МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра информационных технологий и компьютерных систем

БАЗЫ ДАННЫХ

Методические указания
к выполнению лабораторных работ по дисциплине
для студентов дневной формы обучения
по направлениям подготовки
09.03.01 «Информатика и вычислительная техника»,
09.03.04 «Программная инженерия»

Севастополь
СевГУ
2023

УДК 004.65(076.5)
ББК 32.973-018.2я73
Б179

Р е ц е н з е н т :
В. В. Альчаков - канд. техн. наук, доцент кафедры
«Информатика и управление в технических системах»

Ответственный за выпуск:
А. А. Брюховецкий - канд. техн. наук, доцент, заведующий кафедрой
информационных технологий и компьютерных систем

Составители: О. В. Ченгарь, Т.В. Волкова, Ю.В. Доронина,
А. А. Малицкая

Б179 **Базы данных** : методические указания к выполнению лабораторных работ по дисциплине для студентов дневной формы обучения по направлениям подготовки 09.03.01 «Информатика и вычислительная техника», 09.03.04 «Программная инженерия» / Севастопольский государственный университет, Институт информационных технологий, кафедра информационных технологий и компьютерных систем ; сост.: О. В. Ченгарь, Т. В. Волкова, Ю. В. Доронина, А. А. Малицкая. – Севастополь : СевГУ, 2023. – 102 с. – Текст : электронный.

Приводятся теоретические сведения и задания для выполнения лабораторных работ в процессе изучения учебной дисциплины «Базы данных».

УДК 004.65(076.5)
ББК 32.973-018.2я73

Методические указания рассмотрены и утверждены на заседании кафедры «Информационные технологии и компьютерные системы», протокол № 5 от 04 февраля 2022 г.

© Ченгарь О. В., Волкова Т.В.,
Доронина Ю.В., Малицкая А.А., сост., 2023
© ФГАОУ ВО «Севастопольский
государственный университет», 2023

СОДЕРЖАНИЕ

1. Лабораторная работа № 1. Реляционная алгебра 4
2. Лабораторная работа № 2. Команды определения данных по организации структуры таблиц в MySQL 12
3. Лабораторная работа № 3. Средства манипулирования данными языка SQL. 30
4. Лабораторная работа № 4. Элементарные выборки посредством оператора select. Агрегатные функции sql. 35
5. Лабораторная работа № 5. Ссылочная целостность данных. Создание схемы базы данных 41
6. Лабораторная работа № 6. Концептуальный и логический уровни проектирования БД 46
7. Лабораторная работа № 7. Разработка сложных запросов к базе данных. Запросы на основе нескольких таблиц. Коррелированные вложенные подзапросы 47
8. Лабораторная работа № 8. Создание пользовательских представлений и индексов 56

Библиографический список 60

Приложение 1. Концептуально-логические схемы баз данных, используемые в вариантах заданий 61

Приложение 2. Варианты заданий к л/р №3 70

Приложение 3. Варианты заданий к л/р №4 71

Приложение 4. Варианты заданий к л/р №6 72

Приложение 5. Варианты заданий к л/р №7 99

ЛАБОРАТОРНАЯ РАБОТА № 1
РЕЛЯЦИОННАЯ АЛГЕБРА

Цель работы:

Научиться формировать запросы на языке реляционной алгебры.

Порядок выполнения работы:

1. Ознакомьтесь с примерами решения задач реляционной алгебры.
2. В схеме базы данных выберите отношение, согласно индивидуального варианта (ПРИЛОЖЕНИЕ 1).
3. В документе MS Word подготовьте исходную таблицу, соответствующую структуре выбранного отношения.
4. Заполните таблицу данными (осмысленными) - не менее 15 кортежей.
5. Продумайте и напишите 5 различных словесных запросов к данной таблице. Запросы должны быть сформированы таким образом, чтобы выражение реляционной алгебры, его реализующее, содержало не менее 2-х операций. Также необходимо предусмотреть, чтобы в запросах были реализованы все, рассмотренные в лекции, реляционные операции.
6. К каждому запросу напишите выражение реляционной алгебры, реализующее искомый запрос.
7. Для всех 5 заданий представьте результирующие отношения (исходя из заполненных данных в исходной таблице). Существует два теоретических языка запросов:
   1) язык реляционной алгебры,
   2) язык реляционного исчисления.

Теоретические сведения

1.1 Теоретические языки запросов

Эти языки представляют собой минимальные возможности реальных языков манипулирования данными и эквивалентны друг другу по своим выразительным возможностям. Существуют правила преобразования запросов между ними. В реляционной алгебре операнды и результаты всех действий являются отношениями. Языки реляционной алгебры являются процедурными, т. е. задают алгоритм исполнения запроса. Операторы реляционной алгебры состоят из операндов, в роли которых выступают отношения, и реляционных операций.

Языки реляционного исчисления являются непроцедурными (описательными или декларативными) и позволяют выражать запросы с помощью предиката первого порядка, т. е. высказывание в виде функций, которому должны удовлетворять кортежи или домены отношений (исчисление кортежей, исчисление доменов).

Запрос к БД содержит лишь информацию о желаемом результате (язык SQL).

Отношение R содержит 2 части: заголовок и тело. Заголовок содержит фиксированное множество параметров: <имя атрибута: имя домена>

{<A1:D1>, <A2:D2>...<An:Dn>}

Все имена атрибутов разные.

Тело содержит множество кортежей. Каждый кортеж содержит множество пар: <имя атрибута: значение атрибута>

{<A1:Vi1>,<A2,Vi2>,...,<An,Vin>}

i=1,2,...,m, где m – количество кортежей в этом множестве.

Значения Vij должны принадлежать соответствующим доменам.

m – число кортежей – называется кардинальным числом отношения,

n – число атрибутов – степень отношения.

1.2 Примеры операций реляционной алгебры

Совместимость структур отношений или совместимость по типу означает совместимость имен атрибутов и типов соответствующих доменов.

При выполнении бинарной операции участвующие в операции отношения обязательно должны быть совместимы по типу.

Операция объединения

R1 union R2

Объединением двух совместимых отношений R1 и R2 является отношение R, содержащее все элементы исходных отношений с исключением повторений. Заголовки отношений R1 и R2 должны быть идентичны.

Пример

Отношение R1 – множество поставщиков из Лондона

S#	SName	Status	City
S1	Smith	20	London
S4	Clark	20	London

Отношение R2 – множество поставщиков, которые поставляют деталь P1.

S#	SName	Status	City
S1	Smith	20	London
S2	Jones	10	Paris

R (R1 union R2)

S#	SName	Status	City
S1	Smith	20	London
S2	Jones	10	Paris
S4	Clark	20	London

Операция вычитания

R1 minus R2

Вычитание совместимых отношений R1 и R2 с одинаковыми заголовками есть отношение, тело которого состоит из множества кортежей, принадлежащих R1, но не принадлежащих R2.

Для вышеуказанного примера нужно найти множество поставщиков, находящихся в Лондоне, но не поставляющих деталь P1.

R{{S4, Clark, 20, London}}

R2 minus R1 ≠ R1 minus R2.

Операция пересечения

R1 intersect R2

Пересечением двух совместимых отношений R1, R2 с одинаковыми заголовками называют отношение R с телом, содержащим кортежи, одновременно принадлежащим обоим исходам.

Пример

Найти поставщиков из Лондона, производящих деталь P1.

{{S1, Smith, 20, London}}

Декартово произведение отношений

R1 times R2

Произведение отношений R1 степени k1 и R2 степени k2, которые не имеют одинаковых атрибутов, есть такое отношение R степени k1+k2, заголовок которого представляет собой сцепление заголовков отношений R1 и R2. А тело имеет все возможные кортежи такие, что первые k1 элементов кортежа принадлежит множеству R1, а последние k2 элементов – множеству R2.

При необходимости получить произведение двух отношений, имеющих одинаковые имена атрибутов, производится операция переименование ***rename.***

Пример

R1 – множество всех поставщиков. R1 {S1, S2, S3, S4, S5}

R2 – множество всех номеров деталей R2 {P1, P2, P3, P4, P5, P6}

Результат произведения – отношение, содержащее все пары типа поставщик-деталь.

R{{S1, P1}, {S1,P2}, {S1, P3}...{S5,P6}}

Выборка

R where f

Выборка отношения R по формуле f представляет собой новое отношение с таким же заголовком и телом, состоящим из таких кортежей отношения R, которые удовлетворяют истинности логического выражения, заданного формулой f.

Для записи формул используются операнды (имена атрибутов), константы, логические операции (AND, OR, NOT), операции сравнения и скобки.

Пример

P where Weight>14

P#	PName	Color	Weight	City
P1	Nut	Red	12	London
P2	Cam	Blue	12	Paris

Операция проекции

A[X, Y, ..., Z]

Проекция отношения A по атрибутам X, Y, Z, где каждый из атрибутов принадлежит A, представляет собой отношение C с заголовком {X, Y, ..., Z} и телом, содержащим кортежи отношения A за исключением повторяющихся кортежей (вертикальное подмножество отношения A). Повторение одинаковых атрибутов в списке {X, Y, Z} запрещено.

Варианты операции проекции:

1) Отсечение списка атрибутов подразумевает образование всех атрибутов (тождественная проекция)

2) Выражение вида A[] обозначает пустую проекцию, результатом которой является пустое множество.

3) Операция проекции может применяться к произвольному отношению, в том числе и к результату выборки.

(S WHERE CITY='PARIS')[S#] – проекция из отношения, которое является результатом выборки по заданному условию.

S#
S2
S3

Операция соединения

Cf(R1, R2)

Соединение отношений R1, R2 по условию, заданному формулой f, представляет собой отношение R, которое можно получить путем декартового произведения отношений R1, R2 с последующим применением к результату операций выборки по формуле f.

(R1 TIMES R2)WHERE A*B – атрибуты A и B должны быть определены на одном и том же домене. Атрибут A принадлежит R1, B принадлежит R2.

Операция * должна иметь смысл для этого домена.

Частный случай *-соединения:

1) Эквисоединение (формула задает равенство операндов)

2) Естественное соединение – **JOIN** – применяется к двум отношениям, имеющим общий атрибут, т. е. атрибут, имеющий одно и то же имя и определенный на одном и том же домене.

Атрибут может быть сложным, т. е. представляет собой совокупность атрибутов.

(S RENAME CITY AS SCITY) TIMES (P RENAME CITY AS PCITY) WHERE SCITY>PCITY – пары поставщиков и деталей, находящихся в разных городах.

S#	SName	Status	SCity	P#	PName	Color	Weight	PCity
S2	Jones	10	Paris	P1	Nut	Red	12	London
S2	Jones	10	Paris	P4	Screw	Red	14	London

Естественное соединение

S JOIN P

Если отношение не имеет общих атрибутов, то операция естественного соединения эквивалентна декартовому произведению.

Соединение производится по общему столбцу.

Пример

Найти всех поставщиков и детали, которые находятся в одном городе.

S#	CName	Status	City	P#	PName	Color	Weight
S1	Smith	20	London	P1	Nut	Red	12
S1	Smith	20	London	P4	Screw	Red	14

Деление отношений

R1 DIVIDEBY R2

Результатом деления отношения R1 с атрибутами A и B на отношение R2 с атрибутом B, где A и B – простые или составные атрибуты, причем, B – общий атрибут, определенный на одном и том же домене (множестве доменов составного атрибута) является отношение R с заголовком A и телом, состоящим из кортежей r таких, что в отношении R1 имеются кортежи (r, S), причем множество кортежей S включает множество значений атрибутов B отношения R2.

Пример

Получить поставщиков, поставляющих заданное множество деталей.

DEND(делимое) DIVIDEBY DOR(делитель)

DEND

S#	P#
S1	P1
S1	P2
S1	P3
S1	P4
S1	P5
S1	P6
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4
S4	P5

1) DOR

P#
P1
P2
P3
P4
P5
P6

Результат:

S#
S1

2) DOR

P#
P2
P4

Результат:

S#
S1
S4

3) DOR

P#
P1

Результат:

S#
S1
S2

1.3 Дополнительные операции

Переименование – позволяет изменить имя атрибута

RENAME <исходное отношение><старое имя атрибута> AS <новое имя атрибута>, [<старое имя атрибута> AS <новое имя атрибута>…]

Расширение – порождает новое отношение, которое отличается от исходного наличием добавляемого атрибута

EXTEND <исходное отношение> ADD <выражение> AS <новый атрибут>

Имя нового атрибута не должно входить в заголовок исходного отношения и не должно использоваться в выражении.

Помимо обычных арифметических операций и операций сравнения можно использовать функции

COUNT, SUM, AVG, MAX, MIN

Пример

Выбрать детали, вес которых больше 1000.

(EXTEND P ADD (WEIGHT*454) AS GMWT) WHERE GMWT>1000

Подведение итогов (группирование)

SUMMARIZE – выполняет вертикальные или групповые вычисления

SUMMARIZE <исходное отношение> BY (<список атрибутов>) ADD <выражение> AS <новый атрибут>

Исходное отношение задается именем отношения или заключенным в круглые скобки выражением реляционной алгебры.

Список атрибутов – разделенные запятыми имена атрибутов, по которым происходит группировка.

Выражение – скалярное выражение, аналогичное EXTEND.

Новый атрибут – имя нового атрибута.

Результатом операции SUMMARIZE является отношение R с заголовком, состоящим из атрибутов списка, расширенное новым атрибутом.

<список атрибутов> A1, A2, … A_N

Пример

Вычислить количество поставок по каждому поставщику.

SUMMARIZE SP BY (S#) ADD COUNT AS SP_NUM

S#	SP_NUM
S1	6
S2	2
S3	3
S4	1

Вычисление общего и среднего числа поставок.

SUMMARIZE SP BY (P#) ADD SUM QTY AS SUMP AVGQTY AS AVGP

Операторы реляционной алгебры позволяют изменять тело существующего отношения.

<выражение-цель>:=<выражение-источник>

Как правило, <выражение-цель> - имя отношения,

<выражение-источник> - выражение реляционной алгебры.

Пример

S:=SUNION{{<S#:'S6'>,<Sname:'WHITE'>, <STATUS:'50'>,<CITY:'MADRID'>}} – добавление кортежа.

Более удобной является операция вставки INSERT.

Вставка

INSERT <выражение-источник> INTO <выражение-цель>

Оба выражения должны быть совместимы по структуре.

Пример

INSERT (S WHERE CITY='LONDON') INTO TEMP

Операция обновления

UPDATE <выражение-цель><список элементов>,

где <список элементов>:<атрибут>:=<скалярное выражение> [,<атрибут>:=<скалярное выражение>...]

Пример

UPDATE P WHERE COLOR='RED' CITY='ROME'

Удаление

DELETE <выражение-цель>

DELETE S WHERE STATUS<20

Операция реляционного сравнения

может использоваться для прямого сравнения двух отношений.

<выражение1>*<выражение2>

* : = # ≤ ≥ < >

Пример

TEMP:=S[CITY]=P[CITY]

1.4 Основные правила записи выражений реляционной алгебры

Результатом произвольной реляционной операции является отношение, которое в свою очередь может участвовать в другой реляционной операции.

Свойство замкнутости позволяет записывать вложенные отношения реляционной алгебры. При этом необходимо учитывать:

1. Приоритет выполнения операции (пересечение более приоритетно, чем объединение)
2. Тождественное преобразование позволяет по-разному записывать одно и то же выражение.

