ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Б1.Б.15 СУБД и базы данных

Направление подготовки (специальность) *09.03.01 "Информатика и вычислительная техника"*

Профиль образовательной программы *"Автоматизированные системы обработки информации и управления"*

Форма обучения *очная*

СОДЕРЖАНИЕ

1.Конспект лекций.....3
1.1 Лекция № 1 *Введение в базы данных* 3
1.2 Лекция № 2 *Обзор современных систем управления базами данных*.....4
1.3 Лекция № 3 *Архитектура СУБД*.....6
1.4 Лекция № 4 *Модели данных*.....8
1.5 Лекция № 5,6 *Реляционная модель данных* 10
1.6 Лекция №7,8 *Реляционная алгебра и язык SQL* 17
1.7 Лекция № 9 *Проектирование концептуальной модели данных*.....20
1.8 Лекция № 10 *Проектирование логической модели данных* 27
1.9 Лекция № 11 *Физическая модель данных*.....33
1.10 Лекция № 12,13 *Администрирование базы данных* 34
1.11 Лекция № 14 *Словарь данных*.....38
1.12 Лекция № 15 *Общая характеристика баз знаний и экспертных систем* 41
1.13 Лекция № 16 *СУБД ACCESS*.....43
1.14 Лекция № 17 *Создание локального приложения в СУБД* 48
2. Методические материалы по выполнению лабораторных работ.....51
2.1 Лабораторная работа № ЛР-1,2 *Введение в базы данных* 51
2.2 Лабораторная работа № ЛР-3,4 *Обзор современных систем управления базами данных*.....53
2.3 Лабораторная работа № ЛР-5,6 *Архитектура СУБД*.....54
2.4 Лабораторная работа № ЛР-7,8 *Модели данных* 54
2.5 Лабораторная работа № ЛР-9,10,11,12 *Реляционная модель данных* 57
2.6 Лабораторная работа № ЛР-13,14,15,16 *Реляционная алгебра и язык SQL*.....60
2.7 Лабораторная работа № ЛР-17,18 *Проектирование концептуальной модели данных* 66
2.8 Лабораторная работа № ЛР-19,20 *Проектирование логической модели данных.* 68
2.9 Лабораторная работа № ЛР-21, 22 *Физическая модель данных* 70
2.10 Лабораторная работа № ЛР-23,24,25 *Администрирование базы данных*.....71
2.11 Лабораторная работа № ЛР-26,27 *Словарь данных* 71
2.12 Лабораторная работа № ЛР-28,29 *Общая характеристика баз знаний и экспертных систем* 71
2.13 Лабораторная работа № ЛР-30,31 *СУБД ACCESS*.....75
2.14 Лабораторная работа № ЛР-32,33, 34 *Создание локального приложения в СУБД*....82

1. КОНСПЕКТ ЛЕКЦИЙ

1.1 Лекция № 1 (2 часа).

Тема: «Введение в базы данных»

1.1.1 Вопросы лекции:

1. Понятие, назначение баз данных.
2. Основные компоненты баз данных.

1.1.2 Краткое содержание вопросов:

1. Понятие, назначение баз данных.

В современных базах данных хранятся не только данные, но и информация. **База данных** (БД) – организованная структура, предназначенная для хранения информации. Современные БД позволяют размещать в своих структурах не только данные, но и методы (т.е. программный код), с помощью которых происходит взаимодействие с потребителем или другими программно-аппаратными комплексами.

Системы управления базами данных (СУБД) – комплекс программных средств, предназначенных для создания структуры новой базы, наполнения ее содержанием, редактирования содержимого и визуализации информации. Под **визуализацией информации базы** понимается отбор отображаемых данных в соответствии с заданным критерием, их упорядочение, оформление и последующая выдача на устройство вывода или передача по каналам связи.

Существует много систем управления базами данных. Они могут по-разному работать с разными объектами и предоставляют пользователю разные функции и средства. Большинство СУБД опираются на единый устоявшийся комплекс основных понятий.

2. Основные компоненты баз данных.

БД может содержать разные типы объектов. Каждая СУБД может реализовывать свои типы объектов.

Таблицы – основные объекты любой БД, в которых хранятся все данные, имеющиеся в базе, и хранится сама структура базы (поля, их типы и свойства).

Отчеты – предназначены для вывода данных, причем для вывода не на экран, а на печатающее устройство (принтер). В них приняты специальные меры для группирования выводимых данных и для вывода специальных элементов оформления, характерных для печатных документов (верхний и нижний колонтитулы, номера страниц, время создания отчета и другое).

Страницы или страницы доступа к данным – специальные объекты БД, выполненные в коде HTML, размещаемые на web -странице и передаваемые клиенту

вместе с ней. Сам по себе объект не является БД, посетитель может с ее помощью просматривать записи базы в полях страницы доступа. Т.о., страницы – интерфейс между клиентом, сервером и базой данных, размещенным на сервере.

Макросы и модули – предназначены для автоматизации повторяющихся операций при работе с системой управления БД, так и для создания новых функций путем программирования. Макросы состоят из последовательности внутренних команд СУБД и являются одним из средств автоматизации работы с базой. Модули создаются средствами внешнего языка программирования. Это одно из средств, с помощью которых разработчик БД может заложить в нее нестандартные функциональные возможности, удовлетворить специфические требования заказчика, повысить быстродействие системы управления, уровень ее защищенности.

Запросы – служат для извлечения данных из таблиц и предоставления их пользователю в удобном виде. С их помощью выполняют отбор данных, их сортировку и фильтрацию. Можно выполнить преобразование данных по заданному алгоритму, создавать новые таблицы, выполнять автоматическое заполнение таблиц данными, импортированными из других источников, выполнять простейшие вычисления в таблицах и многое другое.

Особенность запросов состоит в том, что они черпают данные из базовых таблиц и создают на их основе временную *результирующую таблицу* (*моментальный снимок*) – образ отобранных из базовых таблиц полей и записей. Работа с образом происходит быстрее и эффективнее, нежели с таблицами, хранящимися на жестком диске.

Обновление БД тоже можно осуществить посредством запроса. В базовые таблицы все данные вносятся в порядке поступления, т.е. они не упорядочены. Но по соответствующему запросу можно получить отсортированные и отфильтрованные нужным образом данные.

Формы – средства для ввода данных, предоставляющие пользователю необходимые для заполнения поля. В них можно разместить специальные элементы управления (счетчики, раскрывающиеся списки, переключатели, флажки и прочее) для автоматизации ввода. Пример, заполнение определенных полей бланка. При выводе данных с помощью форм можно применять специальные средства их оформления.

1.2 Лекция № 2 (2 часа).

Тема: «Обзор современных систем управления базами данных»

1.2.1 Вопросы лекции:

1. Определение СУБД

2. Современные системы управления базами данных.

1.2.2 Краткое содержание вопросов:

1. Определение СУБД.

База данных (БД) – набор логически взаимосвязанных данных, описывающий информационное состояние объектов в различных предметных областях и обрабатываемые компьютерной техникой.

Системой управления базами данных является программная и языковая среда для создания, управления и обработки информационных баз.

Назначение СУБД:

- работа с базами на внешней (диски, ленты и т. д.) и оперативной памяти;
- совместный доступ пользователей;
- контроль изменений, архивирование и восстановление баз;
- предоставление языка доступа для обработки информации;
- утилиты для создания, модификации и управления базами.

2. Современные системы управления базами данных.

Технология “Клиент-сервер” – технология, разделяющая приложение- СУБД на две части: клиентскую (интерактивный графический интерфейс, расположенный на компьютере пользователя) и сервер, собственно осуществляющий управление данными, разделение информации, администрирование и безопасность, находящийся на выделенном компьютере. Взаимодействие “клиент-сервер” осуществляется следующим образом: клиентская часть приложения формирует запрос к серверу баз данных, на котором выполняются все команды, а результат исполнения запроса отправляется клиенту для просмотра и использования. Данная технология применяется, когда размеры баз данных велики, когда велики размеры вычислительной сети, и производительность при обработке данных, хранящихся не на компьютере пользователя (в крупном учреждении обычно имеет место именно такая ситуация). Если технология“клиент-сервер”на применяется, то для обработки даже нескольких записей весь файл копируется на компьютер пользователя, а только затем обрабатывается. При этом резко возрастает загрузка сети, и снижается производительность труда многих сотрудников.

MicrosoftAccess, MicrosoftVisualFoxPro, MicrosoftVisualBasic обеспечивают средства для создания клиентских частей в приложениях“клиент-сервер”, которые сочетают в себе средства просмотра, графический интерфейс и средства построения запросов, а Microsoft SQL Server является на сегодняшний день одним из самых мощных серверов баз данных.

OLE 2. 0 (ObjectLinkingandEmbedding – связывание и внедрение объектов) – стандарт, описывающий правила интеграции прикладных программ. Применяется для использования возможностей других приложений. OLE 2. 0 используется для определения и совместного использования объектов несколькими приложениями, которые поддерживают данную технологию. Например, использование в среде Access таблиц Excel и его мощных средств построения диаграмм или использование данных, подготовленных Access, в отчетах составленных в редакторе текстов Word (связывание или включение объекта).

OLE Automation (Автоматизация OLE) –компонент OLE, позволяющий программным путем устанавливать свойства и задавать команды для объектов другого приложения. Позволяет без необходимости выхода или перехода в другое окно использовать возможности нужного приложения. Приложение, позволяющее другим прикладным программам использовать свои объекты называетсяOLE сервером. Приложение, которое может управлять объектами OLE серверов называется OLE контроллер или OLE клиент. Из рассмотренных программных средств в качестве OLE серверов могут выступать MicrosoftAccess, а также MicrosoftExcel, Word и Graph.... MicrosoftVisualFoxPro 3. 0 и 5. 0 может выступать только в виде OLE клиента. RAD (RapidApplicationDevelopment – Быстрая разработка приложений) –подход к разработке приложений, предусматривающий широкое использование готовых компонентов и/или приложений и пакетов (в том числе от разных производителей). ODBC (OpenDatabaseConnectivity – открытый доступ к базам данных) –технология, позволяющая использовать базы данных, созданные другим приложением при помощи SQL.

SQL (StructuredQueryLanguage – язык структурированных запросов) – универсальный язык, предназначенный для создания и выполнения запросов, обработки данных как в собственной базе данных приложения, так и с базами данных, созданных другими приложениями, поддерживающими SQL. Также SQL применяется для управления реляционными базами данных.

VBA (VisualBasicforApplications – VisualBasic для Приложений) – разновидность объектно-ориентированного языка программирования VisualBasic, встраиваемая в программные пакеты.

1.3 Лекция № 3 (2 часа).

Тема: «Архитектура СУБД»

1.3.1 Вопросы лекции:

1. Архитектура СУБД.

2. Уровни архитектуры.

1.3.2 Краткое содержание вопросов:

1. Архитектура СУБД.

В среде СУБД можно выделить пять основных компонентов:

1) Аппаратное обеспечение. Одни СУБД предназначены для работы только с типами ОС или оборудования, другие могут работать с широким кругом аппаратного обеспечения и различными ОС. Для работы СУБД обычно требуется некоторый минимум оперативной и дисковой памяти, но ее может быть недостаточно для достижения приемлемой производительности системы.

2) Программное обеспечение. Этот компонент включает операционную систему, программное обеспечение самой СУБД, прикладные программы, включая и сетевое программное обеспечение, если СУБД используется в сети. Обычно приложения создаются на языках третьего поколения, таких как C, COBOL, Fortran, Ada или Pascal, или на языках четвертого поколения, таких как SQL, операторы которых внедряются в программы на языках третьего поколения.

3) Данные – наиболее важный компонент с точки зрения конечных пользователей. База данных содержит как рабочие данные, так и метаданные, т.е. "данные о данных".

4) Подсистема средств проектирования представляет собой набор инструментов, упрощающих проектирование и реализацию баз данных и их приложений. Как правило, этот набор включает в себя средства для создания таблиц, форм, запросов и отчетов.

5) Подсистема обработки обеспечивает обработку компонентов приложений, созданных с помощью средств проектирования. Например, в Access 2003 имеется компонент, реализующий построение формы и связывающий элементы формы с данными таблиц.

2) Уровни архитектуры

Существует три основных уровня архитектуры или три уровня описания элементов данных. Это внешний, концептуальный и внутренний уровни, которые формируют так называемую трёхуровневую архитектуру.

Внешний уровень - уровень, на котором данные воспринимаются пользователями, тогда как СУБД и операционная система воспринимают данные на внутреннем уровне.

Концептуальный уровень представления данных осуществляет отображение внешнего уровня на внутренний и обеспечивает требуемую независимость друг от друга.

Внешний уровень - это представление данных с точки зрения пользователей. Этот уровень описывает пользовательскую часть БД, т.е. часть, которая относится к каждому

пользователю. Внешний уровень состоит из нескольких внешних представлений Баз Данных. Каждый пользователь имеет дело с представлением «реального мира», выраженным в наиболее удобной для него форме. Внешнее представление содержит только те сущности, атрибуты и связи, которые интересны пользователю. Другие сущности, атрибуты и связи, которые ему неинтересны, также могут быть представлены в БД, но пользователь может даже не подозревать об их существовании.

Концептуальный уровень - это обобщающее преставление БД. Этот уровень описывает то, какие данные хранятся в БД, а также связи, существующие между ними. Этот уровень содержит логическую структуру всей БД (с точки зрения администратора БД). Фактически это полное представление требований к данным со стороны организации, которое не зависит от любых соображений относительно способа их хранения. На концептуальном уровне представлены следующие компоненты:

- все сущности, их атрибуты и связи;
- накладываемые на данные ограничения;
- семантическая информация о данных (связанная со значением, смыслом);
- информация о мерах обеспечения безопасности и поддержки целостности данных.

1. 4 Лекция № 4 (2 часа).

Тема: «Модели данных»

1.4.1 Вопросы лекции:

1. Понятие модели данных.
2. Иерархическая модель данных
3. Сетевая модель данных
4. Реляционная модель данных.

1.4.2 Краткое содержание вопросов:

1. Понятие модели данных

Под моделью данных понимают некоторую формальную теорию представления и обработки данных, включающую методы описания типов и логических структур данных (аспект структуры), методы манипулирования данными (аспект манипуляции) и методы описания и поддержки целостности данных (аспект целостности).

Модель данных – это способ моделировать, инструмент. Модель базы данных – это результат использования этого способа для проектирования базы данных.

Выделяют 5 моделей данных: иерархическая, сетевая, (не)реляционная и объектно-ориентированную. К ранним моделям данных принято относить иерархическую и сетевую

модели. СУБД, реализующие их, появились первыми и заложили основы технологий баз данных. Общие черты ранних моделей:

1. «вырастание» из практики. Сначала появлялись СУБД, а потом формулировались положения соответствующей модели данных. Как следствие, в основе иерархических и сетевых СУБД не лежит строгий и формальный матем. аппарат, а модели данных имеют скорее описательный характер.

2. организация доступа к данным на уровне отдельных записей. Иерархическая и сетевая модели предполагают операции с отдельными записями – поиск конкретной записи, переходы к следующей/предыдущей записям и так далее. Соответственно, и языки для работы с данными в иерархических и сетевых – императивные (после SQL была добавлена).

3. слаборазвитая (по сравнению с реляционными БД) система ограничений целостности.

2. Иерархическая модель данных

Иерархическая модель является первой и появилась вместе с наиболее известной СУБД, ее реализующей – системой IMS (Information Management System) разработки компании IBM. Эта СУБД была выпущена в 1968 году и существует и развивается до сих пор. Данные организуются в виде иерархии. Основными понятиями данной модели являются база данных, сегмент и поле.

Поле – это минимальная единица данных, которая представляет собой неделимое целое. То есть, например, если нам нужно хранить в поле адрес, то выделить в нем элементы (например, регион, город, улицу) и получать доступ к ним средствами СУБД невозможно, это нужно реализовывать на уровне прикладной программы. Набор полей образует сегмент (или запись). В сегменте должен быть определен ключ – поле или набор полей, значение которого может однозначно идентифицировать конкретный экземпляр сегмента. Сегменты образуют между собой связи типа «родитель-потомок». При этом у сегмента может быть произвольное количество потомков, но только один родитель. В вершине иерархии корневой сегмент - без родителя. Все сегменты в совокупности образуют древовидную структуру, которая и является базой данных. Схема базы данных – это набор таких деревьев.

3. Сетевая модель данных

СУБД, реализующие сетевую модель данных, появились почти одновременно с иерархическими СУБД. Появилась в 1971 году. Наиболее известная СУБД, реализующая архитектуру CODASYL, IDMS (принадлежит компании ComputerAssociates и продается под наименованием CA IDMS/DB).

Сетевую модель данных можно считать расширением иерархической модели. Если в последней у записи-потомка может иметься только один предок, то в сетевой – произвольное количество. Сама СУБД представляет собой наборы экземпляров записей и наборы экземпляров связей. Т.е., сетевая СУБД допускает наличие между двумя наборами экземпляров записи любого количества связей. При этом две записи могут меняться ролями родителя и потомка. Хотя, структура базы данных усложняется, это с лихвой окупается за счет исключения необходимости дублирования данных, которое происходило бы при моделировании сложных связей в иерархической БД. Как и иерархические, сетевые СУБД предполагают манипуляцию данными на уровне отдельных записей. Типичные операции: поиск записей, создание, уничтожение, изменение, включение в связь, исключение из связи, изменение связи.

Т.о., сетевые СУБД предоставляют больше возможностей, но одновременно, требуют больших вычислительных ресурсов и более сложных алгоритмов для работы с данными.

4. Реляционная модель данных.

Реляционная модель данных ведет свою историю с публикации в 1970 году работы Эдгара Кодда «Реляционная модель данных для больших совместно используемых банков данных». Она основывается на математическом «фундаменте». Ключевое понятие реляционной модели – отношение (англ. Relation) определено в терминах теории множеств. Соответственно, операции над отношениями также основываются на операциях над множествами. Сам Кодд предложил восемь операций, которые, в совокупности с отношениями, образуют реляционную алгебру. Предложенный набор операций избыточен, так как часть операций может быть выражена через другие. Но множество операций выбрано исходя из соображений удобства их использования. Уже из приведенного описания понятно, что реляционная модель ориентирована на работу с множествами записей, а не с отдельными экземплярами. Этим она радикально отличается от остальных моделей данных. Простота и мощь реляционной модели позволили реализующим ее СУБД завоевать первенство на рынке средств управления базами данных и вытеснить остальные решения на периферию.

1. 5 Лекция № 5,6 (4 часа).

Тема: «Реляционная модель данных»

1.5.1 Вопросы лекции:

1. Понятие домена, атрибута, кортежа, отношения.

2. Табличное представление отношения.

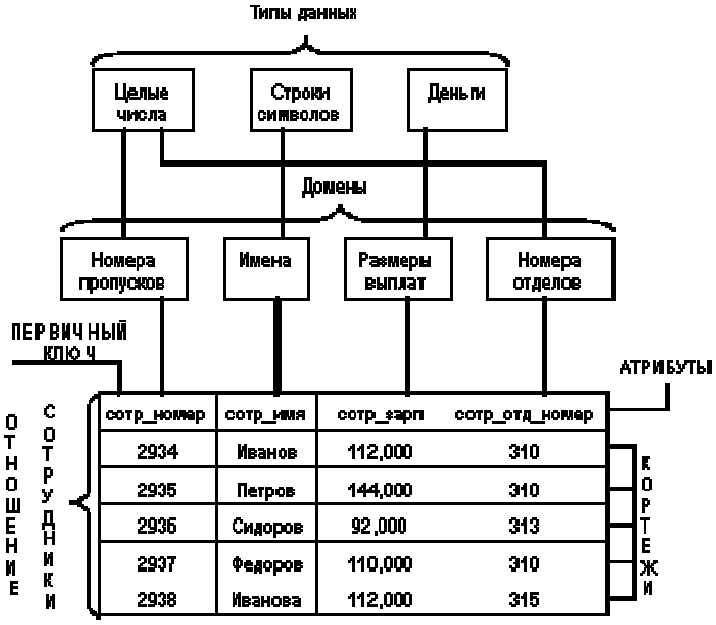
3. Схема отношения.

4. Первичные и внешние ключи

1.5.2 Краткое содержание вопросов:

1. Понятие домена, атрибута, кортежа, отношения.

Первые три относятся к элементам данных, остальные – к структурам, объединяющим элементы.



Для реляционной модели данных тип используемых данных не важен. Требование, чтобы тип данных был простым, нужно понимать так, что в реляционных операциях не должна учитываться внутренняя структура данных. Конечно, должны быть описаны действия, которые можно производить с данными как с единым целым, например, данные числового типа можно складывать, для строк возможна операция конкатенации и т.д.

В реляционной модели данных с понятием тип данных тесно связано понятие **домена**, которое можно считать уточнением типа данных.

Понятие домена более специфично для баз данных, хотя и имеет некоторые аналогии с подтипами в некоторых языках программирования. В самом общем виде домен определяется заданием некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу типа данных. Если вычисление этого логического выражения дает результат "истина", то элемент данных является элементом домена.

Наиболее правильной интуитивной трактовкой понятия домена является понимание домена как допустимого потенциального множества значений данного типа.

Следует отметить также семантическую нагрузку понятия домена: данные считаются сравнимыми только в том случае, когда они относятся к одному домену. В нашем примере значения доменов "Номера пропусков" и "Номера групп" относятся к типу целых чисел, но не являются сравнимыми. Заметим, что в большинстве реляционных СУБД понятие домена не используется.

Домен характеризуется следующими свойствами:

- Домен имеет уникальное имя (в пределах базы данных).
- Домен определен на некотором простом типе данных или на другом домене.
- Домен может иметь некоторое логическое условие, позволяющее описать подмножество данных, допустимых для данного домена.

Домен несет определенную смысловую нагрузку.

