ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»
Кафедра «Автоматизированные системы обработки информации и управления**»**

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Б1.В.ДВ.08.01 СУБД и базы данных

Направление подготовки (специальность) 27.03.04 У правление в технических системах

Профиль образовательной программы Интеллектуальные системы обработки информации и управления

Квалификация выпускника бакалавр

Форма обучения очная

СОДЕРЖАНИЕ

1. КОНСПЕКТ ЛЕКЦИЙ

1. 1 Лекция № 1(2 часа).

Тема: «Введение в базы данных. Обзор современных систем управления базами данных**»**

1.1.1 Вопросы лекции:

1. Понятие, назначение баз данных.
2. Основные компоненты баз данных.
3. Современные системы управления базами 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 и формы .**

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.

3. 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 (диалект) объектно-ориентированного языка программирования VisualBasic, встраиваемая в программные пакеты.

1.2 Лекция № 2 (2 часа).

Тема: «Реляционная модель данных»

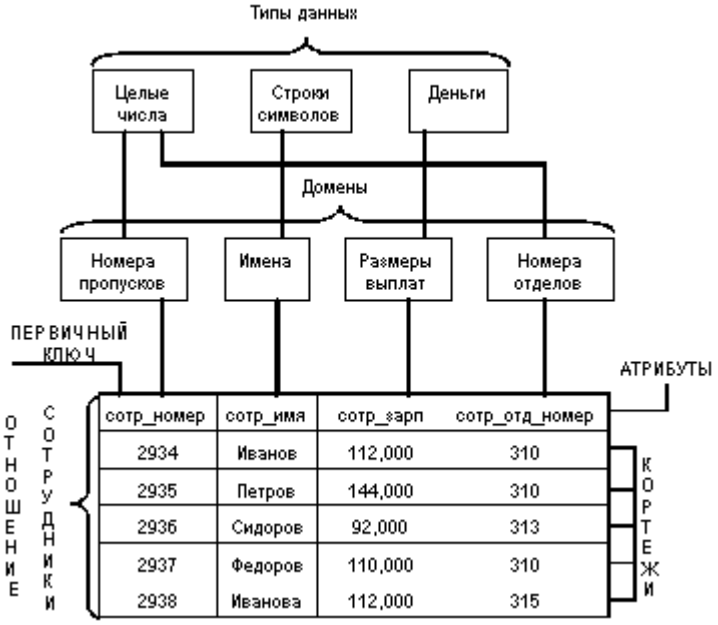
1.2.1 Вопросы лекции:

1. Понятие домена, атрибута, кортежа, отношения.
2. Табличное представление отношения.
3. Схема отношения.
4. Первичные и внешние ключи.

1.2.2 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.

Атрибутесть 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 созданиявнешних 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 именам, 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 *родительскимотношением*, 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.

1.3 Лекция №3 (2 часа).

Тема: «Проектирование концептуальной модели данных»

1.3.1 Вопросы лекции:

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

1.3.2 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 отношенный

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,[2] 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).

Аномалиейназывается 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.

2. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ПРОВЕДЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

2.1 Практическое занятие №1 (2 часа).

Тема: «Модели данных»

2.2.1 Задачи для работы:

1.Сетевая модель данных

2.Реляционная модель данных

3.1.2 Краткое описание проводимого занятия:

1.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.

2.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.

2.1.3 Результаты и выводы: в результате выполнения практической работы были изучены основные модели данных.

2.2 Лабораторная работа №2 (2 часа).

Тема: «Реляционная модель данных»

2.2.1 Задание для работы:

1.Понятие домена, атрибута, кортежа, отношения

2.2.2 Краткое описание проводимого занятия:

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

2.2.3 Результаты и выводы: в ходе выполнения работы изучены понятие домена, атрибута, кортежа, отношения.

2.3Лабораторная работа №3 (2 часа).

Тема: «Реляционная алгебра и язык SQL»

2.3.1 Задание для работы:

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

2.3.2 Краткое описание проводимого занятия:

Язык для взаимодействия с БД SQL появился в середине 70-х и был разработан в рамках проекта экспериментальной реляционной СУБД System R. Исходное название языка SEQUEL (StructeredEnglishQueryLanguage) только частично отражает суть этого языка. Конечно, язык был ориентирован главным образом на удобную и понятную пользователям формулировку запросов к реляционной БД, но на самом деле уже являлся полным языком БД, содержащим помимо операторов формулирования запросов и манипулирования БД средства определения и манипулирования схемой БД; определения ограничений целостности и триггеров; представлений БД; возможности определения структур физического уровня, поддерживающих эффективное выполнение запросов; авторизации доступа к отношениям и их полям; точек сохранения транзакции и откатов. В языке отсутствовали средства синхронизации доступа к объектам БД со стороны параллельно выполняемых транзакций: с самого начала предполагалось, что необходимую синхронизацию неявно выполняет СУБД.

