МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Тульский государственный университет»

Политехнический институт
Кафедра «Технологические системы пищевых, полиграфических
и упаковочных производств»

Утверждено на заседании кафедры
«Технологические системы пищевых,
полиграфических и упаковочных про-
изводств»
«26» января 2022 г., протокол № 6

Заведующий кафедрой
_____В.В. Прейс

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ПРАКТИЧЕСКИМ (СЕМИНАРСКИМ) ЗАНЯТИЯМ ПО ДИСЦИПЛИНЕ (МОДУЛЮ)

«Компьютерные технологии»

основной профессиональной образовательной программы высшего образования – программы бакалавриата

по направлению подготовки
29.03.03 Технология полиграфического и упаковочного производства

с направленностью (профилем)
Технология полиграфического производства

Формы обучения: заочная

Идентификационный номер образовательной программы: 290303-01-22

Тула 2022 год

ЛИСТ СОГЛАСОВАНИЯ
методических указаний по выполнению практических занятий
дисциплины (модуля)

Разработчик:

Проскуряков Н.Е., профессор, докт. техн. наук, профессор
(ФИО, должность, ученая степень, ученое звание)

(подпись)

СОДЕРЖАНИЕ:

Практическое занятие № 1

Проектирование реляционных баз данных

Цель работы: ознакомиться с общей характеристикой процесса проектирования, изучить структуру информационно-логической модели информационной системы. Рассмотреть проектную документацию.

Теоретические сведения

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

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.

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.

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 несанкцио-ниро-ванного 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.

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 информа-

ционной 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. 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 предметной области (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 воз-

можных 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.

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.

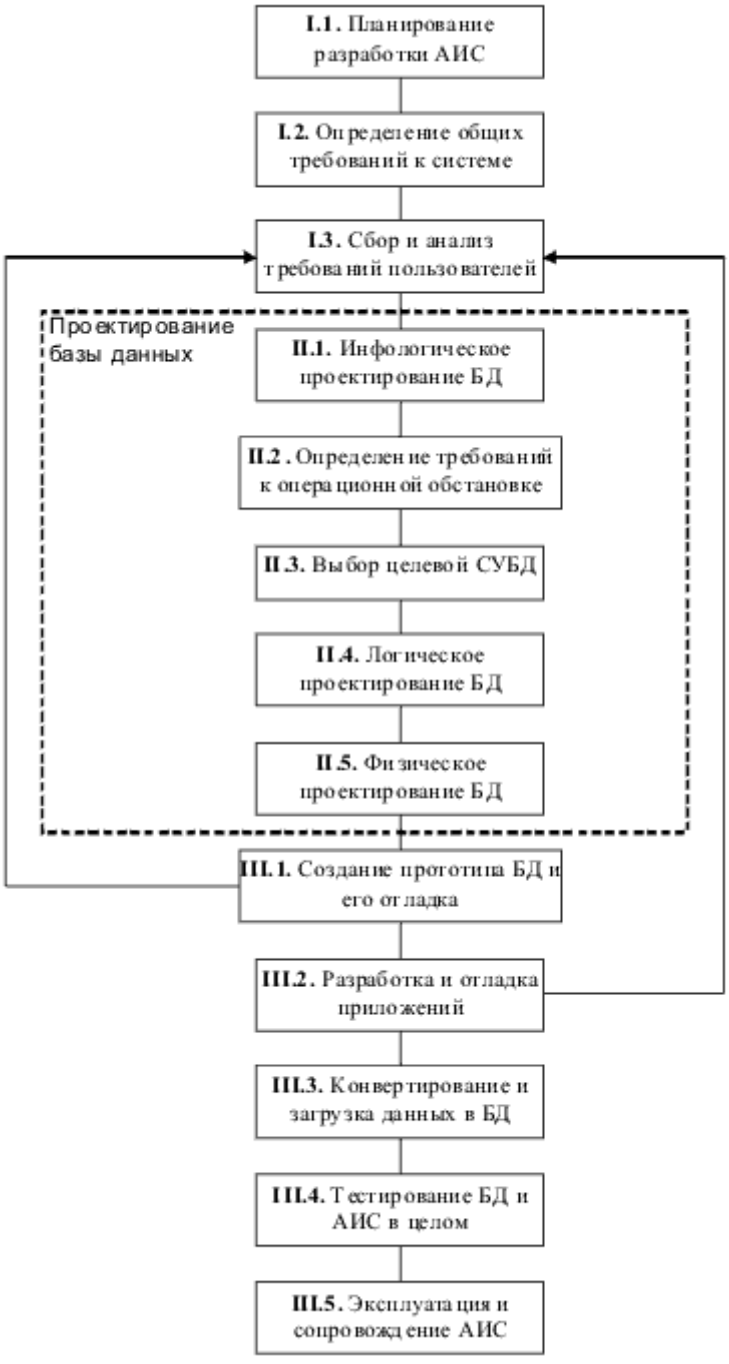

Рис. 1. Жизненный цикл приложения баз данных

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:

- 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.
- 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.
- 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 про-граммного 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 внесения изменений в систему. Это может быть вызвано изменениями предметной 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 предотвратить 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.

