Министерство образования Российской Федерации
Владимирский государственный университет
Кафедра информационных систем
и информационного менеджмента

УПРАВЛЕНИЕ ДАННЫМИ

Методические указания к лабораторным работам

Составитель
В.В. ВЕРШИНИН

Владимир 2004

ЭЛЕКТРОННАЯ ВЕРСИЯ
ЭЛЕКТРОННАЯ ВЕРСИЯ

УДК 681.32

Рецензент
Кандидат технических наук,
доцент Владимирского государственного университета
С.Г. Мосин

Печатается по решению редакционно-издательского совета
Владимирского государственного университета

Управление данными. Метод. указания к лаб. работам / Владим. гос. ун-т; Сост.: В.В. Вершинин. Владимир, 2004. 40 с.

Содержат описание и задания к лабораторным работам по курсу «Управление данными». Включают работы по изучению и освоению стандартных средств для работы с базами данных, используя стандарт языка SQL на примере системы управления базами данных InterBase, разработанной фирмой Borland International. Также рассматриваются визуальные CASE-средства для моделирования структуры баз данных.

Каждая работа содержит предварительный теоретический материал, который может быть использован при подготовке, выполнении и защите лабораторных работ. Индивидуальные задания к лабораторным работам предлагается выполнять последовательно, начиная с первой. Это позволяет получить конкретные знания о предметной области и изучаемом материале, а также приобрести практические навыки по созданию, хранению и доступу к информации, хранящейся в базах данных.

Предназначены для студентов специальностей 071900, 073700, 220100 и других, изучающих дисциплины соответствующего профиля.

Табл. 6. Рис. 2. Библиогр.: 5 назв.

УДК 681.32

ЭЛЕКТРОННАЯ ВЕРСИЯ
ЭЛЕКТРОННАЯ ВЕРСИЯ

ПРЕДИСЛОВИЕ

Объемы информации, с которыми приходится работать в современном мире, имеют тенденцию к постоянному увеличению и учету новых сведений. В связи с этим возникает задача адекватного представления этой информации в хранилищах данных. Также немаловажное значение играют другие вопросы, связанные с удобством и быстротой доступа, поиска и извлечения нужных данных из их общего объема.

Выбор структур данных, рациональное применение технических и аппаратных средств совместно с языковыми средствами доступа к данным способствуют успешной и эффективной работе.

Предлагаемый в данных методических указаниях к лабораторным работам материал по курсу «Управление данными» ставит целью познакомить студентов с различными аспектами создания, изменения и манипуляции данными при построении приложений, ориентированных на работу с базами данных. В качестве языка манипулирования данными выбран язык SQL.

Первая лабораторная работа посвящена изучению синтаксических конструкций языка SQL, предназначенных для создания и модификации структуры таблиц и баз данных. Здесь же рассматриваются команды для непосредственного добавления, изменения и удаления данных в таблицах. Вторая работа посвящена изучению методологии «сущность-связь». В качестве программной поддержки при изучении методологии выбрано CASE–средство ERWin, которое позволяет наиболее полно изучить рассматриваемую методологию моделирования структуры баз данных. Третья работа посвящена изучению средствам языка SQL при выборке данных из таблиц. Четвертая и пятая работы знакомят студентов с такими объектами СУБД, как представления, хранимые процедуры и генераторы. Рассматриваются средства языка SQL, позволяющие манипулировать такими объектами и использовать их при решении различных задач по управлению данными.

ЭЛЕКТРОННАЯ ВЕРСИЯ
ЭЛЕКТРОННАЯ ВЕРСИЯ

Лабораторная работа № 1

ЗНАКОМСТВО С СУБД INTERBASE.
ВЫПОЛНЕНИЕ ЭЛЕМЕНТАРНЫХ ОПЕРАЦИЙ ПРИ РАБОТЕ С СУБД

1. Цель работы

Изучить технические возможности системы управления базами данных (СУБД) InterBase при работе с простейшими базами данных (БД) и средства формирования SQL запросов к СУБД.

2. Общие сведения

Под базой данных понимается некоторая унифицированная структура данных, совместно используемая одним или несколькими пользователями и хранящая информацию рассматриваемой предметной области. Задача базы данных состоит в хранении всех представляющих интерес данных в одном или нескольких местах, причем таким способом, который заведомо исключает ненужную избыточность. В хорошо спроектированной базе данных избыточность данных исключается и вероятность сохранения противоречивых данных минимизируется. Таким образом, создание баз данных преследует две основные цели: понизить избыточность данных и повысить их надежность.

2.1. Основные функции InterBase

Языковые средства манипуляции данными позволяют манипулировать данными, осуществляя их выборку из таблиц. Логика и правила работы самого приложения с полученными данными описываются уже языками высокого уровня. Языком выборки данных из таблиц является SQL, правила которого определяются стандартом SQL-92. Язык SQL, реализованный в различных СУБД допускает отклонение от этого стандарта.

СУБД InterBase подчиняется правилам стандарта SQL-92. Это позволяет обеспечивать целостность ссылок на уровне связи с возможностью каскадирования операций, обновляемость представлений и внешние соединения. СУБД InterBase Server предоставляет набор библиотек, которые содержат функции InterBase API для поддержания работы и взаимодействия клиентских приложений и СУБД посредством SQL. InterBase поддерживает и предоставляет широкий набор функций, которые позволяют строить клиентские приложения различной сложности (в том числе и сетевые), в которых выполняется хранение и обработка данных с использова-

ЭЛЕКТРОННАЯ ВЕРСИЯ

нием SQL и СУБД. Ключевые функции InterBase представлены в табл. 1.

Таблица 1

Основные функции InterBase

Функции СУБД	Описание
Поддержка сетевых протоколов	На всех платформах InterBase поддерживает NCP/IP; для Windows NT-каналы NetBEUI; для NetWar - IPX/SPX
Соответствие минимальной конфигурации SQL-92	ANSI стандарт SQL доступен через Interactive SQL tool или клиентские приложения
Доступ к базам данных	Одно приложение может обращаться к нескольким базам данных одновременно
Минимальный уровень блокировки строк	Сервер блокирует только те записи, которые клиент модифицирует вместо блокировки полной страницы базы данных
BLOB-данные и фильтры BLOB	Динамически изменяемый размер типа данных позволяет хранить неформатированные данные, такие как графика и текст
Декларативная ссылочная целостность	Автоматическая поддержка логических связей между таблицами по внешним (FOREIGN) и первичным (PRIMARY) ключам
Хранимые процедуры	Программы, хранимые элементы в базе данных для расширения возможностей запросов поиска и изменения данных
Триггеры	Программы, которые запускаются, когда в связанных с ними таблицах добавляются, модифицируются или удаляются данные
Индикация событий	Выдача сообщений приложению от СУБД. Дает возможность клиентским приложениям получать асинхронные уведомления об изменениях в базе данных
Обновляемые представления	Представления, виртуальные (таблицы), которые могут отражать изменения данных сразу, как только изменения происходят

InterBase также поддерживает особенности расширенного SQL, некоторые из них представлены в расширении SQL стандарта SQL3. Сюда входят хранимые процедуры, триггеры и прочие синтаксические конструкции, расширяющие функциональные возможности СУБД при работе с данными. InterBase разрешает одновременное обращение многих клиентских приложений к одной базе данных. Клиентское приложение может также обращаться к нескольким базам данных одновременно. Основные технические характеристики СУБД InterBase представлены в табл. 2. Указанные в таблице характеристики и ограничения следует учитывать при

ЭЛЕКТРОННАЯ ВЕРСИЯ

проектировании и разработке пользовательских приложений. Следует заметить, что существуют реализации этой СУБД под различные платформы.

Таблица 2

Спецификации InterBase

Характеристики	Описание
Максимальное число клиентов, подсоединенных к одному серверу	Это число определяется комбинацией таких факторов как производительность ОС, аппаратные ограничения и потребности отдельного клиента на сервере. Удобная работа с InterBase Server может быть предоставлена 150 клиентам. Однако это не гарантируется. Поскольку некоторые ОС могут не поддерживать доступ 150 сетевых соединений
Максимальный размер БД	Максимальный размер одного файла составляет 2 Гб для Windows 95, 4 Гб - Windows NT, Unix. Размер также определяется возможностям ОС
Максимальный размер открытых одновременно БД	Без ограничений, определяется системными ресурсами
Максимальное количество таблиц в БД	2^{16} (65 356), т.к. таблицы нумеруются 16-битовым целочисленным представлением
Максимальное количество столбцов (атрибутов) в таблице	2^{32}, так как столбцы нумеруются целочисленным 32-битовым представлением

2.2. Порядок создания и запуска сервера

После завершения инсталляции пакета InterBase, сервер базы данных запускается по умолчанию при каждой загрузке ОС. Параметры запуска сервера могут быть изменены при помощи утилиты InterBase Configuration. Возможны следующие параметры запуска InterBase-сервера:

Windows Startup – автоматический запуск сервера вместе с загрузкой операционной системы;

Manual Startup – запуск InterBase сервера вручную либо с использованием панели меню, либо запуск командного файла regcfg.exe из директории, куда установлен сервер;

Disabled – после запуска InterBase-сервера на панели задач автоматически появляется иконка. Если указать данный параметр, иконка отображаться не будет.

Основной утилитой для доступа к базам данных и администрирования самой СУБД InterBase является IBConsole, которую можно найти в меню «Пуск→Программы→InterBase→IBConsole». Для создания сервера необходимо выбрать пункт Server→Register… Если создается локальный

ЭЛЕКТРОННАЯ ВЕРСИЯ

сервер, т.е. сервер подключение к которому будет происходить с той же машины (локальной), на которой он установлен, то необходимо сделать активной опцию Local Server. По умолчанию параметры доступа к серверу следующие: **Login**: *sysdba*; **Password**: *masterkey*.