Отличие домена от понятия подмножества состоит именно в том, что домен отражает семантику, определенную предметной областью. Может быть несколько доменов, совпадающих как подмножества, но несущие различный смысл. Например, домены "Вес детали" и "Имеющееся количество" можно одинаково описать как множество неотрицательных целых чисел, но смысл этих доменов будет различным, и это будут различные домены.

Атрибут есть имя, поставленное в соответствие домену и представляющее семантически значимое свойство объекта ПО. Если домену поставлено в соответствие имя, то говорят, что на домене определен атрибут. Атрибут принимает значения на домене.

На одном и том же домене можно определить произвольное число атрибутов. Атрибуты, определенные на общем домене, наследуют его свойства.

Отношение интуитивно можно понимать как таблицу, заголовком которой является строка атрибутов, а значимыми строками – строки их значений, или как плоский файл, однако это неточные представления.

Определение 3. Множество кортежей SR, соответствующих одной и той же схеме R, называется отношением.

Отношение характеризуется:

- арностью (степенью) – числом пар <домен, атрибут> в схеме;
- мощностью – числом кортежей, составляющих тело отношения.

Отношение является единственной структурной единицей РМД.

Кортеж, соответствующий данной схеме отношения, - это множество пар {имя атрибута, значение}, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения. "Значение" является допустимым значением домена

данного атрибута (или типа данных, если понятие домена не поддерживается). Тем самым, степень или "арность" кортежа, т.е. число элементов в нем, совпадает с "арностью" соответствующей схемы отношения. Попросту говоря, кортеж - это набор именованных значений заданного типа.

Отношение - это множество кортежей, соответствующих одной схеме отношения. Иногда, чтобы не путаться, говорят "отношение-схема" и "отношение-экземпляр", иногда схему отношения называют заголовком отношения, а отношение как набор кортежей - телом отношения. На самом деле, понятие схемы отношения ближе всего к понятию структурного типа данных в языках программирования. Было бы вполне логично разрешать отдельно определять схему отношения, а затем одно или несколько отношений с данной схемой.

Свойства отношений РМД.

Отношения РМД обладают рядом свойств, отличающих их как от обычных теоретико-множественных отношений, так и от таблиц.

Уникальность кортежей. Так как отношение есть множество кортежей, в нем не может быть дубликатов кортежей, то есть каждый кортеж встречается в отношении только один раз. Из этого свойства следует, что каждое отношение имеет некоторый набор атрибутов, значения которых уникально идентифицирует кортежи. Этот набор атрибутов называют возможным ключом отношения.

Неупорядоченность кортежей. Это также следствие того, что отношение – множество кортежей. Множества неупорядоченны, если их упорядоченность специально не оговорена. Заметим, что в реальных структурах хранения данные так или иначе упорядочены. Однако учет этой упорядоченности в процедурах манипулирования данными сделал бы прикладные программы зависящими от физических структур хранения. Поэтому введение каких-либо гипотез об упорядоченности в концептуальную модель данных было бы ошибкой.

Неупорядоченность атрибутов. Это свойство следует из определения схемы отношения как множества пар <домен, атрибут>. Неупорядоченность атрибутов делает возможной модификацию схем отношений путем удаления атрибутов, вставки новых и переименования атрибутов и позволяет относительно просто определить ряд полезных операций над отношениями.

Уникальность атрибутов. Одноименные атрибуты недопустимы, поскольку это может привести к появлению в схеме отношения дубликатов пар (домен, атрибут), что противоречит определению. Кроме того, только уникальность атрибутов может обеспечить возможность отнесения значения из кортежа к определенному домену.

Атомарность значений атрибутов. Свойство следует из определения атрибута. Атрибут принимает значения на домене, а домен – подмножество простого типа. Таким образом, в реляционной теории не рассматриваются так называемые ненормализованные отношения.

Изменяемость отношений. Реляционная модель данных рассматривает отношение как структурный тип. Тип определяется схемой отношения. Все кортежи – знаки типа – удовлетворяют одной и той же схеме. Тело отношения может изменяться во времени. Отдельные кортежи могут добавляться или удаляться. Могут изменяться значения атрибутов в существующих кортежах. Поэтому можно говорить об экземпляре (текущем значении) отношения с заданной схемой.

Экземпляр (значение) отношения – это набор кортежей с заданной схемой, существующий в некоторый фиксированный момент времени.

2. Табличное представление отношения.

При табличном представлении каждому кортежу отношения взаимно-однозначно соответствует строка (запись) в таблице. Но не всякая таблица представляет некоторое отношение. Таблица — это прямоугольная сетка, в ячейках которой может находиться все, что угодно. При этом различные строки этой таблицы могут содержать в соответствующих столбцах одинаковые данные. Другими словами, таблицы могут содержать идентичные строки. В этом случае совокупность всех строк таблицы не является множеством, а значит, не представляет собой отношения. Напомню, что множество — это совокупность различных элементов. Таким образом, любое отношение (в определенном ранее смысле) можно представить в виде таблицы, но обратное утверждение, вообще говоря, не верно — не всякая таблица представляет какое-нибудь отношение.

При представлении отношения явным образом в виде таблицы, ее нельзя рассматривать просто как вместилище данных, в которое можно добавлять новые записи или удалять старые. Добавление, удаление и изменение данных в такой таблице приводит либо к другому отношению, либо к никакому отношению (если в таблице окажутся одинаковые записи).

Табличное представление отношений — это чрезвычайно плодотворный технологический прием, обеспечивший создание и широкое практическое применение баз данных. То обстоятельство, что отношения — просто множества, позволило изучать проблемы реляционных баз данных на теоретическом уровне, в мире математики (алгебра множеств и исчисление предикатов), а не только в мире технологии. В результате

разработка СУБД, конкретных баз данных, языков манипулирования данными, в том числе и 8<ЗЬ, водрузилась на твердый теоретический фундамент.

Далее мы будем рассматривать отношения, представляя их в виде таблиц. Введем несколько терминов, которые обычно применяются в теории отношений: Так, говоря о некотором отношении Я(А9 А29..., Ап), мы имеем в виду следующее.

□ Отношение имеет имя Л. Например, сведения о товарах, хранящихся на складе, образуют некоторое отношение, которому можно дать имя склад.

□ Множества А, А29..., Ап , для которых определено отношение Л, имеют различные имена, называемые атрибутами отношения. Мы будем считать, что А19 А29..., Ап — это атрибуты отношения. Совокупность всевозможных значений любого множества с именем А1 (/ = 1, 2,..., п) называется **доменом** атрибута Аг. Заметим еще раз, что в отношении не может быть двух и более одинаковых атрибутов, хотя их домены могут быть и одинаковыми (в смысле равенства множеств). Например, отношение склад может быть определено для атрибутов наименование, количество, цена и поставщик. Структуру этого отношения можно записать так: склад (наименование, количество, цена, поставщик). Каждый атрибут имеет свой домен — множество возможных значений. Например, доменом атрибута количество может быть множество неотрицательных чисел. При табличном представлении отношения каждому атрибуту соответствует заголовок столбца, а домену — множество значений в этом столбце.

□ Порядок перечисления атрибутов в структуре отношения не имеет значения. Иначе говоря, перестановка атрибутов местами оставляет само отношение прежним, хотя вид таблицы, представляющей это отношение, меняется. Например, отношения склад (наименование, количество, цена, поставщик) и склад (поставщик, наименование, количество, цена) считаются эквивалентными.

□ Элементами отношения являются кортежи — последовательности значений атрибутов отношения. В отношении не может быть двух и более одинаковых кортежей, а порядок расположения кортежей не имеет значения. При табличном представлении отношения каждому кортежу соответствует строка таблицы. Строки таблицы мы будем называть записями.

3. Схема отношения

Схема отношения базы данных — это именованное множество пар {имя атрибута, имя домена (или типа, если понятие домена не поддерживается)}. Если все атрибуты одного отношения определены на разных доменах, осмысленно использовать для именования атрибутов имена соответствующих доменов (не забывая, конечно, о том, что

это является всего лишь удобным способом именования и не устраняет различия между понятиями домена и атрибута).

Схема базы данных (в структурном смысле) — это набор именованных схем отношений.

Кортеж, соответствующий данной схеме отношения в базе данных, — это множество пар {имя атрибута, значение}, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения. «Значение» является допустимым значением домена данного атрибута (или типа данных, если понятие домена не поддерживается). Тем самым, степень или «арность» кортежа, т.е. число элементов в нем, совпадает с «арностью» соответствующей схемы отношения. Попросту говоря, кортеж — это набор именованных значений заданного типа.

Отношение — это множество кортежей данной базы данных, соответствующих одной схеме отношения. Иногда, чтобы не путаться, говорят «отношение-схема» и «отношение-экземпляр», иногда схему отношения называют заголовком отношения, а отношение как набор кортежей — телом отношения. На самом деле, понятие схемы отношения в базе данных ближе всего к понятию структурного типа данных в языках программирования.

Число атрибутов в отношении называют степенью (или -арностью) отношения.

Мощность множества кортежей отношения называют мощностью отношения.

4. Первичные и внешние ключи

Первичный ключ (англ. *primarykey*) — в реляционной модели данных один из потенциальных ключей отношения, выбранный в качестве основного ключа (или ключа по умолчанию).

Если в отношении имеется единственный потенциальный ключ, он является и первичным ключом. Если потенциальных ключей несколько, один из них выбирается в качестве первичного, а другие называют «альтернативными».

С точки зрения теории все потенциальные ключи отношения эквивалентны, то есть обладают одинаковыми свойствами *уникальности* и *минимальности*. Однако в качестве первичного обычно выбирается тот из потенциальных ключей, который наиболее удобен для тех или иных практических целей, например для создания внешних ключей в других отношениях либо для создания кластерного индекса. Поэтому в качестве первичного ключа, как правило, выбирают тот, который имеет наименьший размер (физического хранения) и/или включает наименьшее количество атрибутов.

Другой критерий выбора первичного ключа — сохранение уникальности со временем. Всегда существует вероятность того, что некоторый потенциальный ключ

перестанет быть таковым в долговременной перспективе или при изменении требований к системе. Например, если номер студенческой группы включает последнюю цифру года поступления, то номера групп для идентификации групп уникальны только в течение 10 лет. Поэтому в качестве первичного ключа стараются выбирать такой потенциальный ключ, который с наибольшей вероятностью не утратит уникальность.

Внешний ключ (англ. *foreignkey*) — понятие теории реляционных баз данных, относящееся к ограничениям целостности базы данных.

Неформально выражаясь, *внешний ключ* представляет собой *подмножество атрибутов* некоторой переменной отношения R_2, значения которых должны совпадать со значениями некоторого потенциального ключа некоторой переменной отношения R_1.

Формальное определение. Пусть R_1 и R_2 — две переменные отношения, не обязательно различные. Внешним ключом FK в R_2 является подмножество атрибутов переменной R_2 такое, что выполняются следующие требования:

1. В переменной отношения R_1 имеется потенциальный ключ CK такой, что FK и CK совпадают с точностью до переименования атрибутов (то есть переименованием некоторого подмножества атрибутов FK можно получить такое подмножество атрибутов FK', что FK' и CK совпадают как по именами, так и по типам атрибутов).

2. В любой момент времени каждое значение FK в текущем значении R_2 идентично значению CK в некотором кортеже в текущем значении R_1. Иными словами, в каждый момент времени множество всех значений FK в R_2 является (нестрогим) подмножеством значений CK в R_1.

При этом для данного конкретного внешнего ключа $FK \rightarrow CK$ отношение R_1, содержащее потенциальный ключ, называют *главным*, *целевым*, или *родительским* отношением, а отношение R_2, содержащее внешний ключ, называют *подчинённым*, или *дочерним* отношением.

Поддержка внешних ключей также называется соблюдением ссылочной целостности. Реляционные СУБД поддерживают автоматический контроль ссылочной целостности.

1.6 Лекция №7,8 (4 часа).

Тема: «Реляционная алгебра и язык SQL»

1.6.1 Вопросы лекции:

1. Особенности языков описания и манипулирования данными в реляционной модели языки запросов, основанные на реляционном исчислении.

2. Структурный язык запросов SQL

1.6.2 Краткое содержание вопросов:

1. Особенности языков описания и манипулирования данными в реляционной модели языки запросов, основанные на реляционном исчислении.

Реляционные языки по своей структуре значительно ближе к естественным языкам, чем логические языки. Это приводит к тому, что процедуры семантического этапа при переводе текстов на язык Яс и обратного перевода получаются для этого случая более простыми, чем аналогичные процедуры для логических языков. Но с точки зрения эквивалентных преобразований в реляционных языках нет возможности построить систему процедур, относительно которой можно было бы, как это сделано для исчисления высказываний или исчисления предикатов, доказать ее полноту и эффективность.

Реляционные языки различного типа хорошо описаны в отечественной литературе. Во второй из этих работ, посвященной некоторым философско-методологическим проблемам построения реляционных баз данных, приведена и формальная алгебраическая модель для реляционных алгебр.

Использование *реляционного языка SQL* поддерживает основные виды обработки данных таблиц. Самым мощным оператором языка SQL является оператор SELECT, обеспечивающий формирование результирующих таблиц.

Как и в *реляционном языке данных*, язык данных сетевой модели РГБД КОДАСИЛ предусматривает средства включения новых записей, обновления существующих, а также средства удаления записей.

Предоставляет для пользователя наряду с *реляционным языком программирования PAL* (ParadoxApplicationLanguage) [42] также и другие интерфейсы.

Такое постепенное обобщение описаний в *реляционных языках* приводит в конце концов к построению минимального описания с максимальной свободой в заполнении переменных конкретными понятиями и со свободными параметрами типа параметра q в примере 2.10. Это минимальное описание, сохраняющее смысл понятия (например, понятия колонна движущихся однородных объектов), называется структурным фреймом описания.

2. Структурный язык запросов SQL.

SQL (англ. StructuredQueryLanguage — язык структурированных запросов) — универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных.

Вопреки существующим заблуждениям, SQL в его чистом (базовом) виде является информационно-логическим языком, а не языком программирования. Вместе с тем стандарт языка спецификацией SQL/PSM предусматривает возможность его процедурных расширений, с учётом которых язык уже вполне может рассматриваться в качестве языка программирования.

SQL основывается на реляционной алгебре.

В начале 70-х годов SQL являлся лишь языком запросов (ЯЗ). Он, по сути, содержал только предложение SELECT, которое позволяло формулировать запросы для выборки данных из базы. Затем язык был дополнен двумя другими компонентами, необходимыми для работы с базами данных. Первый из них — средства для определения структуры базы данных, которые в терминологии теории баз данных называются языком определения данных (ЯОД). Второй — средства, позволяющие заполнять базу данными, изменять их и удалять. Этот компонент в теории баз данных называется языком манипулирования данными (ЯМД). Также было принято решение, что весь интерфейс с базами данных должен обеспечиваться одним языком, вследствие чего SQL оброс множеством функций, необходимых для управления базами данных. Приведем некоторые из них:

– определение, переопределение и удаление таблиц базы данных и других ее объектов (доменов, представлений, индексов, триггеров, хранимых процедур, функций и т. д.);

– указание физической организации данных;

– поддержка ограничений целостности и непротиворечивости базы данных;

– защита данных от несанкционированного доступа посредством определения пользователей (с именами и паролями) и ролей, прав доступа к данным и прав на изменение состояния базы данных;

– манипулирование данными в таблицах базы, включая вставку, изменение и удаление значений;

– поиск данных в нескольких таблицах и упорядочение полученных результатов;

– организация резервного копирования и восстановления базы данных;

– поддержка целостности транзакций;

– поддержка пользовательских процедур и функций, расширяющих функциональные возможности SQL.

SQL существует в двух формах. В интерактивном SQL пользователь непосредственно вводит команды и получает результат. Команды встроенного SQL

включаются в тексты программ на других языках. В этом случае обращение к базе данных, а также обработка результатов производится этими программами.

1.7 Лекция № 9 (2 часа).

Тема: «Проектирование концептуальной модели данных»

1.7.1 Вопросы лекции:

1. Анализ данных
2. Нормализация отношений
3. Графическое представление.

1.7.2 Краткое содержание вопросов:

1. Анализ данных

Одна из первых задач, с решением которых сталкивается разработчик программной системы - это изучение,осмысление и *анализ предметной области*. Дело в том, что предметная область сильно влияет на все аспекты проекта: требования к системе, взаимодействие с пользователем, модель хранения данных, реализацию и т.д.

Анализ предметной области, позволяет выделить ее сущности, определить первоначальные требования к функциональности и определить границы проекта. Модель *предметной области* должна быть документирована, храниться и поддерживаться в актуальном состоянии до этапа реализации. Для документирования могут быть использованы различные средства.

Для управления обсуждением области действия проекта можно использовать методику "будет - не будет". В простейшем случае - это *список* с двумя столбцами, в одном из которых записывается, что проект будет делать, а во втором - что не входит в проект. Такой *список*, формируется заинтересованными лицами при рассмотрении каждой *бизнес-цели проекта*, используя любую технику, например метод "мозгового штурма" (см. тему "Выявление требований"). Полученные характеристики позволяют четко определить границы проекта и довольно просто преобразуются в предположения, которые фиксируются в документе.

Функциональная *область действия* определяет услуги, предоставляемые системой, и вначале до конца неизвестны. В определении услуг системы может помочь *список* "Действующее лицо/Цель", в котором перечислены все цели пользователя, поддерживаемые системой. При его разработке в первую графу вписываются имена основных действующих лиц, т.е. тех, кто имеет цели, во вторую графу - цель каждого действующего лица, а в третью - приоритет или предположение о том, в какую версию войдет эта услуга. Формы списков приведены на рисунке.

Для определения основных функций продукта можно использовать, например, краткое описание варианта использования. Описание каждой функции можно представить также в виде списка, состоящего из трех *граф*: действующее лицо, цель и краткое описание варианта использования.

Анализ предметной области является основой для *анализа осуществимости* проекта и определения образа (концепции) продукта и границ проекта.

Анализ осуществимости

Разработка новых программных систем должна начинаться с *анализа осуществимости*. На основании анализа предметной области, общего описания системы и ее назначения необходимо принять решение о продолжении или завершении проекта. Для этого необходимо ответить на следующие вопросы.

- Отвечает ли система бизнес-целям организации-заказчика и организации-разработчика?
- Можно ли реализовать систему, используя известные технологии и не выходя за пределы заданной стоимости и заданного времени?
- Можно ли объединить систему с другими уже эксплуатируемыми системами?

Для ответа на первый (и главный) вопрос нужно опросить заинтересованных лиц, например, менеджеров подразделений, в которых будет использоваться система, для выяснения того,

- что произойдет с организацией, если система не будет введена в эксплуатацию;
- как система будет способствовать целям бизнеса;
- какие текущие проблемы поможет решить система и т.д.

После получения и обработки информации готовится отчет, в котором должны быть даны рекомендации относительно продолжения разработки системы.

Постановку бизнес-задачи надо обсуждать с Заказчиком, или будущим Владельцем системы.

Вопросы, которые ему стоит задать, это:

- Почему вообще пошла речь о создании системы?
- В чём Вы видите её назначение?
- Какие бизнес-возможности она должна реализовать?
- Какие проблемы должна решить?

В качестве "Стандарта" на вопросы такого рода смотрите *шаблон документа* "*Stakeholder* Request", например, из *RUP*. Бизнес-требования может выразить

Заказчик или Эксперты *предметной области*. Они обычно фиксируются в виде списка из 10-30 ключевых свойств продукта - в качестве шаблона см. Vision из *RUP*.

Бизнес-моделирование надо проводить на основе информации от, а лучше совместно с экспертами *предметной области*. Вопросы по сути сводятся к "Что, почему, когда, как и кем происходит в *предметной области* и как оно взаимосвязано?":

- Каковы основные понятия предметной области, их определения и взаимосвязи? Результат можно оформить в виде глоссария и/или концептуально-семантической модели предметной области.
- На основании каких правил - международных, федеральных, муниципальных, районных и т.д. законов, указов, стандартов, спецификаций, регламентов и т.д. - происходит то, что происходит в предметной области? Результат оформляете в виде структурированного списка или прикрепляете к элементам концептуальной модели.
- Что реально (какие процессы, события, факты) происходит и в какой последовательности, взаимосвязи? Результат оформляете в виде сценариев описания бизнес-процессов (что достаточно универсально) или диаграмм *SADT* (*IDEF0*, IDEF3, *DFD*) / *ARIS*(eEPC и т.д.) / UML (BusinessUse-caseDiagram (BUC) + *ActivityDiagram* + *SequenceDiagram*). Это один из сложнейших этапов.
- Какими свойствами обладает каждое из выделенных понятий - структурными и поведенческими? Результат описывается в виде таблиц с атрибутами Концептуальных сущностей или Детальной концептуальной моделью - ER - *IDEF1X* / UML *ClassDiagram*(*BOM*).

Существует российский стандарт *функционального моделирования* Р 50.1.028-2001, созданный на основе *IDEF0*.

Определение требований - частично Бизнес-требования и Требования, проистекающие из *предметной области* вы уже определили выше, теперь осталось исследовать *Пользовательские требования* и *Системные требования* и ограничения к отдельным аспектам качества системы. *Пользовательские требования*, как можно понять, нужно выявлять из общения с потенциальными пользователями системы. Вопросы:

- На какую систему будет похожа создаваемая?
- С какими системами и как давно вы работаете?
- Какое у вас образование?
- Каковы ваши ожидания от системы - что и как она должна делать, какие задачи помогать решать, как должна выглядеть?
- Какие шаги необходимо предпринять для решения каждой задачи?
- В каком случае вы будете считать, что система "Хороша"?

Результаты анкетирования/интервьюирования обычно представляют в виде пользовательских историй (*User Story*, Agile) или Пользовательских сценариев (Use-case), также возможно их диаграммное *представление* средствами диаграмм потока *работ* (IDEF3),*ARIS*, Activity/State *UML Diagram*. Подробнее про работу с Пользователями могут рассказать специалисты по Проектированию взаимодействия, интерфейсам и эргономике.