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.

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).

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 абстракций:

- 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:

1. **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).
2. **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*.
3. **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.

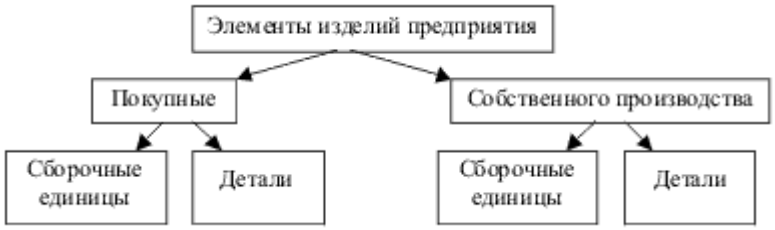

Рис. 8.2. Использование обобщений при объединениях

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:

1. 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.

2. Выделение групп пользователей системы. Каждая группа выполняет определённые задачи и обладает разными правами доступа к системе.

3. 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.

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.

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 100М, 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.

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).

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.

Порядок выполнения практической работы:

1. Собрать предварительную информацию об исследуемом предприятии.
2. Сформулировать видение выполнения проекта и границы проекта.
3. Составить отчет об обследовании.
4. Получить следующие данные:
   - Краткая информация о компании (профиль клиента).
   - Цели проекта.
   - Подразделения и пользователи системы.

Практическое занятие № 2
Выполнение информационного анализа описания предметной области (ПО), определение логической структуры базы данных

Цель занятия: выполнить информационный анализ предметной области исследуемого предприятия, определение логической структуры и составление ER- диаграммы.

Теоретические сведения

Разработка любой АИС начинается с определения предметной области.

Предметная область информационной системы

Предметная область (ПО) информационной системы рассматривается как совокупность реальных процессов и объектов (**сущностей**), предоставляющих интерес для её пользователей. 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.

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.

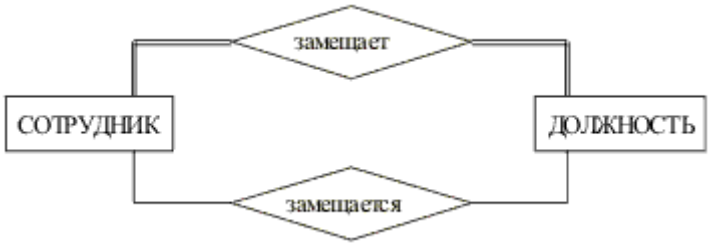

Рис.3. Примеры обязательной и факультативной связей

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).

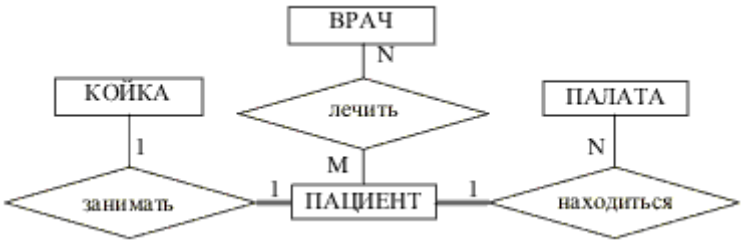

Рис.4. Примеры различной кардинальности связей

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. 5,а), бинарные (рис. 5,б) и тернарные (рис.5,в) связи. (На практике связи с большей степенью редко используются). 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)

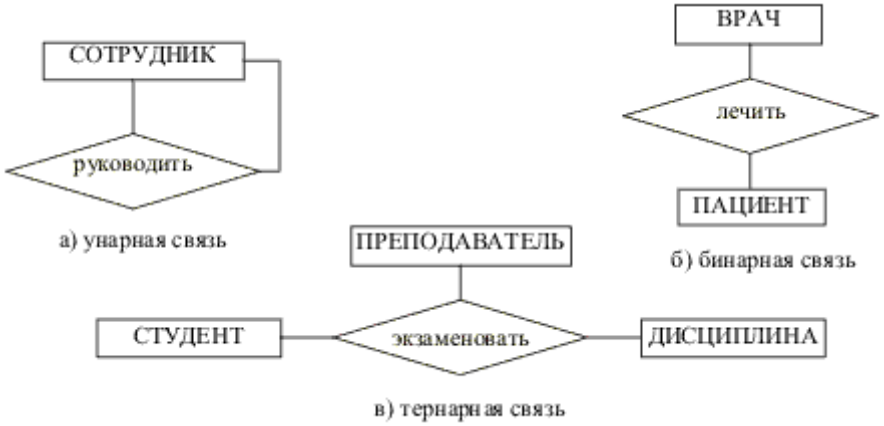

Рис.5. Примеры связей различной степени

Различают тип *связи* и *экземпляр связи*. Тип связи определяется её именем, 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 обрабатыва-ются 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.

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 данным через СУБД.

Рис.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.

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.7 Уровни представления данных

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

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.8 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. 9,а) и составным (включающим наряду с элементами данных и другие агрегаты, рис. 9,б).