A WHERE C1 AND C2 ~ A (WHERE C1) INTERSECT (A WHERE C2)

A WHERE C1 OR C2 ~ (A WHERE C1) UNION (A WHERE C2)

3. Составляя выражение, нужно обеспечить совместимость участвующих в операции отношений. При необходимости изменения заголовков следует выполнять переименование атрибутов.

Преимуществом реляционного исчисления перед реляционной алгеброй можно считать то, что пользователю не требуется самому строить алгоритм выполнения запроса. Пользователь указывает только, какие данные нужно получить.

Исчисление предикатов – математическая основа реляционного исчисления.

Существует два варианта исчислений:

- исчисление кортежей,
- исчисление доменов.

ЛАБОРАТОРНАЯ РАБОТА № 2
КОМАНДЫ ОПРЕДЕЛЕНИЯ ДАННЫХ ПО ОРГАНИЗАЦИИ СТРУКТУРЫ ТАБЛИЦ В MySQL

Цель работы:

Научиться создавать в базе данных таблицы и изменять их структуру для выполнения элементарных действий на удаленном и локальном компьютере.

Порядок выполнения работы:

1) Изучить методические рекомендации по выполнению работы.
2) Рассмотреть предложенную схему базы данных (ПРИЛОЖЕНИЕ 1).
3) Для таблицы, указанной в варианте задания, описать логическую структуру, которая содержит последовательность расположения атрибутов в отношении и их основные характеристики. Описание оформить в виде таблицы:

Название таблицы в БД					
Атрибут		Признак ключа	Формат атрибута		
Имя (обозначение)	Полное наименование		Тип	Длина	Точность (для чисел)/ множество значений (для ENUM и SET)/ знач. по умолчанию и т.д.
имя 1					
…					
имя N					

4) Создать новую базу данных с помощью SQL-редактора программы phpMyAdmin, название которой будет отражать предметную область (согласно варианта).
5) С помощью SQL-редактора программы phpMyAdmin создать таблицу в соответствии с вариантом задания (табл. 1.1). Заполнить таблицу данными (2-3 записи).
6) С помощью SQL-редактора программы phpMyAdmin произвести следующие изменения структуры таблицы:
   - добавить новый столбец в таблицу и поместить его после второго столбца;
   - поменять местами второй столбец таблицы и тот, который был добавлен;
   - удалить добавленный столбец из таблицы.
7) С помощью SQL-редактора программы phpMyAdmin создать, а затем удалить произвольную таблицу.

Теоретические сведения

2.1 Администрирование сервера баз данных mysql. Разработка простейшего клиентского приложения

Установка базового пакета Denwer.

Базовый пакет содержит большинство необходимых программ и утилит: Apache, SSL, SSI, PHP5 с поддержкой GD, MySQL5 с поддержкой транзакций, систему управления виртуальными хостами, систему управления запуском и завершением всех компонентов Денвера, phpMyAdmin — система управления MySQL через Web-интерфейс, эмулятор sendmail и SMTP-сервера. Пакет предназначен для быстрой настройки компонентов, необходимых для разработки WEB-приложений с использованием СУБД MySQL и скриптов PHP.

Для установки пакета воспользуйтесь инсталлятором, который можно загрузить на сайте www.denwer.ru. На этом же сайте находится подробная инструкция по установке системы.

Создание новой базы данных

Запустить WEB-браузер. Инициализировать работу приложенииphpMyAdmin. Для этого в строке адреса ввести ссылку http://localhost/Tools/phpmyadmin/index.php.

В разделе «Actions/MySQLlocalhost/Createnewdatabase» ввести произвольное имя базы данных, например lab1, нажать кнопку «Create» (рисунок 2.1).

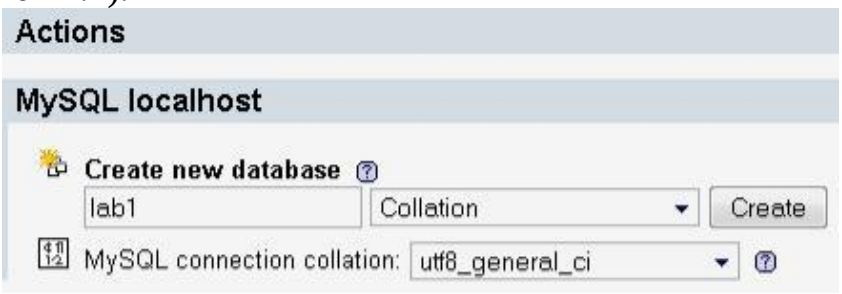


Рисунок 2.1 – Создание новой базы данных

Обратите внимание на SQL-скрипт, расположенный вверху страницы после создания базы данных (рисунок 2.2).

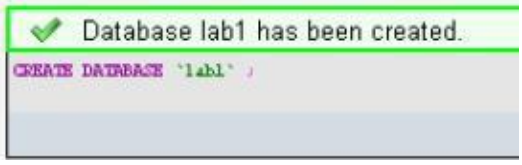


Рисунок 2.2 – SQL-скрипт для создания базы данных.

Создание отношения (таблицы)

Так как БД является моделью реального мира перед созданием таблицы необходимо описать предметную область, которую она представляет. Рассмотрим простейшую модель: телефонный справочник. Данные этой модели определяются следующими атрибутами: уникальный

номер абонента (он должен создаваться автоматически), фамилия, имя, отчество, дата рождения (рисунок 2.3).

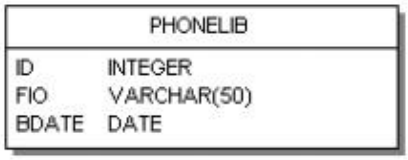


Рисунок 2.3 – Атрибуты отношения «Телефонный справочник»

Для создания таблицы в среде phpMyAdmin воспользуемся интерфейсом CreateNewTable (рисунок 2.4).

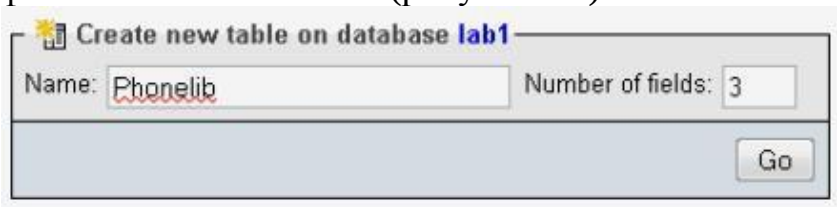


Рисунок 2.4 – Интерфейс создания таблицы

Описываем в интерфейсе имена атрибутов и их типы (рисунок 2.5).

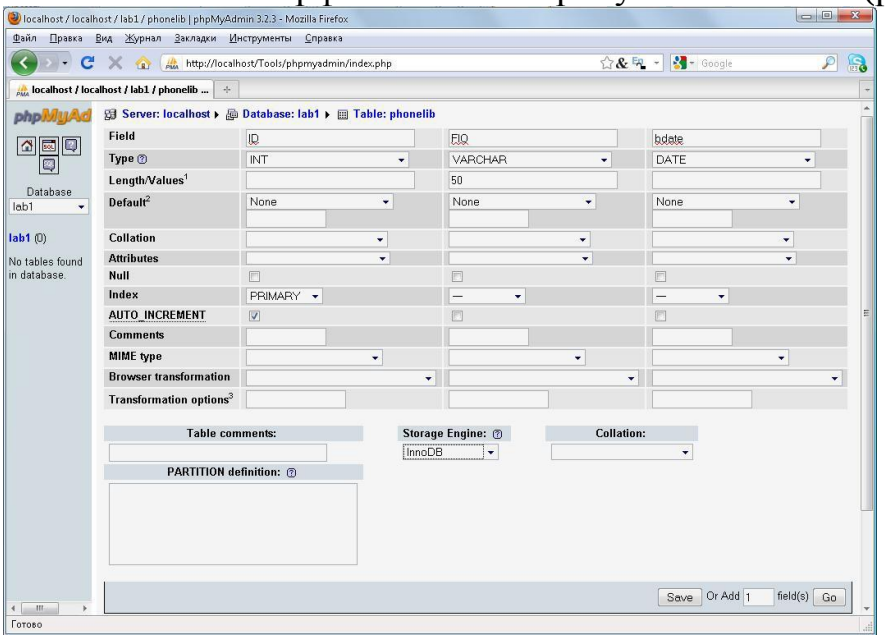


Рисунок 2.5 – Описание атрибутов

Обратите внимание, что поле для поля ID установлен признак AUTO_INCREMENT, для того чтобы при добавлении новой записи (картежа) значение поля генерировалось автоматически. На будущее, установим для этого атрибута признак PRIMARY в поле index. Атрибут FIO (фамилия, имя, отчество) является текстовым, поэтому для него указываем длину 50 (в символах).

Для создания таблицы после описания атрибутов нажимаем кнопку «Save».

Среда phpMyAdmin автоматически генерирует SQL-скрипт для создания новой таблицы:

```
CREATE TABLE `lab1`.`phonelib` (
`ID` INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`FIO` VARCHAR( 50 ) NOT NULL ,
`bdate` DATE NOT NULL
) ENGINE = InnoDB;
```

Синтаксис операции CREATETABLE в MySQL, в целом, соответствует стандарту SQL-92, однако имеет ряд особенностей. В частности, фрагмент фразы `ENGINE = InnoDB` определяет тип «движка» для работы с данными, каждый из которых имеет свои характерные особенности.

MyISAM. Движок по умолчанию. Не поддерживает транзакций, средняя надежность хранения данных. Превосходная производительность чтения данных (через SELECT). Блокирует всю таблицу при записи в неё данных, отчего маленькая производительность при частых записях.

InnoDB. Транзакционный тип движка, применяемый при интенсивных операциях записи, благодаря возможности блокировки уровня строк таблицы. Великолепная восстанавливаемость и высокая надежность хранения данных. Поддерживает внешние ключи. Именно этот движок рекомендуется использовать при выполнении лабораторных работ.

Ввод данных в таблицу

Для ввода данных в таблицу выбираем операцию «Insert» (рисунок 2.6).

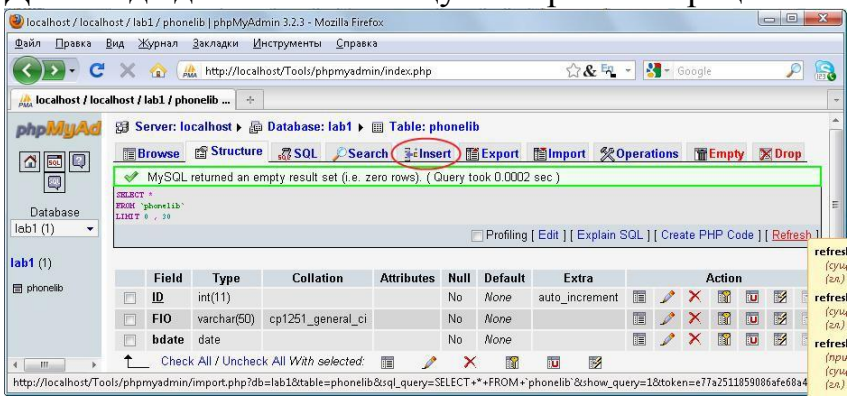


Рисунок 2.6 – Переход к вводу данных

Заполняем колонку «VALUE» в интерфейсе ввода данных и нажимаем на кнопку «Go» (рисунок 2.7). Обратите внимание, что поле ID оставляем пустым.

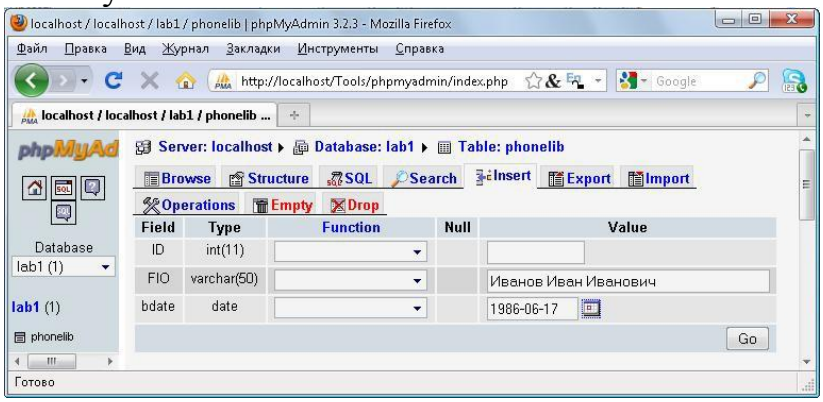


Рисунок 2.7 – Ввод данных в таблицу

SQL-скрипт добавления записи имеет вид: INSERTINTO `lab1`.`phonelib` (`ID`, `FIO`, `bdate`) VALUES (NULL, 'Иванов Иван Иванович', '1986-06-17'). Переходим на вкладку «Browse», для того чтобы убедиться в правильности заполнения данных. При этом выполняется SQL-скрипт выборки данных из таблицы SELECT*FROM `phonelib` (рисунок 2.8).

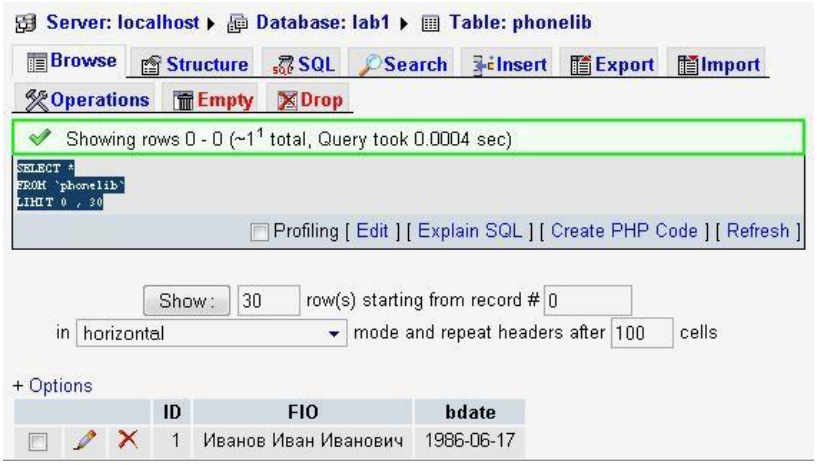


Рисунок 2.8 – Выборка данных из таблицы PhoneLib

Администрирование базы данных. Создание новых пользователей, паролей и привилегий

Пользователь имеет определенные привилегии (может редактировать, удалять, создавать новые таблицы и позиции в базе данных) и авторизуется по заданному логину и паролю.

При покупке хостинга у большинства хостинг-провайдеров для вас автоматически создается база данных и аккаунт пользователя с логином и паролем, имеющий все необходимые привилегии. На нормальном хостинге есть точно такой же phpMyAdmin посредством которого вы сможете управлять таблицами и их содержимым, а также импортировать данные из локальной базы данных в базу данных хостинга. Логин и пароль для доступа к базе данных, а также ссылка на phpMyAdmin высылается хостинг-провайдером на ваш e-mail.

На локальном же компьютере нам придется самим создать пользователя. Для этого выбираем базу данных lab1 и вкладку Priveleges (рисунок 2.9).

