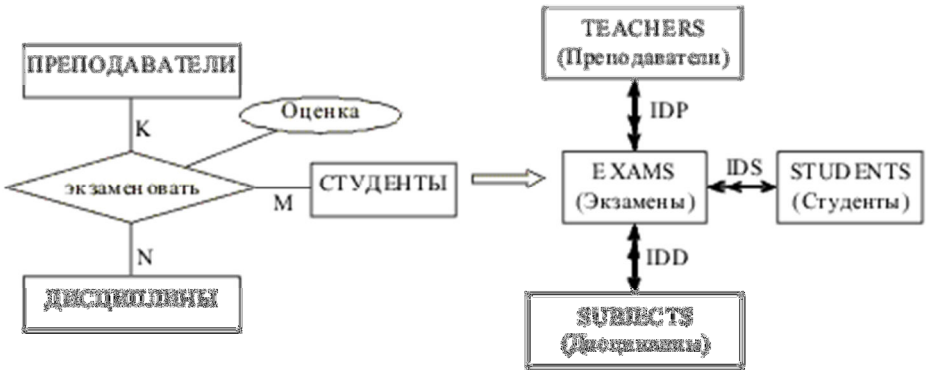

Рис. 8.5. Преобразование связи с атрибутами

35. Связь 1:1 реализуется в рамках одной таблицы. Исключение из этого правила составляют ситуации, когда связанные сущности существуют независимо друг от друга. Например, связь между сущностями *ВОДИТЕЛИ* и *ТРАНПОРТНЫЕ СРЕДСТВА* при условии, что за каждым транспортным средством закреплён один водитель. Эта схема будет включать две таблицы, а связь между ними можно реализовать с помощью уникального (возможно, необязательного) внешнего ключа в той таблице, которая будет считаться подчинённой.
36. Унарная связь 1:n (между сущностями одного типа) реализуется с помощью внешнего ключа, определённого в той же таблице, что и первичный ключ. Например, для отражения в таблице *СОТРУДНИКИ* связи *руководить*, нужно добавить в неё поле *Руководитель*. Это поле будет внешним ключом, ссылающимся на первичный ключ этой же таблицы (рис. 2.9). Такой ключ позволяет отразить иерархию сотрудников, когда у каждого сотрудника может быть только один непосредственный руководитель, а у директора поле *Руководитель* будет неопределённым (null).

37. Бинарная связь типа n:m реализуется с помощью промежуточной таблицы. Например, для сущностей *КНИГИ* и *АВТОРЫ* и связи *написать* промежуточная таблица будет содержать два внешних ключа: идентификатор книги и идентификатор автора, написавшего эту книгу (рис. 8.6). В эту промежуточную таблицу также вносятся те атрибуты, которые характеризуют эту связь (например, номер автора в списке авторов этой книги).

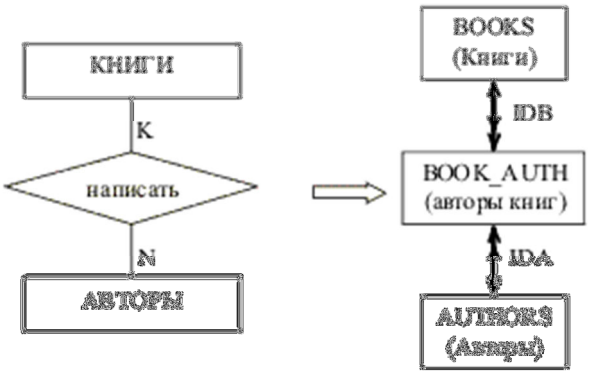

Рис. 8.6. Преобразование бинарной связи 1:n между сущностями разных типов

38. Унарная связь n:m реализуется с помощью промежуточной таблицы. Например, для отражения связи *ассоциируется* между терминами таблицы *КЛЮЧЕВЫЕ СЛОВА*, нужно добавить таблицу *АССОЦИАЦИИ*, в которой будут два внешних ключа на таблицу*КЛЮЧЕВЫЕ СЛОВА* (рис. 8.6).