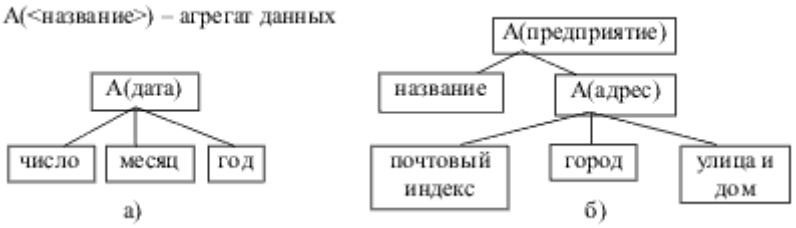

Рис.9 Примеры агрегатов: а) простой и б) составной агрегат

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. 10.

Рис.10 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. 11).

Рис. 11 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.

Порядок выполнения практической работы:

1. Составить ER - диаграмму предметной области обследуемого предприятия.
2. Провести классификацию пользователей по уровню доступа к данным.

Практическое занятие № 3
Алгоритмы выявления функциональных зависимостей данных.

Цель работы: выявить функциональные зависимости данных исследуемого предприятия

Теоретические сведения.

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. Транзак-

ция либо выполняется полностью, т.е. выполняются все входящие в неё операции, либо не выполняется совсем, если в процессе её выполнения возникает ошибка.

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 целостности. Статические ограничения присущи всем состояниям ПО, а динамические определяют возможность перехода ПО из одного состояния в другое. Примерами статических ограничений целостности могут служить требование уникальности 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.

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 декартовом 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 различные множества (домены). **Декартово произведение** этих множеств определяется следующим образом:

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

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

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

Подмножество декартова произведения доменов называется **отношением**.

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*).

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 отношений. Атрибутам реляционного отношения назначаются имена, уникальные в рамках отношения. Обращение к отношению происходит по его имени, а обращение к атрибуту – по имени отношения и имени атрибута.

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 обязательного 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 сущностей разных типов (родительской и подчинённой сущностей).

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.

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:

1. **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.
2. **Аномалия удаления**: при удалении записей обо всех поставках определённого поставщика все данные об этом поставщике будут утеряны.
3.

4. **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.

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 A_i таких, 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. Суббо-тин Д.	Язык про-граммиро-вания СИ	Садчи-ков П. Седов П.	учеб-ник	1990	384
10	22.18	МК	Дже-хани Н.	Язык АДА	Краси-лов А. Перми-нов О.	учеб-ник	1988	552
35	32.97	ВТ	Соло-вьев Г. Ники-тин В.	Операцион-ные си-стемы ЭВМ		учеб-ное по-собие	1992	208
11	32.81	Кибер-нетика	Попов Э.В.	Общение с ЭВМ на естествен-ном языке	Некра-сов А.	учеб-ник	1982	360
44	32.97		ПУ для ПЭВМ		Витен-берг Э.	спра-вочник	1992	208
89	32.973	ЭВМ	Коутс Р.Б Влей-минк И.	Интерфейс «человек-компьютер»	Шань-гин В.	учеб-ник	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 про-граммиро-вания СИ	Садчи-ков П.	учеб-ник	1990	384
20	22.18	МК	Суббо-тин Д.	Язык про-граммиро-вания СИ	Седов П.	учеб-ник	1990	384
10	22.18	МК	Дже-хани Н.	Язык АДА	Краси-лов А.	учеб-ник	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 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	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=(x_1, x_2,…)). 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	Киберне-тика	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

. 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 *Тип*, 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 $X \rightarrow\rightarrow Z$, и при этом $Y \rightarrow X$ или $Z \rightarrow Y$, 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*.

Порядок выполнения практической работы:

1. Выполнить построение реляционных отношений предметной области исследуемого предприятия
2. Выполнить нормализацию отношений

Практическое занятие № 4
Объектно-ориентированный подход к организации баз данных.

Цель работы: ознакомиться с концепцией объектно-ориентированного проектирования баз данных и СУБД

Теоретические сведения.

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.

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 дан-ных

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.

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 позволяет различать и поддерживать два независимых взгляда на БД: "взгляд" пользователя, воплощаемый в "логическом" представлении данных, и "взгляд" системы – "физическое" представление (организация хранимых данных).

Для инициализации базы данных разработчик средствами конкретной СУБД описывает логическую структуру БД, её организацию в среде хранения и пользовательские представления данных (соответственно концептуальную схему БД, схему хранения и внешние схемы). Обрабатывая эти схемы, СУБД создаёт пустую БД требуемой структуры и предоставляет средства для наполнения её данными предметной области и дальнейшей эксплуатации.

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).

Правила Кодда для реляционной СУБД (РСУБД)

1. Явное представление данных (The Information Rule). Информация должна быть представлена в виде данных, хранящихся в ячейках. Данные, храня-щиеся 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 дан-ными, 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.

Контрольные вопросы:

1. Исторические аспекты появления реляционного подхода создания баз данных и его содержание

2. Основные характеристики ООСУБД

3. Свойства и виды отношений

4.Основные правила Кодда для реляционной СУБД

Порядок выполнения работы:

1.Ознакомиться с теоретическими сведениями

2. Ответить на контрольные вопросы

Практическое занятие № 5
Сравнение объектно-ориентированных и объектно-реляционных СУБД. Достоинства и недостатки.