Рассмотрим эти свойства языка немного более подробно.

13.1.1. Запросы и операторы манипулирования данными

Как известно, 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.

13.1.2. Операторы определения и манипулирования схемой БД

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.

13.1.3. Определения ограничений целостности и триггеров

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.

Более точно, ограничения целостности делились на два класса: проверяемые после выполнения оператора манипулирования данными и проверяемые при завершении транзакции или при выполнении специального оператора INFORCE INTEGRITY. Типы предикатов, которые можно использовать в операторах определения ограничений целостности разных классов, различаются. В операторах первого класса проверяется, фактически, текущий кортеж, с которым производится манипулирование. Во втором случае проверяются указанные в ограничении целостности отношения, т.е. все их

кортежи. Различается и определяемая в языке реакция системы на нарушения ограничений целостности разных классов. В первом случае нарушение ограничения целостности приводит к откату транзакции в точку, непосредственно предшествующую операции манипулирования данными, выполнение которого вызвало нарушение ограничения целостности. Во втором случае ограничение приводит к полному откату транзакции к ее началу.

Очень важным механизмом, определенным в языке SQL System R, является механизм триггеров. В контексте System R этот механизм рассматривался главным образом как средство автоматического поддержания целостности БД. При определении триггера указывалось условие проверки его применимости (имя отношения и тип операции манипулирования данными), условие применимости триггера (логическое выражение, построенное по правилам, близким к правилам для ограничений целостности первого класса) и действие, которое должно быть выполнено над БД в случае истинности условия применимости. Такое действие могло быть выражено с помощью произвольного оператора манипулирования данными. Во время выполнения действия могли срабатывать другие триггеры и т.д.

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.

13.1.4. Представления базы данных

В языке допускалось использование хранимых отношений БД и представляемых отношений. Наиболее удачным решением было использование для определения представлений общего аппарата операторов выборки. Любой оператор выборки может быть использован для определения представления.

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.

Массу проблем, исследований и предложений породила потенциальная возможность выполнения операторов манипулирования данными над представлениями. Понятно, что эта возможность легко реализуема для простых представлений, но в более сложных случаях не только реализация, но и семантика операций становится нетривиальной. Кстати, в System R операторы манипулирования данными допускались только над простыми представлениями.

13.1.5. Определение управляющих структур

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.

13.1.6. Авторизация доступа к отношениям и их полям

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.

13.1.7. Точки сохранения и откаты транзакции

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.

13.1.8. Встроенный SQL

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.

13.1.9. Динамический SQL

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.

2.3.3 Результаты и выводы: в ходе выполнения работы изучена реляционная алгебра и язык SQL.

2.4 Лабораторная работа №4 (2 часа).

Тема: «Администрирование базы данных»

2.4.1 Задание для работы:

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

2.4.2 Краткое описание проводимого занятия:

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.Последовательность действий жизненного цикла:

1) Анализ предметной области (что это будет за база);

2) Анализ запросов (те требования и типичные вопросы, которые могут возникнуть у пользователя - те поля, которые надо предусмотреть);

3) Интеграция ваших представлений (формализованное описание ваших представлений об этой предметной области - это техническое задание);

4) Выбор средств реализации (объем физической памяти, финансовые затраты и т. д.);

5) Физическое проектирование (создание оболочки - программист пишет программы или уже сам);

6) Непосредственное создание базы данных (интеграция в нее данных - на этом этапе трестируется работоспособность базы данных);

7) Эксплуатация базы данных (у нас в процессе эксплуатации возникает не только необходимость пользоваться, возможно реорганизация, реструктуризация и реформирование базы данных).

2.4.3 Результаты и выводы: в ходе выполнения работы изучены функцияи администрирования базы данных.

2.5 Лабораторная работа №5 (2 часа).

Тема: «Словарь данных»

2.5.1 Задание для работы:

1. **Словарь данных**

2.5.2 Краткое описание проводимого занятия:

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

2.5.3 Результаты и выводы: в ходе выполнения работы получен опыт работы со словарем данных.

2.6 Лабораторная работа №6 (2 часа).

Тема: «Создание локального приложения в СУБД»

2.6.1 Задание для работы:

1. **Создание локального приложения в СУБД**

2.6.2 Краткое описание проводимого занятия:

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.

Поэтому, хотя и допустимо хранить фиксированный набор названий жанров, правильнее поступить с ними по аналогии с названиями фирм: выделить в отдельную группу, а в основных записях помещать только ссылки.