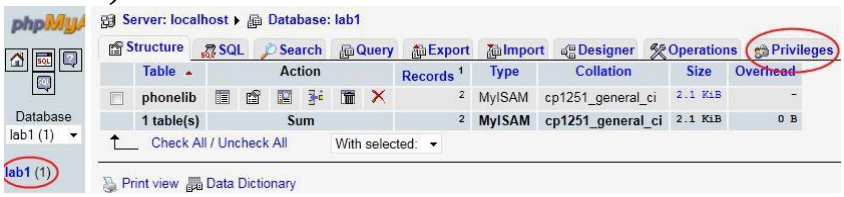


Рисунок 2.9 – Редактирование привилегий

Переходим по ссылке Add a new user и в интерфейсе администратора создаем нового пользователя и пароль (рисунок 2.10). В нашем случае имя пользователя будет совпадать с паролем (student).

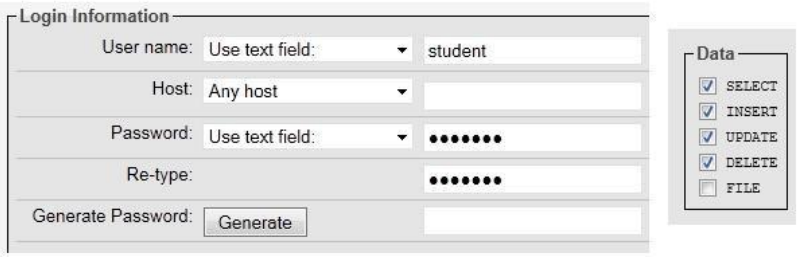


Рисунок 2.10 – Создание нового пользователя, пароля и привилегий

В расположенном ниже интерфейсе определяем права пользователя при работе с базой данных. Как минимум, для обеспечения функциональности необходимо разрешить добавлять записи в таблицу и выбирать данные из нее (операции Insert и Select). В нашем случае разрешим также обновлять и удалять записи (Update и Delete).

Разработка клиентского приложения

Клиент — это аппаратный или программный компонент вычислительной системы, посылающий запросы серверу.

Программа, являющаяся клиентом, взаимодействует с сервером, используя определённый протокол. Она может запрашивать с сервера какие-либо данные, манипулировать данными непосредственно на сервере, запускать на сервере новые процессы и т. п. Полученные от сервера данные клиентская программа может предоставлять пользователю или использовать как-либо иначе, в зависимости от назначения программы. Программа-клиент и программа-сервер могут работать как на одном и том же компьютере, так и на разных. Во втором случае для обмена информацией между ними используется сетевое соединение.

Веб-приложение — клиент-серверное приложение, в котором клиентом выступает браузер, а сервером — веб-сервер и сервер баз данных. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются межплатформенными сервисами. Для создания веб-приложений на стороне сервера используются разнообразные технологии и любые языки программирования, способные осуществлять вывод в стандартную консоль.

В данном курсе приложение-клиент будет разрабатываться средствами языков HTML, JAVA-скрипт, PHP.

Рассмотрим структуру информации после установки пакета разработчика «Denwer».

Предположим, что папка «C:\WebServers», в которую был установлен Denwer проектируется на виртуальный диск S. Тогда в папе «S:\usr\local\mysql-5.1\data\lab1\» будет находится разработанная нами база данных. Для создания клиентского приложения необходимо войти в папку «S:\home\localhost\www\» и создать в ней каталог Lab1. Создаем файл «S:\home\localhost\www\Lab1\index.html», в котором будет размещен HTML-скрипт титульной страницы проекта:

```
<HTML>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<meta name="Author" content="СергеевГеоргийГеоргиевич">
<META NAME="ROBOTS" CONTENT="ALL">
<METANAME="Keywords" CONTENT="лабораторная работа, MySQL, соединение с базой данных">
<META NAME="Description" CONTENT="Лабораторнаяработа '1. Соединение с базой данных">
</head>
<title>Лабораторная работа '1. Соединение с базой данных</title>
<BODY>
<A HREF="phlib.php">Переченьабонентов</A>
</BODY>
</HTML>
```

Если в обозревателе теперь ввести адрес «http://localhost/lab1/», увидим ссылку «Перечень абонентов».

Задачей приложения-клиента в данной лабораторной работе является вывод состояния отношения в виде таблицы. Для этого необходимо выполнить следующие действия:

1) установить соединение с базой данных;
2) выполнить выборку данных;
3) отобразить данные в виде таблицы.

Для общения с MySQL из PHP понадобятся следующие функции:

1) создать соединение с MySQL:

intmysql_connect (string hostname, string username, string password).

Параметры:

Hostname – имя хоста, на котором находится база данных,

Username – имя пользователя,

Password – пароль пользователя.

Функция возвращает параметр типа int, который больше 0, если соединение прошло успешно, и равен 0 в противном случае.

2) выбрать базу данных для работы:

intmysql_select_db(string database_name, intlink_identifier).

Параметры:

Database_name – имя базы данных;

link_identifier – ID соединения, которое получено в функции *mysql_connect*. (параметр необязательный, если он не указывается, то используется ID от последнего вызова mysql_connect).

Функция возвращает значение true или false.

3) *запрос к базе данных:*

intmysql_query(string query, intlink_identifier).

Параметры:

Query – строка, содержащая запрос,

link_identifier – см. предыдущую функцию.

Функция возвращает ID результата или 0, если произошла ошибка.

4) *В цикле выбираем записи, полученные в результате запроса:*

arraymysql_fetch_array (resourceresult [, intresult_type]).

Параметры:

result – указатель на ресурс результатов выборки, второй (необязательный)– флаг, отвечающий за содержимое возвращаемого функцией ассоциативного массива.

Функция возвращает массив, соответствующий извлечённому ряду, или FALSE, если рядов больше нет. Функция *mysql_fetch_array()* – это расширенная версия *mysql_fetch_row()*. Помимо сохранения данных в числовых индексах результирующего массива, она также хранит данные в ассоциативных индексах, используя в качестве ключей имена полей.

5) закрыть соединение

intmysql_close (intlink_identifier).

Параметры:

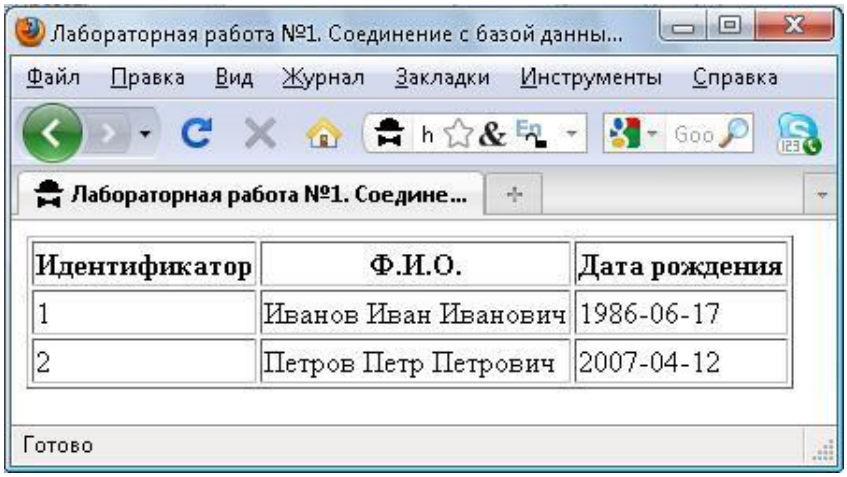
link_identifier – см. выше.

Функция возвращает значение true или false.

Скрипт для выдачи списка абонентов в виде таблицы, расположенный в файле «S:\home\localhost\www\Lab1\phlib.php» будет иметь вид:

```
<HTML>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<meta name="Author" content="Сергеев Георгий Георгиевич">
<META NAME="ROBOTS" CONTENT="ALL">
<METANAME="Keywords" CONTENT="лабораторная работа, MySQL, соединение с базой данных">
<META NAME="Description" CONTENT="Лабораторная работа №1. Соединение с базой данных">
</head>
<title>Лабораторная работа №1. Соединение с базой данных</title>
<BODY>
<?
/* Переменные для соединения с базой данных */
$hostname = "localhost";
$username = "student";
$password = "student";
$dbName = "lab1";
/* Таблица MySQL, в которой хранятся данные */
$userstable = "phonelib";
/* создать соединение */
mysql_connect($hostname,$username,$password) OR DIE("Не могу создать соединение");
/* выбрать базу данных. Если произойдет ошибка - вывести ее */
mysql_select_db($dbName) or die(mysql_error());
/* составить запрос для выборки информации */
$query = "SELECT * FROM $userstable";
/* Выполнить запрос. */
$result=mysql_query($query) or die(mysql_error());
echo "<TABLE BORDER=1>";
echo"<TR><TH>Идентификатор</TH><TH>Ф.И.О.</TH><TH>Датарождения</TH></TR>";
/* Выбрать очередную запись из таблицы. */
while($row=mysql_fetch_array($result))//  берем результаты из каждой строки
{
echo;
/* Вывести ее в виде HTML*/
echo "<tr><td>".$row['ID']."</td><td>".$row['FIO']."</td><td>".$row['bdate']."</td></tr>";
};
echo "</TABLE>";
/* Закрыть соединение */
mysql_close();
?>
</BODY>
</HTML>
```

Результат работы скрипта приведен на рисунке 2.11.



Идентификатор	Ф.И.О.	Дата рождения
1	Иванов Иван Иванович	1986-06-17
2	Петров Петр Петрович	2007-04-12

Рисунок 2.11 – Результат выполнения скрипта выборки из базы данных

2.2 Создание новой базы данных

Инициализировать работу приложения phpMyAdmin.

В разделе «Actions/ MySQL localhost/ Create new database» ввести произвольное имя базы данных, например lab1, нажать кнопку «Create» (рисунок 2.12).

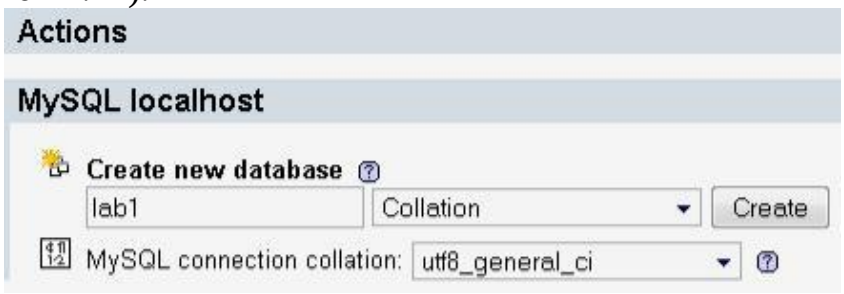


Рисунок 2.12 – Создание новой базы данных

Обратите внимание на SQL-скрипт, расположенный вверху страницы после создания базы данных (рисунок 2.13).

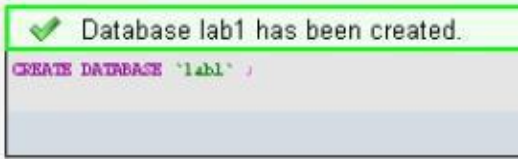


Рисунок 2.13 – SQL-скрипт для создания базы данных.

Создание отношения (таблицы) с помощью интерфейса оболочки phpMyAdmin

Так как БД является моделью реального мира перед созданием таблицы необходимо описать предметную область, которую она представляет. Рассмотрим простейшую модель: телефонный справочник. Данные этой модели определяются следующими атрибутами: уникальный номер абонента (он должен создаваться автоматически), фамилия, имя, отчество, дата рождения (рисунок 2.14).

PHONELIB	
ID	INTEGER
FIO	VARCHAR(50)
BDATE	DATE

Рисунок 2.14 – Атрибуты отношения «Телефонный справочник»

Для создания таблицы в среде phpMyAdmin можно воспользоваться и интерфейсом CreateNewTable (рисунок 2.15).

Рисунок 2.15 – Интерфейс создания таблицы

Описываем в интерфейсе имена атрибутов и их типы (рисунок 2.16).

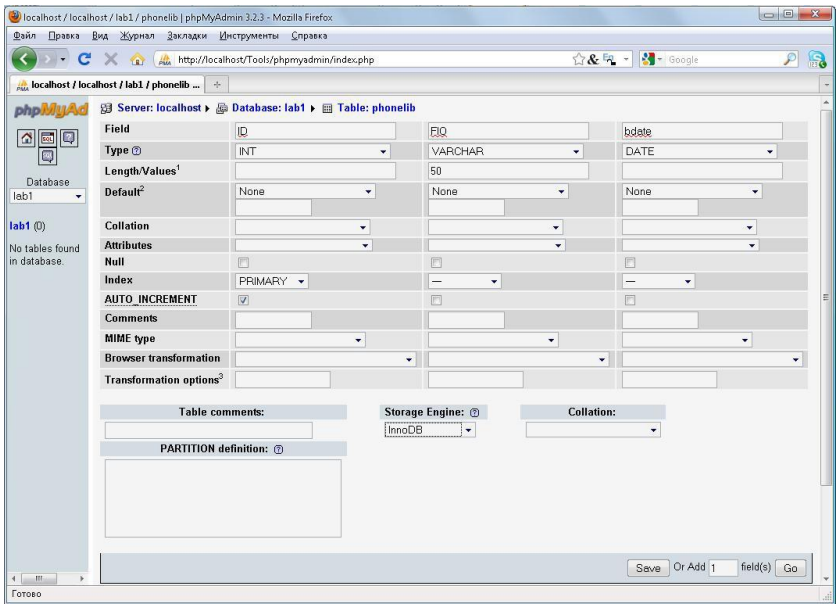


Рисунок 2.16 – Описание атрибутов

Обратите внимание, что поле для поля ID установлен признак AUTO_INCREMENT, для того чтобы при добавлении новой записи (картежа) значение поля генерировалось автоматически. На будущее, установим для этого атрибута признак PRIMARY в поле index. Атрибут FIO (фамилия, имя, отчество) является текстовым, поэтому для него указываем длину 50 (в символах).

Для создания таблицы после описания атрибутов нажимаем кнопку «Save».

Среда phpMyAdmin автоматически генерирует SQL-скрипт для создания новой таблицы:

```
CREATE TABLE `lab1`.`phonelib` (
`ID` INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`FIO` VARCHAR( 50 ) NOT NULL ,
`bdate` DATE NOT NULL
) ENGINE = InnoDB;
```

Синтаксис операции CREATETABLE в MySQL, в целом, соответствует стандарту SQL-92, однако имеет ряд особенностей. В частности, фрагмент фразы `ENGINE = InnoDB` определяет тип «движка» для работы с данными, каждый из которых имеет свои характерные особенности.

MyISAM. Движок по умолчанию. Не поддерживает транзакций, средняя надежность хранения данных. Превосходная производительность чтения данных (через SELECT). Блокирует всю таблицу при записи в неё данных, отчего маленькая производительность при частых записях.

InnoDB. Транзакционный тип движка, применяемый при интенсивных операциях записи, благодаря возможности блокировки уровня строк таблицы. Великолепная восстанавливаемость и высокая надежность хранения данных. Поддерживает внешние ключи. Именно этот движок рекомендуется использовать при выполнении лабораторных работ.

Синтаксис команды CREATE TABLE

CREATE TABLE <имя таблицы> (<опред_столбца> [,<опред_столбца> | <ограничения> ...]);

Обязательно надо указать имя таблицы и определить как минимум один столбец. Пользователь, создавший таблицу, получает все права на нее.