Цель работы: провести сравнительный анализ реляционного и объектно-реляционного подхода к проектированию баз данных.

Теоретические сведения.

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,).

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.

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. 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.

Основные функции реляционной СУБД

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

1. **Поддержка многопользовательского режима доступа.** База данных создаётся для решения многих задач многими пользователями. Это подразумевает возможность одновременного доступа многих пользователей к данным. Данные в БД являются разделяемым ресурсом, и РСУБД должна обеспечивать разграничение доступа к ним.
2. **Обеспечение физической целостности данных.** Проблема обеспечения физической целостности данных обусловлена возможностью разрушения данных в результате сбоев и отказов в работе вычислительной системы или в результате ошибок пользователей. Развитые РСУБД позволяют в большинстве случаев восстановить потерянные данные. Восстановление данных чаще всего основано на периодическом создании резервных копий БД и ведении журнала регистрации изменений (журнала транзакций) (см. 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.

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

Отношение R

A	B	C
a	b	c
c	a	d
c	b	d

а)

Секция $\Pi_{AC}(R)$

A	C
a	c
c	d

б)

Рис.2.10. Пример проекции отношения

2. **Селекция** (selection)

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

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

Отношение R

A	B	C
a	b	c
c	a	d
c	b	d

а)

Секция $\sigma_{C=d}(R)$

A	B	C
c	a	d
c	b	d

б)

Рис. 2.11. Пример селекции отношения

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

Это бинарная операция над разносхемными отношениями, 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

а)

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

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

Отношение S

A	B	C
g	h	a
h	d	d

Разность R–S

A	B	C
c	a	d
c	h	c

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.

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

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

Отношение R

A	B	C	D

Отношение S

C	D

Частное R/S

A	B

a	b	c	b
c	f	g	h
a	v	c	b
a	b	g	h
c	v	g	h
c	f	c	b

g	h
c	b

a	b
c	f

а) б)

Рис. 2.15. Пример операции деления

Языком обработки данных, основанным на реляционной алгебре, является SQL

Порядок выполнения практической работы:

1. Выполнить сравнительный анализ рассмотренных ранее методов проектирования применительно к обследуемому предприятию.
2. Обосновать выбор того или иного средства проектирования.

Практическое занятие № 6
Принцип проектирования и использования многомерных баз данных

Цель работы: ознакомиться с принципами создания и использования многомерных баз данных.

Теоретические сведения:

Вот уже более 30-и лет базы данных являются одной из одной из наиболее широко востребованных информационных технологий. Некоторые авторы утверждают, 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:

1. **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).

2. **Работа с неточными данными**. Информация в базах данных часто содержит ошибки или является неполной. Результаты запроса по такой БД могут сильно отличаться от реального положения дел. Процессор запросов, работающий с вероятностями, коэффициентами доверия, коэффициентами полноты и т.д. позволил бы учитывать степень достоверности данных при принятии решений на основе этих данных.

3. **Новые пользовательские интерфейсы**. Это одно из наиболее актуальных направлений современных информационных технологий. Конечные пользователи не знают язык запросов (SQL), и для получения информации из БД вынуждены пользоваться интерфейсами, которые для них создают программисты. В приложения обычно включают некоторый набор готовых запросов и возможность сформулировать произвольный запрос с помощью некоего конструктора. Но для того, чтобы воспользоваться конструктором, пользователь должен знать структуру базы данных и хорошо разбираться в 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.

4. Проблемы **оптимизации запросов**. Помимо остающейся актуальной задачи поиска новых способов оптимизации, можно выделить ещё две серьёзные проблемы оптимизации: обработка неструктурированных запросов (возможно, на ограниченном естественном языке), и оптимизация группы запросов. Работа с неструктурированными запросами особенно актуальна в свете использования баз данных в поисковых системах (в том числе, при поиске в Internet). А оптимизация группы одновременно выполняющихся запросов позволит улучшить характеристики СУБД с точки зрения быстродействия.

5. **Интеграция разнородных и слабо формализованных данных**. Изначально базы данных предназначались для хранения и обработки фактографических хорошо структурированных данных. Но огромное количество данных представлено в различных графических и мультимедийных форматах. Включение в СУБД способов обработки подобных данных позволяет исполь-

зовать 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.

6. **Организация доступа к базам данных через Internet**. Многие web-сайты содержат динамическую информацию, например, о товарах и ценах в Internet-магазинах. В локальных системах такая информация традиционно хранится в базах данных. Интеграция СУБД в web-среду позволяет 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.

7. **Самоадаптация**. Современные СУБД имеют широкие возможности по настройке баз данных под конкретную предметную область и аппаратные средства. Но использование этих возможностей – достаточно сложная задача, которая требует наличия высококвалифицированного администратора БД. Для упрощения настройки и сопровождения БД СУБД должна брать на себя большинство функций настройки и выполнять их в автоматическом или автоматизированном режиме.