Системные требования нужно выяснять у IT-специалистов Заказчика, если таковые имеются, из специфики контекста использования системы, опыта построения аналогичных систем (у IT-Экспертов-Архитекторов) и Специалистов по отдельным аспектам системы, значимым для данного проекта (Юристы, Эргономисты, и т.д.) и Заказчика:

- Будет ли система единичной или тиражируемой?
- В каких странах она будет работать?
- Насколько важна информация, хранящаяся, обрабатываемая и передающаяся системой?
- Каков возможный ущерб от потери той или иной информации?
- Сколько пользователей будет работать с системой сегодня, завтра, через год?

Переработанный результат оформляется в виде Системных требований (*Software* Requirement *Specification*, стандарт IEEE-STD-830-1998, или ТЗ ГОСТ 34-602-89 или неформально в виде SupplementarySpecificaion из *RUP*).

Приложения для настольных компьютеров подобны широкоугольным объективам в том смысле, что в типичных случаях они отображают значительный объем информации, который позволяют предоставлять пользователю экраны большого размера. В отличие от этого мобильные приложения напоминают увеличительное стекло или объектив с переменным фокусным расстоянием. Они предоставляют пользователю возможность быстро просматривать необходимые подробные данные, быстро переходить к ограниченным наборам данных и получать к ним *доступ*, а также принимать решения в реальном масштабе времени. Как правило, мобильные приложения пре доставляют более специализированный набор сценариев по сравнению с приложениями, ориентированными на настольные компьютеры. Очень важно точно определиться с тем, на каких сценариях должно специализироваться ваше *приложение*.

Прежде чем приступать к реальной разработке приложения, определите *подмножество* функциональных средств, к которым *пользователь* сможет получать быстрый *доступ* в манере, свойственной мобильным устройствам. В случае

создания нового приложения, аналога которого для настольных компьютеров не существует, выпишите ключевые сценарии, которые пользователи смогут выполнять с помощью вашего приложения, а также порядок действий пользователя, обеспечивающий использование этих сценариев на мобильном устройстве. Во многих случаях вам будет легче придать этим сценариям реальные очертания, если вы подготовите соответствующие рисунки или создадите прототипы. Если подразумевается определенная *группа* конечных пользователей, пообщайтесь с ними и предоставьте им возможность поработать некоторое время с экспериментальными версиями своих приложений, чтобы они могли дать о них свои отзывы.

Оптимальный подбор предоставляемых средств определяет все остальное

Если вы правильно выделите ключевые сценарии и возможности вашего приложения, то это окажет определяющее влияние на всю оставшуюся часть процесса разработки. Наличие явно сформулированного описания того, как *конечные* пользователи будут использовать ваше *приложение*, и детальное понимание их потребностей окажут вам неоценимую помощь при настройке производительности приложения, а также проектировании пользовательского интерфейса, коммуникационной системы и модели памяти.

Если вы не определите важнейшие с вашей точки зрения сценарии и возможности, то в результате вы получите бессистемную смесь средств, объединенных в одно *приложение*. Отсутствие явного списка основных функций приложения или разделения функций на группы в соответствии с их приоритетами приведет к тому, что пользовательский *интерфейс* не будет оптимизирован для эффективного решения ключевых задач. Например, если ожидается, что *пользователь* в основном будет заинтересован во вводе данных, то вы должны оптимизировать пользовательский *интерфейс* таким образом, чтобы обеспечить как можно более точное и надежное выполнение операций ввода. И наоборот, если ввод данных используется лишь изредка, то вариант пользовательского интерфейса ввода, оптимизированного не самым идеальным образом, может оказаться вполне допустимым, что позволит перебросить ресурсы проектирования и разработки на другие направления. Лишь только если группой разработчиков будут идентифицированы, перечислены и согласованы наиболее важные сценарии, эксплуатационные характеристики приложения могут быть настроены для их выполнения должным образом, а *конечные* пользователи не будут лишены важных для них средств из-за недосмотра.

Чтобы процесс разработки мог быть успешно завершен, составьте *список* ключевых требовании, которым должно удовлетворять *приложение*, и возможностей, которые оно должно обеспечивать, и пусть этот *список* будет первым разделом вашего основного

2. Нормализация отношений

Нормальная форма — свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки или изменения данных. Нормальная форма определяется как совокупность требований, которым должно удовлетворять отношение.

Процесс преобразования отношений базы данных к виду, отвечающему нормальным формам, называется нормализацией. Нормализация предназначена для приведения структуры БД к виду, обеспечивающему минимальную логическую избыточность, и не имеет целью уменьшение или увеличение производительности работы или же уменьшение или увеличение физического объёма базы данных.[1] Конечной целью нормализации является уменьшение потенциальной противоречивости хранимой в базе данных информации. Как отмечает К. Дейт, общее назначение процесса нормализации заключается в следующем:

- исключение некоторых типов избыточности;
- устранение некоторых аномалий обновления;
- разработка проекта базы данных, который является достаточно «качественным» представлением реального мира, интуитивно понятен и может служить хорошей основой для последующего расширения;
- упрощение процедуры применения необходимых ограничений целостности.

Устранение избыточности производится, как правило, за счёт декомпозиции отношений таким образом, чтобы в каждом отношении хранились только первичные факты (то есть факты, не выводимые из других хранимых фактов).

Процесс проектирования БД с использование метода НФ является итерационным и заключается в последовательном переводе отношения из 1НФ в НФ более высокого порядка по определенным правилам. Каждая следующая НФ ограничивается определенным типом функциональных зависимостей и устранением соответствующих аномалий при выполнении операций над отношениями БД, а также сохранении свойств предшествующих НФ.

Используемые термины

Атрибут— свойство некоторой сущности. Часто называется полем таблицы.

Домен атрибута — множество допустимых значений, которые может принимать атрибут.

Кортеж — конечное множество взаимосвязанных допустимых значений атрибутов, которые вместе описывают некоторую сущность (строка таблицы).

Отношение — конечное множество кортежей (таблица).

Схема отношения — конечное множество атрибутов, определяющих некоторую сущность. Иными словами, это структура таблицы, состоящей из конкретного набора полей.

Проекция — отношение, полученное из заданного путём удаления и (или) перестановки некоторых атрибутов.

Функциональная зависимость между атрибутами (множествами атрибутов) X и Y означает, что для любого допустимого набора кортежей в данном отношении: если два кортежа совпадают по значению X, то они совпадают по значению Y. Например, если значение атрибута «Название компании» — CanonicalLtd, то значением атрибута «Штаб-квартира» в таком кортеже всегда будет MillbankTower, London, UnitedKingdom. Обозначение: {X} -> {Y}.

Нормальная форма— требование, предъявляемое к структуре таблиц в теории реляционных баз данных для устранения из базы избыточных функциональных зависимостей между атрибутами (полями таблиц).

Метод нормальных форм (НФ) состоит в сборе информации о объектах решения задачи в рамках одного отношения и последующей декомпозиции этого отношения на несколько взаимосвязанных отношений на основе процедур нормализации отношений.

Цель нормализации: исключить избыточное дублирование данных, которое является причиной аномалий, возникших при добавлении, редактировании и удалении кортежей(строк таблицы).

Аномалией называется такая ситуация в таблице БД, которая приводит к противоречию в БД либо существенно усложняет обработку БД. Причиной является излишнее дублирование данных в таблице, которое вызывается наличием функциональных зависимостей от не ключевых атрибутов.

Аномалии-модификации проявляются в том, что изменение одних данных может повлечь просмотр всей таблицы и соответствующее изменение некоторых записей таблицы.

Аномалии-удаления— при удалении какого-либо кортежа из таблицы может пропасть информация, которая не связана на прямую с удаляемой записью. Аномалии-добавлениявозникают, когда информацию в таблицу нельзя поместить, пока она не полная, либо вставка записи требует дополнительного просмотра таблицы.

3. Графическое представление.

Схема базы данных (от англ. *Databaseschema*) — её структура, описанная на формальном языке, поддерживаемом СУБД. В реляционных базах данных схема определяет таблицы, поля в каждой таблице (обычно с указанием их названия, типа, обязательности), и ограничения целостности(первичный, потенциальные и внешние ключии другие ограничения).

Схемы в общем случае хранятся в словаре данных. Хотя схема определена на языке базы данных в виде текста, термин часто используется для обозначения графического представления структуры базы данных.

Основными объектами графического представления схемы являются таблицы и связи, определяемые внешними ключами.

1.8 Лекция № 10 (2 часа).

Тема: «Проектирование логической модели данных»

1.8.1 Вопросы лекции:

1. Отображение на реляционную модель
2. Отображение на иерархическую модель
3. Отображение на сетевую модель

1.8.2 Краткое содержание вопросов:

1. Отображение на реляционную модель

ORM (англ. Object-RelationalMapping, *объектно-реляционное отображение*) — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Существуют как проприетарные, так и свободные реализации этой технологии.

Решение проблемы хранения данных существует — это реляционные системы управления базами данных. Использование реляционной базы данных для хранения объектно-ориентированных данных приводит к семантическому разрыву, заставляя программистов писать программное обеспечение, которое должно уметь как обрабатывать данные в объектно-ориентированном виде, так и уметь сохранить эти данные в реляционной форме. Эта постоянная необходимость в преобразовании между двумя разными формами данных не только сильно снижает производительность, но и создает трудности для программистов, так как обе формы данных накладывают ограничения друг на друга.

Реляционные базы данных используют набор таблиц, представляющих простые данные. Дополнительная или связанная информация хранится в других таблицах. Часто

для хранения одного объекта в реляционной базе данных используется несколько таблиц; это, в свою очередь, требует применения операции JOIN для получения всей информации, относящейся к объекту, для её обработки. Например, для хранения данных записной книжки, скорее всего, будут использоваться как минимум две таблицы: люди и адреса, и, возможно, даже таблица с телефонными номерами.

Так как системы управления реляционными базами данных обычно не реализуют реляционного представления физического уровня связей, выполнение нескольких последовательных запросов (относящихся к одной «объектно-ориентированной» структуре данных) может быть слишком затратно. В частности, один запрос вида «найти такого-то пользователя и все его телефоны и все его адреса и вернуть их в таком формате», скорее всего, будет выполнен быстрее серии запросов вида «Найти пользователя. Найти его адреса. Найти его телефоны». Это происходит благодаря работе оптимизатора и затратам на синтаксический анализ запроса.

Некоторые реализации ORM автоматически синхронизируют загруженные в память объекты с базой данных. Для того чтобы это было возможным, после создания объект-в-SQL-преобразующего SQL-запроса (класса, реализующего связь с DB) полученные данные копируются в поля объекта, как во всех других реализациях ORM. После этого объект должен следить за изменениями этих значений и записывать их в базу данных.

Системы управления реляционными базами данных показывают хорошую производительность на глобальных запросах, которые затрагивают большой участок базы данных, но объектно-ориентированный доступ более эффективен при работе с малыми объёмами данных, так как это позволяет сократить семантический провал между объектной и реляционной формами данных.

При одновременном существовании этих двух разных миров увеличивается сложность объектного кода для работы с реляционными базами данных, и он становится более подвержен ошибкам. Разработчики программного обеспечения, основывающегося на базах данных, искали более легкий способ достижения постоянства их объектов.

2. Отображение на иерархическую модель

Иерархическая модель организует данные в виде древовидной структуры.

К основным понятиям иерархической структуры относятся: уровни элемент (узел), связь. Дерево представляет собой иерархию элементов называемых узлами. Узел - это совокупность атрибутов данных, описывающих некоторый объект. На самом верхнем уровне иерархии имеется один и только один узел - корень. Каждый узел, кроме корня, связан с одним узлом на более (высоком уровне, называемым исходным для данного узла.

Ни один элементов не имеет более одного исходного узла. Каждый элемент может быть связан с одним или несколькими элементами на более низком уровне. Они называются порожденными (рис 3.15).

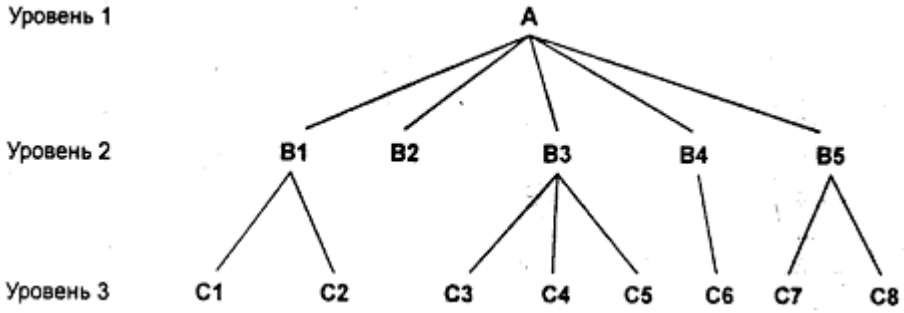


Рис. 3.15 Иерархическая структура БД

К каждой записи базы данных существует только один (иерархический) путь от корневой записи. Например, как видно из рис. 3.15, для записи С4 путь проходит через записи А и В3.

Иерархическая модель данных имеет много общих черт с сетевой моделью данных, хронологически она появилась даже раньше, чем сетевая. Допустимыми информационными конструкциями в иерархической модели данных являются отношение, веерное отношение и иерархическая база данных. В отличие от ранее рассмотренных моделей данных, где предполагалось, что информационным отображением одной предметной области является одна база данных, в иерархической модели данных допускается отображение одной предметной области в несколько иерархических баз данных.

Понятия отношения и веерного отношения в иерархической модели данных не изменяются.

Иерархической базой данных называется множество отношений и веерных отношений, для которых соблюдаются два ограничения:

1. Существует единственное отношение, называемое корневым, которое не является зависимым ни в одном веерном отношении.

2. Все остальные отношения (за исключением корневого) являются зависимыми отношениями только в одном веерном отношении.

Рис. 3.16 Иерархическая база данных для вуза: а - исходная структура; б - с добавленными сведениями о группах дипломников.

Схема иерархической БД по составу компонентов идентична сетевой базе данных. Названные выше ограничения поддерживаются иерархическими СУБД.

На рис. 3.16 изображена структура иерархической базы данных о студентах и преподавателях вуза, удовлетворяющая всем ограничениям, указанным в определении.

В графических иллюстрациях структуры приводятся ключи соответствующих отношений.

Если понадобится в рамках данной иерархической структуры указать для групп, выполняющих дипломное проектирование, связь с соответствующей выпускающей кафедрой, то установить веерное отношение Р:(Кафедра, Группа) невозможно, так как Группа не может быть зависимым отношением дважды (она уже является зависимой для отношения Факультет). Зафиксировать связь студенческих групп с выпускающей кафедрой можно путем выделения соответствующих групп в отдельное отношение с ключом В - группа, что приводит к появлению избыточной информации.

Ограничение, которое поддерживается в иерархической модели данных, состоит в невозможности нарушения требований, фигурирующих в определении иерархической базы данных. Это ограничение обеспечивается специальной укладкой значений отношений в памяти ЭВМ. Ниже мы рассмотрим одну из простейших реализации укладки иерархической БД.

На рис. 3.17 приводится связь значений отношений из иерархической базы данных, структура которой показана на рис. 3.16, а. Каждое значение представляется соответствующей величиной первичного ключа.

Далее эта информация организуется в линейную последовательность значений, как это показано на рис. 3.17, б.

Необходимо отметить, что существуют различные возможности прохождения иерархически организованных значений в линейной последовательности. Принцип, применяемый для иерархических баз данных, называется концевым прохождением.

3.Отображение на сетевую модель

На разработку этого стандарта большое влияние оказал американский ученый Ч.Бахман. Основные принципы *сетевой модели данных*были разработаны в середине 60-х годов, эталонный вариант *сетевой модели данных* описан в отчетах рабочей группы по языкам баз данных (*COnference* on *DAtaSYstem* Languages) CODASYL (1971 г.).

Сетевая модель данных определяется в тех же терминах, что и *иерархическая*. Она состоит из *множества* записей, которые могут быть владельцами или членами групповых отношений. *Связь* между записью-владельцем и записью-членом также имеет вид 1:N.

Основное различие этих моделей состоит в том, что в *сетевой модели запись* может быть членом более чем одного группового отношения. Согласно этой модели каждое групповое *отношение* именуется и проводится различие между его типом и экземпляром. Тип группового отношения задается его именем и определяет свойства общие для всех экземпляров данного типа. Экземпляр группового отношения представляется записью-владельцем и множеством (возможно пустым) подчиненных записей. При этом имеется следующее ограничение: *экземпляр записи* не может быть членом двух экземпляров групповых отношений одного типа (т.е. сотрудник из примера в п..1, например, не может работать в двух отделах).

Иерархическая структура рис. 4.2 преобразовывается в сетевую модель, следующим образом (см. рис. 4.3):

- деревья (a) и (b), показанные на рис. 4.2, заменяются одной сетевой структурой, в которой запись СОТРУДНИК входит в два групповых отношения;
- для отображения типа M:N вводится запись СОТРУДНИК_КОНТРАКТ, которая не имеет полей и служит только для связи записей КОНТРАКТ и СОТРУДНИК, (см. рис. 4.3). Отметим, что в этой записи может храниться и полезная информация, например, доля данного сотрудника в общем вознаграждении по данному контракту.

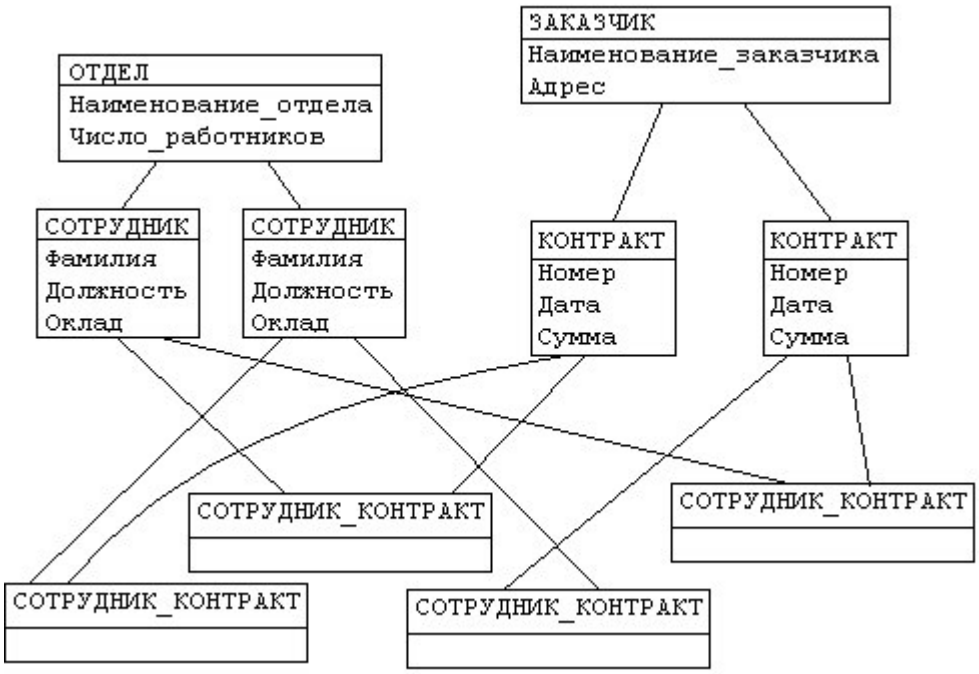


Рис. 4.3. Сетевая модель базы данных

Каждый экземпляр группового отношения характеризуется следующими признаками:

Способ упорядочения подчиненных записей:

- произвольный,
- хронологический /очередь/,
- обратный хронологический /стек/,
- сортированный.

Если *запись* объявлена подчиненной в нескольких групповых отношениях, то в каждом из них может быть назначен свой способ упорядочивания.

Режим включения подчиненных записей:

- автоматический - невозможно занести в БД запись без того, чтобы она была сразу же закреплена за неким владельцем;
- ручной - позволяет запомнить в БД подчиненную запись и не включать ее немедленно в экземпляр группового отношения. Эта операция позже инициируется пользователем.

Режим исключения.

Принято выделять три класса членства подчиненных записей в групповых отношениях:

1. **Фиксированное.** Подчиненная запись *жестко связана* с записью владельцем и ее можно исключить из группового отношения только удалив. При *удалении*

записи-владельца все подчиненные записи автоматически тоже удаляются. В рассмотренном выше примере фиксированное членство предполагает групповое отношение "ЗАКЛЮЧАЕТ" между записями "КОНТРАКТ" и "ЗАКАЗЧИК", поскольку контракт не может существовать без заказчика.

2. **Обязательное.** Допускается переключение подчиненной записи на другого владельца, но невозможно ее существование без владельца. Для удаления записи-владельца необходимо, чтобы она не имела подчиненных записей с обязательным членством. Таким отношением связаны записи "СОТРУДНИК" и "ОТДЕЛ". Если отдел расформировывается, все его сотрудники должны быть либо переведены в другие отделы, либо уволены.

3. **Необязательное.** Можно исключить запись из группового отношения, но сохранить ее в базе данных не прикрепляя к другому владельцу. При *удалении записи*-владельца ее подчиненные записи - необязательные члены сохраняются в базе, не участвуя более в групповом отношении такого типа. Примером такого группового отношения может служить "ВЫПОЛНЯЕТ" между "СОТРУДНИКИ" и "КОНТРАКТ", поскольку в организации могут существовать работники, чья деятельность не связана с выполнением каких-либо договорных обязательств перед заказчиками.

1. 9 Лекция № 11 (2 часа).

Тема: «Физическая модель данных»

1.9.1 Вопросы лекции:

1. Виды моделей данных.

2. Физическая модель данных

1.9.2 Краткое содержание вопросов:

1. Виды моделей данных

Выделяют следующие модели:

- инфологическая модель;

- даталогическая модель;

- физическая модель.

Инфологическая модель данных (ИМД) создается по результатам проведения исследований предметной области. Она представляет собой концептуальную модель базы данных, не привязанную к какой-либо конкретной СУБД.