Определение столбцов

<опред_столбца>=

<имя столбца>**{**<тип данных>|**COMPUTED** [**BY**] (<выражение>) | <имя домена>**}**

[**DEFAULT** { literal | **NULL** | **USER**}]

[**NOT NULL**]

[<ограничение столбца>]

Здесь

COMPUTED [**BY**] – “Вычисляется как”

DEFAULT – значение по умолчанию

[<ограничение столбца>] – ограничение целостности столбца

Обязательно должно быть указано имя столбца. Далее должен быть указан тип данных столбца или выражение для его вычисления, или имя домена.

Опционально могут быть указаны:

- значение по умолчанию
- ограничения целостности (PRIMARY KEY, UNIQUE, FOREIGN KEY)
- атрибут NOT NULL (обязательно должен быть указан у PRIMARY KEY и FOREIGN KEY)

Пример определения таблицы:

```
CREATE TABLE salary_history (
emp_no          INTEGER NOT NULL,
change_date     DATE DEFAULT 'NOW' NOT NULL,
updater_id      VARCHAR(20) NOT NULL,
old_salary      NUMERIC(8,3) NOT NULL,
percent_change DOUBLE PRECISION DEFAULT 0 NOT NULL
new_salary COMPUTED BY (old_salary + old_salary * percent_change / 100),
PRIMARY KEY (emp_no, change_date, updater_id),
FOREIGN KEY (emp_no) REFERENCES EMPLOYEE (emp_no)
ON UPDATE CASCADE
ON DELETE CASCADE);
```

Приведем пояснения к оператору. Объявлены поля:

- поле *EMP_NO* целочисленное, не может принимать значение NULL.
- поле *CHANGE_DATE* имеет тип DATE**,** значение по умолчанию – текущая дата, не может принимать значение NULL.
- поле *UPDATER_ID* – строка переменной длины с максимальной длиной 20 символов, не может принимать значение NULL.
- поле *OLD_SALARY* – содержит денежное значение, которое хранится с точностью до трех знаков после запятой, не может принимать значение NULL.
- поле *PERCENT_CHANGE* вещественное число, по умолчанию имеет значение 0, не может принимать значение NULL.
- Поле *NEW_SALARY* вычисляется как *OLD_SALARY* + *OLD_SALARY* * *PERCENT_CHANGE* / 100

Объявлены ограничения:

- объявляется составной первичный ключ, состоящий из трех полей: *EMP_NO*, *CHANGE_DATE*, *UPDATER_ID.*
- описывается внешний ключ (FOREIGN KEY) - поле *EMP_NO* ссылается (REFERENCES) на поле *EMP_NO* таблицы EMPLOYEE, обновление и удаление из таблицы EMPLOYEE каскадируется.

Указание значений по умолчанию

Значение по умолчанию может быть:

- литералом (символьная строка, числовое значение или дата)
- константой NULL (если значение не введено, полю будет присвоено значение NULL)
- константой USER (в поле будет введено имя пользователя). Длины поля должно хватить для сохранения имени пользователя.

Описатель NOT NULL

Если поле описано как NOT NULL, его значение обязательно должно быть определено. Если пользователь забудет определить такое поле, он не сможет добавить строку в таблицу.

Определение вычисляемых полей

Определение:

<имя столбца> COMPUTED [BY] (<выражение>);

Столбцы, на которые ссылается выражение, должны существовать до определения вычисляемого поля.

Например, в следующей таблице поле «Полное имя» вычисляется с помощью конкатенации полей «Имя» и «Фамилия»:

```
CREATE TABLE employee (
first_name VARCHAR(10) NOT NULL,
last_name VARCHAR(15) NOT NULL,
full_name COMPUTED BY (last_name || ', ' || first_name)
);
```

Описание ограничений целостности

Первичный ключ есть минимальный набор полей, обнозначно идентифицирующих строку. Первичный ключ может быть только один, он описывается ограничением целостности PRIMARY KEY. Кроме первичного ключа, в таблице может быть произвольное количество возможных (альтернативных) ключей. Для описания возможных ключей служит ограничение UNIQUE. Смысл UNIQUE такой же, как и PRIMARY KEY – значение поля (полей), описанных как UNIQUE должно быть уникальным по таблице.

В случае, если первичный или альтернативный ключ состоит из одного поля, его можно описать в той же строке, что и само поле (такое ограничение называется ограничением на уровне столбца). Если первичный или альтернативный ключ состоит из нескольких атрибутов, он описывается после определения всех полей таблицы (такое ограничение называется ограничением на уровне таблицы).

Ограничение на уровне столбца имеет следующий синтаксис:

<col_constraint> = [CONSTRAINT <имя ограничения>] <определение ограничения>.

Т.е. определение ограничения состоит из необязательной части, состоящей из служебного слова CONSTRAINT и имени ограничения, и обязательной части, состоящей из собственно определения ограничения. В случае, если первая часть опущена, сервер генерирует имя ограничения автоматически. Посмотреть имя ограничения можно просмотрев системную таблицу RDB$RELATION_CONSTRAINTS.

Определение ограничения на уровне таблицы описывается так:

<определение ограничения> = {
UNIQUE | **PRIMARY KEY**|**CHECK** (<условие проверки>)
| **REFERENCES** <ИмяДругойТаблицы> [(<имя столбца>[, <имя столбца> …])]
[**ON DELETE** {**NO ACTION** | **CASCADE** | **SET DEFAULT** | **SET NULL**}]
[**ON UPDATE** {**NO ACTION** | **CASCADE** | **SET DEFAULT** | **SET NULL**}]
}

Поле можно определить как первичный ключ, его значение можно сделать уникальным по таблице, либо можно указать, что поле ссылается (REFERENCES) на некоторое поле из другой таблицы. Таблица, на которую ссылается поле, называется родительской. Смысл ограничения REFERENCES следующий: значение в поле в дочерней таблицы можно занести только в том случае, если это значение есть в соответствующем поле родительской таблицы. Например, если у нас есть таблица клиентов, и таблица покупок, поле “Номер клиента” из таблицы покупок должно ссылаться на поле “Номер клиента” из таблицы клиентов. Таким образом в таблицу покупое невозможно будет занести номер несуществующего клиента. Вторая часть описателя REFERENCES описывает, какие действия необходимо предпринять при удалении из родительской таблицы (ON DELETE) и при обновлении родительской таблицы (ON UPDATE). Всего возможно четыре варианта:

- NO ACTION - ничего не предпринимать
- CASCADE – каскадировать. В случае удаления из родительской таблицы будут удалены все связанные записи из дочерней таблицы. Например, удаление клиента повлечет удаление всех его покупок. При модификации записи из родительской таблицы будут модифицированы также записи в дочерней таблице. Например, если клиент поменял номер с пятого на десятый, во всех его покупках поле “Номер клиента” также изменит свое значение с 5 на 10.
- SET DEFAULT – поле получает свое значение по умолчанию
- SET NULL – поле устанавливается в NULL

Какое именно действие необходимо предпринимать зависит от смысла таблиц. При удалении клиента удалять все его покупки имеет смысл. Но при

удалении желтого цвета из таблицы цветов не стоит удалять все желтые автомобили.

Ограничения целостности на уровне таблицы описывается так же, как и ограничение на уровне столбца, за исключением определения самого ограничения (уф). Оно описывается так:

<ограничение> = {{**PRIMARY KEY | UNIQUE**} (<имя столбца> [,<имя столбца> …])
| **FOREIGN KEY** (<имя столбца> [,<имя столбца> …]) **REFERENCES** <ИмяДругойТаблицы>
[**ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}**]
[**ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}**]

Отличие в том, что в PRIMARY KEY, UNIQUE и FOREIGN KEY можно описать более одного столбца.

<u>Синтаксис команды ALTER TABLE</u>

```
ALTER [IGNORE] TABLE tbl_name alter_spec [, alter_spec ...]

alter_specification:
        ADD [COLUMN] create_definition [FIRST | AFTER column_name
]
  или   ADD [COLUMN] (create_definition, create_definition,...)
  или   ADD INDEX [index_name] (index_col_name,...)
  или   ADD PRIMARY KEY (index_col_name,...)
  или   ADD UNIQUE [index_name] (index_col_name,...)
  или   ADD FULLTEXT [index_name] (index_col_name,...)
  или   ADD [CONSTRAINT symbol] FOREIGN KEY index_name
(index_col_name,...)
            [reference_definition]
  или   ALTER [COLUMN] col_name {SET DEFAULT literal | DROP
DEFAULT}
  или   CHANGE [COLUMN] old_col_name create_definition
              [FIRST | AFTER column_name]
  или   MODIFY [COLUMN] create_definition [FIRST | AFTER
column_name]
  или   DROP [COLUMN] col_name
  или   DROP PRIMARY KEY
  или   DROP INDEX index_name
  или   DISABLE KEYS
  или   ENABLE KEYS
  или   RENAME [TO] new_tbl_name
  или   ORDER BY col
  или   table_options
```

Команда используется для модификации структуры существующей таблицы. С помощью ALTER TABLE можно:

- добавить новый столбец к таблице
- удалить столбец из таблицы
- удалить ограничение целостности со столбца или с таблицы целиком
- изменить имя столбца, его тип, и позицию столбца в таблице

Перед тем как использовать ALTER TABLE

- сохраните данные таблицы
- удалите все ограничения целостности со столбца

Сохранение данных это процесс, состоящий из пяти шагов. Например, один из столбцов таблицы имеет тип CHAR(3), его надо поменять на CHAR(5). Для этого надо:

1. Создать временныи столбец, определив его так же, как существующий.
2. Переписать данные из существующего столбца во временный
3. Модифицировать временный столбец
4. Переписать данные из временного столбца обратно
5. Уничтожить временный столбец

1. Добавляем новый столбец

```
ALTER TABLE EMPLOYEE ADD TEMP_NO CHAR(3);
```

2. Переписываем

```
UPDATE EMPLOYEE SET TEMP_NO = OFFICE_NO;
```

3. Модифицируем

```
ALTER TABLE ALTER TEMP_NO TYPE CHAR(4);
```

4. Переписываем обратно

```
UPDATE EMPLOYEE SET OFFICE_NO = TEMP_NO;
```

5. Удаляем

```
ALTER TABLE DROP TEMP_NO;
```

Удаление столбца

ALTER TABLE <имя таблицы> DROP <имя столбца>[, <имя столбца>...];

Например:

```
ALTER TABLE EMPLOYEE
DROP EMP_NO,
DROP FULL_NAME;
```

Команда ALTER TABLE не сможет выполниться при следующих условиях:

- если данные в таблице после удаления нарушают ограничения PRIMARY KEY или UNIQUE

- если столбец является частью первичного ключа, ограничения UNIQUE, или столбец является внешним ключом
- столбец используется в представлении (view), в триггере, или используется при вычислении другого поля.

В случае, если что-либо мешает удалить столбец, надо сначала удалить это «что-то», а затем удалять столбец. Отсюда мораль – таблицу надо создавать один раз и на всю жизнь. Семь раз мерить, потом резать. Кажется, что легче уничтожить таблицу целиком и создать ее заново, чем что-либо в ней изменить. Уничтожать таблицу жалко, если в ней уже есть данные, и невозможно, если таблица связана с другими таблицами (сначала надо удалить связи). Вывод: к созданию таблицы надо подходить очень ответственно, так как любые ошибки исправляются тяжело.

Добавление столбца

ALTER TABLE <имя таблицы> ADD <определение столбца>

Например:

ALTER TABLE employee ADD emp_no INTEGER NOT NULL;

Добавление ограничения целостности на таблицу

ALTER TABLE <имя таблицы> ADD [CONSTRAINT <имя ограничения>] <определение ограничения>;

Можно добавить ограничения PRIMARY KEY, FOREIGN KEY и UNIQUE.

Например:

ALTER TABLE EMPLOYEE ADD CONSTRAINT DEPT_NO UNIQUE(PHONE_EXT);

Удаление ограничения целостности, наложенного на столбец

ALTER TABLE <имя таблицы>

DROP CONSTRAINT <имя ограничения>;

Например:

ALTER TABLE PROJECT

DROP CONSTRAINT TEAM_CONSTRT;

Если имя ограничения небыло указано явно при создании таблицы, сервер генерирует имя автоматически. Посмотреть имена ограничений можно в системной таблице RDB$RELATION_CONSTRAINTS:

SELECT * FROM RDB$RELATION_CONSTRAINTS

Модификация столбца в таблице:

Можно изменить имя столбца, тип столбца, позицию столбца в таблице.

ALTER TABLE <имя таблицы> ALTER [COLUMN] <имя столбца> <определение изменения>

<определение изменения>:

- для изменения имени: TO <новое имя>
- для изменения типа данных: TYPE <новый тип>
- для изменения позиции: MODIFY <определение столбца> AFTER <столбец после которого>

Примеры:

1. Изменение позиции EMP_NO после EMP_NUM

ALTER TABLE EMPLOYEE MODIFY EMP_NO CHAR(20) AFTER EMP_NUM;

2. Изменение имени EMP_NO на EMP_NUM

ALTER TABLE EMPLOYEE ALTER EMP_NO TO EMP_NUM;

3. Изменение типа

ALTER TABLE EMPLOYEE ALTER EMP_NUM TYPE CHAR(20);

Преобразование из не-символьных в символьные типы возможны при следующих ограничениях.

- BLOB и массивы не преобразуются.
- Новое определение поля должно вмещать старые данные (CHAR(3) не вместит CHAR(4))

ЛАБОРАТОРНАЯ РАБОТА № 3
СРЕДСТВА МАНИПУЛИРОВАНИЯ ДАННЫМИ ЯЗЫКА SQL

Цель работы:
Изучение операторов INSERT, UPDATE, DELETE и простейших форм оператора SELECT.

Порядок выполнения работы:
1) Изучить методические рекомендации по выполнению работы.
2) Открыть в phpMyAdmin созданную в л/р №2 базу данных, содержащую одну таблицу в соответствии с вариантом задания.
3) Используя интерфейсные свойства оболочки phpMyAdmin, добавить в таблицу не менее 10 записей произвольного содержания, но имеющих смысловое значение. В 4-5 записях содержимое поля должно соответствовать условию, указанному в варианте задания.
4) Вызвать диалог SQL-редактора программы phpMyAdmin. Ввести 4-5 запросов для добавления записи в таблицу, при этом среди них обязательно должны быть запросы:
- на добавление одного кортежа в таблицу;
- на добавление неполных данных в таблицу (если возможно);
- на добавление одного кортежа, но без соблюдения определенного порядка следования полей;
- на добавление одновременно нескольких кортежей в таблицу.

5) Изменить содержимое полей последней введенной записи с помощью SQL-запроса.
6) Изменить содержимое любого поля всех записей, удовлетворяющих условию (по варианту), с помощью SQL-запроса.
7) Удалить все записи, удовлетворяющие условию, с помощью SQL-запроса. Такие записи должны существовать в таблице, для чего повторно выполнить п.3).

Теоретические сведения

К базовым средствам манипулирования данными языка SQL относятся «поисковые» варианты операторов UPDATE и DELETE. Эти варианты называются поисковыми, потому что при задании соответствующей операции задается логическое условие, налагаемое на строки адресуемой оператором таблицы, которые должны быть подвергнуты модификации или удалению. Кроме того, в такую категорию языковых средств входит оператор INSERT, позволяющий добавлять строки в существующие таблицы.

