МОДЕЛИ ПРЕДСТАВЛЕНИЯ И ОБРАБОТКИ ДАННЫХ И ЗНАНИЙ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Под редакцией В. В. Голенкова

В 3-х частях

Часть 1

Н. А. ГУЛЯКИНА, Т. Л. ЛЕМЕШЕВА, В. В. АГАШКОВ

Рекомендовано Учебно-методическим объединением вузов Республики Беларусь по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для студентов учреждений, обеспечивающих получение высшего образования по специальности «Искусственный интеллект»

Минск БГУИР 2007

УДК 004.65+004.82 (075.8)
ББК 32.973 – 018.2 я 73
Г 94

Гулякина, Н. А.

Г 94 Модели представления и обработки данных и знаний. Лабораторный практикум : учеб.-метод. пособие; под ред. В. В. Голенкова : В 3 ч. Ч.1 / Н. А. Гулякина, Т. Л. Лемешева, В. В. Агашков. – Минск : БГУИР, 2007. – 44 с. : ил.

ISBN 978-985-488-177-5 (ч.1)

В первой части лабораторного практикума содержатся теоретические сведения и практические задания по представлению и обработке данных и знаний. В теоретической части приводятся основные модели представления данных и знаний, языки и программные средства для их обработки, а также примеры их использования. В практической части приводятся контрольные вопросы и варианты индивидуальных компьютерных заданий, которые касаются различных аспектов реализации приведенных моделей.

УДК 004.65+004.82 (075.8)
ББК 32.973 – 018.2 я 73

ISBN 978-985-488-177-5 (ч.1)
ISBN 978-985-488-178-2

Содержание

Предисловие

Лабораторный практикум создан по материалам лекций и лабораторных работ, проводимых в рамках учебного курса «Модели представления знаний, базы данных и интеллектуализация информационно-поисковых систем» для студентов БГУИР, с необходимыми поправками и дополнениями. В данном практикуме приводятся теоретические сведения и варианты индивидуальных заданий для четырех лабораторных работ по курсу, которые затрагивают тему организации доступа к реляционным базам данных в интерактивном режиме и из программных приложений. Приведенный материал соответствует типовой программе учебной дисциплины, утвержденной Министерством образования Республики Беларусь 24 июня 2001г. (регистрационный номер №ТД-185/тип). Он может быть использован для проведения лабораторных работ и тренингов по соответствующему учебному курсу, а также при выполнении курсовых и дипломных проектов и работ в рамках учебного плана специальности «Искусственный интеллект» и смежных специальностей.

Лабораторный практикум содержит теоретический материал и задачи для реализации на компьютере по основам многоуровневой архитектуры представления и переработки информации, трактуемой либо как элементарные данные (числа и символы), либо как система отношений на данных сложной структуры, либо как самоинтерпретируемые метаданные – знания. Задачами практикума являются изучение и приобретение практических навыков использования языков представления и переработки данных и знаний на примерах промышленных систем управления базами данных и знаний, таких как ©IBM DB2, ©Lotus Notes и MySQL.

Выполнение практикума базируется на знаниях и навыках студентов, приобретенных ими при изучении курсов «Введение в специальность», «Математические основы искусственного интеллекта», «Общая теория систем», «Аппаратное и программное обеспечение сетей», «Ассоциативная память и ассоциативные процессоры в интеллектуальных компьютерах», «Организация и функционирование традиционных и интеллектуальных компьютеров».

Объектами изучения курса являются:

- концепции архитектуры систем баз данных и знаний;
- модели представления данных и знаний;

- языковые и инструментальные средства управления/переработки данных и знаний, а также обеспечение их корректности, целостности и актуальности.

Задача курса заключается в формировании у студентов:

- знаний о базовых архитектурах систем управления базами данных (СУБД) и систем, основанных на знаниях;
- знаний о базовых моделях и методах представления данных и метаданных, а также средствах их переработки;
- знаний основных стратегий и методов формализации различных проблемных областей с использованием технологий баз данных и знаний;
- умений и навыков разработки и обслуживания баз данных и знаний в современных инструментальных средах.

Данный компьютерный практикум является дополнением учебно-методического материала, приведенного в учебном пособии «Семантическая модель сложноструктурированных баз данных и баз знаний» / В. В. Голенков, Н. А. Гулякина, О. Е. Елисеева и др. – Минск : БГУИР, 2004. – 263 с.

В работе для более наглядного и четкого восприятия текста используются специальные обозначения. Текст разбивается на нумеруемые темы, в рамках темы – на лабораторные работы. Их наименования выделяются жирным шрифтом разного размера. Каждая лабораторная работа содержит теоретические сведения по изучаемой теме, пример решения задачи, контрольные вопросы, формулировки задач, которые предлагаются студентам в ходе лабораторной работы, и варианты индивидуальных заданий.

Введение

Основными проблемами искусственного интеллекта являются представление и обработка знаний. Решение этих проблем состоит как в разработке эффективных моделей представления знаний, методов получения новых знаний, так и в создании программ, устройств, реализующих эти модели и методы. Всякая интеллектуальная деятельность опирается на знания. В эти знания включаются характеристики текущей ситуации, оценки возможности выполнения действий, законы и закономерности предметной области. База знаний является неотъемлемым компонентом любой интеллектуальной системы. Знания в базе знаний хранятся в явном формализованном виде, в отличие от знаний, которыми владеет человек. Для представления и обработки знаний необходимы специальные модели и формальные языки, а также программные средства, которые автоматизируют процессы представления и обработки знаний. На сегодняшний день существует несколько подходов к представлению и обработке знаний: модели семантических сетей различного вида (однородные, неоднородные); фреймовые модели; логические модели. На практике чаще всего используются смешанные модели: фреймово-логические и логико-семантические.

Необходимость практического использования и совершенствования систем обработки данных привела к появлению баз данных, позволяющих хранить большие объемы информации. До конца 80-х годов большие объемы данных (базы данных) в различных сферах служили главным образом лишь для описания ситуации, сложившейся на производстве. Эти данные позволяли знать, например, состояние материалов на складе, кадровый состав, некоторые характеристики. Роль систем управления базами данных (СУБД) заключалась лишь в поиске конкретной информации. В течение последнего десятилетия задачи и возможности информационных систем довольно значительно изменились. Базы данных проникли во все области информатики. Причем данные могут быть различными: от банальных «фамилия – имя – отчество – возраст – зарплата» до множеств команд или рекомендаций и инструкций. В истории развития СУБД выделяют три модели представления и обработки данных: иерархическая модель, сетевая модель, реляционная модель. На сегодняшний день с развитием объектно-ориентированного подхода в проектировании и программировании возникла новая модель данных – объектно-реляционная, или постреляционная, которая сочетает в себе

достоинства объектно-ориентированного способа описания предметной области и реляционной алгебры для обработки данных предметной области [8]. В настоящее время современные интеллектуальные информационные системы должны хранить и обрабатывать большие объемы как данных, так и знаний.

В первой части практикума рассматриваются элементы языка SQL (Structured Query Language), переход от данных к знаниям и модели представления знаний. Текущая версия стандарта языка SQL принята в 1992 г. (Официальное название стандарта – Международный стандарт языка баз данных SQL (1992) (International Standart Database Language SQL), неофициальное название – SQL2). В конце 2003 г. был принят и опубликован новый вариант международного стандарта SQL3. Все СУБД, претендующие на название «реляционные», реализуют тот или иной диалект SQL. Многие нереляционные системы также имеют в настоящее время средства доступа к реляционным данным. Целью стандартизации является переносимость приложений между различными СУБД. Нужно заметить, что в настоящее время ни одна система не реализует стандарт SQL в полном объеме. Кроме того, во всех диалектах языка имеются возможности, не являющиеся стандартными. Таким образом, можно сказать, что каждый диалект – это надмножество некоторого подмножества стандарта SQL. Это затрудняет переносимость приложений, разработанных для одних СУБД, в другие СУБД.

Язык SQL оперирует терминами, несколько отличающимися от терминов реляционной алгебры, например, вместо «отношений» используются «таблицы», вместо «кортежей» – «строки», вместо «атрибутов» – «колонки» или «столбцы». Отношение в реляционной модели данных не допускает наличия одинаковых кортежей, а таблицы в терминологии SQL могут иметь одинаковые строки. Имеются и другие отличия. Язык SQL является реляционно полным. Это означает, что любой оператор реляционной алгебры может быть выражен подходящим оператором SQL.

Основу языка SQL составляют операторы, условно разбитые на несколько групп по выполняемым функциям. Можно выделить следующие группы операторов (перечислены не все операторы SQL):

1. Операторы DDL (Data Definition Language) – операторы определения объектов базы данных:

CREATE SCHEMA – создать схему базы данных;

DROP SCHEMA – удалить схему базы данных;

CREATE TABLE – создать таблицу;

ALTER TABLE – изменить таблицу;

DROP TABLE – удалить таблицу;

CREATE VIEW – создать представление;

DROP VIEW – удалить представление.

2. Операторы DML (Data Manipulation Language) – операторы манипулирования данными:

SELECT – отобрать строки из таблиц;

INSERT – добавить строки в таблицу;

UPDATE – изменить строки в таблице;

DELETE – удалить строки в таблице;

COMMIT – зафиксировать внесенные изменения;

ROLLBACK – отменить внесенные изменения.

3. Операторы защиты и управления данными:

GRANT – предоставить привилегии пользователю или приложению на манипулирование объектами;

REVOKE – отменить привилегии пользователя или приложения.

Кроме того, есть группы операторов установки параметров сеанса, получения информации о базе данных, операторы статического SQL, операторы динамического SQL.