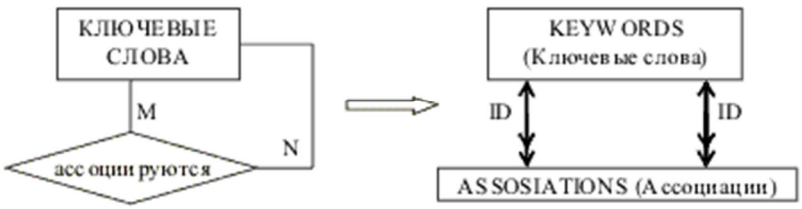

Рис. 8.7. Преобразование унарной связи кардинальности n:m

8.9.2. Выявление нереализуемых связей

К нереализуемым относятся связи кардинальностью 1:n или n:m, обязательные в обе стороны. Например, связь заказы–строки заказов: заказ не может быть пустым, и заказанный товар должен входить в определённый заказ. Т.е. нельзя добавить заказ, пока в нём нет ни одной строки, и нельзя добавить строку в несуществующий заказ. Эта проблема обычно решается так: связь делается необязательной со стороны первичного ключа, а внешний ключ остаётся обязательным. При этом в приложении

необходимо предусмотреть правило обработки пустых заказов (например, их удаление).

К необычным конструкциям данных можно отнести так называемые взаимоисключающие связи, когда подчинённая сущность связана с одной из двух родительских сущностей. Например, счёт в банке может принадлежать либо физическому лицу, либо юридическому, и не может принадлежать и тому, и другому либо не принадлежать никому. Такую связь можно реализовать по-разному, например, введением в таблицу счетов двух внешних ключей (номер_физ_лица и номер_юр_лица) и следующего ограничения целостности:

```
(номер_физ_лица  IS NULL  AND  номер_юр_лица  IS NOT NULL) OR
(номер_физ_лица  IS NOT NULL  AND номер_юр_лица  IS NULL)
```

8.9.3. Определение первичных ключей

В принципе, можно создать таблицу и без первичного ключа. Но наличие у каждой таблицы первичного ключа – хороший стиль проектирования БД. Кроме того, если эта таблица является родительской для какой-либо другой таблицы, то определить первичный или уникальный ключ необходимо, чтобы можно было определить внешний ключ в подчинённой таблице. В качестве ПК следует брать тот уникальный атрибут сущности, по которому чаще всего происходит обращение к данным. Например, для БД налоговой инспекции это ИНН – индивидуальный номер налогоплательщика, а для БД ФОМС (фонда обязательного медицинского страхования) – номер медицинского полиса.

Если у сущности нет уникальных атрибутов, можно рассмотреть уникальные комбинации атрибутов. Но первичный ключ не должен быть длинным, т.к. ссылающийся на него внешний ключ будет занимать много памяти. Поэтому при отсутствии подходящих атрибутов нужно вводить суррогатный первичный ключ, который не несёт смысловой нагрузки и служит только для идентификации записей. (Некоторые СУБД позволяют определять значения такого ключа как AUTOINCREMENT, т.е. числовое поле, значение которого начинается с 1 автоматически увеличивается на 1 при добавлении новой записи).

8.9.4. Определение типов данных атрибутов

Определение типов данных для полей таблиц зависит от требований ПО. Но можно дать следующие общие рекомендации по выбору типов данных:

39. Для коротких символьных значений и символьных строк фиксированной длины следует выбирать тип CHAR. Например, для поля "единица измерения" со значениями 'кг', 'шт.', 'уп.' (char(3)), для поля "пол" (char(1)) и т.п.

40. Для символьных строк переменной длины нужно выбирать тип VARCHAR с указанием максимально возможной длины хранимого значения. Если при добавлении данных длина строки превысит указанное ограничение, система не сможет добавить данные и вернёт сообщение об ошибке.
41. Для числовых атрибутов, не участвующих в сложных расчётах, нужно использовать основной числовой тип реляционных СУБД – тип NUMBER, указывая реально необходимое количество разрядов. Например, для атрибута *Номер сотрудника* это может быть NUMBER(4) (до 10000 человек), а для зарплаты – NUMBER(8, 2) (до 999999.99 рублей).
42. Для числовых атрибутов, которые участвуют в сложных расчётах, следует использовать такие числовые типы, которые хранят данные в машинном (двоичном) представлении. Это ускорит выполнение расчётов.
43. Для числовых атрибутов, имеющих ведущие нули, следует выбирать тип CHAR, а не числовой тип, иначе ведущие нули будут потеряны. Например, для серии и номера паспорта (char(10)).
44. Для хранения дат нужно выбирать тип DATE или его варианты (DATETIME, например). Это позволит использовать арифметику дат и не заботиться о правильности вводимых данных: СУБД сама проверит допустимость даты.
45. Для хранения больших объектов (графических, звуковых и т.п.) следует выбирать специальные типы данных, перечень которых зависит от выбранной СУБД. Это могут быть типы LONG, CLOB (character large object), BLOB (binary large object) и другие.
46. Для семантически одинаковых полей разных таблиц нужно выбирать одинаковые типы данных. Например, ФИО сотрудника и ФИО клиента. Во многих СУБД для упрощения типизации данных можно создать специальные типы данных (create type) и использовать их в качестве типов полей таблиц.