Базовые средства манипулирования данными

Оператор INSERT добавляет одну или более новых строк данных к существующей таблице или виду. INSERT - одна из привилегий базы данных, которая контролируется инструкциями GRANT и REVOKE.

Значения вставляются в столбцы строки по порядку их следования,

если не задан факультативный список столбцов. Если список столбцов задан на множестве всех доступных столбцов, во все неперечисленные столбцы автоматически вставляется или значение по умолчанию, или NULL.

Если факультативный список столбцов упущен, предложение VALUES должно содержать значения для всех столбцов таблицы.

Чтобы вставить одну строку данных, должно присутствовать предложение VALUES и содержать определенный список значений.

Чтобы вставить несколько строк данных, определите <select_expr>, которое возвращает уже существующие данные из другой таблицы. Выбранные строки должны соответствовать списку столбцов.

Предупреждение: Допустимо выбирать из той же таблицы, в которую строки вставляются, но эта практика не рекомендуется, так как подобные действия могут привести к бесконечным вставкам строк (зацикливанию).

Синтаксис:

```
INSERT INTO <object> [(col [, col ...])]
{VALUES (<val> [, <val> ...]) | <select_expr>};
<object> = tablename | viewname
<val> = {<constant> | <expr>| <function> | NULL | USER}
[COLLATE collation]
```

Обратите внимание: Предложение COLLATE не может быть использовано для BLOB значений.

<constant> = num | "string" | charsetname "string"

<expr> = Допустимое выражение SQL, которое возвращает в одиночное значение столбца.

<function> = {CAST (<val> AS <datatype>)
| UPPER (<val>) | GEN_ID (generator, <val>) }

<select_expr> = SELECT возвращающий ноль или более строк, где число столбцов в каждой строке такое же, как число элементов, которые должны быть вставлены.

INTO <object> Имя существующей таблицы или вида, в которую вставляются данные.

col Имя существующего столбца в таблице или виде, в который вставляются значения.

VALUES (<val> [, <val> ...] Список значению для вставки в таблицу или вид. Значения должны быть в том же порядке, как целевые столбцы.

<select_expr> Запрос, который возвращает значения, для вставки в целевые столбцы.

Пример:

Следующая инструкция добавляет строку в таблицу, присваивает

значения двум столбцам:

```
INSERT INTO EMPLOYEE_PROJECT (EMP_NO, PROJ_ID) VALUES
(52, "DI");
```

Оператор UPDATE изменяет одну или более существующих строк в таблице или виде. Факультативное предложение WHERE может быть использовано, чтобы ограничить UPDATE к некоторому подмножеству строк таблицы. Модификации не могут модифицировать секторы массива. Предупреждение: Если предложение WHERE упущено, UPDATE изменяет все строки в таблице. Обратите внимание: Когда модифицируются BLOB столбцы, UPDATE заменяет весь BLOB целиком новым значением.

Синтаксис:

```
UPDATE {table | view}SET col = <val> [, col = <val> ...]
[WHERE <search_condition>;<val> = {col [<array_dim>] |
<constant> | expr> | <function>  | NULL | USER
}<array_dim> = [x:y [, x:y ...]]
```

Обратите внимание: внешние скобки должны присутствовать в ссылке на массив.

<constant> = num | "string" | charsetname "string"

<expr> = Допустимое выражение SQL, которое возвращает одиночное значение.

<function> = {CAST (<val> AS <datatype>) | UPPER (<val>) | GEN_ID (generator, <val>) }

table | view - Имя существующей таблицы или вида для модификции.

SET col = <val> Определяет столбцы для изменения и значения, которые требуется присвоить этим столбцам.

WHERE <search_cond> Модифицировать только найденное. Определяет условия, которым строка должна удовлетворять, чтобы изменится.

Примеры:

Следующая инструкция изменяет столбцы для всех строк таблицы:

```
UPDATE CITIES SET POPULATION = POPULATION * 1.03;
```

Следующая инструкция использует предложение WHERE, чтобы ограничить модификацию столбцов подмножеством строк:

```
UPDATE COUNTRY SET CURRENCY = "USDollar" WHERE COUNTRY =
"USA";
```

Оператор DELETE определяет одну или более строк, чтобы удалить из таблицы или редактируемого (updatable) вида. DELETE - одна из привилегий, которые контролируются инструкциями GRANT и REVOKE.

Для определения строк, которые следует удалить, может быть использовано факультативное предложение WHERE.

Предостережение: Если не используется предложение WHERE, удалены все строки из таблицы.

Синтаксис:

```
DELETE FROM table [WHERE <search_condition>];
```

Table - Имя таблицы из которой удаляются строки.

<search_condition> Условия поиска, которые определяют строки для удаления. Если это предложение не используется, DELETE воздействует на все строки в определенной таблице или виде.

Примеры:

Следующая инструкция удаляет все строки из таблицы:

```
DELETE FROM EMPLOYEE_PROJECT;
```

Следующая инструкция удаляет строку для служащего #141:

```
 DELETE FROM SALARY_HISTORY WHERE EMP_NO = 141;
```

Манипулирование данными таблицы с помощью phpMyAdmin.

Среда phpMyAdmin предоставляет возможности интерактивного манипулирования данными. Рассмотрим этот процесс на примере редактирования данных таблицы, разработанной в предыдущей лабораторной работе.

Для этого запускаем среду и выбираем из перечня нашу базу данных (рисунок 3.1).

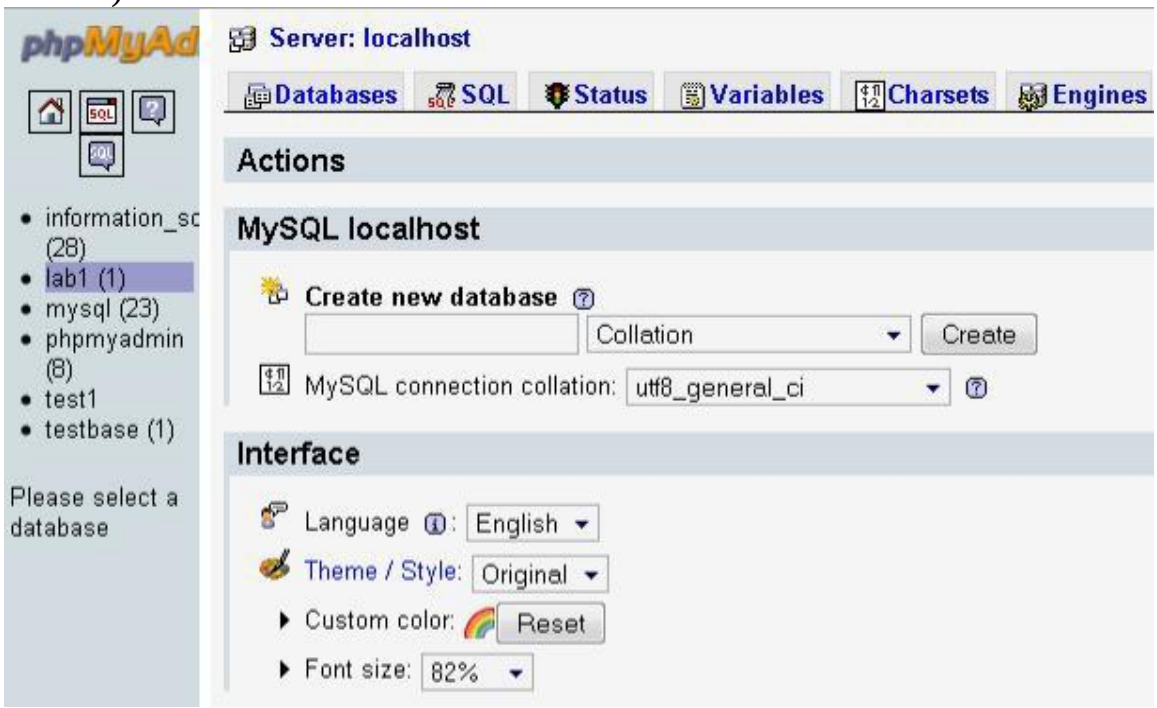


Рисунок 3.1 – Выбор базы данных для редактирования

Далее выбираем для редактирования таблицу phonelib и нужную

запись для редактирования (рисунок 3.2).

Рисунок 3.2 – Выбор таблицы и записи для редактирования

Выполняем редактирование записи. Обратите внимание на текст SQL-скрипта для обновления строки таблицы (рисунок 3.3).

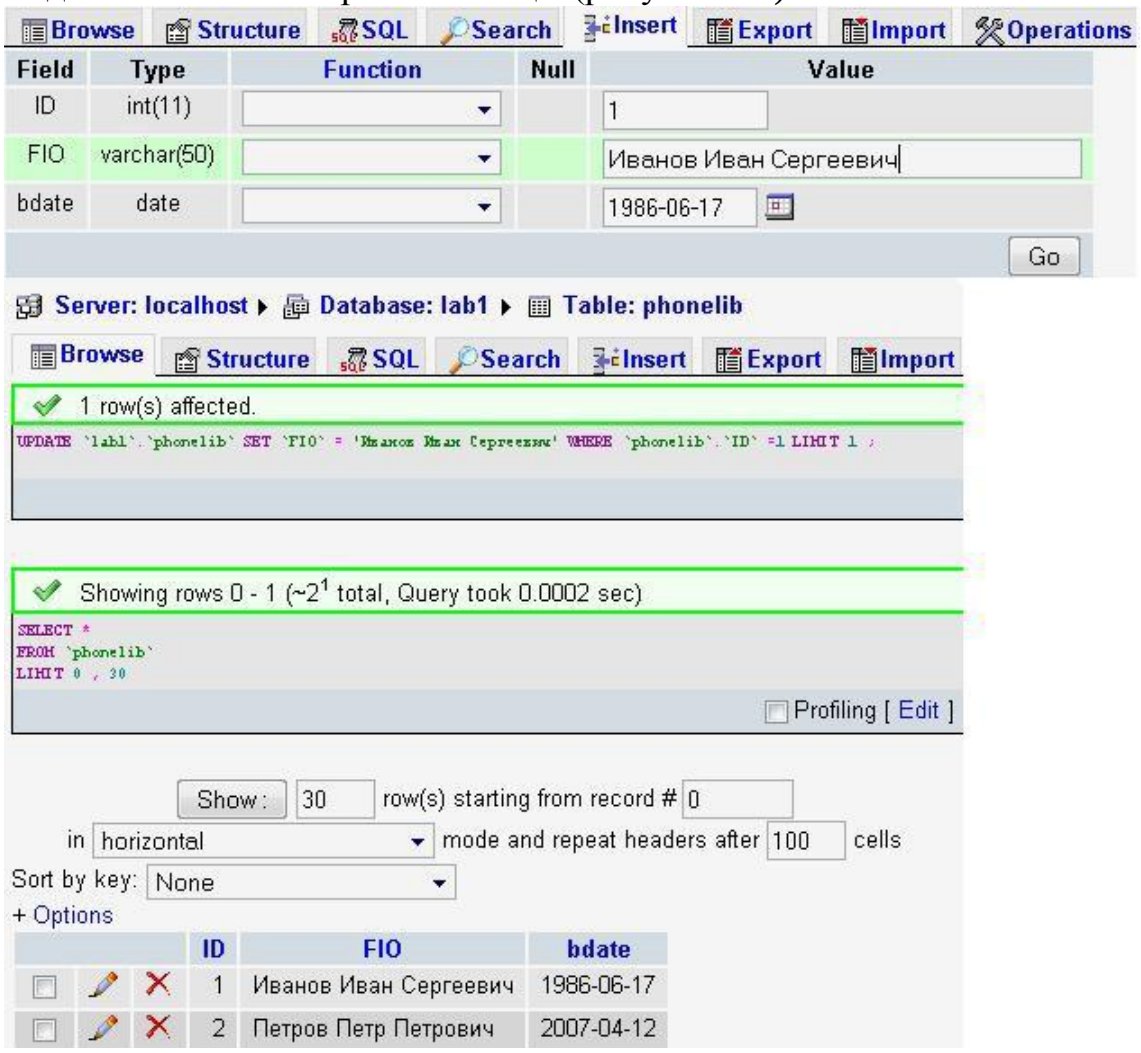


Рисунок 3.3 – Редактирование записи в среде phpMyAdmin

ЛАБОРАТОРНАЯ РАБОТА № 4
ЭЛЕМЕНТАРНЫЕ ВЫБОРКИ ПОСРЕДСТВОМ ОПЕРАТОРА SELECT. АГРЕГАТНЫЕ ФУНУКЦИИ SQL

Цель работы:
Изучение работы оператора SELECT языка SQL, а также изучение возможностей обработки данных с помощью агрегатных функций языка SQL.

Порядок выполнения работы:

1) Изучить методические рекомендации по выполнению работы.
2) Открыть в phpMyAdmin созданную в л/р №2и №3 базу данных, содержащую **одну таблицу**. Обязательно убедиться в наличии в таблице записей необходимых для реализации запросов на выборку. В случае отсутствия таковых, добавить необходимые записи в таблицу.
3) Продемонстрировать все записи таблицы, указанной в варианте задания, с помощью SQL-запроса.
4) С помощью SQL-запроса продемонстрировать действие модификатора DISTINCT.
5) Ограничить вывод данных с помощью SQL-запроса, используя WHERE с простым условием.
6) Ограничить вывод данных с помощью SQL-запроса, используя WHERE и составное условие.
7) С помощью SQL-запроса продемонстрировать действие специальных функций IN, BETWEEN, LIKE и IS NULL в условии.
8) С помощью SQL-запроса продемонстрировать работу специальных функций с условием NOT.
9) Ознакомиться с принципами работы агрегатных функций COUNT, SUM, AVG, MAX, MIN. Составить SQL-запрос с одной из агрегатных функций.
10) С помощью SQL-запроса продемонстрировать использование COUNT(*).
11) С помощью SQL-запроса продемонстрировать выполнение простых вычислений в запросе.
12) Использовать простое вычисление, как параметр агрегатной функции, в SQL-запросе.
13) Ознакомиться с использованием предложения GROUP BY, продемонстрировать его работу с помощью SQL-запроса.
14) Ознакомиться с использованием предложения HAVING, продемонстрировать его работу с помощью SQL-запроса.
15) С помощью SQL-запросов выполнить следующие задания по варианту (ПРИЛОЖЕНИЕ 3):

- осуществить фильтрацию информации по заданному полю;
- получить результат агрегатной функции по полю группировки, (функция и поле группировки указаны в варианте задания);

• выполнить сортировку информации по возрастанию или убыванию значений в поле, выбранном пользователем из списка всех полей таблицы (направление сортировки задано вариантом задания).

Все запросы формулировать на естественном языке и структурированном языке запросов SQL. Проверить работу вышеупомянутых запросов с помощью SQL-редактора программы phpMyAdmin.

Запросы на выборку формулируются к базе данных, определенной вариантом задания к первой и второй лабораторным работам. В ПРИЛОЖЕНИИ 3 приводится имя таблицы БД, к которой необходимо осуществлять запросы, а также данные для выполнения п.15: поле, по которому должна осуществляться фильтрация, поле группировки и агрегатная функция для реализации запроса с группировкой, направление сортировки строк.

Студент может **добавить в таблицу дополнительные неключевые поля**, если сочтет это полезным для смысловой наглядности разрабатываемых запросов.