Описание тестового примера

В качестве тестового примера базы данных для проверки SQL-запросов на протяжении всех лабораторных работ используется гипотетическая база данных «Sample» [1], предлагаемая фирмой IBM в составе СУБД ©DB2 Universal Database [6]. Рекомендуется также использовать для сравнения СУБД ©Microsoft Access и учебную базу данных «Борей», а также СУБД MySQL.

Для того чтобы использовать базу данных «Sample» ее нужно проинсталлировать, при этом пользователь должен обладать правами SYSADM [1]. При использовании Windows-платформы пользователь должен набрать в командной строке [6]:

db2sampl *e*,

где *e* – опциональный параметр, специфицирующий диск, где будет размещена база данных. Иначе, для инсталлирования «Sample» пользователь может воспользоваться графическим интерфейсом, выбрав пункт меню «Start» –> «DB2 for Windows NT» –> «First Steps».

База данных «Sample» состоит из следующих таблиц:

Таблица В1

DEPARTMENT

Название столбца	DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
Тип данных	char(3) not null	varchar(29) not null	char(6)	char(3) not null	char(16)
Описание	Номер отдела	Название, описывающее функции отдела	Номер служащего (EMPNO), являющегося начальником отдела	Номер вышестоящего отдела (DEPTNO)	Расположение отдела

Таблица В2

EMPLOYEE

Название столбца	EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT
Тип данных	char(6) not null	varchar(12) not null	char(1) not null	varchar(15) not null	char(3)
Описание	Номер служащего	Имя служащего	Первая буква отчества	Фамилия	Номер отдела (DEPTNO)

Окончание табл. В2

Название столбца	JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
Тип данных	char(8)	smallint not null	char(1)	date	dec(9,2)	dec(9,2)	dec(9,2)
Описание	Должность	Количество лет обучения	Род (M-male, F-female)	Дата рождения	Годовой доход	Премия за год	Комиссионные за год

Таблица В3

EMP_ACT

Название столбца	EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
Тип данных	char(6) not null	char(6) not null	smallint not null	dec(5,2)	date	Date
Описание	Номер служащего	Номер проекта	Номер задания	Доля времени служащего для проекта	Дата начала выполнения задания	Дата завершения выполнения задания

Таблица В4

ORG

Название столбца	DEPTNUMB	DEPTNAME	MANAGER	DIVISION	LOCATION
Тип данных	smallint not null	varchar(14)	smallint	varchar(10)	varchar(13)

Описание	Номер отдела	Название отдела	Номер менеджера	Подразделение организации	Город

Таблица В5

PROJECT

Название столбца	PROJNO	PROJNAME	DEPTNO	RESPEMP	PRSTAFF
Тип данных	char(6) not null	varchar(24) not null	char(3) not null	char(6) not null	dec(5,2)
Описание	Номер проекта	Название проекта	Номер отдела	Номер служащего, ответственного за проект	Оценка кадрового обеспечения

Окончание табл. В5

Название столбца	PRSTDATE	PRENDATE	MAJPROJ
Тип данных	date	date	char(6)
Описание	Дата начала	Дата окончания	Главный проект для подпроектов

Таблица В6

SALES

Название столбца	SALES_DATE	SALES_PERSON	REGION	SALES
Тип данных	date	varchar(15)	varchar(15)	Int
Описание	Дата продажи	Фамилия служащего	Регион продажи	Количество продаж

Таблица В7

STAFF

Название столбца	ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
Тип данных	smallint not null	varchar(9)	smallint	char(5)	smallint	dec(7,2)	dec(7,2)
Описание	Номер служащего	Имя служащего	Номер отдела	Должность	Годы службы	Текущая зарплата	Комиссионные

Таблица DEPARTMENT содержит информацию об отделах некоторой гипотетической организации, выпускающей некоторую продукцию и распространяющую ее в различных городах. Таблица EMPLOYEE содержит информацию о сотрудниках организации. Таблица EMP_ACT содержит информацию об активности сотрудников, т.е. какие проекты и какие задания в проектах они выполняют и в течение некоторого промежутка времени. Таблица ORG содержит информацию о подразделениях организации. Таблица PROJECT содержит информацию о проектах, выполняемых данной организацией. Таблица SALES содержит информацию о продажах. Таблица STAFF содержит

информацию о служащих, годовой заработной плате, комиссионных, годах службы.

Лабораторная работа №1
Создание схемы базы данных

Теоретические сведения

Схема базы данных – это коллекция именованных объектов в базе данных. Схема позволяет логически классифицировать объекты базы данных, например, таблицы, виды, пользователи, триггеры, функции и пакеты. Сама схема является также объектом базы данных. При создании базы данных, например в DB2, все пользователи имеют права IMPLICIT_SCHEMA. Это дает возможность создавать любые объекты базы данных в рамках данной схемы. Если данная схема запрещена для пользователей группы PUBLIC, то она все равно может быть создана при помощи соответствующего оператора CREATE SCHEMA (см. ниже) пользователем, который наделен правами создания этой схемы [3, 4, 7].

С помощью оператора определения схемы можно за одно обращение к СУБД создать все необходимые объекты, связанные с указанным идентификатором авторизации. Этот оператор имеет следующий синтаксис:

```
<schema definition> ::=
       CREATE SCHEMA <schema name clause>
       [ <schema element>... ]
<schema name clause> ::=
       <schema name>
       |AUTHORIZATION <schema authorization identifier>
       |<schema name> AUTHORIZATION <schema authorization identifier>
<schema element> ::=
       <table definition>
```

При выполнении оператора определения схемы создаются объекты, определяемые элементами схемы (базовые таблицы, представления, ограничения, домены, утверждения целостности, наборы символов, привилегии и т.д.), и их соответствующие описатели. Операторы, используемые для определения элементов схемы, можно использовать независимо, вне оператора определения схемы. Например, пользователь с правами администратора создает схему с именем lab1 для пользователя db2admin:

CREATE SCHEMA lab1 **AUTHORIZATION** db2admin

Созданную ранее схему можно уничтожить. Для уничтожения схемы используется оператор DROP SCHEMA.

Оператор создания таблицы определяется следующими синтаксическими правилами:

```
<table definition> ::=
       CREATE TABLE  <table name> <table element list>
    <table element list> ::=
       <(> <table element> [ { <,> <table element> }... ] <)>
<table element> ::=
       <column definition>
       |<table constraint definition>
```

Определение таблицы должно содержать по крайней мере одно определение столбца. При выполнении оператора определения таблицы создается набор описателей привилегий, определяющих привилегии INSERT, SELECT, UPDATE, DELETE и REFERENCES на этой таблице и INSERT, SELECT, UPDATE и REFERENCES для каждого определения столбца. Все привилегии являются передаваемыми и относящимися к идентификатору авторизации из определения схемы или модуля в зависимости от того, где выполняется оператор определения таблицы.

Определение столбца имеет следующий синтаксис:

```
<column definition> ::=
       <column name> { <data type> <domain name> }
       [ <default clause> ]
       [ <column constraint definition>... ]
       [ <collate clause> ]
<column constraint definition> ::=
       [ <constraint name definition> ]
       <column constraint>
<constraint name definition> ::= CONSTRAINT <constraint name>
<column constraint> ::=
       NOT NULL
       |<unique specification>
       |<references specification>
       |<check constraint definition>
```

Определение столбца может входить в оператор определения таблицы (CREATE TABLE), в оператор изменения схемы таблицы (ALTER TABLE). Пусть T обозначает соответствующую таблицу. Номер столбца соответствует позиции определения этого столбца в операторе CREATE TABLE, т.е. i-й столбец таблицы описывается i-м определением столбца в определении таблицы. Имя и тип данных или домен столбца специфицируется именем столбца и типом данных или именем домена соответственно.

Пользователь A может создавать таблицы в рамках схемы C. Данная команда создаст таблицу student в схеме lab1 с единственным столбцом stud_id типа INT:

CREATE TABLE lab1.student (stud_id INT)

При определении столбца можно указать значение, которое должно присваиваться этому столбцу по умолчанию, если при занесении новых строк в соответствующую таблицу значение данного столбца не задается явно.

Ограничение уникальности определяется следующими синтаксическими правилами:

```
<unique constraint definition> ::=
       <unique specification> <(> <unique column list> <)>
<unique specification> ::=
       UNIQUE | PRIMARY KEY
<unique column list> ::=
       <column name list>
```

Ссылочное ограничение определяется следующими синтаксическими правилами:

```
<referential constraint definition> ::=
       FOREIGN KEY <(> <referencing columns><)> <references specification>
<references specification> ::=
       REFERENCES <referenced table and columns>
       [ <referential triggered action> ]
<referencing columns> ::= <reference column list>
<referenced table and columns> ::=
       <table name>
       [ <(> <reference column list> <)> ]
<reference column list> ::= <column name list>
<referential triggered action> ::=
       <update rule> [ <delete rule> ]
       |<delete rule> [ <update rule> ]
<update rule> ::= ON UPDATE <referential action>
<delete rule> ::= ON DELETE <referential action>
<referential action> ::=
       CASCADE
      |SET NULL
      |SET DEFAULT
      |NO ACTION
```

Ссылающейся таблицей будем называть таблицу, фигурирующую в определении таблицы или изменении схемы таблицы (оператор ALTER TABLE). Таблицей, на которую указывает ссылка, будем называть таблицу, имя которой указывается в разделе referenced table and columns. Ссылающимися столбцами будем называть столбцы, перечисленные в списке столбцов-ссылок (reference column list). Если в разделе referenced table and columns

специфицирован список столбцов, то набор имен столбцов из этого списка должен совпадать с набором столбцов уникальности, содержащимся в ограничении уникальности таблицы, на которую указывает ссылка. Если раздел referenced table and columns не содержит списка столбцов, то для таблицы, на которую указывает ссылка, должно быть определено ограничение уникальности с ключевым словом PRIMARY KEY. Проверочное ограничение целостности определяется следующим синтаксическим правилом:

```
<check constraint definition> ::=
       CHECK <(> <search condition> <)>
```

Если в спецификации оператора удаления столбца указано RESTRICT, то на удаляемый столбец не должно быть ссылок в выражении запроса любого существующего представления и в условии поиска любого существующего ограничения, кроме табличных ограничений, ссылающихся только на этот столбец и входящих в описание данной таблицы. Если в определении удаления столбца указано CASCADE, то любой зависимый объект уничтожается при выполнении неявного оператора REVOKE (см. ниже).