В случае подключения к серверу, запущенному на другой машине, необходимо выбрать опцию Remote Server, в поле Server указать IP-адрес или сетевое имя сервера (компьютера). В поле Network Protocol выбрать протокол (обычно это TCP/IP), ввести регистрационное имя и пароль указанные ранее. Чтобы завершить работу с сервером необходимо в IBConsole выбрать пункт Server→Logout.

2.3. Основные понятия БД

В реляционной теории одним из главных понятий является отношение. Математически отношение определяется следующим образом. Пусть даны *n* множеств D_1, D_2, ..., D_n. Тогда *R* есть **отношение** над этими множествами, если *R* есть множество упорядоченных наборов вида $<d_1, d_2, ..., d_n>$, где: d_1 – элемент из D_1; d_2 – элемент из D_2; ..., d_n – элемент из D_n. При этом наборы вида $<d_1, d_2, ..., d_n>$ называются **кортежами**, а множества D_1, D_2, ..., D_n – **доменами**. Каждый кортеж состоит из элементов, выбираемых из своих доменов. Эти элементы называются атрибутами, а их значения – значениями атрибутов.

Поскольку в базах данных разнотипная информация может располагаться в отдельных файлах, то с точки зрения размещения данных отношение можно отождествлять с таблицей или, иногда, с файлом. Кортеж представляет собой строку в таблице, или, что то же самое, запись. Атрибут же является столбцом таблицы, или полем в записи. Домен же представляется неким обобщенным типом, который может быть источником для типов полей в записи. Таким образом, следующие тройки терминов являются эквивалентными:

- отношение, таблица, файл (для локальных баз данных);
- кортеж, строка, запись;
- атрибут, столбец, поле.

Реляционная база данных представляет собой совокупность отношений, содержащих всю необходимую информацию и объединенных различными связями. Атрибут (или набор атрибутов), который может быть использован для однозначной идентификации конкретного кортежа (строки или записи), называется **первичным ключом**.

ЭЛЕКТРОННАЯ ВЕРСИЯ

Во многих СУБД имеется механизм так называемых **внешних ключей**. Смысл этого механизма состоит в том, что некоему атрибуту (или группе атрибутов) одного отношения назначается ссылка на первичный ключ другого отношения; тем самым закрепляются связи подчиненности между этими отношениями.

2.4. Синтаксис SQL команд описания таблиц, связи таблиц между собой, описания атрибутов и их типов

Structured Query Language (SQL) – это непроцедурный язык, используемый для формулировки запросов к БД в большинстве современных СУБД и в настоящий момент являющийся индустриальным стандартом.

Непроцедурность языка означает, что на нем можно указать, что нужно сделать с базой данных, но нельзя описать алгоритм этого процесса. Все алгоритмы обработки SQL-запросов генерируются самой СУБД и не зависят от пользователя. Язык SQL состоит из набора операторов, которые можно разделить на несколько категорий:

- **Data Definition Language** (DDL) – язык определения данных, позволяющий создавать, удалять и изменять объекты в базах данных;
- **Data Manipulation Language** (DML) – язык управления данными, позволяющий модифицировать, добавлять и удалять данные в имеющихся объектах базы данных;
- **Data Control Language** (DCL) – язык, используемый для управления пользовательскими привилегиями;
- **Transaction Control Language** (TCL) – язык для управления изменениями, сделанными группами операторов;
- **Cursor Control Language** (CCL) – язык для определения курсоров, подготовки операторов SQL к выполнению и некоторых других операций.

2.4.1. Типы данных

Язык SQL поддерживает большое количество типов данных для представления и манипулирования информацией. Наиболее часто используемые типы данных даны в табл. 3. За дополнительной информацией по типам данных, поддерживаемых СУБД InterBase, необходимо обратиться к документации.

ЭЛЕКТРОННАЯ ВЕРСИЯ

Таблица 3

Типы данных, поддерживаемые InterBase

Тип данных	Обозначение	Формат представления
Целочисленные	INTEGER/INT; SMALLINT	INT – 4 байта, SMALLINT – 2 байта
Строковые типы	CHARACTER/CHAR [<число символов>]	
Дробные (вещественные) типы	DECIMAL [(<число цифр> [,<число цифр после запятой>])]	
Дата	DATE	MM/DD/YY; MM.DD.YY; DD:MM:YY и т.д. Зависит от настроек ОС и СУБД
Время	TIME	MM/SS/HH; MM.SS.HH и т.д. Зависит от настроек ОС и СУБД
Денежный тип	MONEY	Формат представления $n; n$; nр и т.д. Зависит от настроек ОС и СУБД

2.4.2. Синтаксис DDL команд

Оператор создания таблиц БД позволяет создавать новые таблицы и указывать связи между ними. Синтаксис:

CREATE TABLE <Имя таблицы> {<Определение столбца> |
[<Ограничение таблицы>]},

где:

<Определение столбца>::=
<имя столбца>
{<имя домена>|<тип данных>}
[<ограничение столбца>]
<Ограничение столбца>::=
UNIQUE|
PRIMARY KEY|
FOREIGN KEY (<имя столбца>)|
REFERENCES <имя таблицы>[(<имя столбца>…)] NOT NULL.

ЭЛЕКТРОННАЯ ВЕРСИЯ

Имена таблиц и определения столбцов должны быть уникальными. Назначение каждого атрибута в <*Ограничение столбца*> приведено далее:

- UNIQUE – определяет уникальное (не повторяющееся значение атрибута);
- PRIMARY KEY – первичный ключ;
- FOREIGN KEY – внешний ключ;
- REFERENCES – определение ссылки на другую таблицу;
- NOT NULL – указывает, что значение атрибута не должно быть пустым;

Оператор изменения структуры таблицы позволяет изменить структуру таблицы, имена атрибутов и их типы. Синтаксис:

```
ALTER TABLE <имя таблицы>
      {ADD <определение столбца>}|
      ALTER <имя столбца>|
      DROP <имя столбца>|
      ADD <ограничение таблицы>.
```

Следует заметить, что если пытаться изменить имя или тип столбца, на который имеется ссылка из другой таблицы, то будет выдано сообщение об ошибке, и изменения нельзя будет произвести. Для этого сначала необходимо разорвать связь между таблицами.

Оператор создания БД позволяет создавать новые БД в СУБД для последующего использования при хранении и доступе к данным. Синтаксис:

```
CREATE DATABASE '<имя файла БД>'
      USER '<имя пользователя>'
      PASSWORD '<пароль пользователя>',
```

где:

<имя файла БД> – имя физического файла, в котором будет храниться БД, расширение в имени указывается явно, как правило “gbd”. Создание базы данных происходит в два этапа. Сначала физически создается файл БД оператором CREATE DATABASE, затем выполняется регистрация БД в СУБД.

Для регистрации БД в СУБД InterBase в меню Database необходимо выбирать пункт Register. В открывшемся окне в разделе Database требуется указать путь к файлу, хранящему БД и ее псевдоним – Alias. В поле Login Information заносится информация о пользователе, создающем базу. Имя пользователя (Login) и пароль (Password). Для успешной регистрации пользователь должен быть предварительно зарегистрирован. Следует помнить, что пользователь с именем sysdba существует всегда по умолчанию.

ЭЛЕКТРОННАЯ ВЕРСИЯ

В качестве рабочего примера, рассматриваемого в данной и в последующих лабораторных работах, возьмем предметную область "Университет", краткое описание которой в виде исходного задания к лабораторным работам приведено в разделе 6 данной работы. В результате анализа данной предметной области был выявлен набор атрибутов и введено их определение. Часть атрибутов, имеющих общую семантику, объединены в виде отдельных таблиц. Например таблица, хранящая сведения о дисциплинах, объединяет в себе атрибуты: идентификатор дисциплины (ID_Disp), тип дисциплины (Type), кафедра ведущая дисциплину (Cafedra) и название дисциплины (Disp_Name). Конечный результат такого анализа представлен в табл. 4.

Таблица 4

Описание таблиц и атрибутов предметной области "Университет"

Название таблицы	Название атрибута	Описание
Disp		Информация о дисциплине
	ID_Disp	Идентификатор дисциплины
	Type	Тип дисциплины (может принимать значения: ЛК – лекция, ПР – практическое занятие, ЛБ – лабораторные занятия, КР – курсовая работа)
	Cafedra	Название ведущей кафедры
	Disp_Name	Название дисциплины
	Phone	Телефон кафедры
Person		Информация о сотруднике университета
	FIO	Фамилия, имя, отчество сотрудника
	Tab_N	Табельный номер сотрудника
	Ac_Degree	Ученая степень сотрудника
Teach_Load		Нагрузка на сотрудника
	Teach_ID	Идентификатор преподавателя, ведущего дисциплину
	Start_Work	Время начала работы

Для осмысленного хранения информации как единых данных по предметной области, имеющих конкретный семантический смысл необходимо связать таблицы между собой. Связь между таблицами выполнена с использованием внешних ключей (`FOREIGN KEY`). Скрипты на языке SQL, описывающие таблицы и связи между ними, представлены далее.

```
CREATE TABLE "Disp"(
  "ID_Disp"      INTEGER not null primary key,
  "Disp_Name"    CHAR(20) not null,
  "Cafedra"      CHAR(50),
  "Phone"        CHAR(8)
);
```

ЭЛЕКТРОННАЯ ВЕРСИЯ

```
CREATE TABLE "Person"(
  "FIO"          CHAR(30) not null,
  "Tab_N"        INTEGER not null,
  "Ac_Degree"    CHAR(20),
  PRIMARY KEY ("Tab_N")
);

CREATE TABLE "Teach_Load"(
  "Teach_ID"     INTEGER NOT NULL PRIMARY KEY,
  "Start_Of_Work"   DATE DEFAULT 'NOW' not null,
  "Person_ID"       INTEGER not null,
  "Disp_ID"         INTEGER not null,
  FOREIGN KEY ("Person_ID") REFERENCES "Person" ("Tab_N"),
  FOREIGN KEY ("Disp_ID") REFERENCES "Disp" ("ID_Disp")
);
```

3. Порядок выполнения работы

3.1. Разработать схемы таблиц для предметной области, взятой из варианта индивидуального задания. Таблиц должно быть не менее трех, например, две таблицы сущностей и связующая таблица. Согласовать с преподавателем схемы таблиц и связи между ними.