Даталогическая модель данных (ДМД) создается на базе ИМД и представляет собой описание предметной области с учетом используемой для создаваемой БД модели данных.

Физическая модель данных (ФМД) – это модель данных, описанная с помощью средств конкретной СУБД. ФМД строится на базе даталогической путем добавления особенностей конкретной СУБД.

2. Физическая модель данных

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

Однако механизмы буферизации и управления файловыми структурами не приспособлены для решения задач собственно СУБД, эти механизмы разрабатывались просто для традиционной обработки файлов, и с ростом объемов хранимых данных они стали неэффективными для использования СУБД. Тогда постепенно произошел переход от базовых файловых структур к непосредственному управлению размещением данных на внешних носителях самой СУБД. И пространство внешней памяти уже выходило из-под владения СУФ и управлялось непосредственно СУБД. При этом механизмы, применяемые в файловых системах, перешли во многом и в новые системы организации данных во внешней памяти, называемые чаще страничными системами хранения информации. Поэтому наш раздел, посвященный физическим моделям данных, мы начнем с обзора файлов и файловых структур, используемых для организации физических моделей, применяемых в базах данных, а в конце ознакомимся с механизмами организации данных во внешней памяти, использующими страничный принцип организации.

1.10 Лекция № 12,13 (4 часа).

Тема: «Администрирование базы данных»

1.10.1 Вопросы лекции:

1. Функция администрирования базы данных
2. Жизненный цикл системы с базой данных

1.10.2 Краткое содержание вопросов:

Администратор базы данных – человек (или группа лиц), имеющий полное представление об одной или нескольких базах данных и контролирующий их проектирование и использование. Отвечает за состояние базы данных в организации (учреждении) на протяжении ее жизненного цикла. Функциями АБД являются [12, 17]:

- решение вопросов организации данных об объектах предметной области и установлении связей между этими данными с целью объединения информации о различных объектах; согласование представлений пользователей;

- координация всех действий по проектированию, реализации и ведению БД; учет перспективных и текущих требований пользователей;

- решение вопросов, связанных с расширением БД в связи с изменением границ предметной области;

- разработка и реализация мер по обеспечению защиты данных от некомпетентного использования, от сбоев технических средств, по обеспечению секретности определенной части данных и разграничению доступа к данным;

- выполнение работ по ведению словаря данных; контроль не избыточности и непротиворечивости данных, их достоверности;

- обеспечение заданной производительности БД, чтобы обработка запросов выполнялась за приемлемое время;

- изменение при необходимости методов хранения данных, путей доступа к ним, связей между данными, форматов данных; определение степени влияния изменений на всю БД;

- координация вопросов технического обеспечения системы аппаратными средствами исходя из требований, предъявляемых БД к оборудованию;

- координация работы системных программистов, разрабатывающих дополнительное программное обеспечение для улучшения эксплуатационных характеристик системы;

- координация работы прикладных программистов, разрабатывающих новые пакеты программ и выполнение их проверки и включение в состав программного обеспечения системы.

При таком делении функций администрирования к *задачам администрирования данных* относятся следующие.

- Разработка стратегии построения информационной системы.

- Предварительная оценка осуществимости и планирование процесса создания базы данных.

- Разработка корпоративной модели данных.

- Определение требований организации к используемым данным.

- Определение стандартов сбора данных и выбор формата их представления.

- Оценка объемов данных и вероятности их роста.

- Определение способов и интенсивности использования данных.

- Определение правил доступа к данным и мер безопасности соответствующих правовым нормам и внутренним требованиям организации.

- Обеспечение полноты всей требуемой документации.

- Поддержка словаря данных.

- Взаимодействие с конечными пользователями для определения новых требований и разрешения проблем, связанных с доступом к данным и недостаточной производительностью их обработки.

2. Жизненный цикл системы с базой данных.

Одним из базовых понятий проектирования БД является понятие их жизненного цикла. Жизненный цикл БД включает жизненный цикл информации, жизненный цикл информационных продуктов (программного обеспечения) и услуг, а также жизненный цикл ИС.

Жизненный цикл информации состоит из её появления, существования и исчезновения.

Жизненный цикл информационных систем (как и использования компьютерных программ) – это различные их состояния, начиная с момента возникновения необходимости в данной ИС и заканчивая моментом её полного выхода из употребления у всех пользователей.

Жизненный цикл существования программного продукта (БД) является непрерывным процессом. Разработка БД охватывает все работы по их созданию. Концепция жизненного цикла продукта, услуги или системы подразумевает, что они ограниченны, по крайней мере, во времени. Особенностью разработки программного продукта является принятие решений на начальных этапах с их реализацией на последующих этапах. При этом важно оценить необходимые для разработки программного продукта материальные, трудовые и финансовые ресурсы, ориентировочные длительности основных этапов его жизненного цикла.

Традиционно выделяются следующие основные этапы жизненного цикла программного обеспечения:

- анализ требований,

- проектирование,

- кодирование (программирование),

- тестирование и отладка,

- эксплуатация и сопровождение.

Разработка БД охватывает все работы по их созданию.

К основным этапам жизненного цикла БД относятся: отладка, тестирование, анализ требований, эксплуатация, кодирование, проектирование, сопровождение. При этом программный продукт может создаваться даже если на него нет заказа.

Под моделью ЖЦ понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении всего ЖЦ.

К основным (базовым) моделям жизненного цикла относят модели: каскад («водопад»), поэтапная и спираль («водоворот»). Важным аспектом проектирования БД является возможность осуществлять тестирование системы непосредственно в процессе ее разработки. Не все модели жизненного цикла БД позволяют это делать.

В каскадной модели переход на следующий этап означает полное завершение работ на предыдущем этапе.

В спиральной модели переход на следующий этап означает создание прототипа разрабатываемого продукта, создание версии разрабатываемого продукта, уточнение характеристик предыдущего этапа.

Наиболее короткий период разработки программного продукта имеет спиральная модель, а наиболее длительный период разработки БД имеет каскадная модель жизненного цикла.

Каскадная модель применяется при разработке небольших проектов и при решении отдельных задач.

Эксплуатация БД включает работы по ее внедрению в том числе конфигурирование БД и рабочих мест пользователей, обеспечение эксплуатационной документацией, проведение обучения персонала и т.д., и непосредственно эксплуатацию, в том числе локализацию проблем и устранение причин их возникновения, модификацию в рамках установленного регламента, подготовку предложений по совершенствованию, развитию и модернизации БД.

Разработчики стремятся сделать максимально возможным период жизненного цикла информационных продуктов и услуг. Для большинства современных компьютерных программ длительность жизненного цикла равна двум–трём годам, хотя встречаются программы, существующие десять и более лет. Увеличить длительность жизненного цикла БД можно, если регулярно осуществлять маркетинговые мероприятия по её поддержке, изменения в БД и программных услугах; если уменьшить цену на БД, провести модификацию БД и др.

Увеличить длительность жизненного цикла БД можно, если регулярно осуществлять маркетинговые мероприятия по её поддержке, изменения в БД и программных услугах; если уменьшить цену на БД, провести модификацию БД и др.

1.11 Лекция № 14 (2 часа).

Тема: «Словарь данных»

1.10.1 Вопросы лекции:

1. Этапы проектирования БД
2. Словарь данных

1.10.2 Краткое содержание вопросов:

1. Этапы проектирования БД

Основные этапы проектирования:

Первый этап. Планирование разработки базы данных.

Второй этап. Определение требований к системе.

Третий этап. Проектирование концептуальной модели БД.

Четвертый этап. Построение логической модели.

Пятый этап. Построение физической модели.

Шестой этап. Оценка физической модели.

Седьмой этап. Реализация БД.

Восьмой этап. Тестирование и оптимизация.

Этап девятый, заключительный. Сопровождение и эксплуатация.

2. Словарь данных

На первом этапе проектирования базы данных необходимо собрать сведения о предметной области, в том числе о назначении, способах использования и о структуре данных, а по мере развития проекта осуществлять централизованное накопление информации о концептуальной, логической, внутренней и внешних моделях данных. Словарь данных является как раз тем средством, которое позволяет при проектировании, эксплуатации и развитии базы данных поддерживать и контролировать информацию о данных.

При сборе информации о данных следует установить правила присвоения имен элементам, добиться однозначного толкования различными подразделениями назначения источников и соглашений по присвоению имен, сформулировать приемлемые для всех пользователей описания элементов данных и выявить синонимы. Этот процесс включает несколько итераций и связан с необходимостью разрешения конфликтных ситуаций. Отдельные подразделения подчас переоценивают свою роль на предприятии, что

приводит при разработке информационной системы к конфликтам. Разработчику в таких случаях придется выступать в роли арбитра. Если вам не по душе слушать крики: «Судью на мыло!», то для обеспечения эффективного сбора и накопления информации о данных желательно, чтобы все, кто имеет отношение к базе данных, пользовались автоматизированным словарем данных.

Словарь данных содержит информацию об источниках, форматах и взаимосвязях между данными, их описания, сведения о характере использования и распределении ответственности. Словарь данных можно рассматривать как "Метабазу данных", в которой хранится информация о базе данных.

Одно из главных назначений словаря данных состоит в документировании данных. Так как база данных обслуживает множество пользователей, крайне необходимо, чтобы они правильно понимали, что представляют собой данные.

Накопление информации в словаре данных целесообразно начинать уже с самой ранней стадии проектирования. В процессе работы разработчики выясняют у пользователей, какой должна быть система, какие данные будут входными, какого рода информацию они хотят получить из системы, вводя имена элемента данных, например «номер счета», «остаток» или «процент» в банковской системе. При этом обе стороны должны трактовать используемые термины однозначно, иначе может случиться так, что разработанная система не будет удовлетворять требованиям пользователей. Поэтому второе важное назначение словаря данных - обеспечить эффективное взаимодействие между различными категориями разработчиков и пользователей.

Таким образом, два важнейших назначения словаря данных состоят в централизованном ведении и управлении данными как ресурсом на всех этапах проектирования, реализации и эксплуатации системы, а также в обеспечении эффективного взаимодействия между всеми участниками проекта.

В случае распределенной базы данных вся она или ее отдельные части могут размещаться на удаленных друг от друга вычислительных машинах, соединённых линиями связи. Одни рабочие станции в сети могут обращаться только к локальной базе данных, а другие - как к локальной, так и к внешним. В этом случае в словарь данных может быть введена информация обо всех местах физического хранения данных, а также ограничения секретности, безопасности и доступа. С помощью этой информации словарь данных может «решить», каким образом удовлетворить запрос пользователя: обратиться к локальной базе данных или, если пользователь обладает соответствующими полномочиями, передать запрос на внешнюю ПЭВМ.

Неавтоматизированный словарь данных не может обеспечить получение по-разному отсортированных списков элементов данных, которыми пользуются разработчики. Один и тот же элемент может неодинаково использоваться в различных приложениях. На ранней стадии проектирования выявляются далеко не все связи между данными. Впоследствии обнаруживается, что данные применяются в разнообразных приложениях. Они могут встречаться, например, во входных и выходных форматах, связанных между собой, и всякий раз рассматриваются в различных контекстах. Чтобы учесть все возможные ограничения, необходимо приложить значительные усилия. Процесс проектирования же становится в таком случае трудно управляемым. Гораздо проще организовать и управлять разработкой с помощью автоматизированного словаря данных.Для успешного применения словаря данных при разработке системы следует централизовать накопление информации в этом едином источнике, из которого программисты смогут копировать описания структур данных и включать их в свои программы на всех этапах проектирования. В случае применения «ручного» или не интегрированного словаря в нем время от времени может происходить нарушение непротиворечивости информации по отношению к фактическому состоянию системы.

В идеальном случае интерфейс между СУБД и словарем данных должен обеспечивать доступ системы словаря к справочникам СУБД, в которых хранится информация о ее текущем состоянии. Модификация типов данных может производиться только после того, как это будет зарегистрировано в словаре данных. Обновление самих данных допускается лишь после проверки их корректности средствами СУБД. Таким образом, словарь данных, СУБД и база данных образуют замкнутый контур.

В идеале словарь данных должен быть неотъемлемой составной частью всей системы обработки данных. За ввод данных в словарь ответственность несет администратор БД. Поскольку словарь данных является центральным звеном системы, необходимо постоянно поддерживать его копию, которая может использоваться для восстановления словаря после возникновения отказа всей системы или в случае непреднамеренного разрушения его рабочей версии. За сохранность словаря данных как жизненно важной части системы с базой

Данных полностью отвечает администрация базы данных. Если словарь данных применяется для разграничения доступа к базе данных, то доступ к нему надо также разграничить. Следует строго ограничить круг лиц, которым разрешено модифицировать словарь данных. В отношении хранимой в словаре информации должен быть реализован режим секретности.

Словарь данных призван помогать пользователю в выполнении следующих функций:

– совместное использование данных с другими пользователями;

– осуществление простого и эффективного управления элементами данных при вводе в систему новых элементов или изменении описания существующих;

– уменьшение избыточности и противоречивости данных;

– определение степени влияния изменений в элементах данных на всю базу данных;

– централизация управления элементами данных с целью упрощения проектирования базы данных и ее расширения.

1.12 Лекция № 15 (2 часа).

Тема: «Общая характеристика баз знаний и экспертных систем»

1.12.1 Вопросы лекции:

1. Базы знаний

2. Экспертные системы

1.12.2 Краткое содержание вопросов:

1. Базы знаний.

База знаний — это особого рода база данных, разработанная для оперирования знаниями (метаданными). База знаний содержит структурированную информацию, покрывающую некоторую область знаний, для использования кибернетическим устройством (или человеком) с конкретной целью. Современные базы знаний работают совместно с системами поиска информации, имеют классификационную структуру и формат представления знаний.

Полноценные базы знаний содержат в себе не только фактическую информацию, но и правила вывода, допускающие автоматические умозаключения о вновь вводимых фактах и, как следствие, осмысленную обработку информации. Область наук об искусственном интеллекте, изучающая базы знаний и методы работы со знаниями, называется инженерией знаний.

Иерархический способ представления в базе знаний набора понятий и их отношений называется онтологией. Онтологию некоторой области знаний вместе со сведениями о свойствах конкретных объектов также можно назвать базой знаний.

2. Экспертные системы

Экспертная система − это компьютерная программа, которая моделирует рассуждения человека-эксперта в некоторой определенной области и использует для этого базу знаний, содержащую факты и правила об этой области, и некоторую процедуру

логического вывода. Экспертные системы предназначены для моделирования и имитации логики опытных специалистов при принятии решения по какому-либо узкому вопросу в определенной предметной области.

ЭС можно считать одним из применений искусственного интеллекта, хотя исследователи искусственного интеллекта не ставили целью построение экспертных систем. ЭС помогают специалистам, когда их собственных знаний, опыта и интуиции недостаточно для самостоятельного решения возникающих проблем. Такие системы представляют собой машинные программы, решающие задачи примерно так же, как решает их эксперт в реальной обстановке. Это позволяет накапливать, систематизировать и использовать знания и профессиональный опыт тех экспертов, которые выполняют конкретные задачи наилучшим образом и в первую очередь в тех областях, где задачи и их решения слабо формализованы или совсем не формализованы.

Типичная экспертная система состоит из следующих основных компонентов: инженер знаний, подсистема приобретения и накопления знаний, база знаний, рабочая область (база данных − БД), создатель заключения (решатель, интерпретатор), подсистема пояснений, интерфейс пользователя, пользователь.

Чтобы спроектировать экспертную систему, специалист, называемой инженером знания (специально подготовленный системный аналитик), тесно работает с одним или большим количеством экспертов в изучаемой области. Инженеры знания пытаются узнавать все относительно способа, которым эксперт принимает решения.

Подсистема приобретения и накопления знаний помогает инженеру знания в регистрации правил заключения и параметров в базе знаний. Знание (правила заключения и параметры), полученное инженером знания, затем с помощью подсистемы приобретения и накопления знаний регистрируется в компьютерной системе в специализированном формате в блоке, названном базой знаний.

База знаний в экспертной системе предназначена для хранения: долгосрочных данных, описывающих рассматриваемую область; правил, описывающих целесообразные преобразования данных этой области; заключений.

Рабочая область (база данных) предназначена для хранения исходных и промежуточных данных решаемой в текущий момент задачи.

Создатель заключения (решатель, интерпретатор), используя исходные данные из рабочей памяти и знания из базы знаний, формирует такую последовательность правил, которая будучи применима к исходным данным, приводит к решению задачи. Один и тот же создатель заключения может использоваться в различных экспертных системах с различной базой знаний.

Подсистема пояснений объясняет, как система получила решения задачи (или почему она не получила решения) и какие знания она при этом использовала, что облегчает эксперту-пользователю тестирование системы и повышает доверие пользователя к полученному результату.

Интерфейс пользователя (диалоговый компонент) ориентирован на организацию дружелюбного общения со всеми категориями пользователей, как в ходе решения задач, так и во время приобретения знаний, объяснения результатов работы.

В разработке экспертной системы участвуют представители следующих специальностей:

1. эксперт в той проблемной области, задачи которой будет решать ЭС;
2. инженер по знаниям, т.е. специалист по разработке ЭС;
3. программист, т.е. специалист по разработке инструментальных средств.

Необходимо отметить, что отсутствие среди участников разработки инженера по знаниям (т.е. замена его программистом) либо приводит к неудаче процесс создания ЭС, либо значительно удлиняет его.

Эксперт определяет знания (данные и правила), характеризирующие проблемную область, обеспечивает полноту и правильность введенных в ЭС знаний.

Инженер по знаниям помогает эксперту выявить и структурировать знания, необходимые для работы ЭС, осуществляет выбор того инструментального средства, которое наиболее подходит для данной проблемной области, и определяет способ предоставления знаний в этом ЭС. Он выделяет и программирует (традиционными средствами) стандартные функции (типичные для данной проблемной области), которые будут использоваться в правилах, вводимых экспертом.

Программист разрабатывает инструментальное средство, содержащее в пределе все основные компоненты ЭС, сопрягает инструментальное средство с той средой, в которой оно будет использовано.

1.13 Лекция № 16 (2 часа).

Тема: «СУБД ACCESS»

1.13.1 Вопросы лекции:

1. Назначение, общая характеристика и структура СУБД ACCESS
2. Работа с Access.

1.13.2 Краткое содержание вопросов:

1. Назначение, общая характеристика и структура СУБД ACCESS

Access - это система управления базами данных(СУБД). Подсистемой управления понимается комплекс программ, который позволяет не только хранить большие массивы данных в определенном формате, но и обрабатывать их, представляя в удобном для пользователей виде. Access дает возможность также автоматизировать часто выполняемые операции (например, расчет заработной платы, учет материальных ценностей и т.п.). С помощью Access можно не только разрабатывать удобные формы ввода и просмотра данных, но и составлять сложные отчеты.

Access является приложением Windows, а поскольку и Windows и Access разработаны одной фирмой (Microsoft), они очень хорошо взаимодействуют друг с другом. СУБД Access работает под управлением Windows; таким образом, все преимущества Windows доступны в Access, например, вы можете вырезать, копировать и вставлять данные из любого приложения Windows в приложение Access и наоборот.

Access - это реляционная СУБД. Это означает, что с ее помощью можно работать одновременно с несколькими таблицами базы данных. Применение реляционной СУБД помогает упростить структуру данных и таким образом облегчить выполнение работы. Таблицу Access можно связать с данными, хранящимися на другом компьютере или на сервере, а также использовать таблицу, созданную в СУБД Paradox или Dbase. Данные Access очень просто комбинировать с данными Excel.

В СУБД Access предусмотрено много дополнительных сервисных возможностей. Мастера помогут вам создать таблицы, формы или отчеты из имеющихся заготовок, сделав за вас основную черновую работу. Выражения используются в Access, например, для проверки допустимости введенного значения. Макросы позволяют автоматизировать многие процессы без программирования, тогда как встроенный в Access язык VBA(VisualBasicforApplications) - специально разработанный компанией Microsoft диалект языка Basic для использования в приложениях MicrosoftOffice - дает возможность опытному пользователю программировать сложные процедуры обработки данных. Просматривая свою форму или отчет, вы сможете представить, как они будут выглядеть в распечатанном виде. И наконец, используя такие возможности языка программирования С,как функции и обращения к Windows API (ApplicationProgrammingInterface - интерфейс прикладных программ Windows), можно написать подпрограмму для взаимодействия Access с другими приложениями - источниками данных.

В MicrosoftAccess добавлено множество новых средств, разработанных для облегчения работы в Интернет и создания приложений для Web. Для доступа к сети Интернет и использования преимуществ новых средств необходимы средства просмотра Web, например MicrosoftInternetExplorer, а также модем. Пользователь имеет возможность

непосредственно подключаться к узлам MicrosoftWeb из программ Office (в том числе и из Access) с помощью команды Microsoft на Webиз пункта меню. При этом можно, например, получить доступ к техническим ресурсам и загрузить общедоступные программы, не прерывая работу с Access.

Система Access содержит набор инструментов для управления базами данных, включающий конструкторы таблиц, форм, запросов и отчетов. Кроме того, Access можно рассматривать и как среду для разработки приложений. Используя макросы для автоматизации задач, вы можете создавать такие же мощные, ориентированные на пользователя приложения, как и приложения, созданные с помощью "полноценных" языков программирования, дополнять их кнопками, меню и диалоговыми окнами. Программируя на VBA, можно создавать программы, по мощности не уступающие самой Access. Более того, многие средства Access, например мастера и конструкторы, написаны на VBA. Мощность и гибкость системы Access делают ее сегодня одной из лучших программ для управления базами данных.

2. Работа с Access.

Мастер(Wizard) - специальная программа, помогающая в решении какой-то задачи или создании объекта определенного типа. Эта программа поможет вам за несколько минут выполнить рутинную работу, на которую без применения этой программы может уйти несколько часов. Программа-мастер задает вопросы о содержании, стиле и формате объекта, а затем создает этот объект без какого-либо вмешательства с вашей стороны. В Access имеется около сотни*мастеров,*предназначенных для проектирования баз данных, приложений, таблиц, форм, отчетов, графиков, почтовых наклеек, элементов управления и свойств.