Спецификация оператора добавления табличного ограничения имеет следующий синтаксис:

```
<add table constraint definition> ::=
       ADD <table constraint definition>
```

Представляемая таблица, или просто представление создается с помощью оператора CREATE VIEW:

```
<view definition> ::=
       CREATE VIEW <table name> [ <(> <view column list> <)> ]
       AS <query expression>
       [ WITH [ <levels clause> ] CHECK OPTION ]
<levels clause> ::=
       CASCADED | LOCAL
<view column list> ::= <column name list>
```

Число имен столбцов в списке столбцов представления должно совпадать со степенью таблицы, специфицированной выражением запроса. Если в определении представления содержится раздел WITH CHECK OPTION, то представление должно быть обновляемым. Если специфицировано WITH CHECK OPTION без задания раздела уровней (levels clause), то по умолчанию полагается указание класса CASCADED. Под операцией обновления понимаются операции INSERT и UPDATE.

Для каждого из упомянутых в этой теме операторов создания элементов схемы базы данных имеются обратные операторы уничтожения этих элементов.

Созданная ранее таблица может быть уничтожена следующим оператором:

```
<drop table statement> ::=
       DROP TABLE <table name> <drop behavior>
```

Рассмотрим пример базы данных слушателей курсов (рис. 1.1).

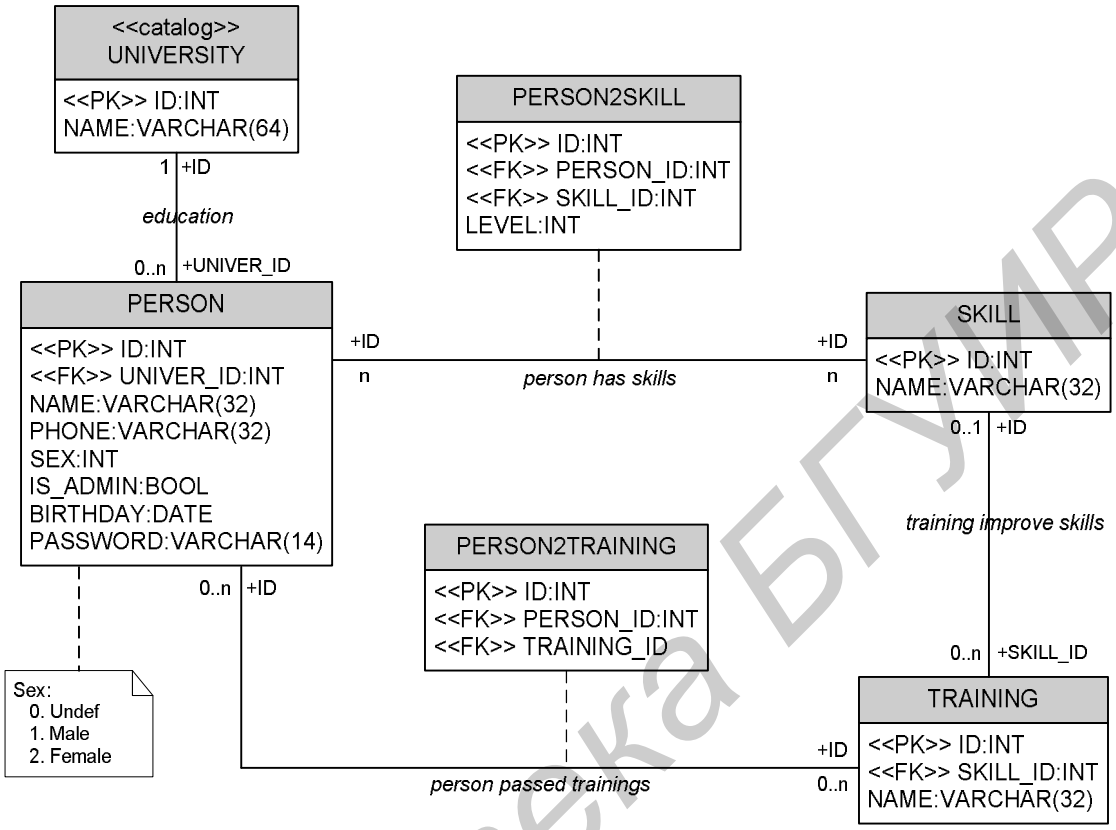


Рис.1.1. U ML-диаграмма базы данных «Слушатели курсов».

База содержит семь таблиц: «PERSON» содержит данные о студентах, изучающих некоторый курс; «TRAINING» содержит данные об изучаемом курсе; «SKILL» содержит данные о приобретаемых в процессе тренинга профессиональных навыках; «PERSON2SKILL» содержит связи между студентами и профессиональными навыками, которые они приобрели; «PERSON2TRAINING» содержит связи между студентом и изучаемым курсом; «UNIVERSITY» содержит список университетов; «SEX» содержит идентификаторы пола.

Для создания схемы базы данных «Слушатели курсов» необходимо выполнить следующий запрос:

```
create table university (
        id int NOT NULL default 0,
        name varchar(64) NOT NULL default '',
        primary key (id) );
create table person (
        id int NOT NULL default 0,
        univer_id int,
```

```
        login varchar(14) NOT NULL default '',
        password varchar(20) NOT NULL default '',
        name varchar(64),
        phone varchar(32),
        birthday date,
        sex int NOT NULL default 0,
        is_admin int NOT NULL default 0,
        primary key (id) );
alter table person
        add constraint p_u foreign key (univer_id) references university (id);
create table skill (
        id int NOT NULL default 0,
        name varchar(32) NOT NULL default '',
        primary key (id) );
create table training (
        id int NOT NULL default 0,
        skill_id int NOT NULL default 0,
        name varchar(32) NOT NULL default '',
        primary key (id) );
alter table training
        add constraint t_s foreign key (skill_id) references skill (id);
create table person2training (
        id int NOT NULL default 0,
        person_id int NOT NULL default 0,
        training_id int NOT NULL default 0,
        primary key (id) );
alter table person2training
        add constraint p2t_p foreign key (person_id) references person (id) on delete cascade,
        add constraint p2t_t foreign key (training_id) references training (id);
create table person2skill (
        id int NOT NULL default 0,
        person_id int NOT NULL default 0,
        skill_id int NOT NULL default 0,
        level int NOT NULL default 0,
        primary key (id) );
alter table person2skill
        add constraint p2s_p foreign key (person_id) references person (id) on delete cascade,
        add constraint p2s_s foreign key (skill_id) references skill (id);
```

Контрольные вопросы

1. Какой идентификатор схемы базы данных в операторе CREATE SCHEMA используется по умолчанию?

2. Что произойдет, если в операторе DROP SCHEMA специфицировано Restrict, а к моменту выполнения этого оператора в нем еще содержатся объекты?

3. Что означает ключевое слово CASCADE в операторе DROP SCHEMA?

4. Может ли имя схемы входить в имя таблицы?

5. Может ли определение таблицы не содержать ни одного определения столбца?

6. Может ли пользователь, который выполнил оператор определения таблицы, передавать при помощи оператора GRANT привилегии другим пользователям, если да, то какие это привилегии?

7. Каким образом задается порядок столбцов в таблице?

8. Поясните понятия check-ограничение таблицы и check-ограничение столбца.

9. Что означает ключевое слово PRIMARY KEY в ограничении уникальности таблицы?

10. Дайте определения понятий «ссылающаяся таблица» и «родительская таблица».

11. Может ли check-ограничение содержать агрегатную функцию?

12. Можно ли изменять схему таблицы?

13. Можно ли добавить столбец из другой таблицы, уже содержащий значения, к новой таблице?

14. Каким образом можно отменить значение столбца по умолчанию?

15. Определите степень любой таблицы из примера.

16. Может ли имя определяемого представления содержаться в какой-либо ссылке на таблицу, содержащейся в выражении запроса (другими словами, допускаются ли рекурсивные определения представлений)?

17. Приведите пример случая, когда представление не является обновляемым.

18. Поясните назначение домена.

Задание

Создать схему реляционной базы данных для заданной предметной области. Все объекты базы данных должны быть определены в операторе создания схемы. Для выполнения задания использовать рекомендуемые преподавателем СУБД. После окончания лабораторной работы необходимо сдать отчет, в котором должна содержаться диаграмма объектов базы данных с пояснениями, распечатка запроса, а также ответы на контрольные вопросы.

Варианты индивидуальных заданий

1. База данных студентов университета.
2. База данных преподавателей университета.
3. База данных подразделений университета.
4. База данных библиотеки.
5. База данных успеваемости студентов.
6. База данных материально-технического оборудования кафедры.
7. База данных студентов, проживающих в общежитии.
8. База данных материально-технического обеспечения общежития.
9. База данных расписания занятий.