3.2. Разработать запросы, полезные для работы в предметной области. Запросы должны содержать реализации следующих операций реляционной алгебры: выборка из всех трех таблиц; проекция и выборка из всех трех таблиц; внутреннее соединение двух таблиц; естественное соединение двух таблиц; объединение двух таблиц (можно использовать объединение выборок из одной и той же таблицы); вычисление всех агрегатных функций; упорядочение данных; группирование данных.

3.3. Запустить программу IBConsole, указать имя пользователя "sysdba", и пароль "masterkey".

3.4. Создать базу данных.

3.5. Создать таблицы

3.6. Заполнить таблицы значениями. В каждой таблице должно быть не менее 5-10 строк.

3.7. В редакторе SQL запросов выполнить разработанные запросы, результаты сохранить в файл, используя команду Query→Save output.

4. Содержание отчета

4.1. Цель работы

4.2. Вариант индивидуального задания к работе.

ЭЛЕКТРОННАЯ ВЕРСИЯ
ЭЛЕКТРОННАЯ ВЕРСИЯ

4.3. Результат анализа индивидуального задания с описанием таблиц и атрибутов внутри таблиц, а также связи между таблицами.

4.4. Тексты всех разработанных SQL скриптов для создания БД и таблиц внутри БД.

4.5. Таблицы с исходными данными.

4.6. Результаты выполнения запросов.

4.7. Выводы по работе.

5. Контрольные вопросы

5.1. Понятие реляционной алгебры.

5.2. Теоретико-множественные операторы реляционной алгебры. Понятие.

5.3. Специальные операторы реляционной алгебры. Понятие.

5.4. Операторы DDL. Назначение.

5.5. Операторы SQL создания БД, таблиц, доменов. Основные разделы и их назначение.

6. Варианты индивидуальных заданий

Каждый вариант индивидуального задания к лабораторной работе представляет собой описание некоторой информационной системы (ИС), которая хранит специфичную информацию, характерную для данной предметной области. Задача студента заключается в выделении основных атрибутов ИС и их классификации при организации структурированного хранения этой информации в БД в соответствии с порядком выполнения работы.

6.1. ИС «*Библиотека*». Минимальный список характеристик: автор книги, название, год издания, цена, является ли книга новым изданием, краткая аннотация; номер читательского билета, ФИО, адрес и телефон читателя, дата выдачи книги читателю и дата сдачи книги читателем.

6.2. ИС «*Оптовая база*». Минимальный список характеристик: код товара, название товара, количество на складе, стоимость единицы товара, примечание (описание товара); номер и ФИО поставщика товара, срок поставки и количество товаров в поставке.

6.3. ИС «*Производство*». Минимальный список характеристик: код изделия, название изделия, год выпуска, объем выпуска, примечание (для каких целей предназначено); код, название, адрес и телефон предприятий производящих изделие и потребителей данного изделия.

ЭЛЕКТРОННАЯ ВЕРСИЯ

6.4. ИС «Сеть магазинов». Минимальный список характеристик: номер, ФИО, адрес, телефон и капитал владельца магазина; номер, название, адрес и телефон магазина; номер, ФИО, адрес, телефон поставщика, а также стоимость поставок данным поставщиком в данный магазин.
6.5. ИС «Авторемонтные мастерские». Минимальный список характеристик: номер водительских прав, ФИО, адрес и телефон владельца автомобиля; номер, ФИО, адрес, телефон и квалификация механика; номер, марка, мощность и цвет автомобиля; номер, название, адрес и телефон ремонтной мастерской.
6.6. ИС «Деканат». Минимальный список характеристик: наименование специальности, код группы, ФИО, дата рождения, домашний адрес, телефон слушателя, примечание: автобиография слушателя; код, название, количество часов и вид контроля предметов, код сессии и оценки каждого слушателя по каждому предмету в каждую сессию.
6.7. ИС «Договорная деятельность организации». Минимальный список характеристик: шифр договора, наименование организации, сроки выполнения, сумма договора, примечание: вид договора; номер, ФИО, адрес, телефон, должность, оклад сотрудников, сроки работы данного сотрудника по данному договору.
6.8. ИС «Поликлиника». Минимальный список характеристик: номер, ФИО, дата рождения пациента; ФИО, должность и специализация лечащего врача, диагноз, поставленный данным врачом данному пациенту, необходимо ли амбулаторное лечение, срок потери трудоспособности, состоит ли на диспансерном учете.
6.9. ИС «Телефонная станция». Минимальный список характеристик: номер абонента, ФИО абонента, адрес, дата установки, наличие блокиратора, задолженность.
6.10. ИС «Спорт». Минимальный список характеристик: ФИО спортсмена, дата рождения, вид спорта, команда, страна, зачетный результат, является ли он достижением, каким (мировой рекорд, олимпийский и т.п.) и за какой год.
6.11. ИС: «Городской транспорт». Минимальный список характеристик: вид транспорта, номер маршрута, дата введения маршрута, начальная остановка, конечная остановка, время в пути.
6.12. ИС «География». Минимальный список характеристик: название страны, столица, площадь территории, является ли страна развитой в экономическом отношении, количество населения, преобладающая национальность.

ЭЛЕКТРОННАЯ ВЕРСИЯ

6.13. ИС «Аэропорт». Минимальный список характеристик: номер рейса, пункт назначения, дата рейса, тип самолета, время вылета, время в пути, является ли маршрут международным, сведения о пассажире: ФИО, дата рождения, место проживания.

6.14. ИС «Персональные ЭВМ». Минимальный список характеристик: фирма-изготовитель, тип процессора, тактовая частота, объем оперативной памяти, объем жесткого диска, дата выпуска; сведения о фирмах, реализующих ЭВМ: наименование, адрес, телефон.

6.15. ИС «Микросхемы памяти». Минимальный список характеристик: маркировка, разрядность, емкость, дата начала выпуска, время доступа, является ли широко используемой, стоимость.

6.16. ИС «Ипподром». Минимальный список характеристик: кличка лошади, масть, возраст, вид забега, является ли лошадь фаворитом; сведения о наезднике: ФИО, возраст, масса, занятое место.

6.17. ИС «Спутники планет». Минимальный список характеристик: название, планета-хозяин, дата открытия, диаметр, период обращения; сведения о планете-хозяине: название, дата открытия, масса, радиус, удаленность от Солнца.

6.18. ИС «Лесное хозяйство». Минимальный список характеристик: наименование зеленого массива, площадь, основная порода, является ли заповедником, дата последней проверки, ФИО лесника; сведения о лесном хозяйстве: название, ФИО директора, курируемые территории.

6.19. ИС «Автотранспортное предприятие». Минимальный список характеристик: номерной знак автомобиля, марка автомобиля, техническое состояние автомобиля, местонахождение автомобиля, средняя скорость, грузоподъемность, расход топлива; сведения о водителе: ФИО, табельный номер, дата рождения, стаж работы, оклад, дата выезда, дата прибытия, место назначения, расстояние, расход горючего, масса груза.

6.20. ИС «Заказы». Минимальный список характеристик: ФИО клиента, номер счета, адрес, телефон, номер заказа, дата исполнения, стоимость заказа, название товара, его цена и количество.

6.21. ИС «Университет». Минимальный список характеристик: номер, ФИО, адрес и должность преподавателя; код, название, количество часов, тип контроля и раздел предмета; код, название, номер заведующего кафедрой; номер аудитории, где преподаватель читает свой предмет.

ЭЛЕКТРОННАЯ ВЕРСИЯ

Лабораторная работа № 2

ИЗУЧЕНИЕ МЕТОДОЛОГИИ ER («СУЩНОСТЬ-СВЯЗЬ»)
И СРЕДСТВ МОДЕЛИРОВАНИЯ ДАННЫХ
С ИСПОЛЬЗОВАНИЕМ CASE-СРЕДСТВА ERWIN

1. Цель работы

Изучить методологию ER. Научиться проектировать структуру БД с использованием методологии ER и CASE средства ERWin.

2. Теоретические сведения

2.1. Общая характеристика ERWin. Назначение

Семейство продуктов ERWin предназначено для моделирования и создания баз данных произвольной сложности на основе диаграмм "сущность-связь". В настоящее время ERWin является наиболее популярным пакетом моделирования данных благодаря поддержке широкого спектра СУБД самых различных классов: SQL-серверов (Oracle, Informix, Sybase SQL Server, InterBase, MS SQL Server, Progress, DB2, SQLBase, Ingress, Rdb и др.).

Информационная модель представляется в виде диаграмм "сущность-связь", отражающих основные объекты предметной области и связи между ними. Дополнительно определяются атрибуты сущностей, характеристики связей, индексы и бизнес-правила, описывающие ограничения и закономерности предметной области. После создания ER-диаграммы пакет автоматически генерирует SQL-код для создания таблиц, индексов и других объектов базы данных. По заданным бизнес-правилам формируются стандартные триггеры БД для поддержки целостности данных, для сложных бизнес-правил можно создавать собственные триггеры, используя библиотеку шаблонов. Пакет может осуществлять реинжиниринг существующих БД: по SQL-текстам автоматически генерируются ER-диаграммы. Таким образом пакет полностью поддерживает технологию FRE (forward and reverse engineering).