8.9.5. Описание ограничений целостности

На этапе логического проектирования необходимо описать все ограничения целостности, обусловленные предметной областью. Типы ограничений целостности и ключевые слова SQL, которые позволяют описывать эти ограничения, приведены в п.6.1.

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

СУБД проверяет выполнение ограничений целостности при каждой операции модификации данных, если эта операция может нарушить целостность данных. Если ограничения целостности включены в схему

БД, они проверяются автоматически и нельзя внести в базу 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 команд языка DML.

Необходимо обратить особое внимание на поля таблиц, для которых домен определён как список возможных значений. Это ограничение целостности можно реализовать в виде: CHECK(<поле> IN (<список значений>)). Но такой подход имеет следующий недостаток: добавление нового значения в список потребует изменения схемы отношения (команда ALTER TABLE). Можно поступить до-другому: вынести этот список значений в отдельное от-ношение. Например, список типов образования (начальное, неполное среднее, среднее, средне-специальное, незаконченное высшее, высшее) для таблицы *СОТРУДНИКИ*. Таблица *ТИПЫ ОБРАЗОВАНИЯ* будет состоять из одного поля *Название типа*, определённого как первичный ключ. Тогда поле *Образование*таблицы *СОТРУДНИКИ* станет внешним ключом.

Определение списка значений позволяет гарантировать правильность вводимых данных и правильность поиска. Если не ограничивать значения поля, то оператор может ввести данные произвольным образом, например: 'незаконченное высшее', 'незаконч. высшее', 'н. высш.' и т.д. Человек понимает, что это одно и то же, а для СУБД это разные значения, и учесть все возможные комбинации в условии поиска очень сложно.

Если какое-либо ограничение целостности (ОЦ) нельзя реализовать средствами DCL, то возможны следующие способы его реализации:

47. С помощью процедурных объектов БД. Чаще всего для этой цели используются триггеры (trigger). **Триггер** – это процедура БД, которая привязана к конкретной таблице и вызывается автоматически при наступлении определённого события (добавления, удаления или модификации данных этой таблицы). Процедура триггера пишется на том языке, который поддерживается выбранной СУБД (например, PL/SQL для Oracle, Visual Basic для MS SQL Server). Триггер пишется программистом и выполняет те действия, которые обусловлены предметной областью. Например, триггер может осуществлять проверку "возраст принимаемого на работу сотрудника не может быть менее 16-и лет" или присваивать полю "Дата заказа" текущую дату при добавлении нового заказа. Если триггер диагностирует нарушение ограничений целостности, он выдаст сообщение об ошибке и команда модификации данных не будет выполнена (произойдёт автоматический откат, rollback).
48. Программно (т.е. через приложение). Для большей гарантии соблюдения ОЦ желательно проектировать программу так, чтобы

внесение изменений в данные и проверка ОЦ выполнялись в одном единственном месте.

49. Вручную. Ручная процедура обязательно должна быть описана в документации (в руководстве пользователя).

8.9.6. Аномалии модификации данных

После составления концептуальной (логической) схемы БД необходимо проверить её на отсутствие аномалий модификации данных. Дело в том, что при неправильно спроектированной схеме БД могут возникнуть аномалии выполнения операций модификации данных. Эти аномалии обусловлены ограниченностью структуры РМД (отсутствием агрегатов и проч.).