8. **Использование GRID**. GRID – это концепция объединения вычислительных ресурсов в единую сеть. В качестве аналогии здесь можно привести электрические сети: при возникновении потребности пользователь просто подключается к сети и получает электричество. Точно 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 ресурсы для выполнения задач пользователя по доступу к БД Oracle и перераспределяют нагрузку на узлы сети с целью оптимизации использования вычислительных ресурсов и повышения общей производительности системы.

9. **Сохранность данных**. Количество накопленных цифровых данных в мире огромно. Но со временем устаревают и форматы хранения данных, и средства доступа к ним. Происходит также старение носителей: размагничиваются магнитные ленты и диски, изменяются оптические и физические свойства носителя. Поэтому даже архивированные данные могут стать недоступными, особенно если нет устройства для чтения устаревшего носителя или от-

сутствует 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).

10. **Технологии разработки данных и знаний** (data mining и knowledge mining). Технологии разработки данных предназначены для поиска неочевидных тенденций и скрытых закономерностей в больших объёмах данных. А knowledge mining – это извлечение знаний из баз данных (или из 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.

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.

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 последней завершённой 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 запись изменений в БД до момента успешного выполнения всех операций, входящих 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.

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 сег-мента 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.

Взаимовлияние транзакций

Транзакции в многопользовательской БД должны быть изолированы друг от друга, т.е. в идеале каждая из них должна выполняться так, как будто выполняется только она одна. В реальности транзакции выполняются одновременно и могут влиять на результаты друг друга, если они обращаются к одному и тому же набору данных и хотя бы одна из транзакций изменяет данные.

В общем случае взаимовлияние транзакций может проявляться в виде:

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

Потеря изменений могла бы произойти при одновременном обновлении двумя и более транзакциями одного и того же набора данных. Транзакция,

закончившаяся последней, перезаписала бы результаты изменений, внесённых предыдущими транзакциями, и они были бы потеряны.