2.2. Основные понятия методологии «сущность-связь»

Разработка базы данных основана на методе проектирования с помощью диаграммы «сущность-связь» (E/R - диаграмма).

ЭЛЕКТРОННАЯ ВЕРСИЯ

E/R-диаграмма – это графическое представление предметов и отношений между ними. Ее цель – точно представить на логическом уровне данные, которые необходимо хранить и обрабатывать.

Атрибуты. Атрибуты представляют данные об объектах, которые необходимо хранить. Атрибуты представляются именами существительными, которые описывают характеристики сущностей.

*Сущност*ь – это множество экземпляров реальных или абстрактных объектов, обладающих атрибутами или характеристиками. Имя сущности отображает тип объекта (обычно имя существительное).

Связь – это некоторая ассоциация между двумя и более сущностями, которая показывает взаимосвязь между ними. Характеризуется типами связей (1:1, 1:n, n:m). Если экземпляр сущности-потомка однозначно определяется своей связью с сущностью-родителем, то связь называется идентифицирующей, в противном случае - неидентифицирующей. Связь может дополнительно определяться с помощью указания степени или мощности (количества экземпляров сущности-потомка, которое может существовать для каждого экземпляра сущности-родителя). Могут быть выражены следующие мощности связей:

- каждый экземпляр сущности-родителя может иметь ноль, один или более связанных с ним экземпляров сущности-потомка;
- каждый экземпляр сущности-родителя должен иметь не менее одного связанного с ним экземпляра сущности-потомка;
- каждый экземпляр сущности-родителя должен иметь не более одного связанного с ним экземпляра сущности-потомка;
- каждый экземпляр сущности-родителя связан с некоторым фиксированным числом экземпляров сущности-потомка.

Концептуальная модель данных (логическая модель) – это описание, диаграмма, изображение или модель предметной области, в ней показаны только объекты (сущности), которыми должна оперировать разрабатываемая система, и связи между ними. Она разрабатывается на начальном этапе проектирования базы данных.

Физическая модель данных описывает данные средствами конкретной СУБД. Отношения, разработанные на стадии формирования логической (концептуальной) модели данных, преобразуются в таблицы, атрибуты становятся столбцами таблиц, для ключевых атрибутов создаются уникальные индексы, домены (набор допустимых значений атрибута) преображаются в типы данных, принятые в конкретной СУБД.

ЭЛЕКТРОННАЯ ВЕРСИЯ

2.3. Понятие нормализации

Нормализация является операцией перемещения атрибутов в подходящие сущности в соответствие с требованиями нормальных форм. Нормализация данных означает проектирование структур данных таким образом, чтобы удалить избыточность и ограничить несвязанные структуры.

Нормализация таблиц – это формальный аппарат ограничений на формирование таблиц, описывающий разбиение таблиц на две или более части и обеспечивающий применение лучших методов добавления, изменения и удаления данных. Можно также сказать, что нормализация – это процесс представления данных в виде простых двумерных таблиц, который позволяет устранить дублирование этих данных и обеспечивает непротиворечивость хранимой в базе информации. *Цель нормализации* – получение проекта БД, в котором любая часть логически законченной информации хранится в одном месте, т.е. исключается избыточность информации. Основа нормализации – аппарат нормализации отношений и приведение отношений к нормальным формам.

Всего существует шесть форм нормальных отношений. Но, как правило, необходимо и достаточно привести БД к третьей нормальной форме, чтобы исключить указанные аномалии при работе с БД. Таблица считается нормализованной на определенном уровне, когда она удовлетворяет условиям, накладываемым соответствующей формой нормализации.

Формы нормализации:

- первая нормальная форма (First Normal Form – 1NF);
- вторая нормальная форма (Second Normal Form – 2NF);
- третья нормальная форма (Third Normal Form – 3NF);
- нормальная форма Бойса-Кодда (Boice-Codd Normal Form – BCNF);
- четвертая нормальная форма (Fourth Normal Form – 4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (Fifth Normal Form – 5NF или PJ/NF);

Теория нормализации основывается на наличии той или иной зависимости между атрибутами отношения.

2.4. Виды зависимостей между отношениями

Для устранения указанных аномалий (а на самом деле *для правильного проектирования модели данных*) применяется метод нормализации отношений. Нормализация основана на понятии функциональной зависимости атрибутов отношения.

ЭЛЕКТРОННАЯ ВЕРСИЯ

Пусть *R* - отношение. Множество атрибутов *Y* **функционально зависимо** от множества атрибутов *X* (*X* **функционально определяет** *Y*) тогда и только тогда, когда для любых кортежей r1,r2∈*R* из того, что *r1X=r2X* следует, что и *r1Y=r2Y*. Другими словами во всех кортежах, имеющих одинаковые значения атрибутов *X*, значения атрибутов *Y* также совпадают в отношении *R*. Символически функциональная зависимость записывается *X*→*Y*.

Замечание. Если атрибуты *X* составляют потенциальный ключ отношения *R*, то *любой* атрибут отношения *R* функционально зависит от *X*.

Пример. В отношении «Университет» можно привести следующие примеры функциональных зависимостей:

{*Cafedra*}→Phone

{*Disp_Name*} →Type

Замечание. Приведенные функциональные зависимости *не выведены* из внешнего вида отношений, представленных в табл. 4. Эти зависимости отражают взаимосвязи, обнаруженные между объектами предметной области, и являются дополнительными ограничениями, определяемыми предметной областью. Таким образом, функциональная зависимость – *семантическое понятие*. Она возникает, когда по значениям одних данных в предметной области можно определить значения других данных. Например, зная табельный номер сотрудника, можно определить его фамилию, по номеру отдела можно определить телефон. Функциональная зависимость *задает дополнительные ограничения* на данные, которые могут храниться в отношениях. Для корректности базы данных (адекватности предметной области) необходимо при выполнении операций модификации базы данных проверять все ограничения, определенные функциональными зависимостями.

На самом деле, функциональную зависимость между атрибутами *A* и *B* в обычном понятии можно выразить еще и так: атрибут *A* функционально зависим от *B* тогда, когда каждому значению *A* в любой момент соответствует единственное значение *B* из всех возможных.

Пусть *K* – ключ, *A1*, *A2*, ..., *An* – некоторые атрибуты в отношении *R*. Пусть *A1*, *A2*, ..., *An*∈*K* (т.е. имеем составной ключ). **Полной функциональной зависимостью** неключевых атрибутов *B1*, *B2*,… *Bm* называется такая зависимость, при которой каждый *B1*, *B2*,… *Bm* функционально зависит от *K* (т.е. от всей совокупности атрибутов ключа), но не находится в функциональной зависимости ни от какой части составного ключа.

ЭЛЕКТРОННАЯ ВЕРСИЯ

Пример. В отношении «Университет» можно привести следующие примеры полных функциональных зависимостей:

{*Cafefra*, *Disp_Name, Type*}→FIO

Транзитивная зависимость наблюдается в том случае, если один из двух неключевых атрибутов функционально зависит от ключа, а другой неключевой атрибут зависит только от первого.

Нетранзитивная зависимость имеется в том случае, если ни один из неключевых атрибутов функционально не зависит от любого другого неключевого атрибута.

Многозначная зависимость имеется тогда, когда атрибут *A* однозначно определяет атрибут *B* в том случае, если для каждого значения атрибута *A* существует хорошо определенное множество соответствующих значений атрибута *B*.

Первая нормальная форма (1НФ) – это обычное отношение. Согласно данному определению отношений, любое отношение автоматически уже находится в 1НФ. Свойства отношений (свойства 1НФ) в этом случае будут следующими:

– в отношении нет одинаковых кортежей;

– кортежи не упорядочены;

– атрибуты не упорядочены и различаются по наименованию;

– все значения атрибутов атомарны.

Если предположить, что все атрибуты предметной области «Университет» размещены в одной таблице, то она будет находящегося в 1НФ.

Отношение *R* находится во **второй нормальной форме** (**2НФ**) тогда и только тогда, когда оно находится в 1НФ и все ее поля, не входящие в первичный ключ, связаны полной функциональной зависимостью с первичным ключом. Т.е. любое неключевое поле однозначно идентифицируется полым набором ключевых полей.

Замечание. Если потенциальный ключ отношения является простым, то отношение автоматически находится во 2НФ.

Таблица, находящаяся во 2НФ должна удовлетворять следующим требованиям: таблица должна содержать данные об одном типе объектов; каждая таблица должна содержать одно поле, или несколько полей, образующих уникальный идентификатор или первичный ключ для каждой строки; все, не входящие в первичный ключ поля, должны однозначно определяться этим ключом.

Если в таблице есть хотя бы одно поле, не зависящее от первичного

ЭЛЕКТРОННАЯ ВЕРСИЯ

ключа, то в первичный ключ необходимо включить дополнительные столбцы. Если таких нет, то добавить его. Если таблица не находится во 2НФ, то нужно выполнить ее *декомпозицию*.

Для нашего примера результатом декомпозиции будет:

Disp

ID_Disp	Disp_Name	Type
1	САПР	ЛК
2	САПР	ПР
3	АрхЭВМ	ЛК
4	АрхЭВМ	КП
5	ЗИ	ЛК
6	ЗИ	ЛБ
7	Инф-ка	ПР

Cafedra

ID_Caf	Cafedra	Phone
1	ВТ	211718
2	ИСИМ	344248
3	ИЗИ	317442

Person

Tab_N	FIO
1111	Ланцов В.Н.
2222	Мамаев А.А
3333	Буланкин В.Б.
4444	Алешкин А.А.