Рассмотрим эти аномалии на примере отношения со следующими атрибутами (атрибуты, входящие в ключ, выделены подчёркиванием):

*ПОСТАВКИ (**Номер поставки**, **Название товара**, Цена товара, Количество, Дата поставки,Название поставщика, Адрес поставщика)*

Различают три вида аномалий: аномалии обновления, удаления и добавления. Аномалия обновления может возникнуть в том случае, когда информация дублируется. Другие аномалии возникают тогда, когда две и более сущности объединены в одно отношение. Например:

50. **Аномалия обновления**: в отношении *ПОСТАВКИ* она может возникнуть, если у какого-либо поставщика изменился адрес. Изменения должны быть внесены во все кортежи, соответствующие поставкам этого поставщика; в противном случае данные будут противоречивы.
51. **Аномалия удаления**: при удалении записей обо всех поставках определённого поставщика все данные об этом поставщике будут утеряны.
52.
53. **Аномалия добавления**: в нашем примере она возникнет, если с поставщиком заключен договор, но поставок от него ещё не было. Сведения о таком поставщике нельзя внести в таблицу *ПОСТАВКИ*, т.к. для него не определён ключ (номер поставки и название товара) и другие обязательные атрибуты.

Для решения проблемы аномалии модификации данных при проектировании реляционной БД проводится нормализация отношений.

8.9.7 Нормализация отношений

В рамках реляционной модели данных Э.Ф. Коддом был разработан аппарат нормализации отношений и предложен механизм, позволяющий любое отношение преобразовать к третьей нормальной форме. Нормализация схемы отношения выполняется путём декомпозиции схемы.

Декомпозицией схемы отношения R называется замена её совокупностью схем отношений Aiтаких, что

$$R = \bigcup_i A_i$$

и не требуется, чтобы отношения Ai были непересекающимися. Декомпозиция отношения не должна приводить к потере зависимостей между атрибутами сущностей. Для декомпозиции должна существовать операция реляционной алгебры, применение которой позволит восстановить исходной отношение.

Покажем нормализацию на примере отношения *КНИГИ* (табл. 8.1):

Id - идентификатор (первичный ключ),
Code - шифр рубрики (по ББК – библиотечно-библиографической классификации),
Theme - название рубрики (по ББК),
Title - название книги,
Author - автор(ы),
Editor - редактор(ы),
Type - тип издания (учебник, учебное пособие, сборник и.т.п.),
Year - год издания
Pg - количество страниц

Введём понятие простого и сложного атрибута. **Простой атрибут** – это атрибут, значения которого атомарны (т.е. неделимы). **Сложный атрибут** может иметь значение, представляющее собой конкатенацию нескольких значений одного или разных доменов. Аналогом сложного атрибута может быть агрегат или повторяющийся агрегат данных.

Таблица 8.1. Исходное отношение *КНИГИ*

Id	*Code*	*Theme*	*Author*	*Title*	*Editor*	*Type*	*Year*	*Pg*
20	22.18	МК	Бочков С. Субботин Д.	Язык программирования СИ	Садчиков П. Седов П.	учебник	1990	384

10	22.18	МК	Джехани Н.	Язык АДА	Красилов А. Перминов О.	учебник	1988	552
35	32.97	ВТ	Соловьев Г. Никитин В.	Операционные системы ЭВМ		учебное пособие	1992	208
11	32.81	Кибернетика	Попов Э.В.	Общение с ЭВМ на естественном языке	Некрасов А.	учебник	1982	360
44	32.97		ПУ для ПЭВМ		Витенберг Э.	справочник	1992	208
89	32.973	ЭВМ	Коутс Р.Б Влейминк И.	Интерфейс «человек-компьютер»	Шаньгин В.	учебник	1990	501

Первая нормальная форма (1НФ).

Отношение приведено к 1НФ, если все его атрибуты простые.

Отношение *КНИГИ* содержит сложные атрибуты *Author* ("Авторы") и *Editor* ("Редакторы"). Для приведения к 1НФ требуется сделать все атрибуты простыми и ввести составной ключ отношения (*ID, Author* и *Editor*) (табл. 8.2).