Представим, что одновременно начали выполняться две транзакции:

```
транзакция 1 –  UPDATE      СОТРУДНИКИ
SET      Оклад = 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.", "экономист", 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 несогласован-ность 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 запро-

сах 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.

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.

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 T_i на отношениях R_j. Транзакция T_1 заблокировала данные B_1 в от-ношении R_1 и ждёт освобождения данных B_2 в отношении R_2, которые заблокированы транзакцией T_2, ожидающей освобождения данных B_3 в отношении R_3, заблокированных транзакцией T_3, которая не может продолжить выполнение из-за транзакции T_1. Если не предпринимать никаких дополнительных действий, то эти транзакции никогда не завершатся, т.к. они вечно будут ждать друг друга.

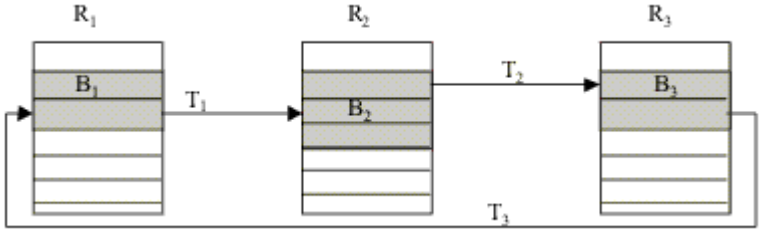

Рис.5.3. Взаимная блокировка трех транзакций

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 после-дующим рестартом через случайный промежуток времени.

5.5. Временные отметки

Использование временных отметок относится к оптимистическим алгоритмам 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 T_i имеет временную отметку t_i, и каждый элемент данных в БД (запись или блок) имеет две отметки: $t_{read}(x)$ – временная отметка транзакции, которая последней считала элемент x, и $t_{write}(x)$ – временная отметка транзакции, которая последней записала элемент x.

При выполнении транзакции T_i система сравнивает отметку t_i и отметки $t_{read}(x)$ и $t_{write}(x)$ элемента $\boldsymbol{x}$ для обнаружения конфликтов:

1. для читающей транзакции $\boldsymbol{T_i}$: если $\boldsymbol{t_i < t_{write}(x)}$, то элемент данных $\boldsymbol{x}$ перезаписан более поздней транзакцией, и его значение может оказаться несогласованным с теми данными, которые эта транзакция уже успела прочитать.
2. для пишущей транзакции:

- если $\boldsymbol{t_i < t_{read}(x)}$, то элемент данных x считан более поздней транзакцией. Если транзакция $\boldsymbol{T}$ изменит значение элемента $\boldsymbol{x}$, то в другой транзакции может возникнуть ошибка.
- если $t_i < t_{write}(x)$, то элемент x перезаписан более поздней транзакцией, и транзакция $\boldsymbol{T}$ пытается поместить в БД устаревшее значение элемента $\boldsymbol{x}$.

Во всех случаях обнаружения конфликта система перезапускает текущую транзакцию $\boldsymbol{T_i}$ 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.

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 дан-ные 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 переза-писаны. При этом будет выдано сообщение об ошибке и опера-

цию 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).

Контрольные вопросы:

1. Что такое хранилище данных и OLAP обработка?
2. Какие проблемы возникают при параллельной обработке транзакций?
3. Какие типы блокировок различают? Как формулируется протоколо блокировки?
4. Какими основными свойствами должна обладать любая из транзакций при работе многопользовательских баз данных?

Порядок выполнения работы:

1. Ознакомиться с теоретическими сведениями
2. Ответить на контрольные вопросы.

Практическое занятие № 7
Технология хеширования и индексирования

Цель работы: ознакомиться с физической организацией данных баз данных

Теоретические сведения

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).
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 независимость 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.

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.

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 разбива-

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