10. База данных программного обеспечения.
11. База данных курсовых проектов.
12. База данных лабораторных работ по некоторой дисциплине.
13. База данных мероприятий (конференции, семинары, выставки).
14. База данных абитуриентов.
15. База данных документооборота.

Лабораторная работа №2
Создание запросов с подзапросами

Теоретические сведения

Первый запрос, который должен выполнить пользователь, начиная работу с базой данных, – это запрос на подключение к базе данных. Для подключения к серверу баз данных в стандарте SQL предусмотрен оператор CONNECT TO. Для управления подключением служат три оператора (CONNECT, SET CONNECTION и DISCONNECT) [3, 4, 7].

С выполнения оператора подключения начинается сеанс взаимодействия с указанным сервером баз данных. Оператор описывается следующими синтаксическими правилами:

```
<connect statement> ::=
       CONNECT TO <connection target>
<connection target> ::=
       <SQL-server name>
       [ AS <connection name> ]
       [ USER <user name> ]
       | DEFAULT
```

Например, для подключения к базе данных «Sample» необходимо в интерактивном режиме набрать следующее предложение:

CONNECT TO sample **USER** db2admin

Введем некоторые начальные понятия, связанные с запросами. Наверное, наиболее важным из таких понятий в SQL/92 является ссылка на таблицу (table reference), которая может встречаться во всех конструкциях, адресуемых к таблицам. Ссылка на таблицу определяется следующими синтаксическими правилами:

```
<table reference> ::=  <table name> [ [ AS ] <correlation name>
       [ <(> <derived column list> <)> ] ]
       |<derived table> [ AS ] <correlation name>
       [ <(> <derived column list> <)> ]
       |<joined table>

<derived table> ::= <table subquery>

<derived column list> ::= <column name list>

<column name list> ::=
       <column name> [ { <,> <column name> }... ]
```

Выражение <table subquery> означает, что таблица формируется при помощи Select-подзапроса (см. ниже). Выражение <correlation name> означает псевдоним, который присваивается объекту базы данных и в дальнейшем может использоваться вместо имени.

Табличное выражение в качестве результата выдает обычную или сгруппированную таблицу:

```
<table expression> ::=
       <from clause>
       [ <where clause> ]
       [ <group by clause> ]
       [ <having clause> ]

<from clause> ::=
       FROM <table reference>
       [ { <,> <table reference> }... ]

<where clause> ::= WHERE <search condition>

<group by clause> ::= GROUP BY <grouping expression>

<having clause> ::= <search condition>
```

Если в списке FROM больше чем одна таблица, то результатом будет декартово произведение всех этих таблиц.

Выражение <group by clause> возвращает сгруппированные строки. В каждой группе выражение <grouping expression> должно быть одинаковым для каждой строки. Например, группируем сотрудников по номеру отдела deptno:

GROUP BY deptno

Выражение <having clause> возвращает истину, если для каждой строки из группы выполняется <search condition>. Если хотя бы одна строка из группы не удовлетворяет условию <search condition>, то выражение <having clause> возвращает ложь и вся группа удаляется из результирующего множества. Если перед HAVING не использовалось GROUP BY, то весь столбец считается одной группой.

Контекстом табличных выражений является запрос. Синтаксис спецификации запросов следующий:

```
<query specification> ::=
       SELECT [ ALL | DISTINCT ] <select list> <table expression>

<select list> ::=
       <*> | <select sublist> [ { <,> <select sublist> }... ]

<select sublist> ::=
       <derived column>
```

```
       |<qualifier><.><*>

<derived column> ::= <value expression> [ <as clause> ]

<as clause> ::= [ AS ] <column name>
```

ALL – отображает все строки результирующей таблицы. Используется по умолчанию.

DISTINCT – отображает все недублирующиеся строки. Максимальная длина значения строкового столбца результирующей таблицы не может быть больше 254 байт. Ключевое слово DISTINCT может быть использовано неоднократно в подзапросах. Дублирующимися называются строки, в которых все значения соответствующих столбцов совпадают. Два NULL-значения считаются равными.

* – отображает в результирующую таблицу все колонки исходной таблицы.

<qualifier> может быть именем таблицы, вида или псевдонимом и должен совпадать с именем в <from-clause>. Количество колонок – не больше 500.

Например, следующие запросы выбирают значения всех столбцов из таблицы employee:

```
SELECT * FROM employee;

SELECT employee.* FROM employee;

SELECT empno,    firstnme,   midinit,
       lastname, workdept,   phoneno,
       hiredate, job,        edlevel,
       sex,      birthdate,  salary,
       bonus,    comm
    FROM employee
```

Соединенная таблица является альтернативным способом порождения новой таблицы на основе использования ранее определенных таблиц. Формально соединенная таблица определяется следующими синтаксическими правилами:

```
<joined table> ::=
       <qualified join>
       |<(> <joined table> <)>

<qualified join> ::=
       <table reference> [ <join type> ] JOIN <table reference>

<join condition> ::= ON <search condition>

<join type> ::=
       INNER
```

```
       |<outer join type> [ OUTER ]

<outer join type> ::=
       LEFT
       |RIGHT
       |FULL
```

Выражение <joined-table> – специфицирует промежуточную таблицу, которая является результатом применения одного из следующих операторов: INNER, LEFT OUTER, RIGHT OUTER, FULL OUTER:

- T1 INNER JOIN T2 – соединение пар строк, для которых истинно условие соединения.
- T1 LEFT OUTER JOIN T2 – соединение пар строк, для которых истинно условие соединения, таким образом, что для каждой строки из T1, для которой нет парной строки из T2, добавляются NULL-значения.
- T1 RIGHT OUTER JOIN T2 – для каждой строки из T2, для которой нет парной строки из T1, добавляются NULL-значения.
- T1 FULL OUTER JOIN T2 – для каждой строки из T1, для которой нет парной строки из T2, добавляются NULL-значения, и для каждой строки из T2, для которой нет парной строки из T1, добавляются NULL-значения. Все колонки, в которые будут записываться значения из T1 и T2, допускают NULL-значения.

При использовании INNER JOIN каждая строка из левой таблицы T1 объединяется с каждой строкой из правой таблицы T2:

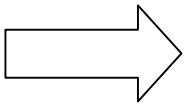
SELECT C.A1, C.A2, C.A3, C.B1 **FROM** T1 **INNER JOIN** T2 **AS** C

T1

A1	A2	A3
A	b	c
D	e	f
D	s	a
B	a	e

T2

A1	B1
a	2
d	4
c	3



C

A1	A2	A3	B1
a	b	c	2
d	e	f	4
d	s	a	4

Если специфицировано <search condition>, то в результирующей таблице остаются только те строки, которые удовлетворяют данному условию.

В стандарте SQL специфицированы три вида подзапросов: скалярный, строчный и табличный. Скалярный подзапрос выдает одно значение некоторого типа; строчный подзапрос выдает одну строку; табличный подзапрос выдает таблицу. В основе каждого вида подзапроса лежит

табличное выражение. Синтаксически соответствующие конструкции определяются следующими правилами:

```
<scalar subquery> ::= <subquery>
<row subquery> ::= <subquery>
<table subquery> ::= <subquery>
<subquery> ::= <(> <query expression> <)>
<query expression> ::=
       <non-join query expression>
       |<joined table>
<non-join query expression> ::=
       <non-join query term>
       |<query expression> UNION [ ALL ]
       |<query term> <query expression> EXCEPT [ ALL ]
       <query term>
<query term> ::=
       <non-join query term>
       |<joined table>
<non-join query term> ::=
       <non-join query primary>
       |<query term> INTERSECT [ ALL ]
       [ <corresponding spec> ] <query primary>
<query primary> ::=
       <non-join query primary>
       |<joined table>
<non-join query primary> ::=
       <simple table>
       |<(> <non-join query expression> <)>
<simple table> ::=
       <query specification>
       |<table value constructor>
       |<explicit table>
<explicit table> ::= TABLE <table name>
```

Предикат позволяет специфицировать условие, результатом вычисления которого может быть true, false или unknown. Допустимы следующие предикаты:

```
<predicate> ::=
       <comparison predicate>
       |<between predicate>
       |<in predicate>
       |<like predicate>
       |<null predicate>
       |<quantified comparison predicate>
       |<exists predicate>
       |<unique predicate>
```

Предикат сравнения предназначен для спецификации сравнения двух строчных значений. Синтаксис предиката следующий:

```
<comparison predicate> ::=
       <row value constructor> <comp op> <row value constructor>
<comp op> ::=
       <equals operator>
       |<not equals operator>
       |<less than operator>
```

```
        |<greater than operator>
        |<less than or equals operator>
        |<greater than or equals operator>
```

Если X и/или Y являются неопределенными значениями, то значение условия «X <comp op> Y» есть unknown.

Предикат BETWEEN позволяет специфицировать условие вхождения в диапазон значений:

```
<between predicate> ::=
       <row value constructor> [ NOT ] BETWEEN
       <row value constructor> AND <row value constructor>
```

Все три строки-операнды должны иметь одну и ту же степень. Пусть X, Y и Z обозначают первый, второй и третий операнды.

«X **NOT BETWEEN** Y **AND** Z» эквивалентно «**NOT** (X **BETWEEN** Y **AND** Z)».
«X **BETWEEN** Y **AND** Z» эквивалентно «X >= Y **AND** X <= Z».