Таблица 8.2. Отношение КНИГИ, приведённое к 1НФ

Id	*Code*	*Theme*	***Author***	*Title*	***Editor***	*Type*	*Year*	*Pg*
20	22.18	МК	Бочков С.	Язык программирования СИ	Садчиков П.	учебник	1990	384
20	22.18	МК	Субботин Д.	Язык программирования СИ	Седов П.	учебник	1990	384
10	22.18	МК	Джехани Н.	Язык АДА	Красилов А.	учебник	1988	552
10	22.18	МК	Джехани Н.	Язык АДА	Перминов О.	учебник	1988	552
35	32.97	ВТ	Соловьев Г.	Операционные системы ЭВМ		учебное пособие	1992	208
35	32.97	ВТ	Никитин	Операционные		учебное	1992	208

			В.	системы ЭВМ		пособие		
11	32.81	Кибернетика	Попов Э.В.	Общение с ЭВМ на естественном языке	Некрасов А.	учебник	1982	360
44	32.97		ПУ для ПЭВМ		Витенберг Э.	справочник	1992	208
89	32.973	ЭВМ	Коутс Р.Б	Интерфейс «человек-компьютер»	Шаньгин В.	учебник	1990	501
89	32.973	ЭВМ	Влейминк И.	Интерфейс «человек-компьютер»	Шаньгин В.	учебник	1990	501

Отношение в 1НФ является информационно-избыточным. Для такого отношения возможны все три вида аномалии. Если потребуется, например, изменить тип издания Джехани Н. «Язык АДА» с учебника на учебное пособие, то обновление должно коснуться двух записей, иначе возникнет нарушение логической целостности данных.

Введём понятие **функциональной зависимости**. Пусть X и Y – атрибуты (группы атрибутов) некоторого отношения. Говорят, что Y функционально зависит от X, если в любой момент времени каждому значению X=x соответствует единственное значение Y=y (X→Y). (При этом любому значению Y=y может соответствовать несколько значений X=(x1, x2,…)). Атрибут X в функциональной зависимости X→Y называется детерминантом отношения.

Проще говоря, функциональная зависимость имеет место, если мы можем однозначно определить значение атрибута (Y), зная значение некоторого другого атрибута (X). Например, если мы знаем название страны, то можем определить название её столицы, а по номеру зачётной книжки студента – группу, в которой он учится.

В нормализованном отношении все неключевые атрибуты функционально зависят от ключа отношения. Неключевой атрибут функционально полно зависит от составного ключа, если он функционально зависит от ключа, но не находится в функциональной зависимости ни от какой части составного ключа.

Вторая нормальная форма (2НФ).

Отношение находится во 2НФ, если оно приведено к 1НФ и каждый неключевой атрибут функционально полно зависит от составного ключа.

Для того чтобы привести отношение ко 2НФ, нужно:

- построить его проекцию, исключив атрибуты, которые не находятся в функционально полной зависимости от составного ключа;
- построить дополнительные проекции на часть составного ключа и атрибуты, функционально зависящие от этой части ключа.

Ключом отношения *КНИГИ* (табл. 8.2) является комбинация полей (***ID, Author, Editor***). Все поля, не входящие в состав ключа, зависят только от идентификатора книги. Поэтому отношение должно быть разбито на два: *КНИГИ* (табл. 8.3) и *КНИГИ–АВТОРЫ–РЕДАКТОРЫ* (табл. 8.4). Эти отношения связаны по внешнему ключу, которым является поле ID.

Таблица 8.3. Отношение *КНИГИ*, приведённое к 2НФ

Id	*Code*	*Theme*	*Title*	*Type*	*Year*	*Pg*
20	22.18	МК	Язык программирования СИ	учебник	1990	384
10	22.18	МК	Язык АДА	учебник	1988	552
35	32.97	ВТ	Операционные системы ЭВМ	учебное пособие	1992	208
11	32.81	Кибернетика	Общение с ЭВМ на естественном языке	учебник	1982	360
44	32.97	ВТ	ПУ для ПЭВМ	справочник	1992	208
89	32.973	ЭВМ	Интерфейс «человек-компьютер»	учебник	1990	501

Таблица 8.4. Отношение *КНИГИ–АВТОРЫ–РЕДАКТОРЫ* (2НФ)

Id	*Author*	*Editor*
20	Бочков С.	Садчиков П.
20	Субботин Д.	Седов П.
10	Джехани Н.	Красилов А. Перминов О.
10		Перминов О.
35	Соловьев Г.	
35	Никитин В.	
11	Попов Э.В.	Некрасов А.
44		Витенберг Э.
89	Коутс Р.Б	Шаньгин В.

89	Влейминк И.	