Теоретические сведения

Большинство операций с базой данных заключается в выборке из нее необходимой пользователю информации. Для этого служит оператор SELECT, позволяющий формировать набор данных, удовлетворяющий условиям поиска и группировки данных. В том числе, оператор SELECT используется для подведения итогов, получения статистической информации и т.д.

Агрегатные функции SQL

Агрегатные функции – это функции, которые работают не с одним значением, взятым из строки таблицы, а с группой значений. Например, рассмотрим таблицу 4.2, в которой хранится информация о продажах товаров:

Таблица 4.2 – Продажи товаров

Дата продажи	№ товара	Группа товара	Цена покупки
11.12.89	1	1	10.00
11.12.89	2	1	20
11.12.89	3	2	15
12.12.89	1	1	100
12.12.89	2	1	50

Группировать поля таблицы можно многими способами. Например, можно сгруппировать продажи по дням, тогда сумма поля «Цена покупки»

даст общую выручку по дням. Группируя по номеру товара, и суммируя по тому же полю получим сумму продаж товара с таким-то номером за все дни. Группируя таблицу по группам товара, получим сумму покупок всех товаров данной группы за все дни. В нашем примере мы вычисляем сумму от некоторой группы, сумма в данном случае – агрегатная функция. Группировать можно не только по одному полю, но и по нескольким.

В случае, если запрос использует агрегатные функции, то в списке полей, которые выводит запрос, могут быть только поля – аргументы агрегатных функций. Все остальные поля обязательно должны быть перечислены в предложении GROUP BY (см. ниже).

Общий алгоритм, по которому работают агрегатные функции следующий: сначала таблица упорядочивается по тем полям, по которым осуществляется группировка; затем от каждой сформированной группы вычисляется требуемое значение, которое и заносится в результат.

Существуют следующие агрегатные функции:

- COUNT считает количество строк, которые вернул запрос;
- SUM суммирует значения данного поля;
- AVG находит среднее значение поля;
- MAX находит максимальное значение поля;
- MIN находит минимальное значение поля.

Рассмотрим работу агрегатных функций на примере базы данных, состоящий из трех таблиц – «Торговый агент», «Покупатель», «Сделка». Таблица 4.3 «Торговый агент» (Salespeople) содержит следующие столбцы:

- SNUM – номер агента;
- SNAME – имя агента;
- CITY – город, где работает агента;
- COMM – комиссионные торгового агента.

Таблица 4.3 – Торговый агент (Salespeople)

SNUM	SNAME	CITY	COMM
1001	Peel	London	0.12
1002	Serres	San Jose	0.13
1004	Motika	London	0.11
1007	Rifkin	Barcelona	0.15
1003	Axelrod	New York	0.10

Таблица 4.4 «Покупатель» (заказчик) состоит из следующих столбцов:

- CNUM – номер покупателя;
- CNAME – имя покупателя;
- CITY – город проживания покупателя;
- RATING – некоторый рейтинг, присвоенный покупателю;
- SNUM – номер торгового агента, за которым закреплен покупатель.

Таблица 4.4 – Покупатель (Customers)

CNUM	CNAME	CITY	RATING	SNUM
2001	Hoffman	London	100	1001
2002	Giovanni	Rome	200	1003
2003	Liu	SanJose	200	1002
2004	Grass	Berlin	300	1002
2006	Clemens	London	100	1001
2008	Cisneros	SanJose	300	1007
2007	Pereira	Rome	100	1004

Таблица 4.5 «Сделка» (заказ) содержит следующие столбцы:

- ONUM – номер заказа;
- AMT – сумма заказа;
- ODATE – дата заказа;
- CNUM – номер покупателя;
- SNUM – номер торгового агента.

Таблица 4.5 – Сделка (Orders)

ONUM	AMT	ODATE	CNUM	SNUM
3001	18.69	10/03/1990	2008	1007
3003	767.19	10/03/1990	2001	1001
3002	1900.10	10/03/1990	2007	1004
3005	5160.45	10/03/1990	2003	1002
3006	1098.16	10/03/1990	2008	1007
3009	1713.23	10/04/1990	2002	1003
3007	75.75	10/04/1990	2004	1002
3008	4723.00	10/05/1990	2006	1001
3010	1309.95	10/06/1990	2004	1002
3011	9891.88	10/06/1990	2006	1001

Ниже приведены тексты запросов к базе данных.

Чтобы найти сумму всех покупок в таблице «Сделки», мы можем ввести следующий запрос: **SELECT SUM** ((*amt*)) **FROM** *Orders*;

Результат его выполнения – число 26658.4

Запрос «**SELECT AVG** (*amt*) **FROM** *Orders*;» вернет среднее значение сумм сделок -2665.84.

Функция **COUNT** отличается от других агрегатных функций тем, что она не выполняет математических действий над значением столбца. Она считает число значений в данном столбце, или число строк в таблице. Если необходимо подсчитать количество различных значений некоторого поля в таблице, функцию COUNT надо использовать с DISTINCT. Например, чтобы подсчитать количество продавцов в настоящее время описанных в таблице сделок, мы можем использовать следующий запрос:

SELECT COUNT (**DISTINCT** *snum*) **FROM** *Orders*;

Результат – 5.

Иногда возникает необходимость решить обратную задачу – подсчитать количество значений поля вместе с повторениями. Для этого существует описатель **ALL** (он подразумевается по умолчанию). Например:

SELECT COUNT (**ALL** *rating*) **FROM** *Customers*;

Подсчитает количество повторений поля rating с повторениями.

Чтобы подсчитать общее число строк в таблице, используйте функцию COUNT со звездочкой вместо имени поля, как, например, в следующем примере.

SELECT COUNT (*) **FROM** *Customers*;

Результат – 7.

Внимание! Только **COUNT** (*) может подсчитывать значения NULL. Все остальные функции игнорируют неопределенные значения.

Простые вычисления.

Столбцы, значение которых может быть получено с помощью простых арифметических действий через другие столбцы в БД, как правило, не хранятся. SQL предоставляет простой способ производить подобные вычисления. Также можно помещать в некоторый столбец константу. Например, чтобы представить комиссионные торгового агента в процентном отношении, а не в виде десятичного числа можно записать такой запрос:

SELECT *snum, sname, city, comm**100 **FROM** *Salespeople* **FROM** *Salespeople*;

Можно дополнить наш запрос знаком процента, выводимым после поля *comm**100:

SELECT *snum, sname, city, comm**100, ' % ' **FROM** *Salespeople*;

Знак процента в данном случае – константное выражение. При выполнении вычислений в запросе допустимы арифметические действия – сложение, вычитание, умножение, деление.

Вычисления в агрегатных функциях.

Допустим, мы хотим вывести 10% от стоимости самой дорогой сделки. Тогда можно записать такой запрос:

SELECT MAX (amt*0.1) **FROM** *Orders*;

Таким образом можно выполнять вычисления с использованием других полей и арифметических действий. Вложение агрегатов не допускается.

Использование GROUP BY.

Как уже было упомянуто ранее, таблицу можно группировать самыми разными способами. В случае, если мы не группируем таблицу вообще, в списке вывода запроса присутствуют только агрегатные функции. Таблица в этом случае рассматривается целиком. В случае, если мы группируем таблицу по какому-то полю, в списке вывода могут присутствовать те поля, относительно которых таблица группируется и агрегатные функции. При

этом все поля, не охваченные агрегатными функциями, должны быть перечислены в предложении GROUP BY.

Предположим, что мы хотим найти сделку с максимальной стоимостью для каждого торгового агента. Можно сделать персональный запрос для каждого из них, выбрав MAX(amt) из таблицы сделок. Можно записать следующий запрос:

SELECT *snum*, **MAX** (*amt*) **FROM** *Orders* **GROUP BY** *snum*;

Результат выполнения запроса представлен в таблице 4.6.

Таблица 4.6

snum	
1001	767.19
1002	1713.23
1003	75.75
1014	1309.95
1007	1098.16

В данном случае таблица группируется относительно номера торгового агента, поэтому поле *snum* присутствует в списке вывода запроса и в предложении GROUP BY.

Использование HAVING.

HAVING подобен WHERE – он задает условия отбора групп так же, как это делает WHERE для строк. Например, если мы хотим узнать, какие торговые агенты имеют заработок от одной сделки более чем 3000, и в какой день, можно использовать следующий запрос:

SELECT *snum, odate*, **MAX** ((*amt*))
FROM *Orders*
GROUP BY *snum, odate*
HAVING MAX ((*amt*)) > 3000.00;
Результат работы запроса представлен в таблице 4.7.

Таблица 4.7

Snum	odate	------
1001	10/05/1990	4723.00
1001	10/06/1990	9891.88
1002	10/03/1990	5160.45

Внимание! В предложении HAVING нельзя проверять имена полей на какое-либо условие – для этого существует WHERE. То, что вы проверяете в HAVING должно иметь только одно значение для группы.

ЛАБОРАТОРНАЯ РАБОТА № 5
ССЫЛОЧНАЯ ЦЕЛОСТНОСТЬ ДАННЫХ. СОЗДАНИЕ СХЕМЫ БАЗЫ ДАННЫХ

Цель работы:

Ознакомиться с принципами построения схемы базы данных и обеспечения целостности данных в базе.

Порядок выполнения работы:

1) Проанализировать схему БД своего варианта задания (вариант тот же, что и в лабораторной работе № 2-4)
2) Выделить, классифицировать и описать **все существующие связи**, определить необходимые ограничения целостности.
3) С помощью SQL-запроса изменить существующую таблицу таким образом, чтобы она могла участвовать в связях с другими таблицами (например, с помощью оператора ALTER TABLE).
4) С помощью SQL-запросов создать еще не созданные таблицы. В процессе создания таблиц установить связи между таблицами.
5) Обязательно добавить во все таблицы записи (можно с помощью интерфейса phpMyAdmin).
6) Составить SQL-запросы на ввод данных в главную и подчиненную таблицу (выбрать любую пару таблиц). Проверить работу ограничений на значения первичного ключа обеих таблиц и внешнего ключа подчиненной таблицы.
7) Составить SQL-запросы на обновление и удаление данных для проверки работы ограничений целостности связей между таблицами. Проверить работу ограничений целостности в случаях установки каскадирования и запрета удаления и обновления данных.

Теоретические сведения

Свое название реляционные базы данных (РБД) получили, в том числе, потому, что таблицы в БД не существуют независимо друг от друга. Таблицы взаимосвязаны друг с другом, т.е. действие, произведенное в одной таблице, вызовет некоторые действия в другой таблице. Существует три основных класса связей между таблицами: один к одному (1:1), один ко многим (1:M), и многие ко многим (M:M). На практике связи первого типа используются редко. Связи третьего типа не реализуются в РБД напрямую, одну связь многие ко многим приводят к двум связям один ко многим

Понятие сущности, связи и целостности базы данных

Процесс проектирования базы данных обычно состоит из трех этапов:

- исследование предметной области;
- анализ данных (сущностей и их атрибутов);
- определение отношений между сущностями и определение первичных и вторичных (внешних) ключей.

В процессе проектирования определяется структура реляционной БД (состав таблиц, их структура и логические связи). Структура таблицы определяется составом столбцов, типом данных и размерами столбцов, ключами таблицы.

К базовым понятиям модели БД «сущность – связь» относятся: сущности, связи между ними и их атрибуты (свойства).

Сущность – любой конкретный или абстрактный объект в рассматриваемой предметной области. Сущности – это базовые типы информации, которые хранятся в БД (в реляционной БД каждой сущности назначается таблица). К сущностям могут относиться: студенты, клиенты, подразделения и т.д. Экземпляр сущности и тип сущности – это разные понятия. Понятие тип сущности относится к набору однородных личностей, предметов или событий, выступающих как целое (например, студент, клиент и т.д.). Экземпляр сущности относится, например, к конкретной личности в наборе. Типом сущности может быть студент, а экземпляром – Петров, Сидоров и т. д.

Связь - взаимосвязь между сущностями в предметной области. Связи представляют собой соединения между частями БД (в реляционной БД – это соединение между записями таблиц).

Сущности - это данные, которые классифицируются по типу, а связи показывают, как эти типы данных соотносятся один с другим. Если описать некоторую предметную область в терминах сущность - связь, то получим модель сущность - связь для этой БД.

Связь устанавливается между двумя общими полями (столбцами) двух таблиц. Существуют связи с отношением «один-к-одному», «один-ко-многим» и «многие-ко-многим».

Отношения, которые могут существовать между записями двух таблиц:

1) один - к - одному, каждой записи из одной таблицы соответствует одна запись в другой таблице;

2) один - ко - многим, каждой записи из одной таблицы соответствует несколько записей другой таблице;

3) многие - к - одному, множеству записей из одной таблице соответствует одна запись в другой таблице;

4) многие - ко - многим, множеству записей из одной таблицы соответствует несколько записей в другой таблице.

Тип отношения в создаваемой связи зависит от способа определения связываемых полей:

1) отношение «один-ко-многим» создается в том случае, когда только одно из полей является полем первичного ключа или уникального индекса.

2) отношение «один-к-одному» создается в том случае, когда оба связываемых поля являются ключевыми или имеют уникальные индексы.

3) отношение «многие-ко-многим» фактически представляется двумя отношениями «один-ко-многим» и третьей (связующей) таблицей,

первичный ключ которой состоит из полей первичного ключа двух других таблиц.

Ключ − это столбец (может быть несколько столбцов), позволяющий установить связь с записями в другой таблице. Существуют ключи двух типов: первичные и вторичные (внешние).

Первичный ключ − это одно или несколько полей (столбцов), комбинация значений которых однозначно определяет каждую запись в таблице. Первичный ключ не допускает значений Null (отсутствия значений) и всегда должен иметь уникальный индекс. Первичный ключ используется для связывания таблицы с внешними ключами в других таблицах.

Внешний (вторичный) ключ - это одно или несколько полей (столбцов) в таблице, содержащих ссылку на поле или поля первичного ключа в другой таблице. Значения в поле внешнего ключа могут повторяться. Внешний ключ определяет способ объединения таблиц.

Из двух логически связанных таблиц одну называют таблицей первичного ключа или главной таблицей, а другую таблицей вторичного (внешнего) ключа или подчиненной таблицей. СУБД позволяют сопоставить родственные записи из обеих таблиц и совместно вывести их в форме, отчете или запросе.

Существует три типа первичных ключей: ключевые поля счетчика (счетчик), простой ключ и составной ключ.

Поле счетчика (Тип данных «Счетчик»). Тип данных поля в базе данных, в котором для каждой добавляемой в таблицу записи в поле автоматически заносится уникальное числовое значение.

Если поле содержит уникальные значения, такие как коды или инвентарные номера, то это поле можно определить как первичный ключ. В качестве ключа можно определить любое поле, содержащее данные, если это поле не содержит повторяющиеся значения или значения Null.

В случаях, когда невозможно гарантировать уникальность значений каждого поля, существует возможность создать ключ, состоящий из нескольких полей. Чаще всего такая ситуация возникает для таблицы, используемой для связывания двух таблиц (связь типа многие - ко -многим).

Необходимо еще раз отметить, что в поле первичного ключа должны быть только уникальные значения в каждой строке таблицы, т.е. совпадение не допускается, а в поле вторичного или внешнего ключа совпадение значений в строках таблицы допускается.