Предикат IN позволяет специфицировать условие вхождения строчного значения в указанное множество значений:

```
<in predicate> ::=
       <row value constructor>
       [ NOT ] IN <in predicate value>

<in predicate value> ::=
       <table subquery>
       |<(> <in value list> <)>

<in value list> ::=
       <value expression> { <,> <value expression> }...
```

Предикат LIKE позволяет проверить, удовлетворяет ли строковое выражение заданному шаблону:

```
<like predicate> ::=
       <match value> [ NOT ] LIKE <pattern>
       [ ESCAPE <escape character> ]
<match value> ::= <character value expression>
<pattern> ::= <character value expression>
<escape character> ::= <character value expression>
```

Например, следующий запрос позволяет выбрать все проекты, в названии которых встречается подстрока «SYSTEM».

SELECT PROJNAME **FROM** PROJECT
WHERE PROJECT.PROJNAME **LIKE** ‘%SYSTEMS%’

Ключевое слово ESCAPE специфицирует символ, который, если встретится в строке, указывает на то, что любой следующий за ним символ

является значимым. Например, на следующий запрос будут найдены все строки, которые начинаются с подстроки «%_\».

SELECT A **FROM** T **WHERE** T.A **LIKE** '\%_\\%' **ESCAPE** '\'

Предикат null позволяет проверить, не является ли значение неопределенным:

```
<null predicate> ::= <row value constructor> IS [ NOT ] NULL
```

Полная семантика предиката null приведена в следующей таблице:

Условие	R **IS NULL**	R **IS NOT NULL**	**NOT** R **IS NULL**	**NOT** R **IS NOT NULL**
Степень 1: null	true	false	false	true
Степень 1: not null	false	true	true	false
Степень > 1: все null	true	false	false	true
Степень > 1: есть null	false	false	true	true
Степень > 1: нет null	false	true	true	false

Предикат сравнения с квантором:

```
<quantified comparison predicate> ::=
       <row value constructor> <comp op> <quantifier> <table subquery>
<quantifier> ::= <all> <some>
<all> ::= ALL
<some> ::= SOME | ANY
```

Степень первого операнда должна быть такой же, как и степень табличного подзапроса. Например, следующий запрос возвращает столбец colA, состоящий из двух значений (2, 3):

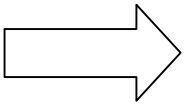
SELECT colA **FROM** tblAB
WHERE colA = **ANY (SELECT** colX **FROM** tblXY **)**

tblAB

colA	colB
1	12
2	12
3	13
4	14

tblXY

colX	colY
2	22
3	33
1	11



C

colA
2
3

Предикат EXISTS:

```
<exists predicate> ::= EXISTS <table subquery>
```

Если мощность табличного подзапроса больше нуля, то значение предиката есть true; иначе – false.

Рассмотрим использование подзапросов на следующем примере: сделать выборку из таблицы department (отделы). Вывести deptno (номер отдела), deptname (название отдела), среднюю зарплату в отделе и количество сотрудников в отделе.

```
SELECT d.deptno, d.deptname, empinfo.avgsal, empinfo.empcount
    FROM department d, TABLE ( SELECT INTEGER(AVG(e.salary)) AS avgsal,
                                      COUNT(*) AS empcount
                               FROM employee e
                               WHERE e.workdept=d.deptno) AS empinfo
```

Результатом выполнения запроса является следующая таблица:

DEPTNO	DEPTNAME	AVGSAL	EMPCOUNT
A00	SPIFFY COMPUTER SERVICE DIV.	42833	4
B01	PLANNING	41250	1
C01	INFORMATION CENTER	30156	3
D01	DEVELOPMENT CENTER	-	0
D11	MANUFACTURING SYSTEMS	24677	9
D21	ADMINISTRATION SYSTEMS	25153	7
E01	SUPPORT SERVICES	40175	1
E11	OPERATIONS	20998	5
E21	SOFTWARE SUPPORT	23827	4

Контрольные вопросы

1. Если в операторе CONNECT TO не указано ключевое слово USER, то какой пользователь используется для доступа к базе данных?

2. Что возвращает следующий запрос: SELECT empno,deptname FROM employee, department WHERE employee.workdept=department.deptno? Поясните алгоритм выполнения запроса.

3. Что означает ключевое слово DISTINCT?

4. Вычислить значение условного выражения и объяснить, как было получено решение:

а) «испытание» ≤ «испытания»

б) «испытуемый» ≥ «испытываемый»

в) «качество» ≥ «качественный»

г) 00111100 < 00111101

д) 00111100 < 0011101

е) 435 < null.

5. Построить запрос, по которому будут найдены все строки, включающие следующую подстроку: «%ibm_class@».

6. Вычислить значение предиката: («Иванов», «Клерк», null, «21/3/1975») IS NOT NULL.

7. Вычислить значение предиката:

SELECT EMPNO

FROM TEMPL

WHERE EXISTS (SELECT * FROM TEMPL **WHERE** SALARY < 100**)**

TEMPL

SALARY	EMPNO
100	1
200	2
100	3

8. Вычислить значение предиката:

employee.salary > **ALL (SELECT** salary **FROM** staff **).**

9. Поясните семантику sql-выражений: a) select a from t order by a, having a > c; b) select a from t having a > c; c) select a from t where a > c.

Задание

Необходимо реализовать указанный в варианте запрос к базе данных «Sample» с использованием конструкции подзапроса.

Варианты индивидуальных заданий

1. Сделать выборку из таблицы department (отделы). Вывести deptno (номер отдела), empno (номер менеджера, где job = «MANAGER»), и среднюю зарплату в этом отделе, где зарплата (salary) менеджера больше средней зарплаты по отделу.

2. Сгруппировать служащих таблицы employee (служащие) по отделам (workdept). Вывести workdept (номер отдела), в котором нет ни одной женщины (sex = «M»).

3. Сделать выборку из таблицы project (проекты). Вывести deptno (номер отдела) и количество проектов в каждом отделе.

4. Сгруппировать записи таблицы sales (продажи) по служащим. Сделать выборку из таблицы sales. Вывести sales_person (фамилия продавца) и количество регионов, в которые они осуществляли продажи (sales).

5. Сделать выборку из таблицы emp_act (активность работников). Вывести empno (номер сотрудника) и количество проектов, в которых он участвует.

6. Сделать выборку из таблицы emp_act (активность работников). Вывести projno (номер проекта) и количество actno (заданий) в каждом проекте.

7. Сделать выборку из таблицы employee (служащие). Вывести job (название должности), количество служащих, занимающих эту должность, и empno (номер служащего). Сгруппировать служащих (empno) по должностям (job).

8. Сделать выборку из таблицы employee (служащие). Вывести workdept (номер отдела), в которых женщин (sex = «F») больше, чем мужчин (sex = «M»), а также количество женщин в этом отделе и количество мужчин.

9. Сделать выборку из таблицы sales (продажи). Вывести sales_person (фамилия служащего), region (регион) и вычислить общее количество (sum()) продаж (sales), которые осуществил данный служащий в данном регионе.

10. Сгруппировать служащих таблицы employee по отделам (workdept), а внутри отдела объединить служащих, у которых дни рождения (birthdate) в одном месяце.

11. Сгруппировать служащих таблицы employee по отделам (workdept), а внутри отдела объединить в группы служащих, у которых одинаковый уровень образования (edlevel).

12. Отсортировать служащих таблицы employee по отделам (workdept), а внутри отдела – по должности (job). Вычислить декартово произведение таблиц employee и department. Вывести в результирующей таблице следующие колонки: empno, workdept, deptname, job.

13. Сделать выборку из таблицы employee (служащие). Для всех уровней образования (edlevel) вывести количество служащих с одинаковым уровнем образования. Сгруппировать служащих (empno) по уровню образования (edlevel). Вывести номер служащего (empno), уровень образования (edlevel), количество служащих с этим уровнем образования в строке для каждого служащего.

14. Сделать выборку из таблицы employee (служащие). Вычислить среднюю зарплату в организации, т.е. по всей таблице employee, и количество служащих, получающих эту зарплату (salary). Вывести номер служащего (empno), зарплату (salary), среднюю зарплату по отделу в строке для каждого служащего.

Лабораторная работа №3
Доступ к реляционным базам данных из внешних приложений

Теоретические сведения

Для создания таблиц в СУБД MySQL необходимо подсоединиться с использованием клиентской консоли к SQL-серверу как пользователь, который имеет права на создание таблиц в базе данных [2]. Пусть пользователь tms_dbuser обладает правами администратора для базы tms_db:

```
C:\> mysql –-user=tms_dbuser –-password=tms_dbpassword
```

Далее подключаемся к базе данных и создаем в ней таблицы.

```
mysql> USE tms_db
mysql> CREATE TABLE abbrev (
    -> word CHAR(255) NOT NULL,
    -> translation TEXT NOT NULL,
    -> INDEX word_index (word),
    -> );
```

Чтобы каждый раз не набирать SQL-запрос вручную, можно подготовить файл сценария в текстовом редакторе и потом запустить его на выполнение с консоли. Подготовьте в текстовом редакторе файл (назовите его, например create.sql). Используйте команды sql из лабораторной работы №1.

Запустить на исполнение сценарий можно любой из следующих команд:

```
mysql> ./ create.sql
```

```
mysql> source create.sql
```

Если текстовый файл находится не в текущем каталоге, то необходимо указать полный путь к файлу. После завершения работы сценария необходимо проверить, создались ли новые таблицы в базе данных:

```
mysql> SHOW TABLES;
```

Результат можно увидеть в консоли (рис. 3.1).

Рис. 3.1. Результат выполнения команды SHOW TABLES

Для того чтобы загрузить данные в таблицы, используем JDBC (Java Database Connectivity) API.

Инструмент JDBC API предоставляет универсальный доступ к данным из Java-программ. Для доступа к базе данных необходим драйвер – специальный Java-класс, с помощью которого создается соединение с SQL-сервером.