Отношение во 2НФ является менее избыточным, чем в 1НФ, но оно также не свободно от аномалий. Например, при удалении книги Попова «Общение с ЭВМ на естественном языке» мы потеряем информацию о том, что есть рубрика «Кибернетика» с кодом 32.81. И внести сведения о новой рубрике нельзя, пока в списке книг не появится хотя бы одна книга по этой рубрике.

Теперь рассмотрим понятие **транзитивной зависимости**. Пусть X, Y, Z – атрибуты некоторого отношения. При этом X→Y и Y→Z, но обратное соответствие отсутствует, т.е. Z не зависит от Y или Y не зависит от X. Тогда говорят, что Z транзитивно зависит от X (X→→Z).

Третья нормальная форма (3НФ).

Отношение находится в 3НФ, если оно находится во 2НФ и в нем отсутствуют транзитивные зависимости.

Для отношения *КНИГИ* (табл. 8.3) атрибут *Theme* зависит от атрибута *Code*, а не от ключа (хотя название рубрики, естественно, соответствует её шифру). Поэтому для приведения отношения к 3НФ (табл. 8.5) нужно выделить из него ещё одно отношение *РУБРИКАТОР* (табл. 8.6).

Таблица 8.5. Отношения *КНИГИ*, приведённые к 3НФ

ID	*Code*	*Title*	*Type*	*Year*	*Pg*
20	22.18	Язык программирования СИ	учебник	1990	384
10	22.18	Язык АДА	учебник	1988	552
35	32.97	Операционные системы ЭВМ	учебное пособие	1992	208
11	32.81	Общение с ЭВМ на естественном языке	учебник	1982	360
44	32.97	ПУ для ПЭВМ	справочник	1992	208
89	32.973	Интерфейс «человек-компьютер»	учебник	1990	501

Табл. 8.6. Отношение*РУБРИКАТОР*

Code	*Theme*
22.18	МК
32.97	ВТ
32.973	ЭВМ
32.81	Кибернетика

Отношения, находящиеся в 3НФ, свободны от аномалий модификации. Но для табл. 8.5 можно ещё вынести в отдельную таблицу поле *Tun*, чтобы реализовать ограничение на домен для этого поля. Таблица *ТИПЫ*

ИЗДАНИЙ будет состоять из одного поля *Название типа*, определённого как первичный ключ. Тогда поле *Тип* таблицы *КНИГИ* станет внешним ключом.

Примечание: если для атрибутов X,Y,Z есть транзитивная зависимость X →→ Z, и при этом Y → X или Z → Y, то такая зависимость не требует декомпозиции отношения. Например, для отношения *АВТОМОБИЛИ* с первичным ключом *Государственный номерной знак* и полями *№ кузова* и *№ двигателя* очевидно, что номера кузова и двигателя зависят как друг от друга, так и от первичного ключа. Но эта зависимость взаимно однозначная, поэтому декомпозиции отношения не нужна.

Введём понятие **многозначной зависимости**. Многозначная зависимость существует, если заданным значениям атрибута X соответствует множество, состоящее из нуля (или более) значений атрибута Y. Если в отношении есть многозначные зависимости, то схема отношения должна находиться в 4НФ.

Перефразируя вышесказанное, многозначная зависимость (X–»Y) имеет место, если по значению некоторого атрибута (X) мы можем определить набор значений другого атрибута (Y). Например, зная название страны, мы можем определить названия всех соседних с нею стран.

Различают тривиальные и нетривиальные многозначные зависимости. Тривиальной называется многозначная зависимость X–»Y, для которой Y d X или X U Y = R, где R – рассматриваемое отношение. Тривиальная многозначная зависимость не нарушает 4НФ. Если хотя бы одно из двух этих условий не выполняется (т.е. Y не является подмножеством X или X U Y состоит не из всех атрибутов R), то такая многозначная зависимость называется *нетривиальной*.

Четвертая нормальная форма (4НФ).

Отношение находится в 4НФ, если оно находится в 3НФ и в нем отсутствуют нетривиальные многозначные зависимости.