Те, кто знакомы с Excel, заметят, что Access во многом похож на Excel. Прежде всего, обе программы являются продуктами для Windows, следовательно, можно использовать свой опыт применения специфичных для Windows соглашений. Данные таблицы или запроса Access отображаются в виде электронной таблицы, которую принято называть *таблицей данных.*Вы обнаружите, что размер строк и столбцов таблицы данных можно изменять так же, как в рабочих таблицах Excel. Фактически режим ввода данных Access ничем не отличается от аналогичного режима Excel. Основное различие между таблицей базы данных (БД) и электронной таблицей - в системе адресации; в электронной таблице адресуется каждая ячейка, а в таблице БД - только поля текущей записи. В электронной таблице каждая ячейка обрабатывается индивидуально, а в таблице БД обработка идет по записям, причем записи обрабатываются однотипным образом. Эти

упрощения для БД позволяют повысить скорость обработки и количество обслуживаемой информации.

Справочная система фирмы Microsoft является, наверное, лучшей среди аналогичных программ как для новичков, так и для опытных пользователей. Access дает возможность использовать контекстно-зависимую справку, для получения которой достаточно нажать правую клавишу мыши. Какие бы вы ни испытывали затруднения при работе с системой, вам поможет появляющаяся на экране справка по интересующей вас теме. Помимо этого справочная система Access имеет удобные и простые в использовании содержание, предметный указатель, систему поиска, журнал хронологии и закладки. В локализованной версии Access 97(как и во всем MicrosoftOffice 97) компания Microsoft добавила новое средство - *Помощник.*Помощник отвечает на вопросы, выдает советы и справки об особенностях используемой программы.

Все составляющие базы данных, такие, как таблицы, отчеты, запросы, формы и объекты, в Access 97 хранятся в едином дисковом файле. Основным структурным компонентом базы данных является таблица. В таблицах хранятся вводимые нами данные. Внешне каждая таблица Access 97 похожа на таблицы, с которыми мы привыкли работать на бумаге, - она состоит из столбцов, называемых *полями,*и строк, называемых*записями.* Каждая запись таблицы содержит всю необходимую информацию об отдельном элементе базы данных. Например, запись о преподавателе может содержать фамилию, имя, отчество, дату рождения, должность и т.п.

При разработке структуры таблицы, прежде всего, необходимо определить названия полей, из которых она должна состоять, типы полей и их размеры. Каждому полю таблицы присваивается уникальное имя, которое не может содержать более 64 символов. Имя желательно делать таким, чтобы функция поля узнавалась по его имени. Далее надо решить, данные какого типа будут содержаться в каждом поле. В Access можно выбирать любые из основных типов данных. Один из этих типов данных должен быть присвоен каждому полю. Значение типа поля может быть задано только в режиме конструктора. Ниже представлены типы данных Access и их описание.

Тип данных	**Описание**
Текстовый (Значение по умолчанию)	Текст или числа, не требующие проведения расчетов, например номера телефонов (до 255 знаков)
Числовой	Числовые данные различных форматов, используемые для проведения расчетов

Дата/время	Для хранения информации о дате и времени с 100 по 9999 год включительно
Денежный	Денежные значения и числовые данные, используемые в математических расчетах, проводящихся с точностью до 15 знаков в целой и до 4 знаков в дробной части
Поле MEMO	Для хранения комментариев; до 65535 символов
Счетчик	Специальное числовое поле, в котором Access автоматически присваивает уникальный порядковый номер каждой записи. Значения полей типа счетчика обновлять нельзя
Логический	Может иметь только одно из двух возможных значений (TrueFalse, Да/Нет)
Поле объекта OLE	Объект (например, электронная таблица MicrosoftExcel, документ MicrosoftWord, рисунок, звукозапись или другие данные в двоичном формате), связанный или внедренный в таблицу Access
Гиперссылка	Строка, состоящая из букв и цифр и представляющая адрес гиперссылки. Адрес гиперссылки может состоять максимум из трех частей: текст, выводимый в поле или в элементе управления; путь к файлу (в формате пути UNC) или к странице (адрес URL). Чтобы вставить адрес гиперссылки в поле или в элемент управления, выполните команду Вставка, Гиперссылка
Мастер подстановок	Создает поле, в котором предлагается выбор значений из списка или из поля со списком, содержащего набор постоянных значений или значений из другой таблицы. Это в действительности не тип поля, а способ хранения поля

В Access существует четыре способа создания пустой таблицы:

– использование мастера баз данных для создания всей базы данных, содержащей все требуемые отчеты, таблицы и формы, за одну операцию. Мастер баз данных создает новую базу данных, его нельзя использовать для добавления новых таблиц, форм, отчетов в уже существующую базу данных;

– мастер таблиц позволяет выбрать поля для данной таблицы из множества определенных ранее таблиц, таких, как деловые контакты, список личного имущества или рецепты;

– ввод данных непосредственно в пустую таблицу в режиме таблицы. При сохранении новой таблицы в Access данные анализируются, и каждому полю присваивается необходимый тип данных и формат;

– определение всех параметров макета таблицы в режиме конструктора.

Независимо от метода, примененного для создания таблицы, всегда имеется возможность использовать режим конструктора для дальнейшего изменения макета таблицы, например для добавления новых полей, установки значений по умолчанию или для создания масок ввода. Однако только четвертый метод позволяет сразу задать ту структуру таблицы, которая вам нужна, и поэтому далее рассмотрим именно этот метод.

1.14 Лекция № 17 (2часа).

Тема: «Создание локального приложения в СУБД»

1.14.1 Вопросы лекции:

1. Локальная СУБД
2. Создание локального приложения в СУБД

1.14.2 Краткое содержание вопросов:

1. Локальная СУБД

Все части локальной СУБД размещаются на компьютере пользователя базы данных. Чтобы с одной и той же БД одновременно могло работать несколько пользователей, каждый пользовательский компьютер должен иметь свою копию локальной БД. Существенной проблемой СУБД такого типа является синхронизация копий данных, именно поэтому для решения задач, требующих совместной работы нескольких пользователей, локальные СУБД фактически не используются.

2. Создание локального приложения в СУБД

Построение распределенных систем имеет важное значение, особенно в условиях бессистемного оснащения предприятия компьютерной техникой, когда многие уже, как правило, работают (или им необходимо работать) с системами, состоящими из множества компьютеров разного типа (серверов, мини-компьютеров, больших машин).

Централизованное хранение данных и доступ к центральной БД в условиях географически распределенной системы приводят к необходимости установления соединений между центральным сервером, хранящим данные, и компьютерами-клиентами. Большинство компьютеров-клиентов отделены от центрального сервера медленными и недостаточно надежными линиями связи, и работа в режиме удаленного клиента становится почти невозможной. Этим можно объяснить существующую

ситуацию, когда в узлах распределенной системы функционируют группы автоматизированных рабочих мест (АРМ), абсолютно не связанные друг с другом.

Содержательная сторона задачи обычно требует обмена данными между группами АРМ, так как изменения в какие-либо данные могут вноситься в одной группе, а использоваться они могут в другой. На практике обмен информацией реализуется регламентной передачей файлов - через модемное соединение или «с курьером».

То, что данные доставляются к месту назначения не системными средствами, а путем экспорта/импорта файлов, приводит к необходимости участия человека в процессе обмена, а это влечет за собой малую оперативность поступления данных и необходимость реализации внешних механизмов контроля целостности и непротиворечивости. В результате возрастает вероятность появления ошибок. Кроме того, реализация всех алгоритмов обмена данными и контроля в этом случае возлагается на прикладных программистов, проектирующих АРМ. Объем работ по программированию и отладке подпрограмм обмена соответствует числу различных АРМ. Это также приводит к повышению вероятности сбоев в системе.

В современной технологии АРМ объединены в локальную сеть. АРМ - клиент выдает запросы на выборку и обновление данных, а СУБД исполняет их. Запросы клиента в соответствии с требованиями задачи сгруппированы в логические единицы работы (транзакции). Если все операции с базой данных, содержащиеся внутри транзакции, выполнены удачно, транзакция в целом также выполняется успешно (фиксируется). Если хотя бы одна из операций с БД внутри транзакции произведена неудачно, то все изменения в БД, происшедшие к этому моменту из транзакции, отменяются (происходит откат транзакции). Такое функционирование обеспечивает логическую целостность информации в базе данных.

При распределенной обработке изменения, проводимые приложением-клиентом, могут затрагивать более чем один сервер СУБД. Для поддержания целостности и в этом случае необходимо применение того или иного транзакционного механизма, реализуемого системными средствами, а не прикладной программой.

Но основной недостаток систем, построенных на распределенных транзакциях, - высокие требования к надежности и пропускной способности линий связи. Поэтому альтернативой распределенным транзакциям считается репликация (дублирование) данных. В таких системах одна и та же информация хранится в различных узлах. Согласование значений и распространение данных по узлам осуществляется автоматически. В зависимости от условий, специфицированных разработчиком, репликация может производиться либо сразу после наступления некоторого события

(скажем, модификации строки таблицы), либо через заранее заданные интервалы времени (каждую минуту, каждый час и т.д.), либо в определенный момент времени (например, ночью, когда загрузка и стоимость линий связи минимальная). Если узел, в который выполняется репликация, в данный момент недоступен, информация об этом сохраняется в вызывающем узле, и репликация осуществляется после восстановления связи. Более того, гарантируется сохранение заданного вызывающим узлом порядка ее выполнения.

Транзакция может вносить изменения (то есть добавлять, удалять и изменять записи) в одну или несколько таблиц базы данных. Выбранные для репликации таблицы специальным образом помечаются. Для каждой такой таблицы или группы ее строк, выбранной по заданному условию, определяется один узел (СУБД), в котором данные таблицы являются первичными. Это тот узел, в котором происходит наиболее активное обновление данных. Репликационному серверу, обслуживающему БД с первичными данными, задается описание тиражирования (replicationdefinition). В этом описании, к примеру, могут быть заданы интервалы значений первичного ключа таблицы или какое-нибудь другое условие, при выполнении которого измененные данные будут тиражироваться из этого узла к подписчикам. Если условие не задано, то описание тиражирования действует для всех записей таблицы. Возможность тиражирования группы записей таблицы означает, в частности, что часть записей может быть первичными данными в каком- то одном узле, а еще часть - в других узлах. В одном или нескольких узлах (СУБД), которым нужны измененные данные, в обслуживающем его репликационном сервере создается подписка (subscription) на соответствующее описание тиражирования. Здесь будет поддерживаться (с небольшой задержкой) копия первичных данных.

Именно в этом сечении - между репликационными серверами - связь может быть медленной или недостаточно надежной. Передаваемые данные в составе транзакции при недоступности узла-получателя записываются в стабильные очереди на диске и затем передаются по мере возможности. Данные могут передаваться в удаленный узел по маршруту, содержащему несколько репликационных серверов. В частности, такая возможность лежит в основе построения иерархических систем репликации.

В одной базе данных могут содержаться как первичные данные, так и данные-копии. Приложение-клиент, работающее со своей СУБД, может вносить изменения напрямую (операторами INSERT, DELETE, UPDATE) только в первичные данные. Для изменения копии данных предназначен механизм асинхронного вызова процедур. Для работы этого механизма в нескольких базах данных создаются процедуры с одинаковым именем и параметрами, но, возможно, с различным текстом. В одной базе данных

процедура помечается как предназначенная к репликации. Вызов этой процедуры вместе со значениями параметров через журнал и механизм репликации передается к узлам-подписчикам, и в их базах данных вызывается одноименная процедура с теми же значениями параметров.

2. МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

2.1 Лабораторная работа №1,2 (4 часа).

Тема: «Введение в базы данных»

2.1.1 Цель работы: Изучить СУБД

2.1.2 Задачи работы:

1. Понятие, назначение баз данных

2. Основные компоненты баз данных

2.1.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. СУБД OpenOffice BASE

2.1.4 Описание (ход) работы:

1.Запустите OpenOffice.Org BASE (Пуск — Программы — OpenOffice.Org — OpenOffice BASE).

2.Появится окно Мастера баз данных.

3.В появившемся окне необходимо оставить вариант «Создать новую базу данных» и нажать кнопку «Далее».

4.Далее появляется окно с предложением действий, выполняемых после сохранения базы данных. Оставьте все, как есть (по умолчанию) и нажмите кнопку «Готово».

5.После этого появится окно с предложением сохранить новую базу данных на диске компьютера. В данном окне найдите свою рабочую папку, в «Имя файла» введите «BASE_1» и нажмите кнопку «Сохранить»:

6.Создание таблиц в OpenOffice BASE.

После создания базы данных появится стартовое окно программы:

- Стартовое окно базы данных кроме вкладок для основных объектов, содержит командные кнопки: Открыть, Конструктор, Создать. С их помощью и выбирается режим работы с базой.

- Кнопка Открыть открывает избранный объект. Если это таблица, о ее можно просмотреть, внести новые записи или изменить те, что были внесены ранее.

- Кнопка Конструктор тоже открывает избранный объект, но по-другому. Она открывает его структуру и позволяет править не содержимое, а устройство. Если это таблица, в нее можно вводить новые поля или изменять свойства существующих полей.

- Если это форма, в ней можно изменять или создавать элементы управления. Очевидно, что этот режим служит не для пользователей базы, а для ее разработчиков.

- Действие кнопки Создать соответствует ее названию. Она служит для создания новых объектов. Этот элемент управления тоже предназначен для проектировщиков базы.

7.В окне «База дынных» Выберите вкладку «Таблицы» и в окне «Задачи» нажмите «Создать в режиме дизайна».

8.В макет таблицы введите следующие данные:

В первый и последний столбец в «тип поля» ввести «Текст [VARCHAR]», а в остальные «Вещественное [REAL]»

9.Теперь нажмите «Сохранить». Появится окно:

Введите «Нач-таб».

Появится окно следующего содержания. Нажмите кнопку «Да».

Далее в окне:

В поле ID В поле «Автозначение» выберите «Да». Данный параметр позволит автоматически генерировать значение идентификаторов строки. Нажмите «Сохранить». Закройте окно.

Теперь интерфейс программы имеет вид (в списке таблиц появилась таблица «Нач-таб»):

10. Нажмите два раза на таблице «Нач-таб» и подготовьте таблицу следующего содержания:

Тип	U заж, в	I, ма	Диаметр баллона, мм	Длина лампы, мм	Тип цоколя
СН-1	150	20	55	90	Р-27
СН-2	65	30	55	90	Р-27
МН-3	48	1	15	35	1Ш-12
МН-4	80	1,5	15	35	1Ш-12
МН-5	50	0,2	9	33	Р-IO
МН-6	60	0,8	6,8	28	Нет

Сохраните заполненную таблицу («Сохранить текущую запись»).

Отсортируйте записи в таблице по полю I, мА путем нажатия на соответствующем значке.

2.2 Лабораторная работа №3,4 (4 часа).

Тема: «Обзор современных систем управления базами данных»

2.2.1 Цель работы: провести обзор современных СУБД.

2.2.2 Задачи работы:

1. Современные системы управления базами данных.

2.2.3 Описание (ход) работы

Современные СУБД в основном являются приложениями Windows, так как данная среда позволяет более полно использовать возможности персональной ЭВМ, нежели среда DOS. Снижение стоимости высокопроизводительных ПК обусловил не только широкий переход к среде Windows, где разработчик программного обеспечения может в меньшей степени заботиться о распределении ресурсов, но также сделал программное обеспечение ПК в целом и СУБД в частности менее критичными к аппаратным ресурсам ЭВМ.

Среди наиболее ярких представителей систем управления базами данных можно отметить: LotusApproach, Microsoft Access, BorlanddBase, BorlandParadox, MicrosoftVisualFoxPro, MicrosoftVisualBasic, а также СУБД Microsoft SQL Server и Oracle, используемые в приложениях, построенных по технологии "клиент-сервер". Фактически, у любой современной СУБД существует аналог, выпускаемый другой компанией, имеющий аналогичную область применения и возможности, любое приложение способно работать со многими форматами представления данных, осуществлять экспорт и импорт данных благодаря наличию большого числа конвертеров. Общепринятыми, также, являются технологи, позволяющие использовать возможности других приложений, например, текстовых процессоров, пакетов построения графиков и т.п., и встроенные версии языков высокого уровня (чаще - диалекты SQL и/или VBA) и средства визуального программирования интерфейсов разрабатываемых приложений. Поэтому уже не имеет существенного значения, на каком языке и на основе какого пакета написано конкретное приложение, и какой формат данных в нем используется. Более того, стандартом "де-факто" стала "быстрая разработка приложений" или RAD (от английского RapidApplicationDevelopment), основанная на широко декларируемом в литературе "открытом подходе", то есть необходимость и возможность использования различных

прикладных программ и технологий для разработки более гибких и мощных систем обработки данных. Поэтому в одном ряду с "классическими" СУБД все чаще упоминаются языки программирования VisualBasic 4.0 и Visual C++, которые позволяют быстро создавать необходимые компоненты приложений, критичные по скорости работы, которые трудно, а иногда невозможно разработать средствами "классических" СУБД. Современный подход к управлению базами данных подразумевает также широкое использование технологии "клиент-сервер".

Таким образом, на сегодняшний день разработчик не связан рамками какого-либо конкретного пакета, а в зависимости от поставленной задачи может использовать самые разные приложения. Поэтому, более важным представляется общее направление развития СУБД и других средств разработки приложений в настоящее время.

2.3 Лабораторная работа №5,6 (4 часа).

Тема: «Архитектура СУБД»

2.3.1 Цель работы: Рассмотреть архитектуру СУБД.

2.3.2 Задачи работы:

1. Архитектура СУБД.

2. Рассмотреть уровни архитектуры.

2.3.3 Описание (ход) работы:

Существует три основных уровня архитектуры или три уровня описания элементов данных. Это внешний, концептуальный и внутренний уровни, которые формируют так называемую трёхуровневую архитектуру.

Внешний уровень - уровень, на котором данные воспринимаются пользователями, тогда как СУБД и операционная система воспринимают данные на внутреннем уровне.

Концептуальный уровень представления данных осуществляет отображение внешнего уровня на внутренний и обеспечивает требуемую независимость друг от друга.

2.4 Лабораторная работа №7,8 (4 часа).

Тема: «Модели данных»

2.4.1 Цель работы: изучить модели данных.

2.4.2 Задачи работы:

1.Понятие модели данных

2.Сетевая модель данных

3.Реляционная модель данных

4.Иерархическая модель данных

2.4.3 Описание (ход) работы:

Сетевой подход к организации данных является расширением иерархического подхода. В иерархических структурах запись-потомок должна иметь в точности одного предка; в сетевой структуре данных у потомка может иметься любое число предков.

Сетевая БД состоит из набора записей и набора связей между этими записями, а если говорить более точно, из набора экземпляров каждого типа из заданного в схеме БД набора типов записи и набора экземпляров каждого типа из заданного набора типов связи.

Тип связи определяется для двух типов записи: предка и потомка. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка. Для данного типа связи *L* с типом записи предка *P* и типом записи потомка *C* должны выполняться следующие два условия:

- каждый экземпляр типа записи *P* является предком только в одном экземпляре типа связи *L*;
- каждый экземпляр типа записи *C* является потомком не более чем в одном экземпляре типа связи *L*.

На формирование типов связи не накладываются особые ограничения; возможны, например, следующие ситуации:

- тип записи потомка в одном типе связи *L1* может быть типом записи предка в другом типе связи *L2*(как в иерархии);
- данный тип записи *P* может быть типом записи предка в любом числе типов связи;
- данный тип записи *P* может быть типом записи потомка в любом числе типов связи;
- может существовать любое число типов связи с одним и тем же типом записи предка и одним и тем же типом записи потомка; и если *L1* и *L2* - два типа связи с одним и тем же типом записи предка *P* и одним и тем же типом записи потомка *C*, то правила, по которым образуется родство, в разных связях могут различаться;
- типы записи *X* и *Y* могут быть предком и потомком в одной связи и потомком и предком – в другой;
- предок и потомок могут быть одного типа записи.

Почти все современные системы основаны на **реляционной** (relational) модели управления базами данных. Название **реляционная** связано с тем, что каждая запись в

такой базе данных содержит информацию, относящуюся только к одному конкретному объекту.

В **реляционной** СУБД все обрабатываемые данные представляются в виде плоских таблиц. Информация об объектах определенного вида представляется в табличном виде: в столбцах таблицы сосредоточены различные атрибуты объектов, а строки предназначены для сведения описаний всех атрибутов к отдельным экземплярам объектов.

Модель, созданная на этапе инфологического моделирования, в наибольшей степени удовлетворяет принципам реляционности. Однако для приведения этой модели к реляционной необходимо выполнить процедуру, называемую **нормализацией**.

Теория нормализации оперирует с пятью **нормальными формами**. Эти формы предназначены для уменьшения избыточности информации, поэтому каждая последующая нормальная форма должна удовлетворять требованиям предыдущей и некоторым дополнительным условиям. При практическом проектировании баз данных четвертая и пятая формы, как правило, не используются. Мы ограничились рассмотрением первых четырех нормальных форм.

Введем понятия, необходимые для понимания процесса приведения модели к реляционной схеме.

Отношение - абстракция описываемого объекта как совокупность его свойств. Проводя инфологический этап проектирования, мы говорили об абстракции объектов и приписывали им некоторые свойства. Теперь же, проводя концептуальное проектирование, мы переходим к следующему уровню абстракции. На данном этапе объектов, как таковых, уже не существует. Мы оперируем совокупностью свойств, которые и определяют объект.

Экземпляр отношения - совокупность значений свойств конкретного объекта.

Первичный ключ - идентифицирующая совокупность атрибутов, т.е. значение этих атрибутов уникально в данном отношении. Не существует двух экземпляров отношения содержащих одинаковые значения в первичном ключе.