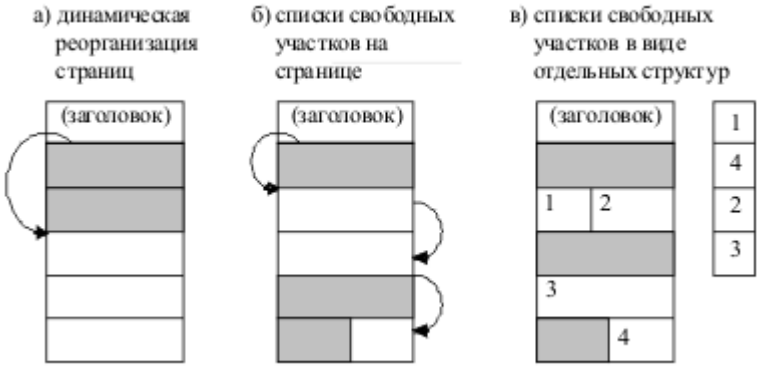

Рис. 4.1. Управление свободным простанством памяти на страницах

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 свободных 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 пространстве памяти и используемые методы доступа называются *схемой хранения*. Схема хранения оперирует в терминах типов объектов.

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).

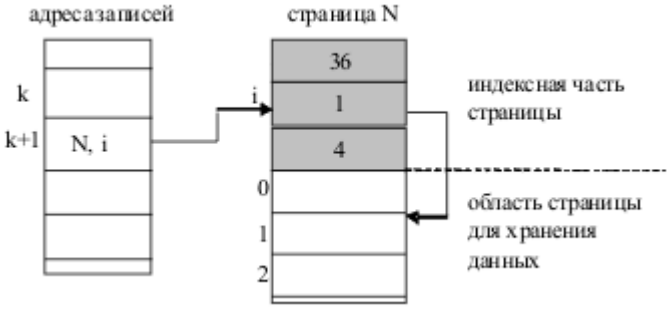

Рис. 4.2. Косвенная адресация с использованием индексируемых страниц

При перемещении записи она остаётся на той же странице, и слот попрежнему 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 вариант 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.

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.

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 запоминании записи (для определения места размещения записи в памяти). В базах данных применяются такие способы доступа по ключу, как индексирование, хеширование и кластеризация.

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.

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

Пространство памяти		
F6:00	Волкова	...
F6:1E	Волков	...
F6:31	Поспелов	...

...

FA:00	Белова	...	
FA:1D	Фридман	...	
FA:2B	Осипов	...	

Рис. 4.3. Пример индекса

Индекс обычно хранится в отдельном файле или отдельной области памяти. 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.

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

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

2000	01.12.95	A2	Партия	114.0	2000	02.12.95	A1
2000	01.12.95	A1	Amos ltd.	52.8			

Рис. 4.4. Пример составного индекса

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).

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.

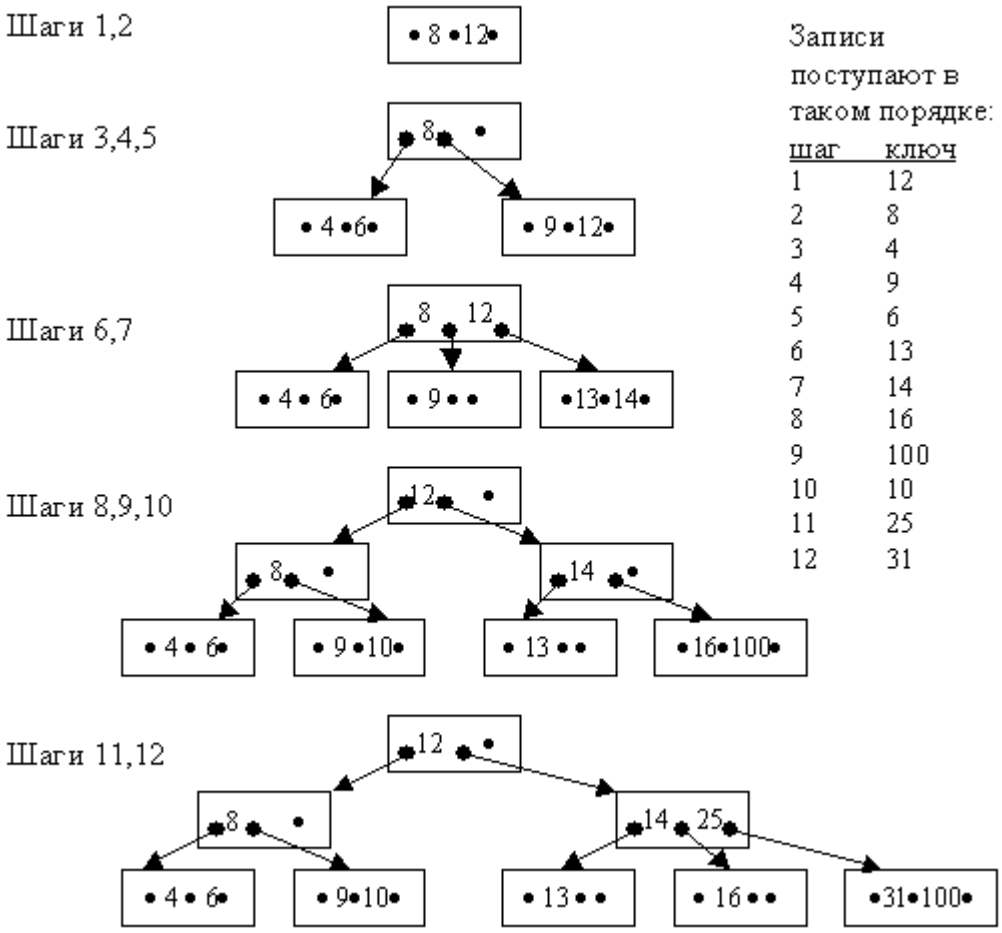

Рис.4.5. Пример построения B-дерева порядка 3

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).

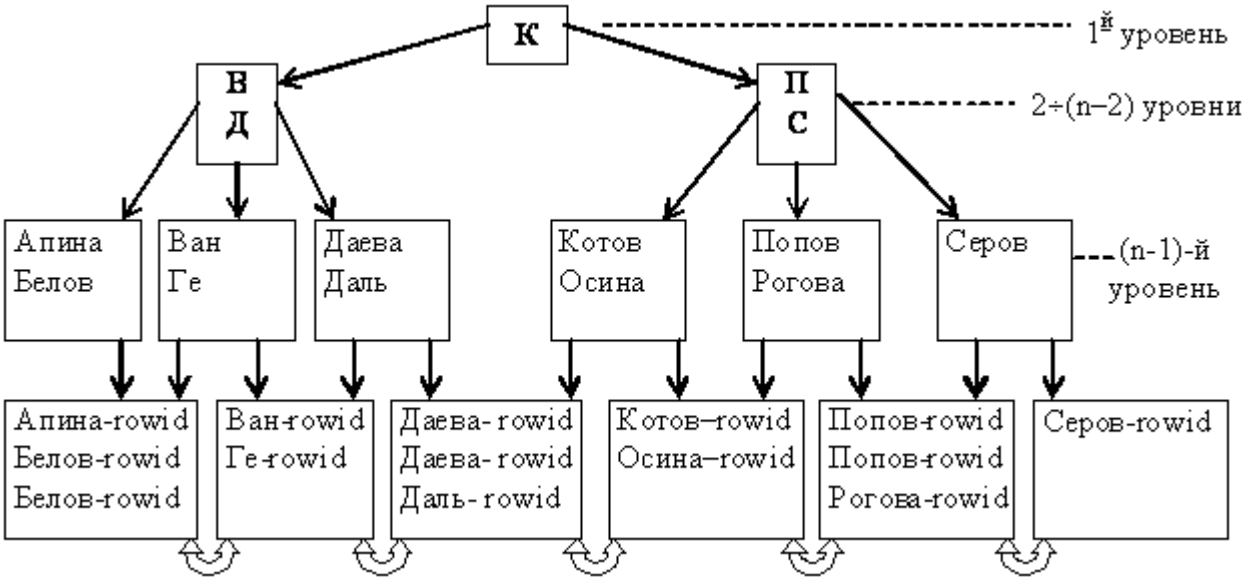

Рис.4.6. Пример индексного блока СУБД Oracle

Организация индексов в СУБД Oracle несколько отличается от рассмотренной 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 X_0 и конечное значения $X_к$ ключа, а также три указателя. Если значение искомого ключа меньше, чем X_0, то происходит обращение по левому указателю; если оно больше, чем $X_к$, то происходит обращение по правому указателю; если оно попадает в диапазон $X_0 \div X_к$ – 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 ин-

декса 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.

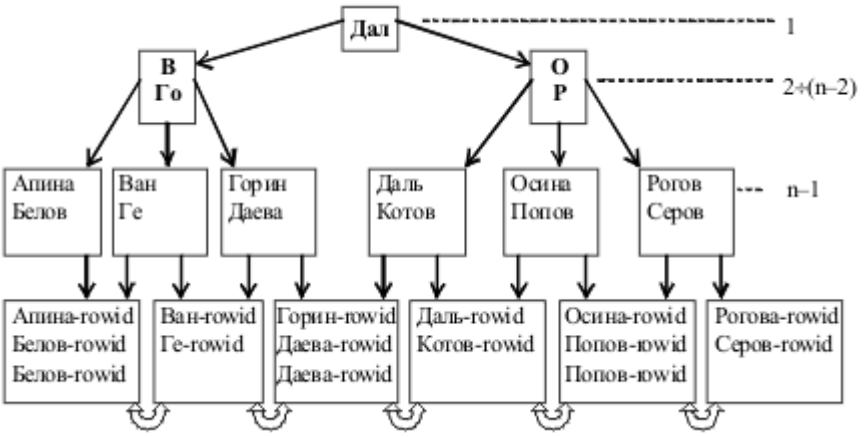

Рис.4.7. Пример перераспределения данных индексного блока СУБД Oracle

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.

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 ин-дексу. 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 блоки-ровки 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 об-новления ведут к потере времени на обновление индекса.

- Не индексируются столбцы, которые часто используются как аргументы выражений или функций: как правило, это не позволяет использовать ин-декс.

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.

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:

$K_1 \approx K_2 \Rightarrow h(K_1) >> h(K_2)$ V $K_2 >> h(K_1)$

2. для разных значений ключа выдавать разные адреса:

$K_1 \neq K_2 \Rightarrow h(K_1) \neq h(K_2)$

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.

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:

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

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:

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