Отрицательный момент в нарушении 4НФ заключается в том, что в одно отношение включаются независящие друг от друга атрибуты. Например, у книги «Язык программирования СИ» (табл. 8.1) два автора и два редактора, но это не значит, что редактор Садчиков редактировал автора Бочкова, а редактор Седов – автора Субботина. А если у книги нет автора или редактора, то соответствующие поля останутся пустыми (null). Т.е. в отношении *КНИГИ–АВТОРЫ–РЕДАКТОРЫ* (табл. 8.4) атрибуты *Author* и *Editor* образуют две многозначные зависимости от ключа, и при этом значения этих атрибутов не зависят друг от друга. Поэтому для приведения отношения к 4НФ нужно разбить его на два отношения *КНИГИ–АВТОРЫ* и *КНИГИ–РЕДАКТОРЫ* (табл. 8.7, 8.8).

Таблица 8.7. Отношение *КНИГИ–АВТОРЫ* (4НФ)	
ID	*Author*
200	Бочков С.
200	Субботин Д.
100	Джехани Н.
300	Крон Г.1
876	Гик Е.Я.
385	Фролов Г.
385	Кузнецов Э.

Таблица 8.8. Отношение*КНИГИ–РЕДАКТОРЫ*(4НФ)	
ID	*Editor*
200	Садчиков П.
300	Баранов А.
876	Кикоин И.
876	Капица С.
440	Витенберг Э.
385	Храмов А.
385	Рожков П.

Обратите внимание, в отношениях, полученных после приведения к 4НФ, первичный ключ (ПК) состоит из всех атрибутов отношения.

Примечание. Есть ещё т.н. третья усиленная форма (НФБК – нормальная форма Бойса-Кодда) и 5НФ. Описание этих форм можно найти в [1].

Нормализация сокращает дублирование данных, но появление новых отношений усложняет схему базы данных.

8.9.8. Денормализация отношений

Иногда после нормализации отношений проводят их денормализацию. Обоснованием денормализации может служить требование обеспечения определённой производительности для критических запросов. В нормализованной БД одна сущность ПО разбивается на несколько отношений, и для получения исходного отношения требуется выполнить операцию соединения. Эта операция занимает много времени, поэтому нормализация может привести к падению производительности БД. Денормализация бывает нескольких видов:

56. Восходящая

    Подразумевает перенос некоторой информации из подчинённого отноше-ния в родительское. Например, для схемы *ЗАКАЗЫ <–>> СТРОКИ ЗАКАЗОВ* обычно нужно знать общую сумму заказа, которая является вычисляемой величиной. Если эти вычисления производятся часто, то для повышения эффективности можно добавить в отношение*ЗАКАЗЫ* поле *Общая сумма*. При этом возникает проблема обеспечения актуальности значения этого поля: пересчёт общей суммы при добавлении новой строки заказа, при удалении и при модификации позиций заказа. Обычно эта проблема решается программно (в приложении) или с помощью триггеров БД.

57. Нисходящая

    В этом случае информация переносится из родительского отношения в подчинённое. Например, в схеме *ДОЛЖНОСТИ <–>> СОТРУДНИКИ* можно в качестве внешнего ключа использовать поле *Название должности*, а не идентификатор. Для этого название в отношении *ДОЛЖНОСТИ* должно быть определено как *unique*. Это позволяет избежать соединения отношений при запросе должности сотрудника.

58. 3. Разбиение одного отношения на два.

    В одно отношение помещаются все атрибуты сущности, которые связаны с экземпляром сущности как 1:1. Но бывает так, что запись имеет большую длину за счёт наличия атрибутов большого объёма (графические данные, текстовые описания и проч.). Если данные этих атрибутов редко используются, то можно выделить в отдельное отношение атрибуты большого объёма. Для связи с исходным отношением вводится уникальный внешний ключ. А для получения исходного отношения создаётся представление (view), которое является соединением двух полученных отношений.

После нормализации/денормализации получается окончательная концептуальная схема БД, на основе которой проводится физическое проектирование. Пример проектирования БД с использованием метода "сущность-связь" и нормализации отношений приведён в [2].

> "Решенные проблемы исчезают в прошлое. Поставленные рожда-ют будущее. В особенности принципиально неразрешимые проблемы. Они вечны. Человек вообще начинается со стремления сделать невоз-можное".
>
> «Зияющие высоты», Александр Зиновьев, советский философ, логик, социолог, публицист

9. 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 около 20%.