Если возникают затруднения с выбором подходящего типа первичного ключа, то в качестве ключа целесообразно ввести поле типа «счетчик».

Связь вида 1:М в реляционных СУБД используется наиболее часто, поэтому ее мы рассмотрим наиболее подробно.

В предыдущих лабораторных работах нами была разработана таблица «Список абонентов». Очевидно, что для этой таблицы первичным ключом является поле счетчика ID, так как его значение однозначно определяет значения полей FIO и bdate. Предоставим СУБД MySQL информацию о том, что это ключевое поле. Это можно сделать с помощью SQL-скрипта вида:

```
ALTER TABLE `phonelib` DROP PRIMARY KEY, ADD PRIMARY KEY(`ID`)
```

или с помощью визуального интерфейса в среде phpMyAdmin (рисунок 5.1).

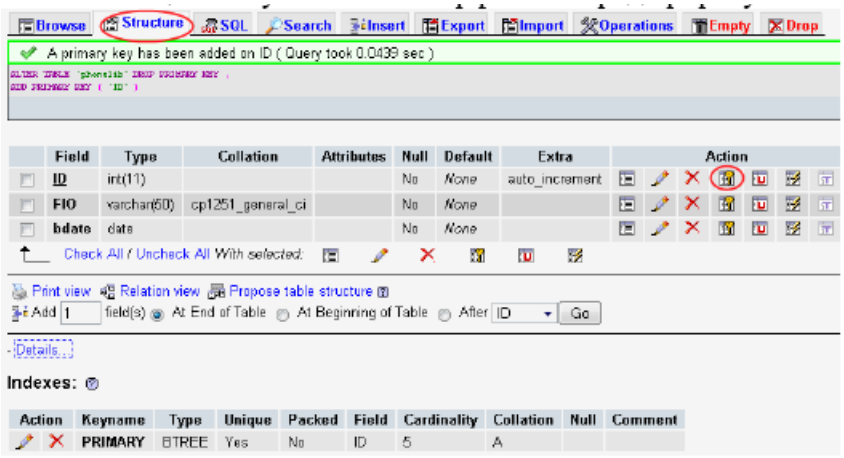


Рисунок 5.1 – Создание первичного ключа в среде phpMyAdmin

Пример. Теперь выполним анализ предметной области: абонент характеризуется фамилией, именем, отчеством и датой рождения. Каждый абонент может иметь несколько телефонных номеров: домашний, мобильный, рабочий и т.п. Отношение сущность-связь для базы данных показано на рисунке 5.2.

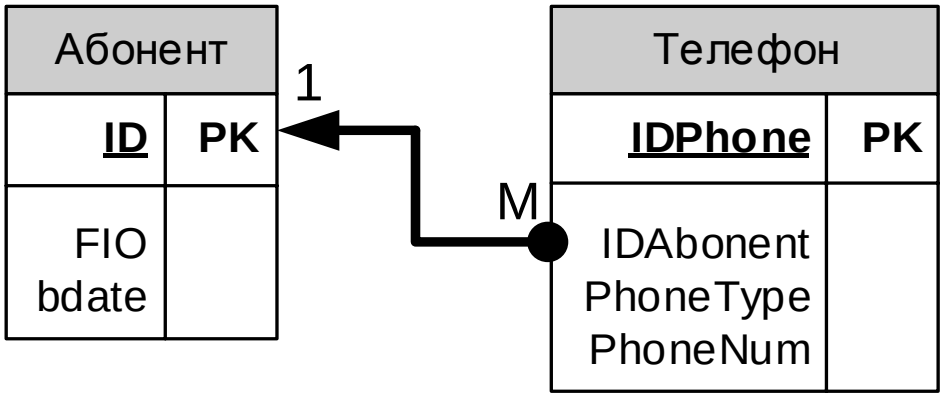


Рисунок 5.2 - Отношение сущность-связь

Очевидно, что поля «ID» и «IDAbonent» в обеих таблицах имеет одинаковый смысл – номер, присвоенный абоненту. В случае если таблицы не имеют связи друг с другом, ничто не мешает нам добавить в таблицу «Телефон» строку, например, (142, «Домашний», «0506521318»), несмотря на то, что в таблице «Абонент» нет абонента с номером 142. Это нарушение целостности данных, так как такая строка может быть занесена, но она не соответствует действительности. Чтобы подобная ситуация стала невозможной, необходимо установить связь между таблицами по полю «ID». В этом случае сервер БД автоматически проследит, чтобы значение поля «IDAbonent» из вставляемой строки существовало в поле «ID» таблицы «Телефон».

Подобной проверки достаточно, чтобы условие целостности данных выполнялось при добавлении данных в таблицы. Но его недостаточно при манипулировании данными. Рассмотрим ситуацию, когда мы удаляем из таблицы «Абонент» некоторую запись, например абонента с номером 1. В случае если таблицы не связанны, удаление абонента повлечет за собой изменение только одной таблицы. В таблице «Телефон» останутся сведения о телефонных номерах абонента с номером 1. Такая ситуация – также нарушение целостности данных, так как абонента в БД нет, а его телефонные номера присутствуют. В случае если таблицы связанны, удаление абонента должно повлечь за собой удаление всех его номеров (тогда говорят, что удаление каскадируется).

Такая же ситуация с модификацией данных в родительской таблице. Если абонент с номером 1 изменил номер на 5, то в связанных таблицах изменение родительской таблицы повлечет за собой автоматическое изменение в полях связи дочерних таблиц.

Сформируем ограничения, обеспечивающие целостность базы данных «Телефонный справочник». Следует отметить, что понятие ссылочной целостности появилось в MySQL сравнительно недавно (начиная с версии 5.0) и большая часть «движков» ее не поддерживает. Поэтому мы и используем «движок» InnoDB при создании таблиц. Интерфейс phpMyAdmin также не имеет возможности визуально устанавливать внешние ключи и определять правила каскадного удаления и обновления базы данных. Поэтому создание дочерней таблицы мы будем выполнять с помощью SQL-скрипта

```
CREATE TABLE `MyPhoneNums` (
  `IDPhone` int(11) NOT NULL AUTO_INCREMENT,
  `IDAbonent` int(11) DEFAULT NULL,
  `PhoneType` varchar(20) DEFAULT NULL,
  `PhoneNum` varchar(12) DEFAULT NULL,
  PRIMARY KEY (`IDPhone`),
  KEY `abonent_key` (`IDAbonent`),
  CONSTRAINT `abonent_key` FOREIGN KEY (`IDAbonent`) REFERENCES `phonelib`
(`ID`) ON DELETE CASCADE ON UPDATE CASCADE) ENGINE=InnoDB AUTO_INCREMENT=2
DEFAULT CHARSET=cp1251;
```

В данном скрипте фраза PRIMARY KEY (‘IDPhone’) означает, что поле счетчика IDPhone является первичным ключом, KEY ‘abonent_key’ (‘IDAbonent’) CONSTRAINT ‘abonent_key’ FOREIGN KEY (‘IDAbonent’) REFERENCES ‘phonelib’ (‘ID’) ON DELETE CASCADE ON UPDATE CASCADE определяет внешний ключ и устанавливает правила каскадного обновления и удаления записей в родительской таблице phonelib.

ЛАБОРАТОРНАЯ РАБОТА № 6
КОНЦЕПТУАЛЬНЫЙ И ЛОГИЧЕСКИЙ УРОВНИ ПРОЕКТИРОВАНИЯ БД

Цель работы

Выполнение лабораторной работы преследует следующие цели:

- ознакомление с методом проектирования БД Сущность-Связь
- изучение Метода нормальных форм (логический уровень проектирования БД)

Порядок выполнения работы

Задание 1. Концептуальный уровень проектирования БД

1) Построить ER-диаграмму типов для данной предметной области.

2) Привести правило перевода данной ER-диаграммы в концептуально-логическую схему реляционной БД.

3) По ER-диаграмме типов построить концептуально-логическую схему реляционной БД. Критерий качества макета БД: отсутствие избыточного дублирования данных и пустых полей в таблицах.

4) Обозначить все потенциальные и внешние ключи.

5) Охарактеризовать связи между таблицами (тип действий при удалении и обновлении данных оговорен в варианте задания). Описать, как должна осуществляться проверка ссылочной целостности при добавлении, удалении и обновлении данных в главной и подчиненной таблице после установки такой связи?

Задание 2. Логический уровень проектирования реляционных баз данных

1-4) ***Продемонстрируйте процедуру нормализации отношений. Для этого представьте все данные о предметной области в виде одного отношения (реляционной таблицы). При заполнении этой таблицы конкретными значениями (заполнить не менее 10 строк), проанализируйте типы связей между объектами предметной области и учтите функциональные зависимости, существующие в предметной области, в том числе, дополнительную функциональную зависимость, оговоренную в варианте задания.***

5) Определить потенциальный ключ (ключи) отношения.

6) Привести диаграмму функциональных зависимостей, существующих в отношении (БД), определить номер нормальной формы, в которой находится отношение.

7) Определить возможные аномалии при работе с отношением (привести примеры).

8) Представить процесс декомпозиции исходного отношения на отношения в последующих нормальных формах. Ограничиться третьей нормальной формой (НФБК). Для каждого этапа декомпозиции выполнить п. 5) – 7).

ЛАБОРАТОРНАЯ РАБОТА № 7
РАЗРАБОТКА СЛОЖНЫХ ЗАПРОСОВ К БАЗЕ ДАННЫХ. ЗАПРОСЫ НА ОСНОВЕ НЕСКОЛЬКИХ ТАБЛИЦ. КОРРЕЛИРОВАННЫЕ ВЛОЖЕННЫЕ ПОДЗАПРОСЫ

Цель работы:

Выполнение лабораторной работы преследует следующие цели:

- изучить способы получения информации из нескольких таблиц;
- записать запросы, демонстрирующие выборки из нескольких таблиц с использованием оператора JOIN и без него;
- изучить способы выполнения и принцип действия рекурсивных запросов;
- научится использовать вложенные подзапросы;
- ознакомиться с возможностями построения вложенных коррелированных подзапросов с применением кванторов.

Порядок выполнения работы:

1) Проанализировать созданную в л/р №5 базу данных.

2) Сформулировать и выполнить с помощью SQL-редактора PhpMyAdmin запросы, соединяющие две таблицы с помощью JOIN и без него.

3) Сформулировать и выполнить с помощью SQL-редактора PhpMyAdmin запросы, соединяющие более чем две таблицы с помощью JOIN и без него.

4) Продемонстрировать с помощью SQL-редактора PhpMyAdmin следующие возможности SQL:

- использование псевдонимов на примере рекурсивного запроса;
- запрос с подзапросом;
- использование агрегатных функций в подзапросе;
- подзапросы, возвращающие единственное и множественные значения;
- подзапросы, использующие вычисление;
- использование подзапросов в HAVING;

5) Ознакомится с принципом и продемонстрировать работу коррелированных подзапросов:

- привести пример соединения таблицы со своей копией;
- привести пример коррелированного запроса, использующего две разные таблицы;
- привести пример запроса с оператором EXIST;
- привести пример запроса с оператором ALL;
- привести пример запроса с оператором ANY.

Разработать и выполнить с помощью SQL-редактора PhpMyAdmin обязательные запросы, заданные вариантом (ПРИЛОЖЕНИЕ 5)

Теоретические сведения

Запросы к нескольким таблицам являются наиболее часто используемым типом запросов, так как реляционные БД нормализованы, и хранимая информация разбита по большому количеству таблиц.

7.1 Простое объединение двух таблиц

Существует два основных способа объединения таблиц – с помощью оператора JOIN языка SQL, и с помощью условия в разделе WHERE. Например, объединить таблицу Customers и таблицу Salespeople из базы данных, рассмотренной в лабораторной работе №3, по полю city можно следующим образом:

```
SELECT Customers.cname,
Salespeople.sname,
Salespeople.city
FROM Salespeople, Customers
WHERE Salespeople.city = Customers.city;
```

Для выполнения данного запроса сервер произведет декартово произведение двух таблиц, после чего выполнит операцию селекции нужных строк, затем – проекцию нужных полей. Декартово произведение – это операция, которая требует для своего выполнения максимальное количество памяти и процессорного времени, по сравнению с другими известными операциями. Данный способ неэффективен.

В случае, если мы используем оператор JOIN, запрос будет выглядеть так:

```
SELECT Customers.cname,
Salespeople.sname,
Salespeople.city
FROM Salespeople JOIN Customers ON Salespeople.city = Customers.city;
```

В данном случае сервер БД не будет производить декартово произведение, сервер воспользуется индексами для определения совпадающих значений поля city, выберет строки с совпадающими значениями, а затем произведет проекцию. Данная операция происходит на порядок быстрее. В случае, если по полям учавствующим в соединении, не создан индекс, сервер вынужден производить декартово произведение.

Можно сделать следующие выводы:

- если объединение таблиц происходит по ключам, выгоднее использовать JOIN;

- если запрос выполняется медленно, нужно создать индекс по полю – параметру объединения (CREATE INDEX)

7.2 Объединение более чем двух таблиц

Объединение более чем двух таблиц по форме не отличается от простого объединения. Например, чтобы объединить три таблицы по некоторому условию, можно записать:

```
SELECT onum, cname, Orders.cnum, Orders.snum
FROM Salespeople, Customers, Orders
WHERE Customers.city <> Salespeople.city AND
Orders.cnum = Customers.cnum AND
Orders.snum = Salespeople.snum;
```

Тот же запрос, переписанный с использованием JOIN, выглядит более сложно:

```
SELECT onum, cname, Orders.cnum, Orders.snum
FROM (Orders JOIN Customers ON Orders.cnum = Customers.cnum) JOIN Salespeople ON Orders.snum = Salespeople.snum
WHERE Customers.city <> Salespeople.city AND
```

7.3 Псевдонимы и рекурсивные объединения

На практике часто встречается ситуация, когда необходимо объединять таблицу саму с собой. Запрос, выполняющий такое объединение, называется рекурсивным. Например, если мы хотим вывести все пары студентов с одинаковым рейтингом, можно записать следующий запрос:

```
SELECT one.cname, two.cname, one.rating
FROM Students one, Students two
WHERE one.rating = two.rating;
```

Здесь в списке вывода SELECT указываются поля cname и rating таблицы one, и поле cname таблицы two. В разделе FROM указывается, что one и two – просто псевдонимы для таблицы Students. Для выполнения запроса сервер создаст две копии таблицы Students, одну с именем one, другую с именем two, выполнит запрос так, как будто это две разные таблицы, и уничтожит копии. Естественно, сервер физически не создает копий таблиц, но с псевдонимами в запросе он работает так, будто это две разные таблицы.

Можно заметить, что вывод запроса будет повторять каждую пару дважды - сначала «Иванов - Петров», затем «Петров - Иванов». Кроме того, вывод содержит строки «Иванов - Иванов», «Петров - Петров». Это происходит потому, что сервер берет первую строку из первого псевдонима и сравнивает ее со всеми строками из второго псевдонима. Будут выбраны строки «Иванов - Иванов» и «Иванов - Петров». Затем он переходит к

следующей строке и снова сравнивает ее со всеми строками из второго псевдонима, и так далее. Будут выбраны строки «Петров - Иванов» и «Петров - Петров». Чтобы избежать дубликатов, надо наложить еще одно условие, налагающие отношение порядка на сравниваемые строки. Например, сравнивать имена.

```
SELECT one.cname, two.cname, one.rating
FROM Students one, Students two
WHERE one.rating = two.rating AND
one.cname < two.cname;
```

7.3 Вложенные подзапросы

Вложенные подзапросы так же служат для получения информации из нескольких таблиц. С их помощью можно выполнять рекурсивные запросы. Для чего существует два способа сделать одно и то же действие? Дело в том, что на практике встречаются ситуации, когда запрос выражается очень сложно через соединения, и легко – через вложенный подзапрос, и наоборот. На конкретном сервере БД запрос с использованием JOIN может выполняться очень долго, а с использованием подзапроса – быстро.