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 $\boldsymbol{w}$ – размер машинного слова (обычно, 2^{31}); $\boldsymbol{A}$ – целое число простое по отношению к $\boldsymbol{w}$; а $\boldsymbol{M}$ – некоторая степень основания системы счисления ЭВМ (2^m). Таким образом, в качестве значения функции берутся $\boldsymbol{M}$ правых значащих цифр дробной части произведения значения ключа и константы $\boldsymbol{A/w}$. 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:

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

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

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).

Контрольные вопросы:

1. Что собой представляет страничная организация данных?
2. Что собой представляет технология хеширования, для чего она используется в структурах хранения БД?
3. В чем состоит основное назначение индексов?
4. Какой индекс называют плотным?
5. В чем состоит технология поиска, вставки и удаления элемента с использованием неплотного индекса?
6. Для чего строятся инвертированные списки.

Порядок выполнения работы:

1. Ознакомиться с теоретическими сведениями
2. Ответить на контрольные вопросы.

Список использованных источников:

1 Основная литература

1. Мурат Е.П. Информатика III : учебное пособие / Мурат Е.П.. — Ростов-на-Дону, Таганрог : Издательство Южного федерального университета, 2018. — 150 c. — ISBN 978-5-9275-2689-5. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/87415.html

2. Старыгина С.Д. Информатика: технологии и офисное программирование : учебное пособие / Старыгина С.Д., Нуриев Н.К., Нургалиева А.А.. — Казань : Казанский национальный исследовательский технологический университет, 2018. — 232 c. — ISBN 978-5-7882-2565-4. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/100670.html.

3. Оболонин И.А. Основы компьютерного проектирования в инфокоммуникационных технологиях : учебно-методическое пособие / Оболонин И.А.. — Новосибирск : Сибирский государственный университет телекоммуникаций и информатики, 2018. — 250 c. — ISBN 2227-8397. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: http://www.iprbookshop.ru/84070.html.

4. Кузнецов С.Д. Основы баз данных: курс лекций. Учеб. пособие. - М.: Интернет-Университет Информ. Технологий, 2005.- 488 с.

5 Джорджес Г. 50 эффективных приемов обработки цифровых фотографий с помощью Photoshop / Г. Джорджес; пер. с англ. С.Д. Панасюка; под ред. В.С. Иващенко. - М. [и др.] : Диалектика, 2006 .— 464 с.

4. Гурский, Ю. Компьютерная графика: Photoshop CS3, CorelDRAW X3, Illustrator CS3 / Ю. Гурский, И. Гурская, А. Жвалевский .— М.[и др.] : Питер, 2008 .— 992 с.

Дополнительная литература

1. Мельников В.П. Информационные технологии. Учебник для ВУЗов. М.: Академия, 2008. – 426 с.

2. Кузин А.В. Базы данных: учеб. пособие для вузов / А. В. Кузин, С. В. Левонисова .— 2-е изд., стер .— М.: Академия, 2008. – 316 с.

3. Грекул В.И. Проектирование информационных систем: курс лекций: учеб. пособие для вузов / В.И. Грекул, Г.Н. Денищенко, Н.Л. Коровкина. - М.: Интернет-Ун-т Информ. Технологий, 2005. — 304с.

4. Волкова, Е.В. Художественная обработка фотографий в Photoshop / Е.В. Волкова. - М.[и др.] : Питер, 2005 .— 269 с.

Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины (модуля)

1. Научная Электронная Библиотека eLibrary - библиотека электронной периодики.- Режим доступа: http://elibrary.ru/ , по паролю.- Загл. с экрана.

2. Электронный читальный зал “БИБЛИОТЕХ” : учебники авторов ТулГУ по всем дисциплинам.- Режим доступа: https://tsutula.bibliotech.ru/, по паролю.- Загл. с экрана

3. ЭБС Издательства «Лань» e.lanbook.- Режим доступа: http://e.lanbook.com/, по паролю.- Загл. с экрана.

4. ЭБС «КнигаФонд» (ООО «Центр цифровой дистрибуции») knigafund.- Режим доступа: http://www.knigafund.ru/.- Загл. с экрана.

5. ЭБС IPRBooks универсальная базовая коллекция изданий. – Режим доступа: http://www.iprbookshop.ru, по паролю.