Отношение *R* находится в **третьей нормальной форме** (**3НФ**) тогда и только тогда, когда отношение находится во 2НФ и ни одно из ее неключевых полей функционально не зависит от любого другого неключевого поля. Или другими словами, таблица находится в 3НФ, если она находится во 2НФ и каждое ее неключевое поле нетранзитивно зависит от первичного ключа.

Очевидно, что для нашего примера все таблицы удовлетворяют условию 3НФ, кроме отношения *Disp*. Так как категория занятия, вообще говоря, зависит от вида дисциплины. Поэтому его надо разбить на два более мелких отношения:

Disp

ID_Disp	Disp_Name	ID_Cat
1	САПР	1
2	САПР	2
3	АрхЭВМ	1
4	АрхЭВМ	4
5	ЗИ	1
6	ЗИ	3
7	Инф-ка	2

Categories

ID_Cat	Type
1	ЛК
2	ПР
3	ЛБ
4	КП

Отношение находится в **нормальной форме Бойса-Кодда** только в том случае, если любая функциональная зависимость между его атрибутами сводится к полной функциональной зависимости от ключа.

Отношение находится в **четвертой нормальной форме**, если оно находится в нормальной форме Бойса-Кодда и все его многозначные зависимости фактически являются функциональными зависимостями от потенциальных ключей.

ЭЛЕКТРОННАЯ ВЕРСИЯ

Отношение находится в **пятой нормальной форме** тогда и только тогда, когда в каждой его полной декомпозиции все проекции содержат возможный ключ. Отношение, не имеющее ни одной сложной декомпозиции, также находится в пятой нормальной форме.

На практике многие логические модели приведены только к третьей нормальной форме, которой достаточно для работы с большинством предметных баз данных.

Создание диаграммы БД «Университет».

Модель предметной области (логического уровня).

Каждой сущности присваивается уникальное имя и номер, разделяемые косой чертой "/" и помещаемые над блоком. Сущности E/1 и E/2 – родительские, сущность E/3 – дочерняя. Результат представлен на рис. 1.

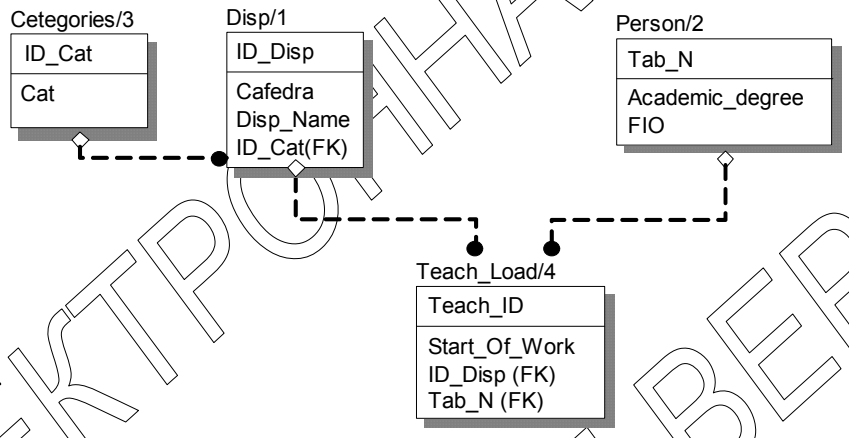

Рис. 1. Логическая модель БД «Университет»

Атрибуты изображаются в виде списка имен внутри блока сущности. Атрибуты, определяющие первичный ключ, размещаются наверху списка и отделяются от других атрибутов горизонтальной чертой. Идентифицирующая связь между сущностью-родителем и сущностью-потомком изображается сплошной линией. Сущность-потомок в идентифицирующей связи является зависимой от идентификатора сущностью. Пунктирная линия изображает неидентифицирующую связь. Сущность-потомок в неидентифицирующей связи будет независимой от идентификатора. Сущности могут иметь также внешние ключи (Foreign Key), которые могут использоваться в качестве части или целого первичного ключа или неключевого атрибута. Внешний ключ изображается с помощью помещения внутрь блока сущности имен атрибутов, после которых следуют буквы FK в скобках.

Результат преобразования модели из логического уровня в физический уровень моделирования представлен на рис. 2.

ЭЛЕКТРОННАЯ ВЕРСИЯ

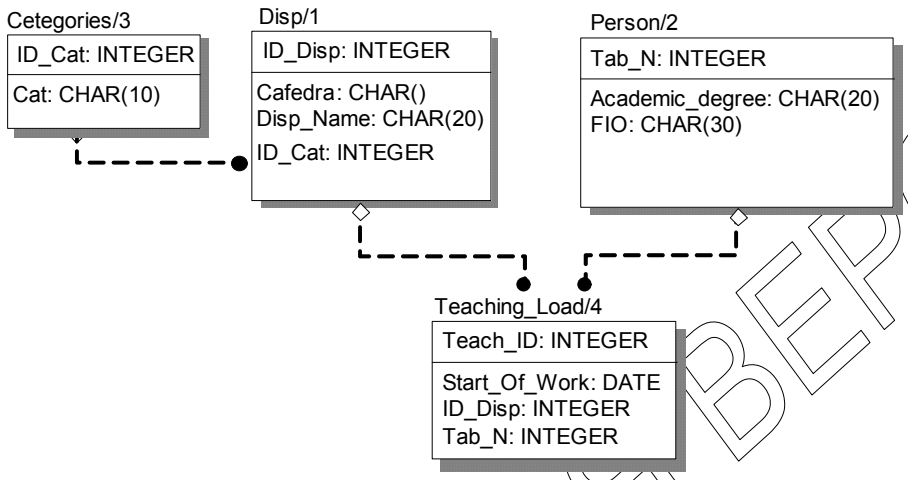

Рис. 2. Физическая модель БД «Университет»

По модели генерируется SQL скрипт:

```
CREATE TABLE Caregories (
    ID_Cat              INTEGER NOT NULL,
    Cat                 CHAR(10),
    PRIMARY KEY (ID_Cat)
);

CREATE TABLE Disp (
    Cafedra             CHAR() NOT NULL,
    ID_Disp             INTEGER NOT NULL,
    Disp_Name           CHAR(20) NOT NULL,
    PRIMARY KEY (ID_Disp),
    FOREIGN KEY (ID_Cat)    REFERENCES Categories
);

CREATE TABLE Person (
    Tab_N               INTEGER NOT NULL,
    Academic_degree     CHAR(20),
    FIO                 CHAR(30),
    PRIMARY KEY (Tab_N)
);

CREATE TABLE Teaching_Load (
    Teach_ID            INTEGER NOT NULL,
    Start_Of_Work       DATE DEFAULT CURRENT SQLID NOT NULL,
    ID_Disp             INTEGER NOT NULL,
    Tab_N               INTEGER NOT NULL,
    PRIMARY KEY (Teach_ID),
    FOREIGN KEY (Tab_N)     REFERENCES Person,
    FOREIGN KEY (ID_Disp)   REFERENCES Disp
);
```

ЭЛЕКТРОННАЯ ВЕРСИЯ

3. Порядок выполнения работы

3.1. Ознакомиться с вариантом индивидуального задания, взятым из предыдущей лабораторной работы.

3.2. Разработать схему ИС в соответствии с методологией ER. Для этого выделить сущности предметной области, установить их атрибуты. Связать сущности между собой. В схеме должно быть не менее трех сущностей. Например, две сущности предметной области и промежуточная (связывающая) сущность. Согласовать с преподавателем схемы, отношения, атрибуты и связи между ними. Привести отношения к третьей нормальной форме там, где это необходимо.

3.3. Запустить ERWin. В меню File выбрать пункт New.

3.4. В открывшемся диалоге Create Model – Select Template, в разделе New Model Type необходимо выбрать тип разрабатываемой модели: Logical/Physical. Это означает, что в разрабатываемой модели будет включен логический уровень (уровень предметной области) и физический уровень (уровень реализации структуры БД для конкретной СУБД). Для физического уровня в разделе Target Database необходимо выбрать вид СУБД, для которой разрабатываются модели – Interbase.

3.5. CASE средство ERWin является стандартным Windows-приложением, в котором реализован редактор моделей БД с поддержкой методологии «сущность-связь», генератор SQL скриптов на основе создаваемых моделей, а также вспомогательные утилиты для работы с моделями. Например, утилиты проверки структуры моделей, сопоставления структуры БД различных версии (с возможностью поиска различий), обратной разработки (Reverse Engineer) и т.д.

3.6. Реализовать модель предметной области (логического уровня) в редакторе ERWin. Смотри пример модели «Университет» (University).

3.7. Проанализировать адекватность SQL описания рассматриваемой модели предметной области.

3.8. Перейти на физический уровень моделирования. Для этого в меню Model выбрать пункт Physical Model. Проанализировать, как изменились представление и структура модели при переходе с логического уровня на физический. Для того чтобы перейти обратно на логический уровень моделирования, необходимо в меню Model выбрать пункт Logical Model.

3.9. Выполнить процедуру прямой разработки БД (Forward Engineer). Процедура прямой разработки БД заключается в формировании SQL скриптов готовых для запуска в интерпретаторе команд используемой

ЭЛЕКТРОННАЯ ВЕРСИЯ

СУБД (в нашем случае это консоль IBConsole СУБД InterBase). Для выполнения прямой разработки БД необходимо в меню Tools выбрать пункт «Forward Engineer/Schema Generation…». Прямая разработка возможна только в том случае, если Вы находитесь на физическом уровне моделирования БД.

3.10. Поскольку модель БД может содержать множество различных объектов БД, то существует возможность формировать соответствующие определения в SQL скрипте. Так как разрабатываемая в работе модель БД предметной области, носит простейший характер, то необходимо указать только часть объектов БД, которые должны быть отображены в SQL скрипте. Поэтому в диалоге «InterBase Schema Generation» среди всех объектов, предлагаемых к включению в SQL скрипт необходимо выбрать только необходимые, важные для выполнения работы. Для этого на закладке Options в списке InterBase 5 Schema Generation необходимо выставить флаги, как это показано в табл. 5.