База данных Компьютерные игры

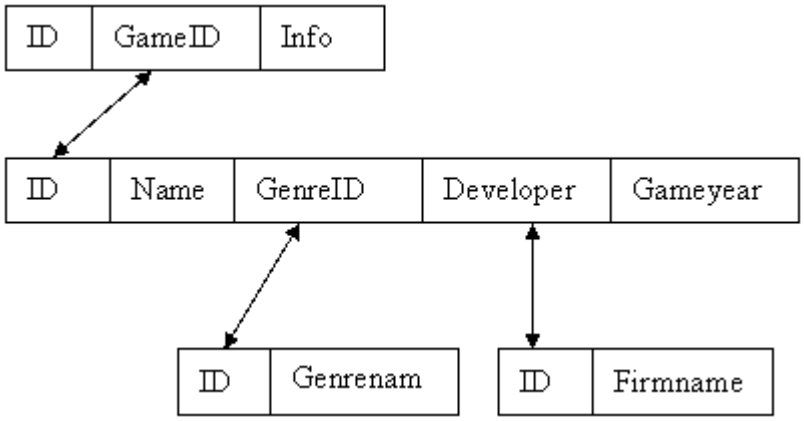
FIRMS Справочник фирм разработчиков игр
ID Ключевое поле
Firmname Название фирмы

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



Понятие базы данных

База данных — это совокупность записей различного типа, содержащая перекрестные ссылки.

В частности, структура записей описания игр будет отличаться от структуры записей описания фирм и записей описания жанров. Записи одного типа внутри базы данных хранятся в*таблицах.* Для фирм используется одна таблица, для игр — другая.

Между записями каждой из таблиц устанавливаются ссылки: в записи таблицы игр имеется ссылка на записи таблицы с названиями фирм, таблицы с названиями жанров и так далее.

Поэтому при работе с базами данных обычно применяются понятия более высокого логического уровня: *запись* и *таблица,* без углубления в подробности их физической структуры.

Таким образом, сама по себе база данных — это только набор таблиц с перекрестными ссылками. Чтобы универсальным способом извлекать из нее группы записей, обрабатывать их, изменять и удалять, требуются специальные программы, которые называются *системами управления базами данных* или сокращенно СУБД.

Среди приведенных нами требований к будущему справочнику можно выделить как требования к обрабатываемой базе данных, так и требовании к программе как системе управления этими данными.

Два последних требования (наличие перекрестных ссылок внутри одной таблицы и между таблицами) относятся к базе данных, остальные (формирование и выполнение запросов, создание отчетов, обеспечение надежности хранения и целостности данных) — к СУБД и различным вспомогательным программам.

Модели баз данных

Таблицы, в которых хранятся данные, состоят из наборов записей одинаковой структуры. Можно сказать, что таблица — это двумерный массив, где строки образованы отдельными записями, а столбцы — полями этой записи. Более точно таблица представляется как одномерный массив переменной длины из записей конкретной структуры (тип **record** Паскаля).

Модель базы данных, состоящей из подобных таблиц, называется *реляционной.*

Практически все ведущие производители СУБД поддерживают именно эту модель баз данных, и в книге в дальнейшем будет рассказываться именно о ней. Реляционная

модель хороша тем, что проста в работе и реализации и позволяет создавать быстро работающие системы.

Имеется еще несколько моделей баз данных. Некоторые из них значительно эффективнее реляционной, но не получили широкого распространения из-за сложности создания подходящих СУБД.

- В *иерархической* модели данные организованы в виде деревьев.
- В *сетевой* модели каждый узел (набор) базы данных взаимодействует с другими узлами посредством сложной системы связей.
- В последнее время признание завоевывает *объектная* модель данных, когда в базе хранятся не только данные, но и методы их обработки в виде программного кода. Это перспективное направление, пока также не получившее активного распространения из-за сложности создания и применения подобных СУБД.

Архитектура СУБД

Приложения, использующие базы данных, обычно принято относить к одной из программных архитектур, имеющих свои плюсы и минусы.

Локальная архитектура

И программа, и база данных расположены на одном компьютере. В такой архитектуре работает большинство настольных приложений.

Файл-серверная архитектура

База данных расположена на мощном выделенном компьютере (сервере), а персональные компьютеры подключены к нему по локальной сети. На этих компьютерах установлены клиентские программы, обращающиеся к базе данных по сети. Преимущество такой архитектуры заключается в возможности одновременной работы нескольких пользователей с одной базой данных.

Недостаток такого подхода — большие объемы информации, передаваемой по сети.Вся обработка выполняется на клиентских местах, где фактически формируется копия базы данных. Это приводит к ограничению максимально возможного числа пользователей и большим задержкам при работе с базой. Эти задержки вызываются тем, что на уровне конкретной таблицы одновременный доступ невозможен. Пока программа на одном из клиентских мест не закончит работу с таблицей (напри-

мер, не выполнит модификацию записей), другие программы не могут обращаться к этой таблице. Это называется *блокировкой на уровне таблицы* и исключает возникновение путаницы в ее содержимом.