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

59. **Хранилища данных и OLAP-обработка**. Хранилище данных – это пред-метно-ориентированный, интегрированный, привязанный ко времени и неизменяемый набор данных, предназначенный для поддержки принятия решений. Хранилище данных позволяют сохранять исторические данные с целью анализа и прогнозирования развития ситуаций. При правильном проектировании хранилище данных даёт высокую отдачу за счёт более качественного управления работой организации (предприятия). Данные в хранилище данных 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).
60. **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.
61. **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 предложенном 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.

62. 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.
63. **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 (гео-информационные системы), издательские системы (с поддержкой вёрстки номеров издания), САПР (системы автоматизации проектирования) и т.д.
64. **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 сохра-нить все преимущества баз данных для использования в web-приложениях. Основными задачами здесь являются:
    a. организация эффективного интерфейса, рассчитанного на неподготов-ленного пользователя;
    b. оптимизация запросов, направленная на уменьшение сетевого трафика;
    c. повышение производительности СУБД в многопользовательском режиме работы.
65. **Самоадаптация**. Современные СУБД имеют широкие возможности по на-стройке 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 БД СУБД должна брать на себя большинство функций настройки и выполнять их в автоматическом или автоматизированном режиме.

66. **Использование GRID**. GRID – это концепция объединения вычислитель-ных ресурсов в единую сеть. В качестве аналогии здесь можно привести электрические сети: при возникновении потребности пользователь просто подключается к сети и получает электричество. Точно так же 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, первые промышленные GRID-системы уже существуют, но поддерживают они только базы данных: это системы Oracle 10G и Oracle 11G (G – это сокращение от GRID). Они динамически выделяют ре-сурсы для выполнения задач пользователя по доступу к БД Oracle и пере-распределяют нагрузку на узлы сети с целью оптимизации использования вычислительных ресурсов и повышения общей производительности систе-мы.

67. **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 (т.е. метаданных).
68. **Технологии разработки данных и знаний** (data mining и knowledge mining). Технологии разработки данных предназначены для поиска неоче-видных тенденций и скрытых закономерностей в больших объёмах дан-ных. А knowledge mining – это извлечение знаний из баз данных (или из хранилища данных). 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.

Более подробно с этими направлениями развития технологии БД можно ознакомиться в [1] и на сайте citforum.ru/database.

10. Список используемых сокращений

DCL data control language, язык контроля данных
DDL data definition language, язык определения данных
DML data manipulation language, язык модификации данных
RBS rollback segment, сегмент отката
АИС автоматизированная информационная система
БД база данных
ИМД иерархическая модель данных
ИПС информационно–поисковая система
КБД ключ базы данных
ОП оперативная память
ОС операционная система
ПО предметная область
ППО прикладное программное обеспечение
РК резервная копия
РМД реляционная модель данных
РСУБД реляционная система управления базами данных
СМД сетевая модель данных
СОД системы обработки данных
СПО системное программное обеспечение
СУБД система управления базами данных

11. Библиографический список

69. 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.
70. Проектирование реляционной базы данных: Метод. указания к курсовому проек-тированию по курсу "Базы данных" / Московский государственный институт элек-троники и математики; Сост.: Карпова И.П. – М., 2003. – 28 с.
71. Изучение основ языка SQL: Метод. указания к лабораторным работам по курсу "Базы данных" / Московский государственный институт электроники и математики; Сост.: И. П. Карпова. М., 2009. – 32 с.
72. Манифест "Системы баз данных третьего поколения". – Журнал «СУБД»,•1995, № 2. – с. 143-159. http://rema.44.ru/resurs/study/ddb/manifest.html
73. Манифест «Системы объектно-ориентированных баз данных». – Журнал «СУБД»,•1995, № 4. – с. 142-155. http://rema.44.ru/resurs/study/ddb/manif_oo.html

74. ГОСТ 20886-85. Организация данных в системах обработки данных. Термины и определения.
75. ГОСТ 34.320-96. Информационные технологии. Система стандартов по базам дан-ных. Концепции и терминология для концептуальной схемы и информационной базы. – Межгосударственный стандарт. Дата введения 01.07.2001.
76. Clegg, Dai and Richard Barker, Case Method Fast-Track. A PAD Approach, Addison-Wesley, 1994.
77. Вендров А.М. CASE-технологии. Современные методы и средства проектирования информационных систем. – http://www.citforum.ru/database/case/index.shtml.