Таблица 5

Установки для формирования структуры БД

Элемент списка InterBase 5 Schema Generation	Состояние флагов
Schema	Все флаги сброшены
View	Все флаги сброшены
Table	Все флаги сброшены Флаг CREATE TABLE установлен
Column	Все флаги установлены Флаг Use DOMAIN сброшен
Index	Все флаги сброшены
Referential Integrity	Флаг Primary Key (PK) установлен, переключатель установлен на CREATE/PK; Флаг Foreign Key (PK) установлен, переключатель установлен на CREATE/FK
Trigger	Все флаги сброшены
Other Options	Все флаги сброшены

3.11. Выбрать предварительный просмотр формируемого скрипта. Для этого нажать кнопку Preview. Проанализировать структуру полученного скрипта.

3.12. Текст SQL скрипта скопировать в буфер обмена и вставить в IBConsole. Запустить его на исполнение. Посмотреть результат.

3.13. Вернуться к логической модели предметной области изменить вид/тип связи между сущностями. Повторно сгенерировать SQL скрипт. Проанализировать изменения в сгенерированном скрипте.

ЭЛЕКТРОННАЯ ВЕРСИЯ

4. Содержание отчета

4.1. Цель работы.

4.2. Вариант индивидуального задания.

4.3. Перечень выделенных сущностей, атрибутов и связей между ними для данной предметной области.

4.4. ER диаграмма предметной области.

4.5. Тексты всех сгенерированных SQL скриптов.

4.6. Результаты исследования изменений в описания модели на языке SQL при изменении типов связей между сущностями (см. п.3.13).

4.7. Выводы по работе.

5. Контрольные вопросы

5.1. Понятие сущности.

5.2. Понятие атрибута сущности. Виды атрибутов.

5.3. Первичный ключ. Назначение. Особенности.

5.4. Внешний ключ сущности. Назначение.

5.5. Понятие связи между сущностями. Виды и типы связей.

5.6. Идентифицирующие и неидентифицирующие связи между сущностями. Особенности.

5.7. Нормальные формы отношений. Виды.

Лабораторная работа № 3

ИЗУЧЕНИЕ СРЕДСТВ ФОРМИРОВАНИЯ SQL ЗАПРОСОВ К СУБД INTERBASE

1. Цель работы

Изучить возможности языка SQL для выборки данных из базы данных.

2. Теоретические сведения

2.1. Краткие сведения о реляционной алгебре

Третья часть реляционной модели, манипуляционная часть, утверждает, что доступ к реляционным данным осуществляется при помощи реляционной алгебры или эквивалентного ему реляционного исчисления.

В реализациях конкретных реляционных СУБД сейчас не используется в чистом виде ни реляционная алгебра, ни реляционное исчисление. Фактическим стандартом доступа к реляционным данным стал язык SQL.

ЭЛЕКТРОННАЯ ВЕРСИЯ

Язык SQL представляет собой смесь операторов реляционной алгебры и выражений реляционного исчисления, использующий синтаксис, близкий к фразам английского языка и расширенный дополнительными возможностями, отсутствующими в реляционной алгебре и реляционном исчислении. Вообще, язык доступа к данным называется реляционно-полным, если он по выразительной силе не уступает реляционной алгебре (или, что то же самое, реляционному исчислению), т.е. любой оператор реляционной алгебры может быть выражен средствами этого языка. Именно таким и является язык SQL. Реляционная алгебра представляет собой набор операторов, использующих отношения в качестве аргументов и возвращающих отношения в качестве результата. Таким образом, реляционный оператор f выглядит как функция с отношениями в качестве аргументов:

$$R = f(R_1, R_2, ..., R_n).$$

Реляционная алгебра является замкнутой, так как в качестве аргументов в реляционные операторы можно подставлять другие реляционные операторы, подходящие по типу:

$$R = f(f_1(R_{11}, R_{12}, ...), f_2(R_{21}, R_{22}, ...), ...).$$

Таким образом, в реляционных выражениях можно использовать вложенные выражения сколь угодно сложной структуры.

Каждое отношение обязано иметь уникальное имя в пределах базы данных. Имя отношения, полученного в результате выполнения реляционной операции, определяется в левой части равенства. Однако можно не требовать наличия имен от отношений, полученных в результате реляционных выражений, если эти отношения подставляются в качестве аргументов в другие реляционные выражения. Такие отношения будем называть неименованными отношениями. Неименованные отношения реально не существуют в базе данных, а только создаются в момент вычисления значения реляционного оператора.

Традиционно определяют восемь реляционных операторов, объединенных в две группы:

а) теоретико-множественные операторы: объединение, пересечение, вычитание, декартово произведение;

б) специальные реляционные операторы: выборка, проекция, соединение, деление.

Операторы, которые выражаются через другие реляционные операторы, будут **зависимыми**. **Независимые операторы** – это операторы, которые не могут быть выражены через любые другие операторы. Например,

ЭЛЕКТРОННАЯ ВЕРСИЯ

операторы деления и соединения будут зависимыми операторами, так как выражаются через операторы декартового произведения, выборки и проекции. А оператор объединения будет независимым оператором.

2.2. Общий синтаксис оператора SELECT

Оператор SELECT возвращает данные из таблицы, вида или хранимой процедуры. Различные инструкции SELECT выполняют следующие действия:

- Возвращают одиночную строку или часть строки из таблицы.
- Непосредственно возвращают список строк или список частичных строк из таблицы.
- Возвращают связанные строки или частичные строки из соединения двух или более таблиц.
- Возвращают все строки или частичные строки из объединения двух или более таблиц.

Любая инструкция SELECT содержит два обязательных предложения (SELECT, FROM) и возможно другие предложения (WHERE, GROUP BY, HAVING, UNION, PLAN, ORDER BY). Табл. 6 объясняет назначение каждого предложения.

Таблица 6

Конструкции языка SQL, входящие в состав оператора SELECT

Предложение	Назначение
SELECT	Список столбцов, которые возвращаются.
FROM	Определяет таблицы, в которых ищутся значения.
WHERE	Определенное условие поиска, которое используется, чтобы выбрать необходимые строки из множества всех строк. Предложение WHERE может содержать инструкцию SELECT, которая упоминается как *подзапрос*
GROUP BY	Группирует возвращенные строки, основываясь на общих значениях столбцов. Используется совместно с HAVING
UNION	Комбинирует результаты двух или более инструкций SELECT, создавая одиночную динамическую таблицу, исключая повторяющиеся строки
ORDER BY	Определяет порядок сортировки строк, возвращенных SELECT, по умолчанию в возрастающем порядке (ASC) или в убывающем порядке (DESC)
PLAN	Определяет план запроса, который будет использоваться оптимизатором запроса вместо обычного выбора

2.3. Синтаксис команды SELECT

```
SELECT [DISTINCT | ALL] {* | <значение> [, <значение> ...]}
FROM <имя таблицы> [, <имя таблицы> ...]
```

ЭЛЕКТРОННАЯ ВЕРСИЯ

```
[WHERE <условие поиска>]
[UNION <select_выражение>]
[ORDER BY <порядок сортировки>]

<function> = {
COUNT (* | [ALL] <значение> | DISTINCT <значение>)
  | SUM ([ALL] <значение> | DISTINCT <значение>)
  | AVG ([ALL] <значение> | DISTINCT <значение>)
  | MAX ([ALL] <значение> | DISTINCT <значение>)
  | MIN ([ALL] <значение> | DISTINCT <значение>)
}

<search_condition> = {<значение> <operator>

{<значение> | (<select_one>)}
  | <значение> [NOT] BETWEEN <значение> AND <значение>
  | <значение> [NOT] LIKE <значение> [ESCAPE <значение>]
  | <значение> [NOT] IN (<значение> [, <значение> ...] |
<select_list>)
  | <значение> IS [NOT] NULL
  | <значение> {[NOT] {= | < | >} | >= | <=}
    {ALL | SOME | ANY} (<select_list>)
  | EXISTS (<select_expr>)
  | SINGULAR (<select_expr>)
  | <значение> [NOT] CONTAINING <значение>
  | <значение> [NOT] STARTING [WITH] <значение>
  | (<условие поиска>)
  | NOT <условие поиска>
  | <условие поиска> OR <условие поиска>
  | <условие поиска> AND <условие поиска>}

<operator> = {=|<|>|<=|>=|!<|!>|<>|!=}

<порядок> =
{col|int} [COLLATE collation] [ASC|DESC]   [, <порядок>]
```

Далее дадим расшифровку каждой инструкции входящей в оператор *SELECT*:

• *SELECT [DISTINCT|ALL]* – определяет данные для поиска. Параметр *DISTINCT* удаляет повторяющиеся значения из возвращенных данных. Параметр *ALL* – параметр по умолчанию, возвращает все данные;

• *{*|<значение>[, <значение> ...]}* - указание диапазона выборки данных из таблицы. Может принимать одно из следующих значений:

ЭЛЕКТРОННАЯ ВЕРСИЯ

* - возвращает все столбцы из определенной таблицы; <val> [, <val> ...] - возвращает определенный список столбцов и значений;

• FROM <имя таблицы> [, <имя таблицы> ...] - список таблиц, видов и сохраненных процедур, из которых возвращаются данные. Список может включать JOIN, объединения могут быть вложенными;

• WHERE <условие поиска> – определяет условия, которые ограничивают подмножество возвращаемых строк из всех доступных строк;

• UNION - комбинирует две или более таблиц, которые имеют полностью либо частично одинаковую структуру;

• ORDER BY <порядок> - пределяет порядок, в котором строки возвращены;

2.4. Примеры полезных SELECT запросов

```
SELECT COUNT(*) AS "Количество преподавателей"
  FROM "Person";
```

Инструкция выбирает ФИО работников, которые преподают:

```
SELECT "FIO"
FROM "Person"
WHERE "Person"."Tab_N" IN
      (SELECT "Teaching_Load"."Person_ID"
       FROM "Teaching_Load");
```

Инструкция выбирает ФИО преподавателей, которые читают предмет "Управление данными":

```
SELECT "FIO"
FROM "Person"
WHERE "Person"."Tab_N" IN
      (SELECT "Teaching_Load"."Person_ID"
       FROM "Teaching_Load" WHERE
"Teaching_Load"."Disp_ID" IN
              (SELECT "ID_Disp" FROM "Disp"
               WHERE "Disp"='Управление данными'));
```

Следующая инструкция выбирает, какие дисциплины может читать каждый из преподавателей в зависимости от объема часов:

```
SELECT "Person"."FIO",
       "Person"."Hours_Load",
       "Disp"."Disp",
       "Disp"."Hours"
       FROM "Person", "Disp"
WHERE "Person"."Hours_Load" >= "Disp"."Hours"
ORDER BY "Person"."FIO";
```

ЭЛЕКТРОННАЯ ВЕРСИЯ

3. Порядок выполнения работы

3.1. Запустить IBConsole.