Клиент-серверная архитектура

В такой архитектуре на сервере не только хранится база данных, но и работает программа СУБД, обрабатывающая запросы пользователей и возвращающая им наборы записей. При этом программы пользователей уже не работают напрямую с базой данных как набором физических файлов, а обращаются к СУБД, которая выполняет операции. Нагрузка с клиентских мест при этом снимается, так как большая часть работы происходит на сервере. СУБД автоматически следит за целостностью и сохранностью базы данных, а также контролирует доступ к информации с помощью службы паролей. Клиент-серверные СУБД допускают блокировку на уровне *записи* и даже отдельного *поля.* Это означает, что с таблицей одновременно может работать любое число пользователей, но доступ к функции изменения конкретной записи или одного из ее полей обеспечен только одному из них.

Основной недостаток этой архитектуры — не очень высокая надежность. Если сервер выходит из строя, вся работа останавливается.

Распределенная архитектура

В сети работает несколько серверов, и таблицы баз данных распределены между ними для достижения повышенной эффективности. На каждом сервере функционирует своя копия СУБД. Кроме того, в подобной архитектуре обычно используются специальные программы, так называемые *серверы приложений.* Они позволяют оптимизировать обработку запросов большого числа пользователей и равномерно распределить нагрузку между компьютерами в сети. Если, помимо работы с данными, требуется выполнить интенсивные вычисления (например, анализ сложной информации), программы для выполнения этих задач (они обычно называются *компонентами)* могут автоматически запускаться на более мощных сетевых компьютерах. Это практически полностью снимает нагрузку с клиентских мест.Такая архитектура также называется *компонентной.*

Недостаток распределенной архитектуры заключается в довольно сложном и дорогостоящем процессе ее создания и сопровождения (администрирования), а также в высоких требованиях к серверным компьютерам.

Интернет-архитектура

Доступ к базе данных и СУБД (расположенным на одном компьютере или в сети) осуществляется из браузера по стандартному протоколу. Это предъявляет минимальные требования к клиентскому оборудованию. Такие программы называют «тонкими клиентами», потому что они способны работать даже на ПК с процессором 80386.

Благородя стандартизации всех протоколов и интерфейсов взаимодействия в Интернете такие системы легко создавать и внедрять. Например, можно не организовывать локальную сеть, а обращаться к серверу через Интернет или использовать протоколы Интернета в локальной сети (в таком случае говорят о технологиях интранет). В этом случае не требуется разрабатывать специальные клиентские программы или придумывать собственные спецификации обмена данными между сервером и клиентскими местами. Достаточно использовать готовые браузеры и

Программные решения.

Реализация работы с СУБД в системе Delphi

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,

ВНИМАНИЕ Если в таблице имеется ключевое поле, это означает, что все значения в нем уникальны и неповторимы. Содержимое ключевого поля в одной таблице повторяться не может. То есть, в таблице Gomes не может быть двух описаний (записей), посвященных одной игре,

а в таблице Firms не может быть двух записей, посвященных одной фирме.

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

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).

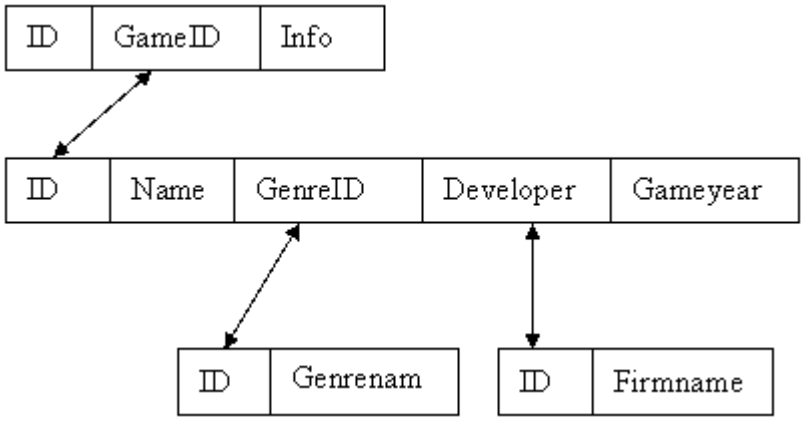
При регистрации в системе Воссозданной группы таблиц как целостной базы данных поможет приложение SQL Explorer (Проводник SQL), запускаемое командой Database>>Explore (База данных > Проводник). В левой части окна приводится список всех зарегистрированных в системе *BDE баз* данных, вправой — свойства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



2.6.3 Результаты и выводы: в ходу выполнения работы получен навык создания локального приложения в СУБД.