Список JDBC-драйверов поддерживается фирмой Sun и находится по адресу: http://industry.java.sun.com/products/jdbc/drivers.

Для СУБД MySQL будем использовать драйвер MM.MySQL, его можно найти по адресу http://mmmysql.sourceforge.net. MM.MySQL – драйвер четвертого типа. Что это означает:

1. Драйверы первого типа являлись мостами ODBC-JDBC, они требовали наличия какого-либо ODBC-драйвера.

2. Драйверы второго типа известны как драйверы, частично написанные на Java. Они переводили вызовы JDBC API в непосредственно вызовы API базы данных. На машине с клиентским приложением должна была быть установлена соответствующая бинарная клиентская библиотека от производителя СУБД (Oracle, DB2, Sybase).

3. Драйверы третьего типа – драйверы, написанные на чистом языке Java, которые трансформировали вызовы JDBC API в независимый от базы данных протокол. Драйвер не соединялся непосредственно с базой данных, а соединялся с связующим промежуточным сервером (middleware server). Для перенастройки Java-приложения на другую базу данных требовалось поменять настройки промежуточного сервера. Недостаток этого метода – большая загрузка системы из-за дополнительного связующего звена. С другой стороны, преимуществом является то, что если приложение обращается ко многим типам баз данных, оно сможет делать это, используя лишь один JDBC-драйвер.

4. Драйверы четвертого типа – это драйверы, написанные на чистом языке Java, которые связываются непосредственно с базой данных. Разработчики считают этот тип драйверов оптимальным, так как он обеспечивает хорошую производительность и позволяет программистам использовать все специфичные для конкретной СУБД функции. Конечно, тесная привязанность к конкретному типу СУБД может уменьшить гибкость, особенно если вы захотите перенастроить на другой тип СУБД. Преимуществом драйвера является то, что вы можете писать распределенные приложения (distributed

applications), когда хост клиентского приложения напрямую связывается с хостом сервера базы данных (апплеты и др.)

Файл с драйвером mm.mysql-2.0.7-you-must-unjar-me.jar необходимо распаковать, например:

```
c:\> jar -xvf
 mm.mysql-2.0.7-you-must-unjar-me.jar
```

Далее файл драйвера mm.mysql-2.0.7-bin.jar необходимо скопировать в новый каталог, например c:\mysql\jdbc (создайте его сами).

Для того чтобы Java-программы смогли создавать экземпляры данного класса на этапе выполнения, он должен быть указан в системной переменной окружения CLASSPATH. Это можно сделать либо на этапе запуска программы через командную строку:

```
java -cp
   путь_к_классу_драйвера\
   имя_jar_архива_драйвера.jar имя_программы
```

либо через Control Panel в Windows NT (Start –> Settings –> Control Panel –> –>System –> закладка Environment) (рис. 3.2).

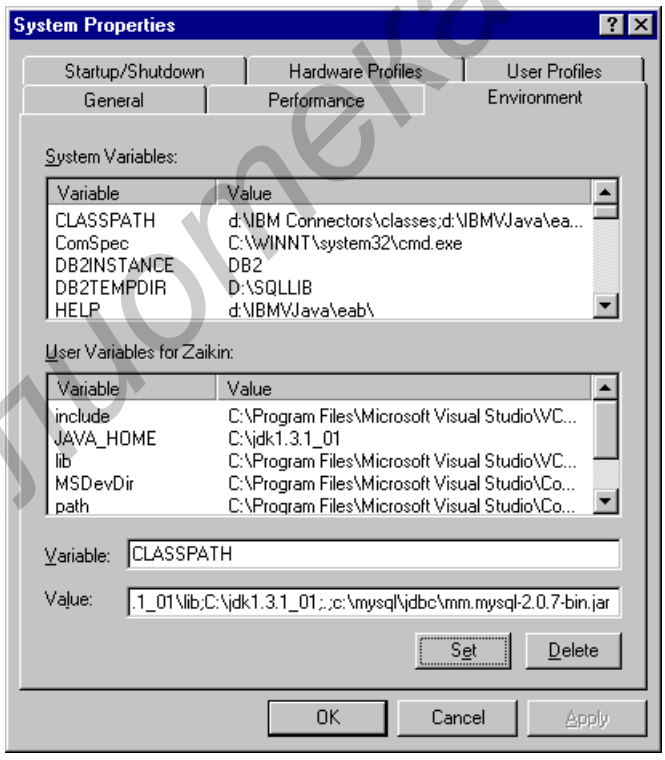


Рис. 3.2. Результат выполнения команды SHOW TABLES

Теперь напишем класс, устанавливающий соединение с сервером базы данных. Соединение с базой данных выполняется в несколько шагов.

Шаг 1. Создаем экземпляр класса драйвера и загружаем в JVM (при этом класс регистрируется в менеджере драйверов):

```
static {
   DriverManager.registerDriver(new org.gjt.mm.mysql.Driver());
}
```

или непосредственно при помощи ClassLoader:

```
Class.forName("org.gjt.mm.mysql.Driver");
```

или используя системное свойство jdbc.drivers:

```
java -Djdbc.drivers=org.gjt.mm.mysql.Driver имя_программы
```

в самой программе:

```
System.setProperty("jdbc.drivers", "org.gjt.mm.mysql.Driver");
```

Используя двоеточие, можно загружать несколько драйверов. Преимущества этого метода – отсутствие необходимости модификации кода при смене класса драйвера. Если загружено несколько драйверов, их приоритет следующий:

1. JDBC-драйверы, загруженные через свойство jdbc.drivers при инициализации JVM.

2. JDBC-драйверы, загруженные динамически (runtime).

Так как свойство jdbc.drivers проверяется единожды при вызове методов класса DriverManager, перед первой установкой соединения с БД важно удостовериться, что драйверы правильно зарегистрированы.

Шаг 2. Составляем правильный URL для создания соединения (Connection), который включает в себя следующие части:

- jdbc – с этой строки начинаются все URL для работы с JDBC API;
- имя СУБД (mysql);
- имя хоста (опционально порт) с SQL сервером (localhost);
- имя БД (mdictdb);
- имя пользователя БД (mdictuser);
- пароль пользователя БД (mdictpassword):

```
static String DBUrl = "jdbc:mysql://localhost/mdictdb?
  user=mdictuser&password=mdictpassword";
```

Шаг 3. Получаем соединение от менеджера драйверов. Он автоматически по URL определяет, какой JDBC драйвер следует использовать для доступа к данной СУБД:

```
Connection c = DriverManager.getConnection(DBUrl);
```

Альтернативный способ – задание имени пользователя и пароля в параметрах метода getConnection():

```
Connection c = DriverManager.getConnection(DBUrl, "mdictuser", "mdictpassword");
```

Далее приведен код класса доступа к базе данных by.bsuir.mpzbd.DBUtil:

```
package by.bsuir.mpzbd;
import java.sql.*;
public class DBUtil {
    public static Connection createConnection(String DBUrl) {
       Connection c = null;
       try {
        Class.forName("org.gjt.mm.mysql.Driver").newInstance();
        System.out.println("Driver loaded !");
        try {
            c = DriverManager.getConnection(DBUrl);
             System.out.println("Connection Created !");
         }catch (SQLException e) {
            System.out.println("SQLException: " + e.getMessage());
            System.out.println("SQLState:     " + e.getSQLState());
            System.out.println("VendorError:  " + e.getErrorCode());
          }
       }catch (Exception e1) {
            System.err.println("Unable to load driver.");
            e1.printStackTrace();
        }
        return c;
    }
}
```