3.2. Подключиться к БД предметной области индивидуального задания, выданного на первую лабораторную работу.

3.3. Разработать не менее пяти SQL запросов к БД. При этом необходимо согласовать все запросы с преподавателем.

4. Содержание отчета

4.1. Цель работы.

4.2. Задание.

4.3. Синтаксис SELECT запроса.

4.4. Разработанные SELECT запросы с назначением и описанием каждого из них.

4.5. Результаты выполнения SQL запросов.

4.6. Выводы по работе.

5. Контрольные вопросы

5.1. Понятие реляционной алгебры. Назначение.

5.2. Реляционно-полные операторы.

5.3. Оператор декартового произведения. Назначение. Выражение через SELECT запрос.

5.4. Оператор соединения. Назначение. Выражение через SELECT запрос.

5.5. Оператор объединения. Назначение. Выражение через SELECT запрос.

5.6. Оператор вычитания. Назначение. Выражение через SELECT запрос.

Лабораторная работа № 4

РАБОТА С ОБЪЕКТАМИ СУБД INTERBASE, ПОДДЕРЖИВАЮЩИМИ ЦЕЛОСТНОСТЬ СТРУКТУРЫ БД (ТРИГГЕРЫ, ГЕНЕРАТОРЫ)

1. Цель работы

Познакомиться с объектами БД, поддерживающими целостность баз данных: триггеры, генераторы. Изучить средства языка SQL для создания, изменения и удаления объектов БД.

ЭЛЕКТРОННАЯ ВЕРСИЯ

2. Общие сведения

2.1. Триггер

Триггер – отдельная программа, ассоциированная с таблицей или представлением, которая автоматически выполняет действия, когда строка в таблице или представлении вставлена, модифицирована или удалена.

Триггер никогда не вызывается явно. Наоборот, когда приложение или пользователь пытаются выполнить инструкцию *INSERT*, *UPDATE* или *DELETE* над строкой в таблице, любые триггеры, связанные с этой таблицей и операцией, выполняются автоматически. Триггеры состоят из заголовка и тела.

Заголовок триггера содержит:

- имя триггера уникальное внутри базы данных, которое отличает триггер от всех остальных;
- имя таблицы, определяющее таблицу, с которой связан триггер;
- инструкции, которые определяют, когда триггер выполняется.

Тело триггера содержит:

- необязательный список *локальных переменных* и их типов данных;
- блок инструкций на языке процедур и триггеров *InterBase*, заключенный между ключевыми словами BEGIN и END. Эти инструкции выполняются, когда триггер запускается. Блок может содержать в себе другой блок, так чтобы могло быть много уровней вложений.

Создание триггера выполняется командой CREATE TRIGGER, синтаксис которой представлен далее:

```
CREATE TRIGGER <имя триггера> FOR <имя таблицы>
  [ACTIVE|INACTIVE]
  {BEFORE|AFTER}
  {DELETE|INSERT|UPDATE}
  [POSITION <число>]
AS <тело триггера>
terminator

<тело триггера> =
  [<список переменных>] <тело триггера>

<список переменных> =
  [DECLARE VARIABLE <имя переменной> <тип>; ...]

<тело триггера> =
BEGIN
  [<инструкция SQL> ...]
END
```

ЭЛЕКТРОННАЯ ВЕРСИЯ

где: <*имя триггера*> – имя триггера, которое должно быть уникальным среди всех процедур, таблиц и видов; *ACTIVE|INACTIVE* – определяет активен триггер, т.е. будет запускаться, или нет (по умолчанию *ACTIVE*); *BEFORE|AFTER* – задает момент запуска триггера, то есть запуск до или после операции над данными таблицы; *DELETE|INSERT|UPDATE* – определяет операцию над таблицей, которая вызывает запуск триггера; *POSITION* <*число*> – необязательный параметр, который определяет порядок срабатывания триггеров. Число должно быть в диапазоне от 0 до 32767. Чем меньше число, тем раньше сработает данный триггер. Триггеры с одинаковой позицией срабатывают в произвольном порядке объявления. Если позиция триггера не задана, то по умолчанию позиция 0; <*инструкция SQL*> – любая одиночная инструкция в языке процедур и триггеров InterBase. Каждая инструкция (исключая BEGIN и END) должна завершатся точкой с запятой(;); *terminator* – символ, который указывает на завершение тела триггера.

Удаление триггера выполняется инструкцией

DROP TRIGGER <имя триггера>

В теле триггера можно использовать две специальные переменные, "new" и "old". Переменная "old" ссылается на старое значение столбца текущей строки. Переменная "new" ссылается на новое значение столбца текущей строки.

Формат:

new <*имя столбца*> или *old.*<*имя столбца*>,

где <*имя столбца*> – имя столбца таблицы, для которой создается триггер.

Команда определения терминатора.

Совместно с командами CREATE PROCEDURE и CREATE TRIGGER используется команда SET TERM. Процедуры и триггеры определены, используя язык процедур и триггеров, в котором инструкция всегда заканчивается точкой с запятой. Процедура или триггер непосредственно должна быть завершена символом, отличным от точки с запятой.

Текстовый файл, содержащий определение CREATE PROCEDURE или CREATE TRIGGER, должен включать одну команду SET TERM перед определением и, соответственно, после определения. Начальный SET TERM определяет новый завершающий символ; конечный SET TERM восстанавливает точку с запятой (;) как признак окончания инструкции

SET TERM <*новый разделитель*> <*старый разделитель*>.

ЭЛЕКТРОННАЯ ВЕРСИЯ

2.2. Генератор

Генератор – это механизм, позволяющий создавать последовательные номера, которые могут быть вставлены в столбец при использовании функции GEN_ID(). Генератор используется, чтобы гарантировать уникальное значение числовых полей, используемых в PRIMARY KEY. Например номер счета, который должен уникально идентифицировать ассоциированную строку.

База данных может содержать любое количество генераторов. Генераторы глобальны для базы данных и могут быть использованы и обновлены в любой транзакции.

Когда генератор создан, SET GENERATOR может установить или изменить его текущее значение. Генератор может быть использован для триггера, процедуры или SQL инструкции, которая вызывает GEN_ID().

Создание генератора выполняется инструкцией

```
CREATE GENERATOR <имя генератора>.
```

Установка значения генератора в заданное значение

```
SET GENERATOR <имя генератора> TO <значение>.
```

Получить текущее значение генератора можно функцией

```
GEN_ID(<имя генератора>, <шаг генерации>).
```

2.3. Примеры использования генератора и триггера

- Инструкция создает генератор ID_DISP_GEN и триггер CREATE_DISP. Триггер использует генератор для создания последовательных числовых ключей с приращением 1 для столбца Disp.ID_Disp:

```
CREATE GENERATOR "ID_DISP_GEN";
SET GENERATOR "ID_DISP_GEN" TO 1;
SET TERM ^ ;

CREATE TRIGGER "CREATE_DISP" FOR "Disp"
  BEFORE INSERT
    POSITION 0
  AS BEGIN
    "Disp"."ID_Disp" = GEN_ID(ID_DISP_GEN, 1);
END^
SET TERM ; ^
```

Проверить работу триггера и генератора можно, используя оператор добавления записи:

```
INSERT INTO "Disp" ("Disp", "Caf")
VALUES ('Управление данными', 'ИСИМ');
COMMIT;
```

ЭЛЕКТРОННАЯ ВЕРСИЯ

- Создать триггер TLog, который будет протоколировать удаляемые из таблицы Person записи (таблица LOG должна быть создана).

```
SET TERM ^ ;
CREATE TRIGGER "TLog" FOR "Person" AFTER DELETE AS
BEGIN
    INSERT INTO "LOG" ("Date", "User", "Information")
        VALUES ('NOW', USER, old.FIO);
END^
SET TERM ; ^
```

3. Порядок выполнения работы

3.1. Запустить консоль сервера БД IBConsole.

3.2. Зарегистрировать БД предметной области, выданной в качестве индивидуального задания к первой работе.

3.3. Создать объекты БД генератор и триггер, используя соответствующие инструкции SQL. При этом число триггеров должно быть не менее трех, срабатывающих в различных условиях и для различных таблиц. Число генераторов должно быть не менее одного для каждой таблицы.

3.4. Опробовать работу созданных объектов БД. Для этого необходимо выполнить обращение с ним и к таблицам БД.

3.5. Выполнить операции, в которых используется генератор и/или при котором срабатывает созданный триггер.

4. Содержание отчета

4.1. Цель работы.