Простой атрибут - атрибут, значения которого неделимы.

Сложный атрибут - атрибут, значением которого является совокупность значений нескольких различных свойств объекта или несколько значений одного свойства.

В классической теории баз данных, модель данных есть формальная теория представления и обработки данных в системе управления базами данных(СУБД), которая включает, по меньшей мере, три аспекта:

– аспект структуры: методы описания типов и логических структур данных в базе данных;

– аспект манипуляции: методы манипулирования данными;

– аспект целостности: методы описания и поддержки целостности базы данных.

Аспект структуры определяет, что из себя логически представляет база данных, аспект манипуляции определяет способы перехода между состояниями базы данных (то есть способы модификации данных) и способы извлечения данных из базы данных, аспект целостности определяет средства описаний корректных состояний базы данных.

Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Эти объекты позволяют моделировать структуру данных, а операторы — поведение данных[1].

Каждая БД и СУБД строится на основе некоторой явной или неявной модели данных. Все СУБД, построенные на одной и той же модели данных, относят к одному типу. Например, основой реляционных СУБД является реляционная модель данных, сетевых СУБД — сетевая модель данных, иерархических СУБД — иерархическая модель данных и т. д.

Иерархическая модель данных — это модель данных, где используется представление базы данных в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней.

Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка (объект более близкий к корню) к потомку (объект более низкого уровня), при этом возможна ситуация, когда объект-предок не имеет потомков или имеет их несколько, тогда как у объекта-потомка обязательно только один предок. Объекты, имеющие общего предка, называются близнецами (в программировании применительно к структуре данных дерево устоялось название братья).

Базы данных с иерархической моделью одни из самых старых, и стали первыми системами управления базами данных для мейнфреймов. Разрабатывались в 1950-х и 1960-х, например, InformationManagementSystem (IMS) фирмы IBM.

2.5 Лабораторная работа №9,10,11,12 (8 часов).

Тема: «Реляционная модель данных»

2.5.1 Цель работы: рассмотреть реляционную модель данных.

2.5.2 Задачи работы:

1.Понятие домена, атрибута, кортежа, отношения

2. Табличное представление отношения

3.Схема отношения

2.5.3 Описание (ход) работы:

Понятие *домена* более специфично для баз данных, хотя и имеет некоторые аналогии с подтипами в некоторых языках программирования. В самом общем виде домен определяется заданием некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу типа данных. Если вычисление этого логического выражения дает результат "истина", то элемент данных является элементом домена.

Кортеж, соответствующий данной схеме отношения, - это множество пар {имя атрибута, значение}, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения. "Значение" является допустимым значением домена данного атрибута (или типа данных, если понятие домена не поддерживается). Тем самым, степень или "арность" кортежа, т.е. число элементов в нем, совпадает с "арностью" соответствующей схемы отношения. Попросту говоря, кортеж - это набор именованных значений заданного типа.

Отношение - это множество кортежей, соответствующих одной схеме отношения. Иногда, чтобы не путаться, говорят "отношение-схема" и "отношение-экземпляр", иногда схему отношения называют заголовком отношения, а отношение как набор кортежей - телом отношения. На самом деле, понятие схемы отношения ближе всего к понятию структурного типа данных в языках программирования. Было бы вполне логично разрешать отдельно определять схему отношения, а затем одно или несколько отношений с данной схемой.

Кортеж, соответствующий данной схеме отношения, - это множество пар {имя атрибута, значение}, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения. Попросту говоря, кортеж - это набор именованных значений заданного типа.

Понятие *реляционный* (англ.*relation* -отношение) связано с разработками известного американского специалиста в области систем баз данных Е. Кодда.

Эти модели характеризуются простотой структуры данных, удобным для пользователя табличным представлением и возможностью использования формального аппарата алгебры отношений и реляционного исчисления для обработки данных.

Реляционная модель ориентирована на организацию данных в виде двумерных таблиц. Каждая ***реляционная таблица*** представляет собой двумерный массив и обладает следующими свойствами:

- каждый элемент таблицы - один элемент данных;

- все столбцы в таблице однородные, т.е. все элементы в столбце имеют одинаковый тип (числовой, символьный и т.д.) и длину;

- каждый столбец имеет уникальное имя;

- одинаковые строки в таблице отсутствуют;

- порядок следования строк и столбцов может быть произвольным.

Пример. Реляционной таблицей можно представить информацию о студентах, обучающихся в вузе (рис. 7).

N личного дела	Фамилия	Имя	Отчество	Дата рождения	Группа
	Сергеев	Петр	Михайлович	01.01.76	
	Петрова	Анна	Владимировна	15.03.75	
	Анохин	Андрей	Борисович	14.04.76	

Рис. 7. Пример реляционной таблицы

Отношения представлены в виде *таблиц,* строки которых соответствуют *записям,* а столбцы - *полям.*

Поле, каждое значение которого однозначно определяет соответствующую запись, называется ***простым ключом*** (ключевым полем). Если записи однозначно определяются значениями нескольких полей, то такая таблица базы данных имеет***составной ключ***. В примере, показанном на рис. 7, ключевым полем таблицы является "N личного дела".

Чтобы связать две реляционные таблицы, необходимо ключ первой таблицы ввести в состав ключа второй таблицы (возможно совпадение ключей); в противном случае нужно ввести в структуру первой таблицы *внешний ключ* - ключ второй таблицы.

Схемой отношения называется перечень имен атрибутов данного отношения с указанием домена, к которому они относятся:

$$S_R = (A_1, A_2, A_n),\ A_i \subseteq D_i.$$

Если атрибуты принимают значения из одного и того же домена, то они называются θ-сравнимыми, где θ – множество допустимых операций сравнений, заданных для данного домена. Например, если домен содержит числовые данные, то для

него допустимы все операции сравнения, тогда $\theta=\{=, <>, >=, <=, <, >\}$. Однако, и для доменов, содержащих символьные данные, могут быть заданы не только операции сравнения по равенству и неравенству значений. Если для данного домена задано лексикографическое упорядочение, то он имеет также полный спектр операций сравнения.

Схемы двух отношений называются ***эквивалентными***, если они имеют одинаковую степень и возможно такое упорядочение имен атрибутов в схемах, что на одинаковых местах будут находиться сравнимые атрибуты, то есть атрибуты, принимающие значения из одного домена:

Пусть $S_{R_1} = (A_1, A_2, ..., A_n)$ – схема отношения R_1. $S_{R_2} = (B_{i1}, B_{i2}, ..., B_{in})$ – схема отношения R_2 после упорядочения имен атрибутов. Тогда

$$S_{R_1} \sim S_{R_2} \Leftrightarrow \begin{cases} 1.\ n = m, \\ 2.\ A_j, B_{ij} \subseteq D_j. \end{cases}$$

Таким образом, для эквивалентных отношений выполняются следующие условия:

- Таблицы имеют одинаковое количество столбцов.
- Таблицы содержат столбцы с одинаковыми наименованиями.
- Столбцы с одинаковыми наименованиями содержат данные из одних и тех же доменов.
- Таблицы имеют одинаковые строки с учетом того, что порядок столбцов может различаться.

Все такие таблицы есть различные *изображения* одного и того же отношения.

2.6 Лабораторная работа №13,14,15,16 (8 часов).

Тема: «Реляционная алгебра и язык SQL»

2.6.1 Цель работы: изучить реляционную алгебру и язык SQL.

2.6.2 Задачи работы:

1. Особенности языков описания и манипулирования данными в реляционной модели

2. языки запросов, основанные на реляционном исчислении.

3. структурный язык запросов SQL.

2.6.3 Описание (ход) работы:

Двумя фундаментальными языками запросов к реляционным БД являются языки реляционной алгебры и реляционного исчисления. При всей своей строгости и

теоретической обоснованности эти языки редко используются в современных реляционных СУБД в качестве средств пользовательского интерфейса. Запросы на этих языках трудно формулировать и понимать. SQL представляет собой некоторую комбинацию реляционного исчисления кортежей и реляционной алгебры, причем до сих пор нет общего согласия, к какому из классических языков он ближе. При этом возможности SQL шире, чем у этих базовых реляционных языков, в частности, в общем случае невозможна трансляция запроса, сформулированного на SQL, в выражение реляционной алгебры, требуется некоторое ее расширение.

Существенными свойствами подъязыка запросов SQL являются возможность простого формулирования запросов с соединениями нескольких отношений и использование вложенных подзапросов в предикатах выборки. Вообще говоря, одновременное наличие обоих средств избыточно, но это дает пользователю при формулировании запроса возможность выбора более понятного ему варианта.

В предикатах со вложенными подзапросами в SQL System R можно употреблять теретико-множественные операторы сравнения, что позволяет формулировать квантифицированные запросы (эти возможности обычно труднее всего понимаются пользователями и поэтому в дальнейшем в SQL появились явно квантифицируемые предикаты).

Существенной особенностью SQL является возможность указания в запросе потребности группирования отношения-результата по указанным полям с поддержкой условий выборки на всю группу целиком. Такие условия выборки могут содержать агрегатные функции, вычисляемые на группе. Эта возможность SQL главным образом отличает этот язык от языков реляционной алгебры и реляционного исчисления, не содержащих аналогичных средств.

Еще одним отличием SQL является необязательное удаление кортежей-дубликатов в окончательном или промежуточных отношениях-результатах. Строго говоря, результатом оператора выборки в языке SQL является не отношение, а мультимножество кортежей. В тех случаях, когда семантика запроса требует наличия отношения, уничтожение дубликатов производится неявно.

Самый общий вид запроса на языке SQL представляет теоретико-множественное алгебраическое выражение, составленное из элементарных запросов. В SQL System R допускались все базовые теретико-множественные операции (UNION, INTERSECT и MINUS).

Работа с неопределенными значениями в SQL System R до конца продумана не была, хотя неявно предполагалось использование трехзначной логики при вычислении логических выражений.

Операторы манипулирования данными UPDATE и DELETE построены на тех же принципах, что и оператор выборки данных SELECT. Набор кортежей указанного отношения, подлежащих модификации или удалению, определяется входящим всоответствующий оператор логическим выражением, которое может включать сложные предикаты, в том числе и с вложенными подзапросами.

В операторе вставки кортежа(ей) в указанное отношение заносимый кортеж может задаваться как в литеральной форме, так и с помощью внутреннего подоператора выборки.

В число операторов определения схемы БД SQL System R входили операторы создания и уничтожения постоянных и временных хранимых отношений (CREATE TABLE и DROP TABLE) и создания и уничтожения представляемых отношений (CREATE VIEW и DROP VIEW). В языке и в реализации System R не запрещалось использовать операторы определения схемы в пределах транзакции, содержащей операторы выборки и манипулирования данными. Допускалось, например, использование операторов выборки и манипулирования данными, в которых указываются отношения, не существующие в БД к моменту компиляции оператора. Конечно, эта возможность существенно усложняла реализацию и требовалась по существу очень редко.

Оператор манипулирования схемой БД ALTER TABLE позволял добавлять указываемые поля к существующим отношениям. В описании языка определялось, что выполнение этого оператора не должно приводить к недействительности ранее откомпилированных операторов над отношением, схема которого изменяется, и что значения вновь определенных полей в существующих кортежах отношения становятся неопределенными.

Язык SQL System R включал очень мощные средства контроля и поддержания целостности БД. Средства контроля базировались на аппарате ограничений целостности (ASSERTIONS). Фактически, ограничение целостности - это логическое выражение, вычисляемое над текущим состоянием БД, ложность которого соответствует нецелостному состоянию БД. Логическое выражение ограничения целостности могло содержать любой допустимый в языке предикат.

Механизмы ограничений целостности и триггеров System R являлись очень мощными и общими, но реализация их очень трудна и накладна (как уже отмечалось, триггеры так и не были реализованы в System R). Дополнительную сложность в

реализации создавал тот факт, что допускалось (по крайней мере не запрещалось языком) определение ограничений целостности и триггеров в пределах той же транзакции, в которой выполняются операторы манипулирования данными. При наиболее полной реализации требовалось бы большое число дополнительных действий во время выполнения транзакции. Кроме того, в ряде случаев отсутствие зафиксированной семантики соответствующих конструкций языка приводило к неоднозначному пониманию выполнения транзакций.

В языке отсутствуют какие-либо ограничения по поводу использования представлений: в любом операторе SQL, в котором допускается использование имени хранимого отношения, допускается и использование имени представления. В SQL System R ничего не говорится о рекомендуемом способе реализации доступа к представлениям, но при любом способе эффект должен быть таким, как если бы выполнить полную материализацию представления до выполнения оператора.

Внесение в реляционный язык, каким является SQL, явных операторов порождения и уничтожения структур физического уровня, поддерживающих эффективное выполнение запросов к БД, явилось в SQL System R чисто прагматическим решением, обеспечивающим возможность всех видов работ с БД с помощью одного языка.

В SQL System R упоминаются два вида таких структур: индексы и связи (links). Индекс в его абстрактном языковом представлении - это инвертированный файл, обеспечивающий доступ к кортежам соответствующего отношения на основе заданных значений одного или нескольких столбцов, составляющих ключ индекса. Операторы языка позволяли создавать и уничтожать индексы, но никаким образом не давали возможности явно указать на необходимость использования существующего индекса при выполнении оператора выборки, решение об этом возлагалось на реализацию.

С помощью оператора определения индекса можно было выразить два дополнительных утверждения, касающихся логической схемы отношения и физической структуры его хранения. Использование при определении индекса ключевого слова UNIQUE означало, что ключ этого индекса является возможным ключом соответствующего отношения. Фактически это означает наличие дополнительного механизма определения ограничения целостности отношения. Один из индексов для данного отношения мог быть определен с ключевым словом CLUSTERING. Это означает требование физической кластеризации во внешней памяти кортежей отношения с равными или близкими значениями ключа индекса.

Операторы определения связи позволяли в стиле сетевой модели данных организовать во внешней памяти списки кортежей указанного отношения. Как и в случае

индексов, операторы позволяли создавать и уничтожать такие списки, но не давали возможности явно указать на необходимость использования существующих списков при выполнении операторов выборки. Большая трудоемкость поддержания списков при выполнении операторов манипулирования данными и трудность выполнения оценок стоимости их использования при выполнении операторов выборки привели к тому, что механизм связей исчез из языка уже на поздней стадии проекта System R. С тех пор этот механизм, насколько нам известно, не появлялся ни в одном варианте SQL.

Существенной особенностью языка SQL, появившейся в нем с самого начала, является обеспечение защиты доступа к данным средствами самого языка. Основная идея такого подхода состоит в том, что по отношению к любому отношению БД и любому столбцу отношения вводится предопределенный набор привилегий. С каждой транзакцией неявно связывается идентификатор пользователя, от имени которого она выполняется (способы связи и идентификации пользователей не фиксируются в языке и определяются в реализации).

После создания нового отношения все привилегии, связанные с этим отношением и всеми его столбцами, принадлежат только пользователю-создателю отношения. В число привилегий входит привилегия передачи всех или части привилегий другому пользователю, включая привилегию на передачу привилегий. Технически передача привилегий осуществляется при выполнении оператора SQL GRANT. Существует также привилегия изъятия всех или части привилегий у пользователя, которому они ранее были переданы. Эта привилегия также может передаваться. Технически изъятие привилегий происходит при выполнении оператора SQL REVOKE.

Проверка полномочности доступа к данным происходит на основе информации о полномочиях, существующих во время компиляции соответствующего оператора SQL. Подобно тому, что мы отмечали в связи с ограничениями целостности и триггерами, в SQL System R отсутствовали какие-либо ограничения по поводу использования операторов GRANT и REVOKE. Это приводило к существенным техническим затруднениям в реализации, а иногда к неоднозначному пониманию поведения.

Долгое время подход к защите данных от несанкционированного доступа принимался практически без критики, однако в связи с распространяющимся использованием реляционных СУБД в нетрадиционных приложениях все чаще раздается критика. Если, например, в системе БД должна поддерживаться многоуровневая защита данных, соответствующую систему полномочий весьма трудно, а иногда и невозможно построить на основе средств SQL.

В SQL System R существовали два специальных оператора для установки так называемых точек сохранения транзакции и для отката транзакции к ранее установленной точке сохранения. В литературе, относящейся к System R, обсуждение этих возможностей практически не содержится, из чего неявно следует, что они не были реализованы.

Прямолинейная реализация этого механизма не вызывает особых технических затруднений, но и не очень полезна, потому что после выполнения частичного отката транзакции для успешного продолжения работы прикладной программы потребовалось бы и восстановить ее состояние в соответствующей точке, а это никак не поддерживается. Понятно, что при более тщательной проработке должны быть увязаны механизмы точек сохранения и контроля целостности. Например, было бы естественно, чтобы при выполнении оператора ENFORCE INTEGRITY, если какие-либо ограничения целостности нарушаются, происходил автоматический откат транзакции к ближайшей точки сохранения, в которой нарушения целостности БД не было. Это значительно усложнило бы реализацию, но было бы очень полезно. Аналогично, можно было бы использовать механизм точек сохранения при автоматических откатах транзакций по причине возникновения синхронизационных тупиков.

Отметим еще два важных свойства языка SQL System R, которые в разных видах присутствуют во всех развитых последующих вариантах языка.

В SQL System R присутствуют специальные операторы, поддерживающие встраивание операторов SQL в традиционные языки программирования (в System R основным таким языком был PL/1).

Основная проблема встраивания SQL в язык программирования состояла в том, что SQL - реляционный язык, т.е. его операторы большей частью работают со множествами, в то время как в языках программирования основными являются скалярные операции. Решение SQL состоит в том, что в язык дополнительно включаются операторы, обеспечивающие покортежный доступ к результату запроса к БД.

Для этого в язык вводится понятие курсора, с которым связывается оператор выборки. Над определенным курсором можно выполнять оператор OPEN, означающий материализацию отношения-результата запроса, оператор FETCH, позволяющий выбрать очередной кортеж результирующего отношения в память программы, и оператор CLOSE, означающий конец работы с данным курсором.

Дополнительную гибкость при создании прикладных программ со встроенным SQL обеспечивает возможность параметризации операторов SQL значениями переменных включающей программы.

Для упрощения создания интерактивных SQL-ориентированных систем в SQL System R были включены операторы, позволяющие во время выполнения транзакции откомпилировать и выполнить любой оператор SQL.

Оператор PREPARE вызывает динамическую компиляцию оператора SQL, текст которого содержится в указанной переменной символьной строке включающей программы. Текст может быть помещен в переменную при выполнении программы любым допустимым способом, например, введен с терминала.

Оператор DESCRIBE служит для получения информации об указанном операторе SQL, ранее подготовленном с помощью оператора PREPARE. С помощью этого оператора можно узнать, во-первых, является ли подготовленный оператор оператором выборки, и во-вторых, если это оператор выборки, получить полную информацию о числе и типах столбцов результирующего отношения.

Для выполнения ранее подготовленного оператора SQL, не являющегося оператором выборки, служит оператор EXECUTE. Для выполнения динамически подготовленного оператора выборки используется аппарат курсоров с некоторыми отличиями по части задания адресов переменных включающей программы, в которые должны быть помещены значения столбцов текущего кортежа результата.

Подводя итог приведенному краткому описанию основных черт SQL System R, отметим, что несмотря на недостаточную техническую проработку, в идейном отношении язык содержал все необходимые средства, позволяющие использовать его как базовый язык СУБД.

2.7 Лабораторная работа №17,18 (4 часа).

Тема: «Проектирование концептуальной модели данных»

2.7.1 Цель работы: рассмотреть проектирование концептуальной модели данных.

2.7.2 Задачи работы:

1. Анализ данных

2. Нормализация отношений

2.7.3 Описание (ход) работы:

В качестве примера возьмем базу данных компании, которая занимается издательской деятельностью. База данных создаётся для информационного обслуживания редакторов, менеджеров и других сотрудников компании. БД должна содержать данные о сотрудниках компании, книгах, авторах, финансовом состоянии компании и предоставлять возможность получать разнообразные отчёты. В соответствии с

предметной областью система строится с учётом следующих особенностей: каждая книга издаётся в рамках контракта; книга может быть написана несколькими авторами; контракт подписывается одним менеджером и всеми авторами книги; каждый автор может написать несколько книг (по разным контрактам); порядок, в котором авторы указаны на обложке, влияет на размер гонорара; если сотрудник является редактором, то он может работать одновременно над несколькими книгами; у каждой книги может быть несколько редакторов, один из них – ответственный редактор; каждый заказ оформляется на одного заказчика; в заказе на покупку может быть перечислено несколько книг. Выделим базовые сущности этой предметной области: Сотрудники компании. Атрибуты сотрудников – ФИО, табельный номер, пол, дата рождения, паспортные данные, ИНН, должность, оклад, домашний адрес и телефоны. Для редакторов необходимо хранить сведения о редактируемых книгах; для менеджеров – сведения о подписанных контрактах. Авторы. Атрибуты авторов – ФИО, ИНН (индивидуальный номер налогоплательщика), паспортные данные, домашний адрес, телефоны. Для авторов необходимо хранить сведения о написанных книгах. Книги. Атрибуты книги – авторы, название, тираж, дата выхода, цена одного экземпляра, общие затраты на издание, авторский гонорар. Контракты будем рассматривать как связь между авторами, книгами и менеджерами. Атрибуты контракта – номер, дата подписания и участники. Для отражения финансового положения компании в системе нужно учитывать заказы на книги. Для заказа необходимо хранить номер заказа, заказчика, адрес заказчика, дату поступления заказа, дату его выполнения, список заказанных книг с указанием количества экземпляров.

Нормализация отношений - обеспечивает эффективность структур данных в реляционной БД

Этот процесс уменьшает избыточность данных (хранение одинаковых данных в нескольких местах). В результате более рационально используется внешняя память, уменьшается вероятность нарушения согласованности данных.