Скопируйте его в файл DBUtil.java и откомпилируйте. При удачном соединении с сервером баз данных отобразится сообщение «Connection Created !». После этого можно выполнять различные команды доступа к таблицам базы данных. Далее приведено содержимое страницы LookupDB.jsp, в которой реализуется web-интерфейс для работы с базой данных:

```
<html>
<body bgcolor="white">
<title>Trainings</title>
<meta http-equiv="Content-Type" content="text/html" charset="iso-8859-1">
<form method=GET action="http://localhost:8080/LookupDB.jsp">
<font size="3">
<%@ page language="java" %>
<%@ page import="java.util.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.sql.*" %>
<%@ page import="by.bsuir.mpzbd.*" %>
<%@ page import="org.gjt.mm.mysql.Driver" %>
<%  String DBUrl = "jdbc:mysql://localhost/tms_db?user=tms_dbuser&
                   password=tms_dbpassword";
    Connection conn = null;
    conn = DBUtil.createConnection(DBUrl);//создаем подключение к базе данных
    PreparedStatement ps = null;
    String sql;
```

```
    // получаем параметры от jsp
    Enumeration filds = request.getParameterNames();
    if(filds.hasMoreElements()) {
      String trainingID = request.getParameter("trainingID");
      String training_name = request.getParameter("training_name");
      String skill_id = request.getParameter("skill_id");
      String cmd = request.getParameter("command");
      Integer id = null;
      try {
        if (cmd.compareTo("insert")==0) { // если выбрана кнопка "insert"
          if (trainingID!=null & training_name!=null & skill_id!=null) {
             sql="insert into training (id, skill_id, name) values (?, ?, ?)";
             ps = conn.prepareStatement(sql);
             //далее устанавливаем параметры sql-выражения id, name и skill_id
             id = new Integer(trainingID);
             ps.setObject(1, id);
             id = new Integer(skill_id);
             ps.setObject(2, id);
             ps.setObject(3, training_name);
             ps.executeUpdate();
          }
        } else if(cmd.compareTo("delete")==0){//если выбрана кнопка "delete"
            if (trainingID!=null & training_name!=null & skill_id!=null) {
              sql="delete from training where id=? AND name=? AND skill_id=?";
              ps = conn.prepareStatement(sql);
              id = new Integer(trainingID);
              ps.setObject(1, id);
              ps.setObject(2, training_name);
              id = new Integer(skill_id);
              ps.setObject(3, id);
              ps.executeUpdate();
            }
          }
       }catch (SQLException e) {
           System.out.println("SQLException: " + e.getMessage());
           System.out.println("SQLState:     " + e.getSQLState());
           System.out.println("VendorError:  " + e.getErrorCode());
        }
      } // if(filds.hasMoreElements())
%>
<BR><BR>Training ID: <input type=text name="trainingID">
<BR><BR>Training Name: <input type=text name="training_name">
<BR><BR>Skill ID: <input type=text name="skill_id">
<BR><BR><INPUT type=radio name=command value=insert>Insert<BR><INPUT type=radio
name=command value=delete>Delete<BR><input type=submit value=Update>
<TABLE BORDER=1>
<CAPTION>Training table</CAPTION>
<TR bgcolor=lightblue><TH>Training ID</TH><TH>Training Name</TH><TH>Skill ID
</TH></TR>
```

```
<%    // распечатываем таблицу training
      ResultSet rs = null;
      sql = "select id, name, skill_id from training";
      try {
        ps = conn.prepareStatement(sql);
        rs = ps.executeQuery();
        Integer id = new Integer(0);
        String training_name = "";
        Integer skill_id = new Integer(0);
        while(rs.next()) {
          id = (Integer)rs.getObject(1);
          training_name = (String)rs.getObject(2);
          skill_id = (Integer)rs.getObject(3);
%>
<TR bgcolor=silver><TD><%=id%></TD><TD><%=training_name%></TD>
<TD><FONT COLOR=Red><%=skill_id%></TD></FONT></TR>
<%        }
     } catch (SQLException e) {
          System.out.println("SQLException: " + e.getMessage());
          System.out.println("SQLState:     " + e.getSQLState());
          System.out.println("VendorError:  " + e.getErrorCode());
       }
     rs.close();
     ps.close();
     conn.close();
%>
</TABLE></font></form></body></html>
```

После запуска web-сервера загрузите в браузере страницу следующим образом: http://localhost:port/LookupDB.jsp (рис. 3.3).

Рис. 3.3. Внешний вид web-страницы

Контрольные вопросы

1. Приведите последовательность команд для создания пользователя для MySQL-сервера.
2. Какой командой прописывается пользователь в базу данных MySQL с определенными правами?
3. Какая команда сервера MySQL позволяет загрузить на выполнение файл с sql-скриптом?
4. Какая функция класса `DriverManager` загружает JDBC-драйвер в JVM?
5. Для чего предназначены следующие функции:

```
Class.forName("org.gjt.mm.mysql.Driver", newInstance());
DriverManager.registerDriver(new org.gjt.mm.mysql.Driver());?
```

Следующая функция создает подключение к базе данных:

```
Connection c = DriverManager.getConnection(DBUrl);
```

Из каких компонентов состоит передаваемый параметр DBUrl?

Задание

Разработать java-приложение, реализующее доступ к реляционной базе данных, созданной в лабораторной работе №1. Интерфейс между пользователем и сервером баз данных реализовать в виде сервлета или JSP (JavaServer Pages). Каждая страница должна отображать данные в виде таблицы, а также следующие команды: просмотр существующих таблиц (select), удаление записей (delete), добавление записей (insert), модификация записей (update).

Варианты индивидуальных заданий

В качестве индивидуального задания использовать базу данных, созданную в лабораторной работе №1. Пользователь должен иметь возможность выполнять операции выборки, вставки, обновления и удаления со всеми таблицами базы данных.

Лабораторная работа №4
Создание инструментальных средств поддержки онтологического инжиниринга

Теоретические сведения

В учебных экспериментальных предметных областях, где содержится небольшой объем знаний и редко встречаются противоречия, выбор представления не сложен. Для таких предметных областей, как покупки в Internet-сети или управление роботом в изменяющейся физической среде, требуются достаточно гибкие способы представления. Эти представления базируются на общих понятиях, таких, как действия, время, физические объекты, а также ментальные характеристики человека, такие, как убеждения, желания, намерения и цели. Способы представления таких абстрактных понятий называют онтологической инженерией или инженерией знаний [9, 10]. Онтологическая инженерия включает следующие этапы: представление знаний, манипулирование знаниями, общение, восприятие, обучение, поведение. В рамках «представления знаний» решаются задачи, связанные с формализацией и представлением знаний в памяти интеллектуальной системы. Для этого разрабатываются специальные модели представления знаний и языки для описания знаний, выделяются различные типы знаний. Изучаются источники, из которых интеллектуальная система может черпать знания, и создаются процедуры и приемы, с помощью которых возможно приобретение знаний для интеллектуальной системы.

Особенности знаний:

1. Внутренняя интерпретируемость. Каждая информационная единица должна иметь уникальное имя, по которому система находит ее, а также отвечает на запросы, в которых это имя упомянуто. Программа, извлекающая данные из памяти, «понимает», что за ними скрывается. При переходе к знаниям необходимо хранить наряду с информационными единицами их *метаструктуру*.

2. Структурированность. Между информационными единицами должны быть установлены различные теоретико-множественные отношения типа «часть – целое», «род – вид» или «элемент – класс». Каждая информационная единица может быть включена в состав любой другой, и из каждой информационной единицы можно выделить некоторые составляющие ее информационные единицы.

3. *Связность*. Все информационные единицы должны быть связаны между собой, т.е. из каждого узла должен существовать маршрут в любой другой узел. Семантика отношений может носить декларативный или процедурный характер. Например, две или более информационные единицы могут быть связаны отношением «одновременно», «причина – следствие» или отношением «быть рядом». Приведенные отношения характеризуют декларативные знания. Если между двумя информационными единицами установлено отношение «аргумент – функция», то оно характеризует процедурное знание, связанное с вычислением определенных функций. Следует различать *отношения структуризации*, *функциональные отношения*, *каузальные отношения* и *семантические отношения*.

4. Семантическая метрика. На множестве информационных единиц в некоторых случаях полезно задавать отношение, характеризующее ситуационную близость информационных единиц, т.е. силу ассоциативной связи между информационными единицами. При работе с информационными единицами появляется возможность находить знания, близкие к уже найденным.

5. *Активность*. Традиционно в области программного обеспечения существует разделение всех информационных единиц на данные и команды. В результате – данные пассивны, а команды активны. Все процессы инициируются командами, а данные используются этими командами лишь в случае необходимости. В интеллектуальных системах, как и у человека, актуализации тех или иных действий способствуют знания, имеющиеся в системе. Таким образом, выполнение программ в интеллектуальной системе должно инициироваться текущим состоянием информационной базы. Появление в базе фактов или описаний событий, установление связей может стать источником активности системы.

Перечисленные пять особенностей информационных единиц определяют ту грань, за которой данные превращаются в знания, а базы данных перерастают в *базы знаний*. Совокупность средств, обеспечивающих работу с знаниями, образует *систему управления базой знаний*.

Существуют два типа методов представления знаний:

1. Формальные модели представления знаний, в основе которых лежит формальная математическая теория.

2. Неформальные (семантические, реляционные) модели представления знаний, которые не придерживаются формальной теории, в результате чего они не обладают универсальностью, т.е. для каждой предметной области строится конкретная модель.

Каждому из методов представления знаний соответствует свой способ описания знаний:

1. *Логические модели*, основанные на формальной теории. Система, построенная на основе формальной теории, называется *аксиоматической системой* (или *формальной системой*). Формальная теория задается четверкой вида

$$M = <T, P, A, B>,$$

где T – *множество базовых элементов* различной природы, например, слов из некоторого ограниченного словаря, деталей детского конструктора, входящих в состав некоторого набора. Для множества T существует способ определения принадлежности или непринадлежности произвольного элемента к этому множеству;

P – множество *синтаксических правил*. С их помощью из элементов T образуют *синтаксически правильные совокупности*. Например, из слов ограниченного словаря строятся синтаксически правильные фразы, из деталей детского конструктора с помощью гаек и болтов собираются новые конструкции;

A – множество аксиом;

B – множество *правил вывода*, применяя их к элементам A, можно получать новые синтаксически правильные совокупности, к которым снова можно применять правила из B. Так формируется *множество выводимых* в данной формальной системе *совокупностей*.

Примерами аксиоматических систем являются исчисление высказываний и исчисление предикатов.

Недостатками формальных моделей представления знаний являются закрытость и плохая модифицируемость. Модификация и расширение здесь всегда связаны с перестройкой всей формальной системы, что для практических систем сложно и трудоемко. В них очень сложно учитывать происходящие изменения. Поэтому формальные системы как модели представления знаний используются в тех предметных областях, которые хорошо локализуются и мало зависят от внешних факторов.

2. *Сетевые модели*. В основе моделей этого типа лежит конструкция, называемая семантической сетью. Сетевые модели формально можно задать в виде $H = <I, C_1, C_2, ..., C_n, Г>$. Здесь I есть множество информационных единиц; $C_1, C_2, ..., C_n$ – множество типов связей между информационными единицами. Отображение Г задает между информационными единицами, входящими в I, связи из заданного набора типов связей. В зависимости от типов связей, используемых в модели, различают *классифицирующие сети*, *функциональные сети* и *сценарии*. В классифицирующих сетях используются отношения структуризации. Такие сети позволяют в базах знаний вводить разные иерархические отношения между информационными единицами. Функциональные сети характеризуются наличием функциональных отношений. Их часто называют *вычислительными моделями*, т.к. они позволяют описывать процедуры вычислений одних информационных единиц через другие. В сценариях используются каузальные отношения, а также отношения типов «средство – результат», «орудие – действие» и т.п.