4.2. Задание.

4.3. Синтаксис оператора создания триггеров и генераторов.

4.4. Скрипты на языке SQL, реализующие генераторы и хранимые процедуры.

4.5. Описание созданных генераторов и триггеров

4.5. Результаты выполнения SQL запросов.

4.6. Выводы по работе.

5. Контрольные вопросы

5.1. Понятие об объектах БД.

5.2. Понятие триггера. Описание, назначение, принцип работы.

5.3. Понятие генератора. Описание, назначение, принцип работы.

5.4. Понятие ссылочной целостности таблиц. Способы поддержания.

5.5. Способы описания объектов базы данных (триггер и генератор) с использованием языка SQL.

ЭЛЕКТРОННАЯ ВЕРСИЯ

Лабораторная работа № 5

ИЗУЧЕНИЕ СРЕДСТВ ЯЗЫКА SQL ДЛЯ МАНИПУЛИРОВАНИЯ ОБЪЕКТАМИ: ХРАНИМАЯ ПРОЦЕДУРА И ПРЕДСТАВЛЕНИЕ

1. Цель работы

Познакомиться с объектами БД: хранимая процедура и представление. Изучить особенности языка SQL для работы с объектами БД.

2. Теоретические сведения

Представление описывает вид данных, основанный на одной (или нескольких) основной таблице в базе данных. Возвращаемые строки определены инструкцией SELECT, которая перечисляет столбцы исходных таблиц, попадающих в представление. Представление не выполняет сохранение данных. Это позволяет выполнять операции выборки, соединения, объединения и т.д. над представлениями так, как если бы они были таблицами.

Пользователь, который создает представление − его владелец, который имеет все привилегии, включая возможность предоставлять привилегии другим пользователям, триггерам и хранимым процедурам. Пользователь может иметь привилегии к представлению, без необходимости иметь доступ к таблицам, на которых представление основано.

Синтаксис команды создания представления

```
CREATE VIEW <имя представления> [(<имя столбца> [,
<имя столбца> ...])]
  AS <select запрос>;
```

где: <*имя представления*> − имя для представления. Должно быть уникальным среди всех имен представлений, таблиц и процедур в базе данных; <*имя столбца*> − имена столбцов представления. Имена столбцов должны быть уникальны среди всех имен столбцов в представлении. Параметр <*имя столбца*> является обязательным, если представление содержит столбцы, основанные на выражении, иначе необязателен. В этом случае имена столбцов берутся из базовой таблицы; <*select запрос*> − определяет критерии выбора строк, для включения в представление.

Удаление представления выполняется командой

```
DROP VIEW <имя представления>.
```

Пример создания представления:

```
CREATE VIEW "VIEW_Pesron" (
```

ЭЛЕКТРОННАЯ ВЕРСИЯ

```
  "FIO",
  "academic_degree"
) AS
  SELECT "FIO", "academic_degree"
  FROM "Person"
  WHERE "Hours_Load" > 80;
```

Хранимая процедура – это отдельная программа, написанная на языке процедур и триггеров *InterBase*, и сохраненная как часть метаданных базы данных. Хранимые процедуры могут принимать входные параметры и возвращать значения в приложение или туда, откуда они были вызваны.

Язык процедур и триггеров *InterBase* включает все инструкции SQL манипулирования данными и некоторые мощные улучшения, включающие следующие операторы: IF ... THEN ... ELSE, WHILE ... DO, FOR SELECT ... DO, исключительные ситуации и обработку ошибок.

Имеются два вида хранимых процедур:

Процедуры выбора, которые приложения могут использовать вместо таблиц или представления в инструкции SELECT. Процедура выбора должна быть определена для возвращения одного или более значений, иначе результатом выполнения процедуры будет ошибка.

Выполняемые процедуры, которые приложения могут непосредственно вызывать в инструкции EXECUTE PROCEDURE. Выполняемая процедура не требует возвращать значения вызываемой программе.

Хранимые процедуры состоят из заголовка и тела. Заголовок процедуры содержит:

– имя хранимой процедуры, которое должно быть уникальным среди имен процедур и таблиц в базе данных;

– необязательный список входных параметров и их типов данных, которые процедура принимает из вызывающей программы;

– следующий за ключевым словом RETURNS список выходных параметров и их типов данных, в случае, если процедура возвращает значения в вызывающую программу.

Тело процедуры содержит:

– необязательный список локальных переменных и их типов данных;

– блок инструкций на языке процедур и триггеров *InterBase*, ограниченный BEGIN и END. Блок может включать в себе другие блоки, так чтобы имелось несколько уровней вложения.

Так как каждая инструкция в теле хранимой процедуры должна завершатся точкой с запятой, необходимо определить другой символ для за-

ЭЛЕКТРОННАЯ ВЕРСИЯ

вершения инструкции CREATE PROCEDURE в SQL.

```
CREATE PROCEDURE <имя процедуры>
[(<имя параметра> <тип данных> [, <имя параметра>
<тип данных> ...])]
[RETURNS <имя параметра> <тип данных>
[, <имя параметра> <тип данных> ...])]
AS <тело процедуры>
<terminator>

<тело процедуры> =
[<описание переменных>]
BEGIN
  <инструкция SQL>
  [<инструкция SQL> ...]
END
<описание переменных> =
DECLARE VARIABLE var <тип данных>;
[DECLARE VARIABLE var <тип данных>; ...]
```

где: <*имя процедуры*> – имя процедуры, которое должно быть уникальным среди процедур, таблиц и видов; <*имя параметра*> <*тип данных*> – имя входного параметра и его тип, который вызывающая программа использует, чтобы передать значения процедуре. Имя входного параметра уникально для переменных внутри процедуры; *RETURNS* <*имя параметра*> <*тип данных*> – имя возвращаемой переменной и ее тип, который процедура использует, чтобы вернуть значения в вызывающую программу. Имя выходного параметра уникально для переменных внутри процедуры; *statement* – любая одиночная инструкция в языке процедур и триггеров InterBase. Каждая инструкция (исключая BEGIN и END) должна завершатся точкой с запятой (;); <*terminator*> – символ, определенный инструкцией SET TERM, который указывает завершение тела процедуры.

Удаление процедуры выполняется инструкцией

```
DROP PROCEDURE <имя процедуры>;
```

Примеры использования процедур

Следующая процедура max_caf_hour берет название кафедры в качестве входного параметра и возвращает максимальное число часов по плану для одного предмета:

```
SET TERM ^ ;
CREATE PROCEDURE max_caf_hour (Caf CHAR(30))
RETURNS (max_hour integer)
AS
BEGIN
```

ЭЛЕКТРОННАЯ ВЕРСИЯ
ЭЛЕКТРОННАЯ ВЕРСИЯ

```
  SELECT MAX("Hours")
    FROM "Disp"
    WHERE "Caf" = :Caf

INTO :max_hour;
    EXIT;
END ^
SET TERM ; ^
```

3. Порядок выполнения работы

3.1. Запустить консоль сервера БД IBConsole.

3.2. Зарегистрировать БД предметной области, выданной в качестве индивидуального задания к первой работе.

3.3. Создать объекты БД, используя соответствующие инструкции SQL. При этом необходимо создать не менее двух хранимых процедур и не менее трех представлений. Допускается использование представлений внутри хранимых процедур.

3.4. Опробовать работу созданных объектов БД. Для этого необходимо выполнить обращение с ним и к таблицам БД.

3.5. Выполнить операции, в которых используется представление. Вызов хранимой процедуры осуществить явным образом.

4. Содержание отчета

4.1. Цель работы.

4.2. Задание.

4.3. Синтаксис операторов описания хранимых процедур и представлений.

4.4. Разработанные хранимые процедуры и представления.

4.5. Описание созданных объектов и их назначение.

4.6. Результаты выполнения SQL запросов.

4.7. Выводы по работе.

5. Контрольные вопросы

5.1. Понятие об объектах БД.

5.2. Понятие хранимой процедуры. Описание, назначение, принцип работы.

5.3. Понятие представления. Описание, назначение, принцип работы.

5.4. Расширенный синтаксис языка SQL. Команды и инструкции.

ЭЛЕКТРОННАЯ ВЕРСИЯ
ЭЛЕКТРОННАЯ ВЕРСИЯ

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Шкарина Л. Язык SQL: учебный курс. – СПб.: Питер, 2001 – 592 с.

2. Карпова Т. Базы данных модели, разработка, реализация. – СПб., Питер. 1999. – 258 с.

3. Кириллов В.В. Основы проектирования реляционных баз данных. Учеб. пособие. – СПб.: ИТМО, 1994. – 90 с.

4. Мейер М. Теория реляционных баз данных. – М.: Мир, 1987. – 608 с.

5. Фаронов В.В. Программирование баз данных в Delphi 6. Учеб. курс. – СПб.: Питер, 2003. – 352 с.

ОГЛАВЛЕНИЕ

ЭЛЕКТРОННАЯ ВЕРСИЯ

УПРАВЛЕНИЕ ДАННЫМИ
Методические указания к лабораторным работам

Составитель
ВЕРШИНИН Виталий Васильевич

Ответственный за выпуск – зав. кафедрой профессор А.В. Костров

Редактор Е.В. Невская
Компьютерная верстка и дизайн обложки В.В. Вершинин

ЛР №020275. Подписано в печать .06.04.
Формат 60x84/16. Бумага для множит. техники. Гарнитура Таймс.
Печать на ризографе. Усл. печ. л. 2,32 Уч.-изд. л. 2,45 Тираж 150 экз.
Заказ
Редакционно-издательский комплекс
Владимирского государственного университета
600000, Владимир, ул. Горького, 87.

ЭЛЕКТРОННАЯ ВЕРСИЯ
ЭЛЕКТРОННАЯ ВЕРСИЯ