Нормализация представляет собой действия по последовательному преобразованию исходной (ненормализованной) таблицы в нормализованные отношения в первой нормальной форме (1НФ), 2НФ, 3НФ, нормальной форме Бойса-Кодда (НФБК), 4НФ, 5НФ.

На практике, как правило, ограничиваются 3НФ, ее оказывается вполне достаточно для создания надежной схемы БД.

Основные свойства нормальных форм:

- каждая следующая нормальная форма улучшает свойства предыдущей нормальной формы;

- при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются.

При проектировании баз данных упор в первую очередь делается на достоверность и непротиворечивость хранимых данных, причем эти свойства не должны утрачиваться в процессе работы с данными, т.е. после многочисленных изменений, удалений и дополнений данных по отношению к первоначальному состоянию БД.

Для поддержания БД в устойчивом состоянии используется ряд механизмов, которые получили обобщенное название средств поддержки целостности. Эти механизмы применяются как статически (на этапе проектирования БД), так и динамически (в процессе работы с БД).Приведение структуры БД в соответствие этим ограничениям - это и есть нормализация.

В целом суть этих ограничений весьма проста: каждый факт, хранимый в БД, должен храниться один-единственный раз, поскольку дублирование может привести (и на практике непременно приводит, как только проект приобретает реальную сложность) к несогласованности между копиями одной и той же информации. Следует избегать любых неоднозначностей, а также избыточности хранимой информации.

Нормализация отношений - обеспечивает эффективность структур данных в реляционной БД

Этот процесс уменьшает избыточность данных (хранение одинаковых данных в нескольких местах). В результате более рационально используется внешняя память, уменьшается вероятность нарушения согласованности данных.

2.8 Лабораторная работа №19,20 (4 часа).

Тема: «Проектирование логической модели данных»

2.8.1 Цель работы: рассмотреть проектирование логической модели данных.

2.8.2 Задачи работы:

1. Отображение на реляционную модель

2. Отображение на иерархическую модель

2.8.3 Описание (ход) работы:

Логическое проектирование представляет собой необходимый этап при создании БД. **Основной задачей логического проектирования** является разработка логической схемы, ориентированной на выбранную систему управления базами данных (СУБД). Этап логического проектирования в отличие от концептуального проектирования полностью ориентирован на инструментальные средства компьютера.

Процесс логического проектирования состоит из следующих этапов:

1. Выбор конкретной СУБД.

2. Отображение концептуальной схемы на логическую схему.

3. Выбор ключей.

4. Описание языка запросов.

Одним из основных критериев выбора СУБД является оценка того, насколько эффективно внутренняя модель данных, поддерживаемая системой, способна описать концептуальную схему. Существующие СУБД делятся по типам моделей данных на реляционные, иерархические и сетевые. СУБД, ориентированные на персональные компьютеры, как правило, поддерживают реляционную модель данных. Подавляющее большинство современных СУБД – реляционные. Если выбрана реляционная система, то концептуальную схему БД предстоит отображать на реляционную.

При отображении концептуальной схемы на реляционную модель данных каждый прямоугольник схемы отображается в таблицу. При этом следует учитывать ограничения на размеры таблиц, которые накладывает выбранная СУБД.

Аналогично поступают со всеми остальными сущностями (объектами) концептуальной схемы.

Выводы. Концептуальная модель представляет объекты предметной области и их взаимосвязи без указания способов их физического хранения.

Таким образом, концептуальная модель является, по существу, моделью предметной области. При проектировании концептуальной модели все усилия разработчика должны быть направлены в основном на структуризацию данных и выявление взаимосвязей между ними без рассмотрения особенностей реализации и вопросов эффективности обработки. Проектирование концептуальной модели основано на анализе решаемых на этом предприятии задач по обработке данных.

Концептуальная модель включает описания объектов и их взаимосвязей, представляющих интерес в рассматриваемой предметной области и выявляемых в результате анализа данных.

Концептуальная модель транспонируется затем в модель данных, совместимую с выбранной СУБД. Версия концептуальной модели, которая может быть обеспечена конкретной СУБД, называется логической моделью.

Логическая модель отражает логические связи между элементами данных вне зависимости от их содержания и среды хранения.

Логическая модель данных может быть реляционной, иерархической или сетевой. Логическая модель отображается в физическую память, такую, как диск, лента или какой-либо другой носитель информации.

Физическая модель, определяющая размещение данных, методы доступа и технику индексирования, называется внутренней моделью системы.

В современных СУБД выполнение задач физического проектирования автоматизировано.

2.9 Лабораторная работа №21, 22 (4 часа).

Тема: «Физическая модель данных»

2.9.1 Цель работы: рассмотреть физическую модель данных.

2.9.2 Задачи работы:

1. Физическая модель данных

2.9.3 Описание (ход) работы:

Логический уровень - это абстрактный взгляд на данные, на нем данные представляются так, как выглядят в реальном мире, и могут называться так, как они называются в реальном мире, например "Автор", "Издательство" или "Авторский гонорар". Объекты модели, представляемые на логическом уровне, называются сущностями и атрибутами (подробнее о сущностях и атрибутах будет рассказано ниже). Логическая модель данных может быть построена на основе другой логической модели, например на основе модели процессов. Логическая модель данных является универсальной и никак не связана с конкретной реализацией СУБД. **Физическая модель** данных, напротив, зависит от конкретной СУБД, фактически являясь отображением системного каталога. В физической модели содержится информация о всех объектах БД. Поскольку стандартов на объекты БД не существует (например, нет стандарта на типы данных), физическая модель зависит от конкретной реализации СУБД. Следовательно, одной и той же логической модели могут соответствовать несколько разных физических моделей. Если в логической модели не имеет значения, какой конкретно тип данных имеет атрибут, то в физической модели важно описать всю информацию о конкретных физических объектах - таблицах, колонках, индексах, процедурах и т. д. Разделение модели данных на логические и физические позволяет решить несколько важных задач.
Масштабирование. Создание модели данных, как правило, начинается с создания логической модели. После описания логической модели, проектировщик может выбрать

необходимую СУБД и ERwin автоматически создаст соответствующую физическую модель. На основе физической модели ERwin может сгенерировать системный каталог СУБД или соответствующий SQL-скрипт. Этот процесс называется прямым проектированием (ForwardEngineering). Тем самым достигается масштабируемость - создав одну логическую модель данных, можно сгенерировать физические модели под любую поддерживаемую ERwin СУБД. С другой стороны, ERwinспособен по содержимому системного каталога или SQL-скрипту воссоздать о физическую и логическую модель данных (ReverseEngineering). На основе полученной логической модели данных можно сгенерировать физическую модель для другой СУБД и затем сгенерировать ее системный каталог. Следовательно, ERwin позволяет решить задачу по переносу структуры данных с одного сервера на другой. Например, можно перенести структуру данных с Oracle на Informix (или наоборот) или перенести структуру dbf-файлов в реляционную СУБД, тем самым облегчив решение по переходу от файл-серверной к клиент-серверной ИС. Заметим, однако, что формальный перенос структуры "плоских" таблиц на реляционную СУБД обычно неэффективен. Для того чтобы извлечь выгоды от перехода на клиент-серверную технологию, структуру данных следует модифицировать.

2.10 Лабораторная работа №23,24,25 (6 часов).

Тема: «Администрирование базы данных»

2.10.1 Цель работы: изучить администрирование базы данных.

2.10.2 Задачи работы:

1. Функция администрирования базы данных

2. Жизненный цикл системы с базой данных

2.10.3 Описание (ход) работы:

1.Администрирование базы данных – это функция управления базой данных (БД). Лицо ответственное за администрирование БД называется “Администратор базы данных” (АБД) или “DatabaseAdministrator” (DBA).

Функция “администрирования данных” стала активно рассматриваться и определяться как вполне самостоятельная с конца 60-х годов. Практическое значение это имело для предприятий, использующих вычислительную технику в системах информационного обеспечения для своей ежедневной деятельности. Специализация этой функции с течением времени совершенствовалась, но качественные изменения в этой

области стали происходить с началом использования так называемых интегрированных баз данных. Одна такая база данных могла использоваться для решения многих задач.

Таким образом, сформировалось определение БД как общего информационного ресурса предприятия, которое должно находиться всегда в работоспособном состоянии. И как для каждого общего ресурса значительной важности, БД стала требовать отдельного управления. Во многих случаях это было необходимо для обеспечения её повседневной эксплуатации, её развития в соответствии с растущими потребностями предприятия. К тому же БД и технология её разработки постоянно совершенствовались и уже требовались специальные знания высокого уровня для довольно сложного объекта, которым стала база данных. Отсюда функция управления базой данных и получила название “Администрирование базы данных”, а лицо ею управляющее стали называть “Администратор баз данных”.

2.11 Лабораторная работа №26,27 (4 часа).

Тема: «Словарь данных»

2.11.1 Цель работы: Рассмотреть словарь данных.

2.11.2 Задачи работы:

1. Определение словаря данных.
2. Основные термины.

2.11.3 Описание (ход) работы:

В словаре данных хранится информация о самой базе данных. Словарь данных представляет собой набор таблиц, создаваемых во время генерации базы данных сервером. Сервер базы данных, также, заботится об обновлении информации в словаре. Пользователи могут обращаться прямо к словарю, чтобы получить необходимую информацию об объектах базы данных. Однако информация, хранящаяся в таблицах словаря сложна для понимания, поэтому рекомендуется пользоваться представлениями, которые позволяют извлекать данные из словаря в более удобном формате. В СУБД Oracle существует четыре категории представлений, каждая из которых обозначается своим префиксом:

§ *USER_* - объекты, принадлежащие пользователю, т.е. те, которые пользователь создал сам

§ *ALL_* - объекты, к которым пользователь имеет доступ, т.е. собственные объекты и объекты, на доступ к которым у пользователя есть привилегия

§ *DBA_* - все объекты базы данных (только для пользователей с привилегией DBA)

§ *V$* - информация о производительности сервера базы данных.

2.12 Лабораторная работа №28,29 (4 часа).

Тема: «Общая характеристика баз знаний и экспертных систем»

2.12.1 Цель работы: Рассмотреть характеристику баз знаний и экспертных систем.

2.12.2 Задачи работы:

1. Базы знаний.

2.Экспертные системы

2.12.3 Описание (ход) работы:

Обязательной составляющей любой экспертной системы является база знаний. Как уже говорилось ранее, под знанием можно понимать обобщенную и формализованную информацию о свойствах и законах предметной области, с помощью которой реализуются процессы решения задач, преобразования данных и самих знаний, и, которое используется в процессе логического вывода. Поскольку не существует единого определения знаний, то будем рассматривать это понятие через набор его специфических характеристик. Определения: Переменная – набор с априорно неизвестными значениями. Переменная может быть связанной и несвязанной (свободной); в последнем случае переменной не приписано никакого значения. Само значение переменной можно рассматривать как некоторый объект. С этой точки зрения связанная переменная будет представлять пару , где исходным объектом (именованной переменной) является A1, а производным объектом, представляющим значение переменной, является A2. Константа может быть представлена как пара , где C – конкретный объект. В том случае, если для переменной предложено обозначение , то эта также суть константа. Определим основные специфические характеристики знаний: Внутренняя интерпретируемость знаний отождествляется с порождение некоторой функции (способа) связывания переменных с их значениями или иначе, способ порождения пар . Такая функция называется интерпретирующей, способ связывания, соответственно, интерпретацией. Данные являются парами типа . В системе с нечеткой логикой используется бесконечное множество оценок истинности. Например, утверждение «изменение стоимости товара ожидается с достоверностью 0,7» не является ни строго истинным, ни строго ложным, а лишь характеризуется некоторой степенью правдоподобия. Активность знаний. Это свойство знаний адаптироваться в ответ на изменяющиеся факты, т.е. возможность

обучения и самокоррекции. Функциональная целостность. Под этим свойством знаний понимается их непротиворечивость, независимость исходных посылок и разрешимость. Непротиворечивость знаний означает. Что в базе знаний невозможно появление двух взаимоисключающих фактов. Хотя принципиально возможно рассуждение на основе взаимоисключающих посылок. Требование разрешимости заключается в том, что любое истинное знание, формализуемое в базе знаний системы, может быть выведено в не й при помощи поддерживаемых стратегий вывода. Требование разрешимости, к сожалению, для некоторых моделей представления знаний (логики предикатов) не всегда выполнимо. Независимость означает невозможность вывода одного знания из другого формальным способом. Ситуативность. Наличие ситуативных связей определяет совместимость тех или иных знаний, хранимых в системе. В качестве таких связей могут выступать отношения времени, места, действия, причины и т.п.]истина, ложь[. Концепт есть упорядоченное множество семантических целых, представляющих понятие. Кроме этого того чтобы понятия, например К1 и к2 образовывали семантическое целое необходимо выполнение следующих условий: К1 и К2 связаны друг с другом как отношение и один из его аргументов; К1 и К2 связаны друг с другом как действие и его носитель или субъект; К1 и К2 связаны как функция ее аргумент. Для знаний должен выполняться «принцип матрешки», т.е. рекурсивная вложенность одних информационных единиц в другие. Каждая информационная единица может быть включена в состав другой, и из каждой такой единицы могут быть выделены составляющие ее элементы. Устанавливаемые между понятиями связи позволяют строить процедуры анализа знаний на совместимость, противоречивость и другие, которые невозможно реализовать при хранении традиционных массивов данных. Семантическое пространство с метрикой. Это свойство знаний связано с возможностью изменения знаний в системе оценок (метрик) в интервале }Bi{B/ ∈образуют семантическое целое, если каждый Bi в B образует семантическое целое как минимум с одним из Bj, где Bj } B1,B2,..Bn {, т.е. константами, для них не нужна интерпретирующая функция, они сами себя определяют.

В нашей стране современное состояние разработок в области экспертных систем можно охарактеризовать как стадию всевозрастающего интереса среди широких слоев экономистов, финансистов, преподавателей, инженеров, медиков, психологов, программистов, лингвистов. К сожалению, этот интерес имеет пока достаточно слабое материальное подкрепление - явная нехватка учебников и специальной литературы, отсутствие символьных процессоров и рабочих станций искусственного интеллекта, ограниченное финансирование исследований в этой области, слабый отечественный рынок программных продуктов для разработки экспертных систем. Поэтому

распространяются "подделки" под экспертные системы в виде многочисленных диалоговых систем и интерактивных пакетов прикладных программ, которые дискредитируют в глазах пользователей это чрезвычайно перспективное направление. Процесс создания экспертной системы требует участия высококвалифицированных специалистов в области искусственного интеллекта, которых пока выпускает небольшое количество высших учебных заведений страны. Современные экспертные системы широко используются для тиражирования опытам знаний ведущих специалистов практически во всех сферах экономики. Традиционно знания существуют в двух видах - коллективный опыт и личный опыт. Если большая часть знаний в предметной области представлена в виде коллективного опыта (например, высшая математика), эта предметная область не нуждается в экспертных системах. Если в предметной области большая часть знаний является личным опытом специалистов высокого уровня (экспертов), если эти знания по каким-либо причинам слабо структурированы, такая предметная область скорее всего нуждается в экспертной системе.

2.13 Лабораторная работа №30,31 (4 часа).

Тема: «СУБД ACCESS»

2.13.1 Цель работы: Изучить СУБД Access.

2.13.2 Задачи работы:

1. Назначение, общая характеристика и структура СУБД ACCESS

2. Состав БД: таблицы, управляющие и обрабатывающие запросы, формы, отчеты, страницы, макросы, модули.

3. Средства создания и модификации объектов базы данных.

2.13.3 Описание (ход) работы:

База данных представляет собой совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы и отображающих состояние объектов и их взаимосвязей в рассматриваемой предметной области.

Следует учесть, что это определение не является единственно возможным. Информатика в отношении определений чаще всего не похожа на математику с ее полной однозначностью. Если подойти к понятию “база данных” с чисто пользовательской точки зрения, то возникает другое определение: база данных – совокупность хранимых операционных данных некоторого предприятия.

В базе данных предприятия, например, может храниться:

- вся информация о штатном расписании, о рабочих и служащих предприятия;

- сведения о материальных ценностях;
- данные о поступлении сырья и комплектующих;
- сведения о запасах на складах;
- данные о выпуске готовой продукции;
- приказы и распоряжения дирекции и т.п.

Даже небольшие изменения какой-либо информации могут приводить к значительным изменениям в разных других местах.

Пример. Издание приказа о повышении в должности одного работника приводит к изменениям не только в личном деле работника, но и к изменениям в списках подразделения, в котором он работает, в ведомостях на зарплату, в графике отпусков и т.п.

Поскольку основу любой базы данных составляет информационная структура, базы данных делят на три типа: табличные (реляционные), сетевые, иерархические.

Опыт использования баз данных позволяет выделить общий набор их рабочих характеристик:

- полнота – чем полнее база данных, тем вероятнее, что она содержит нужную информацию (однако, не должно быть избыточной информации);
- правильная организация – чем лучше структурирована база данных, тем легче в ней найти необходимые сведения;
- актуальность – любая база данных может быть точной и полной, если она постоянно обновляется, т.е. необходимо, чтобы база данных в каждый момент времени полностью соответствовала состоянию отображаемого ею объекта;
- удобство для использования – база данных должна быть проста и удобна в использовании и иметь развитые методы доступа к любой части информации.

Надо отметить, что база данных – это, собственно, хранилище информации и не более того. Однако, работа с базами данных трудоемкая и утомительная. Для создания, ведения и осуществления возможности коллективного пользования базами данных используются программные средства, называемые системами управления базами данных (СУБД).

Система управления базами данных (СУБД) – это система программного обеспечения, позволяющая обрабатывать обращения к базе данных, поступающие от прикладных программ конечных пользователей. Иными словами, СУБД является интерфейсом между базой данных и прикладными задачами.

Системы управления базами данных позволяют объединять большие объемы информации и обрабатывать их, сортировать, делать выборки по определённым критериям и т.п.

Основные функции СУБД – это:

- определение данных;
- обработка данных;
- управление данными.

Современные СУБД дают возможность включать в них не только текстовую и графическую информацию, но и звуковые фрагменты и даже видеоклипы.

Простота использования СУБД позволяет создавать новые базы данных, не прибегая к программированию, а пользуясь только встроенными функциями.

СУБД обеспечивают правильность, полноту и непротиворечивость данных, а также удобный доступ к ним.

Для менее сложных применений вместо СУБД используются информационно-поисковые системы (ИПС), которые выполняют следующие функции:

- хранение большого объема информации;
- быстрый поиск требуемой информации;
- добавление, удаление и изменение хранимой информации;
- вывод ее в удобном для человека виде.

В информационных системах, которые работают на ПК, совместимых с IBM PC, большое распространение получили так называемые dBASE-подобные системы управления базами данных (СУБД). Известно по крайней мере три семейства таких СУБД (dBASE, FoxPro и Clipper), однако версий оригинальных систем и их адаптированных вариантов гораздо больше. Для пользователей существенным является то, что отличаясь между собой командными языками и форматом индексных файлов, все эти СУБД используют одни и те же оперативные файлы с расширением. DBF, формат которых стал на некоторое время своеобразным стандартом баз данных.

В dBASE-подобных БД фактически использован реляционный подход к организации данных, т.е. каждый файл. DBF представляет собой двумерную таблицу, которая состоит из фиксированного числа столбцов и переменного числа строк (записей). В терминах, принятых в технической документации, каждому столбцу соответствует поле одного из пяти типов (N – числовое. C – символьное, D – дата, L – логическое. M – примечание), а каждой строке – запись фиксированной длины, состоящая из фиксированного числа полей. С помощью командных языков этих СУБД мы создаем и исправляем макеты файлов. DBF (описания таблиц), создаем индексные файлы, пишем

пиктограммы работы с базами данных (чтение, поиск, модификация данных, составление отчетов и многое другое). Характерной особенностью файла DBF является простота и наглядность: физическое представление данных на диске в точности соответствует представлению таблицы на бумаге.

Однако в целом системы, построенные на основе файлов DBF, следует считать устаревшими. Многие механизмы реляционных БД, рассмотренные выше, в dBASE-подобных системах либо не поддерживаются, либо создаются пользователями и программистами «кустарным» способом.

Большую популярность до сего времени имеют и другие СУБД (с другим форматом файлов) – Paradox, Clarion, db_Vista и тд. Следует подчеркнуть, что перечисленные системы ведут родословную от MS-DOS, однако ныне почти все они усовершенствованы и имеют версии для Windows.

Среди современных реляционных систем наиболее популярны СУБД для Windows – Access фирмы Microsoft, Approach фирмы Lotus, Paradox фирмы Borland. Многие из этих систем поддерживают технологию OLE и могут манипулировать не только числовой и текстовой информацией, но и графическими образцами (рисунками, фотографиями) и даже звуковыми фрагментами и видеоклипами.

Перечисленные СУБД часто называют настольными, имея в виду сравнительно небольшой объем данных, обслуживаемых этими системами. Однако с ними часто работают не только индивидуальные пользователи, но и целые коллективы.

Вместе с тем, в центр современной информационной технологии постепенно перемещаются более мощные реляционные СУБД с так называемыми SQL-доступом (SQL – это язык запросов). В основе этих СУБД лежит так называемая технология «клиент-сервис». Среди ведущих производителей таких систем – фирмы Oracle, Centura (Gupta), Sybase, Informix, Microsoft и другие. Появились также объектные и объектно-реляционные СУБД.

В последнее время стали среди СУБД наиболее популярными и используемые в практике Access, Lotus, Oracle.

Разберем наиболее используемую программу Access.

Access – в переводе с английского означает “доступ”. MS Access – это функционально полная реляционная СУБД. Кроме того, MS Access одна из самых мощных, гибких и простых в использовании СУБД. В ней можно создавать большинство приложений, не написав ни единой строки программы, но если нужно создать нечто очень сложное, то на этот случай MS Access предоставляет мощный язык программирования – VisualBasicApplication.