3. *Продукционные модели*. В моделях этого типа используются некоторые элементы логических и сетевых моделей. Из логических моделей заимствована идея правил вывода, которые здесь называются *продукциями*, а из сетевых моделей – описание знаний в виде семантической сети. В результате применения правил вывода к фрагментам сетевого описания происходит трансформация семантической сети за счет смены ее фрагментов, наращивания сети и исключения из нее ненужных фрагментов. Таким образом, в продукционных моделях процедурная информация явно выделена и описывается иными средствами, чем декларативная информация. Вместо логического вывода, характерного для логических моделей, в продукционных моделях появляется *вывод на знаниях*.

4. *Фреймовые модели*. В отличие от моделей других типов во фреймовых моделях фиксируется жесткая структура информационных единиц, которая называется *протофреймом*. В общем виде она выглядит следующим образом:

(Имя фрейма:
Имя слота 1 (значение слота 1);
Имя слота 2 (значение слота 2);
. . . . . ;
Имя слота К (значение слота К)).

Значением *слота* могут быть числа или математические соотношения, тексты на естественном языке или программы, правила вывода или ссылки на другие слоты данного фрейма или других фреймов. В качестве значения слота может выступать набор слотов более низкого уровня, что позволяет во фреймовых представлениях реализовать механизм наследования. При конкретизации фрейма ему и слотам присваиваются конкретные имена и происходит заполнение слотов. Таким образом, из протофреймов получаются *фрейм-экземпляры*:

Протофрейм	Фрейм-экземпляр
(Список работников: Фамилия (значение слота 1); Год рождения (значение слота 2); Специальность (значение слота 3); Стаж (значение слота 4)).	(Список работников: Фамилия (Попов - Сидоров - Иванов - Петров); Год рождения (1985 - 1976 - 1980 - 1978); Специальность (водитель - инженер - сантехник - программист); Стаж (2 - 6 - 3 - 5)).

Контрольные вопросы

1. Сформулируйте определение понятия «онтология».
2. Перечислите этапы онтологического инжиниринга.
3. Перечислите отличительные свойства знаний.
4. Дайте определение операции изоморфного поиска.
5. Приведите пример предметной области, для которой наиболее адекватна модель представления знаний «Семантическая сеть».

Задание

Разработать инструментальные средства онтологического инжиниринга, позволяющие реализовать выбранную модель представления знаний: формальную, логическую или семантическую. Реализовать операции навигации и поиска в базе знаний, а также операции верификации базы знаний и операции вывода новых знаний.

Варианты индивидуальных заданий

1. Задача планирования отдыха. Предметная область туристического агентства, описывающая туристические маршруты содержит следующую базовую информацию:

1) дата, время и место отправления;
2) дата, время и место прибытия;
3) время заселения и время выселения;
4) вид транспорта (автобус, поезд, самолет);

5) адрес пансионата или гостиницы;
6) вид пансионата (частный или государственный, дом отдыха, санаторий);
7) если санаторий, то какие виды процедур (массаж, грязевые ванны и т.д.);
8) в каком году был ремонт пансионата или в каком году был построен;
9) расстояние до моря;
10) тип номера (люкс, блок, без удобств, эллинг);
11) количество мест в номере;
12) есть ли в номере горячая вода;
13) с видом на море;
14) этаж, на котором располагается номер;
15) питание (сколько раз или отсутствует, шведский стол или заказ);
16) достопримечательности (если есть, то какие).

При составлении туристических маршрутов необходимо также помнить о купленных путевках и о количестве мест и билетов.

Для составления туристического маршрута необходима информация о предпочтениях клиента. Например, пользователь должен иметь возможность ввести следующий запрос (возможно на формальном языке): найти дом отдыха для гипертоников, расположение – не дальше 100 метров от моря без перепадов высот, номер с видом на море, трехразовое питание.

Использовать логику предикатов первого порядка. Пример формализации знаний туристического агентства:

SeaDistant(X,Y); // гостиница X удаленность от моря на Y м.
HotWater(X, Y); // в гостинице X горячая вода подается по расписанию Y раз в день
RoomInHostel(X, Y); // номер Y принадлежит гостинице X
Balkon(Y); // в номере Y находится балкон
SeaView(Y); // окна номера Y выходят на море
WC(Y); // в комнате Y есть туалет
Shower(Y); // в комнате Y есть душ
Lux(Y); // комната Y относится к классу «люкс»
Room(Y); // Y – номер
RoomSquare(Y,Z); // Z $м^2$ – площадь комнаты Y
Between(X, Y, Z); // $Y \le X < Z$
HumanRoom(X, Y); // номер Y рассчитан на X человек
X = «Дом творчества им.Коровина»;
Y = B1;
SeaDistant(X,50) & HotWater(X,3) & RoomInHostel(Y,X) & Balkon(Y) & SeaView(Y)
& WC(Y) & Shower(Y) → Lux(Y);
Room(Y) & RoomSquare(Y,Z) & Between(Z, 12, 14) → HumanRoom(2, Y);

2. Задача вывода всевозможных следствий из текущей базы знаний с использованием метода резолюций. Формализовать деятельность следующей организации:

1) в организации работает n сотрудников;
2) каждый сотрудник занимает определенную должность;
3) сотрудник может совмещать не более двух должностей;
4) существуют должности, занимая которую, сотрудник не может совмещать другую должность;
5) каждый сотрудник подчиняется только одному начальнику;
6) начальник также является сотрудником;
7) существует начальник, который никому не подчиняется;
8) должность характеризуется названием, должностными обязанностями и уровнем заработной платы;
9) сотрудники образуют отделы;
10) каждый сотрудник работает хотя бы в одном отделе;
11) у отдела существует только один начальник.

Пример формализации факта:

```
Manager(X,Y); // X является начальником Y
```

Примером выведенного следствия из текущей базы знаний является то, что каждый сотрудник не может работать более чем в одном отделе.

3. Задача нахождения синонимичных и омонимичных понятий в тексте. Предположим, что каждое понятие в тексте описывается своим определением. Например,

```
Треугольник – это многоугольник с тремя вершинами.
Треугольник – это совокупность из трех попарно взаимосвязанных отрезков.
```

Необходимо формализовать каждое определение понятия в виде семантической сети. Далее необходимо найти изоморфное вложение подграфов. В зависимости от того пересекаются графы, один граф включается в другой или не пересекаются, необходимо сделать вывод о том, являются ли эти понятия омонимичными (т.е. их определения не пересекаются), либо одно понятие является подмножеством другого.

4. Задача определения связности объектной модели предметной области. В качестве примера рассмотрим диаграмму UML. Данная диаграмма является связной, если один объект генерирует сообщение, используемое другим объектом либо человеком.

Литература

1. IBM DB2 Universal Database. SQL Reference Version 6. International Business Machines Corporation [Электронный ресурс]. – 1999. – Режим доступа : http://www.software.ibm.com/data/db2/library/.

2. MySQL Reference Manual for version 4.0.17. [Электронный ресурс]. – 2003. – Режим доступа : http://www.mysql.com/.

3. Гарсиа-Молина, Г. Системы баз данных. Полный курс. Database Systems. The Complete Book / Г. Гарсиа-Молина, Д. Уидом, Д. Ульман. – СПб. : Вильямс, 2003. –1088 с.

4. Дейт, Д. К. Введение в системы баз данных / Д. К. Дейт. 7-е изд. – СПб. : Вильямс, 2001. – 1072 с.

5. Семантическая модель сложноструктурированных баз данных и баз знаний: учеб. пособие по курсу «Модели представления знаний, базы данных и системы управления базами данных» для студ. спец. «Искусственный интеллект» всех форм обуч. / В. В. Голенков, Н. А. Гулякина, О. Е. Елисеева и др. – Минск : БГУИР, 2004. – 263 с.

6. Смирнов, С. Н. Работаем с IBM DB2 / С. Н. Смирнов. – М. : Гелиос АРВ, 2001. – 303с.

7. Хансен, Г. Базы данных. Разработка и управление / Г. Хансен, Д. Хансен; пер. с англ. – М. : БИНОМ, 1999. – 700 с.

8. Харрингтон, Д. Проектирование объектно-ориентированных баз данных / Д. Харрингтон. – М. : ДМК, Серия : для программистов, 2001. – 272 с.

9. Рассел, С. Искусственный интеллект: современный подход / С. Рассел, П. Норвиг. – 2- е изд. : пер. с англ. – М. : Вильямс, 2006. – 1408 с.

10. Гаврилова, Т. А. Базы знаний интеллектуальных систем / Т. А. Гаврилова, В. Ф. Хорошевский. – СПб. : Питер, 2000. – 384 с.

Св. план 2006, поз. 41

Учебное издание

МОДЕЛИ ПРЕДСТАВЛЕНИЯ И ОБРАБОТКИ ДАННЫХ И ЗНАНИЙ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Учебно-методическое пособие

В 3-х частях

Часть 1

Гулякина Наталья Анатольевна
Лемешева Татьяна Леонидовна
Агашков Виталий Владимирович

Редактор *Е. Н. Батурчик*
Корректор *М. В. Тезина*
Компьютерная верстка *Е. Н. Мирошниченко*

Пописано в печать 18.07.2007. Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Печать ризографическая. Усл. печ. л. 2,67. Уч.-изд. л. 2,1. Тираж 150 экз. Заказ 289.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.
220013, Минск, П. Бровки, 6