Популярность СУБД Microsoft Access обусловлена следующими причинами:

- Access является одной из самых легкодоступных и понятных систем как для профессионалов, так и для начинающих пользователей, позволяющая быстро освоить основные принципы работы с базами данных;
- система имеет полностью русифицированную версию;
- полная интегрированность с пакетами Microsoft Office: Word, Excel, Power Point, Mail;
- идеология Windows позволяет представлять информацию красочно и наглядно;
- возможность использования OLE технологии, что позволяет установить связь с объектами другого приложения или внедрить какие-либо объекты в базу данных Access;
- технология WYSIWIG позволяет пользователю постоянно видеть все результаты своих действий;
- широко и наглядно представлена справочная система;
- существует набор “мастеров” по разработке объектов, облегчающий создание таблиц, форм и отчетов.

После запуска системы появится главное окно Access (рис. 1). Здесь можно открывать другие окна, каждое из которых по-своему представляет обрабатываемые данные. Ниже приведены основные элементы главного окна Access, о которых необходимо иметь представление.

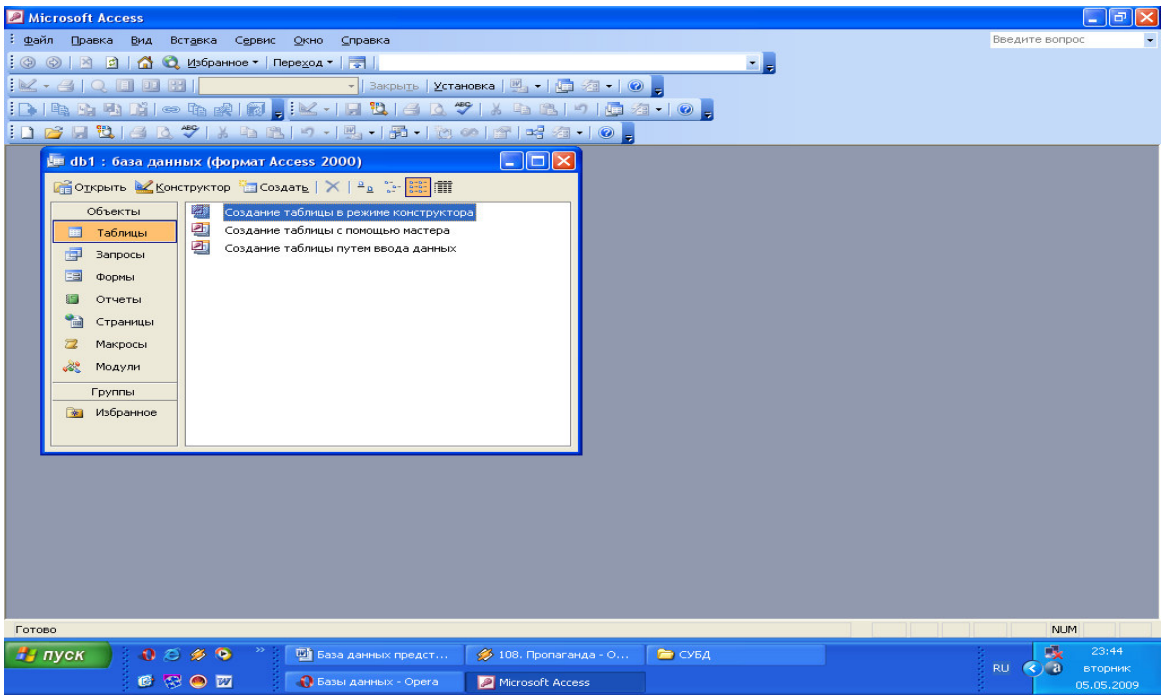


Рис.1. Экран СУБД Access.

В строке заголовка отображается имя активной в данный момент программы. Строка заголовка главного окна Access всегда отображает имя программы MICROSOFT Access.

Пиктограмма системного меню – условная кнопка в верхнем левом углу главного окна практически любого приложения. После щелчка на этой пиктограмме появляется меню, которое позволяет перемещать, разворачивать, сворачивать или закрывать окно текущего приложения и изменять его размеры. При двойном щелчке на пиктограмме системного меню работа приложения завершается.

Панель инструментов – это группа пиктограмм, расположенных непосредственно под полосой меню. Главное ее назначение – ускоренный вызов команд меню. Кнопки панели инструментов тоже могут изменяться в зависимости от выполняемых операций. Можно изменять размер панели инструментов и передвигать ее по экрану. Также можно отобразить, спрятать, создать новую панель инструментов или настроить любую панель инструментов.

Окно базы данных появляется при открытой базе данных. В нем сосредоточены все “рычаги управления” базой данных. Окно базы данных используется для открытия объектов, содержащихся в базе данных, таких как таблицы, запросы, отчеты, формы, макросы и модули. Кроме того, в строке заголовка окна базы данных всегда отображается имя открытой базы данных.

С помощью вкладки объектов можно выбрать тип нужного объекта (таблицу, запрос, отчет, форму, макрос, модуль). Необходимо сказать, что при открытии окна базы данных всегда активизируется вкладка-таблица и выводится список доступных таблиц базы данных. Для выбора вкладки других объектов базы данных нужно щелкнуть по ней мышью.

К основным объектам Access относятся таблицы, запросы, формы, отчеты, макросы и модули.

Таблица – это объект, который определяется и используется для хранения данных. Каждая таблица включает информацию об объекте определенного типа. Как уже известно, таблица содержит поля (столбцы) и записи (строки). Работать с таблицей можно в двух основных режимах: в режиме конструктора и в режиме таблицы.

Запрос – это объект, который позволяет пользователю получить нужные данные из одной или нескольких таблиц. Можно создать запросы на выбор, обновление, удаление или на добавление данных. С помощью запросов можно создавать новые таблицы, используя данные уже существующих одной или нескольких таблиц.

Форма – это объект, в основном, предназначенный для удобного ввода отображения данных. Надо отметить, что в отличие от таблиц, в формах не содержится информации баз данных (как это может показаться на первый взгляд). Форма – это всего лишь формат (бланк) показа данных на экране компьютера. Формы могут строиться только на основе таблиц или запросов. Построение форм на основе запросов позволяет представлять в них информацию из нескольких таблиц.

Отчет – это объект, предназначенный для создания документа, который впоследствии может быть распечатан или включен в документ другого приложения. Отчеты, как и формы, могут создаваться на основе запросов и таблиц, но не позволяют вводить данные.

Макрос – это объект, представляющий собой структурированное описание одного или нескольких действий, которые должен выполнить Access в ответ на определенное событие. Например, можно определить макрос, который в ответ на выбор некоторого элемента в основной форме открывает другую форму. С помощью другого макроса можно осуществлять проверку значения некоторого поля при изменении его содержания. В макрос можно включить дополнительные условия для выполнения или невыполнения тех или иных включенных в него действии.

Работа с формами и отчетами существенно облегчается за счет использования макрокоманд. В MS Access имеется свыше 40 макрокоманд, которые можно включать в макросы. Макрокоманды выполняют такие действия, как открытие таблиц и форм, выполнение запросов, запуск других макросов, выбор опций из меню, изменение размеров открытых окон и т.п. Макрокоманды позволяют нажатием одной (или нескольких одновременно) кнопки выполнять комплекс действий, который часто приходится выполнять в течение работы. С их помощью можно даже осуществлять запуск приложений, поддерживающих динамический обмен данных (DDE), например MS Excel, и производить обмен данными между вашей базой данных и этими приложениями. Один макрос может содержать несколько макрокоманд. Можно также задать условия выполнения отдельных макрокоманд или их набора.

Модуль – объект, содержащий программы на MS Access Basic, которые позволяют разбить процесс на более мелкие действия и обнаружить те ошибки, которые невозможно было бы найти с использованием макросов.

Завершив работу с Access (или с ее приложением), надо корректно закончить сеанс. Простое выключение компьютера - плохой метод, который может привести к возникновению проблем. При работе WINDOWS приложения используют множество файлов, о существовании которых пользователь может даже не подозревать. После

выключения машины эти файлы останутся открытыми, что в будущем может сказаться на надежности файловой системы жесткого диска.

2.14 Лабораторная работа №32,33, 34 (6 часов).

Тема: «Создание локального приложения в СУБД»

2.14.1 Цель работы: создать локальное приложение.

2.14.2 Задачи работы:

1. Определить таблицы
2. Создать приложение.

2.14.3 Описание (ход) работы:

Пусть требуется создать программу, представляющую собой персональный справочник по компьютерным играм. Сейчас информацию о вышедших или разрабатываемых играх можно почерпнуть из самых разных источников: как с многочисленных узлов Интернета, так и из толстых ежемесячных журналов.

Какую информацию должен хранить справочник по играм? Желательно, например, чтобы каждая его запись имела следующие поля:

- название игры;
- жанр;
- удалось ли сыграть;
- фирма-издатель;
- фирма-разработчик;
- год выхода;
- время выхода (сезон);
- возможность сетевой игры;
- ссылка на странички в Интернете;
- ссылка на статьи в журналах;
- оценка в баллах (по 10-балльной шкале);
- оценка графики в баллах;
- оценка звука в баллах;
- комментарии.

Требования к программе и справочнику:

- способность быстро ориентироваться в записанной на диске информации и универсальным методом выделять нужные наборы записей.

- он должен позволять формировать запросы к данным с помощью простых визуальных средств настройки без использования программирования.
- возможность хранить временные наборы записей и формировать на их основе отчеты.
- возможность просмотра, редактирования и удаления записей.
- надежность работы.
- возможность создания *ссылок* на другие записи аналогичной структуры.
- она должна уметь хранить названия фирм (и другие специфические элементы записей) отдельно от основных записей справочника и поддерживать *перекрестные ссылки* между различными группами записей.

База данных Компьютерные игры

FIRMS Справочник фирм разработчиков игр

ID	Ключевое поле
Firmname	Название фирмы

GENRES Справочник Жанр игры

ID	ключевое поле
Genrename	название жанра

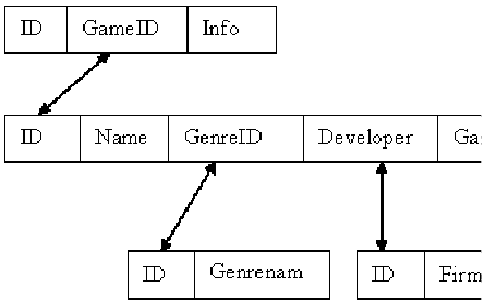
ARTICLES Ссылка на страницы в интернете

ID	Ключевое поле
GameID	Ключ записи из таблицы GEMES
Info	Адрес страницы в интернете

GEMES Основная база

ID	Ключевое поле
Name	Название игры
GenreID	Ключ записи из таблицы GENRES определяющий жанр игры
Developer	Ключ записи из таблицы FIRMS определяющий фирму разработчика
Gameyear	Год выхода игры

Связи между таблицами



Технология BDE для доступа к данным

При создании программ, работающих с базами данных, в системе *Delphi* традиционно используется механизм *BorlandDatabaseEngine (BDE).* В состав *Delphi 7* входит версия*BDE52,* которую, впрочем, можно бесплатно обновлять разными способами (например, обратившись к *Web-узлу* http://www.borland.com/).

Этот механизм реализован в виде набора библиотек, которые обеспечивают для программы, написанной на Паскале, простой и удобный доступ к базам данных независимо от их архитектуры. При использовании механизма *BDE* разработчик может не задумываться о том, как его программа будет работать с базой данных на физическом уровне: локально, в файл-серверной, либо в клиент-сер верно и архитектуре.

Утилиты для работы с СУБД

Создание базы данных Структура базы данных

Теперь, когда мы познакомились с понятием СУБД и принципами работы механизма *BDE,* перейдем к практической реализации справочника игр.

Используемая база данных будет состоять из четырех таблиц. В рамках каждой из них хранятся наборы записей одинаковой структуры.

В первой таблице (назовем ее Firms) будут храниться названия фирм, во второй - жанры (Genres), в третьей — ссылки на Wefc-страницы Интернета и статьи в журналах (Articles). Одной игре может быть посвящено несколько Web-страниц и статей.Четвертая таблица (Games) будет основной.

Создание таблиц

Для создания таблиц в системе *Delphi* 7 имеется приложение DatabaseDesktop (Работа с автономной СУБД). Оно вызывается командой Tools>DatabaseDesktop (Сервис >Работа с автономной СУБД) .

Новая таблица создается командой File>New > Table (Файл>• Создать > Таблица).

В открывшемся диалоговом окне надо выбрать формат таблицы (каждая СУБД хранит данные в собственном формате). Укажите в раскрывающемся списке Tabletype

(Формат таблицы) пункт Paradox 7 (формат СУБД *Paradox* версии 7) и щелкните на кнопке ОК.

Ключевое поле

Как правило, каждая таблица базы данных включает *ключевое* поле. Оно содержит уникальный идентификатор, позволяющий отличить одну запись от другой. Например, каждая фирма и каждая игра могут (и должны) иметь такой неповторимый идентификатор (или *ключ).* В качестве ключей обычно используется цифровое значение.

Во-первых, ключи нужны для повышения эффективности доступа к данным внутри таблицы (это связано с технологией работы реляционных СУБД), а во-вторых, они используются для создания перекрестных ссылок между таблицами. В нашем случае, чтобы сослаться из записи, описывающей игру, на запись таблицы, хранящей название фирмы-производителя, достаточно указать ключ этой записи.

В качестве ключа может выступать, конечно, и название фирмы, но оно будет занимать много места (для ключа-числа достаточно нескольких байтов), что понизит эффективностьработы СУБД. Кроме того, если название изменится, придется корректировать все записи таблицы Games, где упоминается эта фирма, что совсем неудобно. При использовании цифрового ключа достаточно внести изменение только в одну запись таблицы Firms,

Индексация

С понятием ключа тесно связано понятие *индекса.* Если таблица предназначена для выдачи наборов данных на основании всевозможных запросов («отобрать все игры, вышедшие в указанный период», «посмотреть все ролевые игры, в которые я еще не играл»), то такую таблицу желательно проиндексировать по каждому полю, для которого ожидается активное использование в запросах.

Индексы содержат краткую информацию о каждой записи и организованы таким образом, что позволяют очень быстро получать доступ к нужной информации без сканирования всей таблицы.

В СУБД *Paradox 7* ключевое поле всегда индексировано. Такой индекс называется *первичным* и всегда один. Для увеличения скорости сортировки и поиска данных в других полях можно добавить неограниченное число *вторичных* индексов.

Создание таблицы Genres

Проектирование баз данных — это искусство не менее сложное, чем искусство проектирования программ. В современных коммерческих приложениях базы данных состоят из тысяч таблиц, каждая из которых насчитывает сотни полей. Спроектировать

все так, чтобы обеспечивалась эффективная работа при выполнении разных запросов, очень сложно.

Ключевое поле, как правило, указывается первым. Некоторые СУБД требуют этого в обязательном порядке. Например, таков используемый сейчас формат СУБД *Paradox 7.*Отступать от этого порядка не рекомендуется. Название ключевого поля обычно тоже стандартно — ID (от английского *Identifier — идентификатор).*

Этот процесс для таблиц с ключами настолько типичен, что он был автоматизирован.

Щелкните правой кнопкой мыши над областью поля Type (Тип поля) и выберите в контекстном меню значение Autoincrement (Автоприращение). Теперь при добавлении в таблицу новой записи значение данного поля будет автоматически увеличиваться и разработчику не придется отслеживать уникальность первичных ключей.

Пункт столбца Size (размер поля записи) заполняется автоматически, а вот в столбце Key надо указать, что данное поле записи ключевое. Нажатие клавиши ПРОБЕЛ приведет к появлению в этом столбце звездочки — признака ключевого поля. Убрать этот признак можно повторным нажатием той же клавиши.

Тип Alpha- хранит текстовые значения; флажок RequiredField (Обязательное поле)

Создание вторичных ключей

В раскрывающейся список Tableproperties (Свойства таблицы). Выберем строку SecondaryIndexes (Вторичные ключи) и щелкнем на кнопке Define (Определить).

В появившемся диалоговом окне переместим с помощью кнопки со стрелкой поле GenreName в список Indexedfields (Индексированные поля). Поле ID уже используется в качестве первичного индекса. Укажем также, что все значения в поле GenreName уникальные, установив флажок Unique (Уникальное).

По щелчку на кнопке ОК приложение попросит ввести имя для вторичного индекса . Необходимо указать оригинальное имя, не совпадающее с именами полей, например GenrelD.

Перекрестные ссылки

Важный заключительный этап формирования базы данных состоит в создании перекрестных ссылок между таблицами. На этом этапе надо явно указать, например,что ключевое поле таблицы Genres связано с полем GenrelD таблицы Games. Это выполняется так: в раскрывающемся списке TableProperties выбирается пункт ReferentialIntegrity (Целостность ссылок). С помощью кнопки Define (Определить) определяются все связи данной таблицы с ключевыми полями других таблиц.

По щелчку на этой кнопке открывается диалоговое окно, в левой части которого указан список всех полей таблицы, а в правой — другие таблицы в рабочем каталоге.

Зададим связь поля GenrelD с ключевым полем таблицы Genres, которая считается родительской. Выберите поле в левом списке и щелкните на кнопке со стрелкой вправо. В центральной области окна в колонке Chieldfield (Подчиненное поле) появляется название GenrelD [I]. Обозначение [I] указывает на тип этого поля.

В правом списке надо выбрать таблицу Genres.db и щелкнуть на кнопке со стрелкой влево. В столбце Parent'skey (Ключевое поле родительской таблицы) появится надпись ID [+]. Знак «плюс» означает признак ключевого поля (рис. 5.5).

После этого можно щелкнуть на кнопке ОК и в небольшом диалоговом окне ввести произвольное название созданной связи, предположим Genrelnt. После этого снова щелкните на кнопке ОК. В списке ниже кнопки Define (Определить) возникнет новая связь Genrelnt. Таким же способом надо создать связи Devlnt (поле Developer и ключ таблицы Firms) и Publnt (поле Publisher ц ключ таблицы Firms).

Отношения между таблицами

Среди проектировщиков баз данных принята определенная терминология, которую полезно знать и прикладным программистам. Отношения между записями в таблицах делятся на три типа.

Тип отношения **Способ связи Один и одному** Каждой записи соответствует роено одна запись другой таблицы. В принципе, можно вообще обойтись одной таблицей

Один ко многим Примером может служить связь таблиц Games и Genres. В таблице Genres хранится только одна запись, описывающая конкретный жанр, а в таблице Games записей, ссылающихся на этот жанр, может быть много.

Многие ко многим Это отношение не всегда поддерживается в реляционных СУБД, и для его реализации часто вводят промежуточные таблицы. Например, таблица Articles нужна именно для таких целей. Она способна хранить ключи, позволяющие описать такие отношения, когда одна игра обсуждается на нескольких узлах (в нескольких статьях) и, наоборот, один узел посвящен нескольким играм.

Добавление базы данных в BDE

База данных создана в виде набора четырех таблиц, но система *BDE* пока не знает об их существовании. Хотя работать с этими таблицами в принципе можно, указывая полные пути поиска в свойствах соответствующих компонентов, такой подход неправилен. Он не позволяет работать с базой данных как целостным понятием и напрямую обращаться к ней по имени (псевдониму).

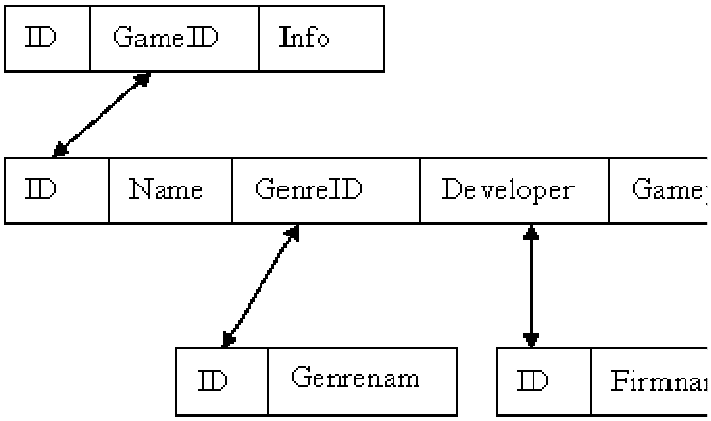
При регистрации в системе Воссозданной группы таблиц как целостной базы данных поможет приложение SQL Explorer (Проводник SQL), запускаемое командой Database>>Explore (База данных > Проводник). В левой части окна приводится список всех зарегистрированных в системе *BDE баз* данных, вправой — свойства текущей базы, выбранной в этом списке.

Создадим новую базу данных. Для этого выполняется команда Object>• New (Объект > Создать) и в диалоговом окне выбора драйвера указывается значение STANDARD.

После щелчка на кнопке ОК в списке появится новый элемент, помеченный зеленым треугольником. Это означает, что регистрация базы данных не завершена. По умолчанию формируется имя базы STANOARD1, изменим его на MyBase. Убедимся, что в свойстве DEFAULT DRIVER (Драйвер по умолчанию) стоит значение PARADOX. В свойстве PATH (Путь поиска для каталога, в котором хранятся таблицы) укажем рабочий каталог (для которого создан псевдоним:WORK:). Этот каталог, как правило, отображается первым при щелчке на кнопке выбора.

Теперь зарегистрированную в системе *BOE* базу надо сохранить. Для этого в контекстном меню объекта MyBase выберем пункт Apply (Применить настройки). На вопрос о необходимости сохранения изменений ответим Yes (Да). Теперь наши таблицы доступны из среды *BDE* под именем базы данных MyBase. Если раскрыть объект MyBase, щелкнув на значке «+»перед его именем, па правой панели SQL Explorer будут показаны все четыре таблицы, а значок базы помечается зеленой рамкой, указывающей, что база данных MyBase открыта.

e:EN-US'>GameID
Info
ID
Name
GenreID
Developer
Gameyear


ID
GameID
Info
ID
Name
GenreID
Developer
ID
Genrenam
ID