Пример запроса с вложенным подзапросом:

```
SELECT *FROM Orders
WHERE snum =( SELECT snum FROM Salespeople WHERE sname = 'Motika');
```

Так как подзапрос стоит после знака равенства, он должен возвращать только одно значение. В случае, если подзапрос вернет более чем одно значение, произойдет ошибка.

Обратите внимание, что при записи подзапроса допустима следующая форма:

<имя/константа> <оператор> <подзапрос>,

а не

<подзапрос> <оператор> <имя/константа> или

< подзапрос > < оператор > < подзапрос >.

Во вложенных подзапросах можно использовать агрегатные функции:

```
SELECT *
FROM Orders
WHERE amt >
(SELECT AVG (amt)
FROM Orders
WHERE odate = 10/04/1990 );
```

Ограничение на вложенный подзапрос то же самое – он должен возвращать единственное значение. В случае, если подзапрос возвращает несколько записей, вместо операций сравнения нужно использовать IN:

```
SELECT *
FROM Orders
WHERE snum IN
(SELECT snum
FROM Salespeople
WHERE city = "LONDON" );
```

Данный запрос более просто записывается с использованием соединения:

```
SELECT onum, amt, odate, cnum, Orders.snum
FROM Orders, Salespeople
WHERE Orders.snum = Salespeople.snum
AND Salespeople.city = "London";
```

Допустимо использовать выражение, основанное на столбце, а не просто сам столбец в предложении SELECT подзапроса:

```
SELECT *
FROM Customers
WHERE cnum =
(SELECT snum + 1000
FROM Salespeople
WHERE sname = Serres );
```

Также допустимы подзапросы в выражении HAVING:

```
SELECT rating, COUNT ( DISTINCT cnum )
FROM Customers
GROUP BY rating
HAVING rating >( SELECT AVG (rating)
FROM Customers
WHERE city = " San Jose');
```

7.4 Построение коррелированных вложенных подзапросов

Вложенные подзапросы и операция JOIN предоставляют большие возможности по манипулированию данными из нескольких таблиц. Существует еще один способ - использовать коррелированные подзапросы, он более сложен для понимания, но в некоторых случаях позволяет формулировать запросы самым коротким образом. Кроме того, есть некоторые вещи, доступные для соединения и недоступные для вложенных подзапросов, и наоборот. Например, при использовании подзапроса становятся возможным использовать агрегатные функции в предикате запроса. При использовании объединений в выводе запроса могут участвовать поля из всех таблиц объединения.

Коррелированным называется подзапрос, который использует псевдоним таблицы определенный не во вложенном запросе, а во внешнем.

При этом вложенный запрос выполняется много раз - для каждой строки внешней таблицы. Пусть требуется найти всех покупателей, совершивших покупки 10.03.1990. Можно записать следующий коррелированный запрос:

```
SELECT *FROM Customers outer
WHERE 10.03.1990 IN
(SELECT odate
FROM Orders inner WHERE outer.cnum = inner.cnum );
```

Логика работы запроса такова: внутренний запрос производит соединение таблицы покупателей и таблицы покупок (outer.cnum = inner.cnum). Для тех строк, у которых условие WHERE выполняется, запрос возвращает дату покупки. Таким образом, для каждого покупателя формируется множество дней, когда он делал покупки. Полученное множество передается во внешний запрос. В случае, если 10.03.1990 входит в сформированное множество, внешний запрос возвращает всю строку (*) строку из таблицы Customers.

Примечание. – Если внешний запрос и внутренний подзапрос обращаются к разным таблицам, то использование псевдонимов не обязательно.

Приведем процесс выполнения коррелированного подзапроса в алгоритмической форме:

1) Выбрать строку из таблицы во внешнем запросе (строка-кандидат).
2) Сохранить значения полей строки-кандидата в псевдониме.
3) Выполнить подзапрос. Везде, где найдена ссылка на псевдоним из внешнего запроса, подставлять значения из текущей строки-кандидата. Использование значения из строки-кандидата называется внешней ссылкой.
4) Оценить предикат внешнего запроса на основе результатов подзапроса. Если предикат верен - строка выбирается для вывода
5) Перейти на следующую строку во внешнем запросе.

Конечно, подобный запрос можно гораздо легче записать другим способом. Например:

```
SELECT DISTINCT *
FROM Customers, Orders
WHERE Customers.cnum = Orders.cnum
AND Orders.odate = 10.03.1990;
```

Более показательным является следующий пример. Пусть нам необходимо вывести имена и номера всех продавцов которые имеют более одного заказчика

```
SELECT snum, sname
FROM Salespeople main
WHERE 1 <
(SELECT COUNT (*)
FROM Customers
WHERE snum = main.snum );
```

В случае, если перед именем поля не указывается имя псевдонима, то данное поле относится к текущему подзапросу. Таким образом, поле snum во вложенном подзапросе относится к таблице Customers, а не к Salespeople. В случае, если такого поля в соответствующей таблице нет, оно ищется в подзапросе верхнего уровня. Так, если бы поле snum отсутствовало в таблице Customers, оно относилось бы к Salespeople.

7.5 Соотнесение таблицы со своей копией

Коррелированные запросы можно писать, основываясь только на одной таблице. Данное свойство делает их незаменимыми при написании аналитических запросов (т.е. запросов, производящих сложный анализ данных). Например, пусть нам необходимо найти все покупки со значениями суммы покупки выше среднего значения для каждого покупателя (т.е. надо найти все покупки данного покупателя, найти среднюю сумму покупок, и выдать все покупки со стоимостью выше средней):

```
SELECT *
FROM Orders outer
WHERE amt > =
(SELECT AVG (amt)   FROM Orders inner
WHERE inner.cnum = outer.cnum );
```

Существуют так называемые специальные операторы SQL, они имеют смысл только для подзапросов. Отличительная особенность специальных операторов - они принимают подзапрос как аргумент, точно так же, как это делает IN.

7.6 Оператор (квантор) EXISTS

EXISTS(X) - булевский оператор. Он получит значение "истинна", если запрос X вернет хоть одну строку. Допустим, нам необходимо узнать список продавцов, у которых есть более одного покупателя

```
SELECT DISTINCT snum
FROM Customers outer
WHERE EXISTS
(SELECT *
FROM Customers inner
WHERE inner.snum = outer.snum
AND inner.cnum < > outer.cnum );
```

Для каждой строки-кандидата из внешнего запроса (продавец, проверяемый в настоящее время), внутренний запрос находит строки, у которых совпадают значения поля snum (номер продавца), но не совпадают значениея поля cnum (номер покупателя). Если не указать DISTINCT, каждый продавец будет выбран один раз для каждого своего заказчика.

На практике очень часто приходится использовать EXIST вместе с NOT. Допустим, нам необходимо вывести список продавцов, за которыми числится только один покупатель

```
SELECT DISTINCT snum
FROM Customers outer
WHERE NOT EXISTS ( SELECT *
FROM Customers inner
WHERE inner.snum = outer.snum
AND inner.cnum < > outer.cnum );
```

Кроме EXISTS в подзапросах возможно использование еще двух операторов - ANY и ALL.

7.7 Оператор ANY

Условие с ANY становится верным, если значение из верхнего подзапроса совпадает по крайней мере с одним значением из вложенного подзапроса. Например если значение - кандидат равно 1, а вложенный подзапрос вернул {1, 2, 3}, условие с оператором ANY станет верным (внешний запрос проверяет на равенство). Например, если нам нужно найти всех продавцов, которые живут в тех же городах, что и покупатели, можно записать следующий запрос:

```
SELECT *
FROM Salespeople
WHERE city = ANY  (SELECT city   FROM Customers );
```

Существует синоним оператора ANY - SOME. Так как SQL похож на английский, то одни предложения верно звучат с ANY, другие - с SOME. Действие операторов эквивалентно.

7.8 Оператор ALL

Действие оператора ALL противоположно оператору ANY. Оператор ALL становится верным, если все значения из вложенного подзапроса равны значению-кандидату из внешнего запроса. Например, если значение-кандидат равно 1, а вложенный подзапрос вернул {1, 1, 1}, оператор ALL станет верным (внешний запрос проверяет на равенство).

Например, если нам необходимо найти всех продавцов, у которых рейтинг выше чем у любого продавца из Рима, можно записать следующий запрос:

```
SELECT *
FROM Customers
WHERE rating > ALL (SELECT rating
FROM Customers
WHERE city = Rome);
```

Операторы ANY и ALL можно выразить через EXIST в коррелированном подзапросе, в явном виде они нужны лишь для упрощения записи запроса. Обратное утверждение неверно, т.е. не все то, что можно выполнить с помощью EXIST, можно сделать с помощью ANY и ALL.

Примечание 1. – Когда говорят, что значение больше (или меньше) чем любое (ANY) из набора значений, это то же самое, что сказать, что оно больше (или меньше) чем любое отдельно взятое из этих значений. И наоборот, сказать что некоторое значение не равно всему (ALL) набору значений, это то же самое, что сказать, что в наборе нет такого же значения.

Примечание 2. – В случае, если подчиненный запрос вернул пустое множество, оператор ALL становится верным, а ANY - ложным.

ЛАБОРАТОРНАЯ РАБОТА № 8
СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ПРЕДСТАВЛЕНИЙ И ИНДЕКСОВ

Цель работы

Ознакомится с принципом работы пользовательских представлений и индексов, продемонстрировать работу с ними на примере.

Порядок выполнения работы

Задание 1. Создание представления

Для заданной вариантом базы данных разработать и выполнить с помощью SQL-редактора PhpMyAdmin следующие запросы:

1) SQL-запрос на создание представления для ввода данных в таблицу (с использованием ограничений);
2) SQL-запрос, который продемонстрирует успешный ввод данных через созданное представление (с учетом заданных ограничений на ввод);
3) SQL-запрос, который продемонстрирует отрицательный результат при вводе данных через созданное представление (с учетом заданных ограничений на ввод) и объяснить причину;
4) SQL-запрос на удаление созданного представления;
5) SQL-запрос на создание представления для вывода неполных данных из двух или более таблиц одновременно;

Задание 2. Создание индекса

Разработать и выполнить с помощью SQL-редактора PhpMyAdmin следующие запросы:

1) SQL-запрос на создание уникального индекса;
2) SQL-запрос на создание индекса на поле, по которому сортируются данные;
3) SQL-запрос на создание индекса на поле, по которому происходит группировка во время агрегации;
4) SQL-запрос на создание составного индекса с учетом селективности колонок (обосновать свой выбор);
5) SQL-запрос на создание составного индекса, причем обосновать очередность перечисления полей в индексе (группировка, сортировка, частота использования) и их количество.

После каждого запроса на создание индекса представить соответствующий запрос на выборку, для которого индексация имеет смысл. После чего обязательно удалять созданный индекс.

Контрольные вопросы

1) Что такое "пользовательское представление"?
2) Условия редактируемости представлений?
3) Назначение пользовательских представлений?
4) Привести примеры пользовательских представлений?

5) Отличие представлений от таблиц?
6) Организация ограничения доступа пользователей к набору данных?
7) Условие обеспечения целостности данных?
8) Внешняя модель и создание представлений при разработке БД?
9) Назначение индексов.
10) Типы индексов.
11) Порядок выполнения запросов на выборку при создании составного индекса.
12) Когда целесообразно использование индексов?
13) Когда не стоит использовать индексы?

Теоретические сведения

Представления – это таблицы чье содержание выбирается или получается из других таблиц. Они работают в запросах и операторах DML точно также, как и основные таблицы, но не содержат никаких собственных данных.

Представления подобны окнам, через которые возможно просматривать информацию, которая фактически хранится в базовой таблице. Представление – это фактически запрос, который выполняется всякий раз, когда представление становится темой команды. Вывод запроса при этом в каждый момент становится содержанием представления.

Можно выделить следующие назначения представления:

- когда необходимо дать пользователю возможность просматривать не всю таблицу, а только ее часть;
- для объединения двух или более таблиц;
- вывод статистических данных.

Индекс – это упорядоченный (буквенный или числовой) список столбцов или групп столбцов в таблице. Таблицы могут иметь большое количество строк, а, так как строки не находятся в каком-нибудь определенном порядке, их поиск по указанному значению может потребовать времени. Индексный адрес – это и забота, и в то же врем обеспечение способа объединения всех значений в группы из одной или больше строк, которые отличаются одна от другой.

В то врем как индекс значительно улучшает эффективность запросов, использование индекса несколько замедляет операции модификации DML (такие как INSERT и DELETE), а сам индекс занимает объем памяти. Следовательно, при создании таблицы необходимо определять, индексировать ее или нет. Индексы могут состоять из многочисленных полей. Если больше чем одно поле указывается для одного индекса, второе упорядочивается внутри первого, третье внутри второго, и так далее.

SQL автоматически создает индекс, когда создается ограничение PRIMARY KEY или UNIQUE.

8.1 Представления

Команда создания представления: CREATE VIEW. Она состоит из слов CREATE VIEW (СОЗДАТЬ ПРЕДСТАВЛЕНИЕ), имени представления которое нужно создать, слова AS (КАК), и далее запроса на выборку данных, который и будет отражать содержимое представления.

CREATE VIEW <имя представления> [(<имя столбца> [,<имя столбца>...])] AS<подзапрос>[WITH CHECK OPTION];

<подзапрос>::=<оператор SELECT>;

Пример:

```
CREATE VIEW PhoneView AS
SELECT pl.vcFamil, pl.vcName, pl.vcSurName,
        dDateBirthDay, vcPhoneNumber
FROM tbPeoples pl, tbPhoneNumbers pn
WHERE pn.idPeoples=*pl.idPeoples
```

Чтобы использовать это представление, сформулируем запрос SELECT:

```
SELECT * FROM PhoneView
```

Результат выполнения запроса:

vcFamil	vcName	vcSurName	dDateBirthDay	vcPhoneNumber
ИВАНОВ	ИВАН	ИВАНОВИЧ	2004-01-31	(978) 102-51-01
ИВАНОВ	ИВАН	ИВАНОВИЧ	2004-01-31	(978) 163-31-52
ПЕТРОВ	ИВАН	ИВАНОВИЧ	1971-04-03	(978) 112-02-46
…	…	…	….	…..

Представление является обновляемым, и может распространять изменения, вносимые в представление на базовые таблицы, только при выполнении некоторых условий. Если условия не выполняются, то представление может служить только для чтения. Термины "обновляемое" и "только для чтения" относятся к содержимому представления, а не к его объявлению.

Условия обновляемости представлений:

– представление является подмножеством одной таблицы, либо другого обновляемого представления (все столбцы базовых таблиц, не включенные в представление, должны позволять состояние NULL);

– оператор SELECT не должен содержать подзапросов, слова DISTINCT, раздела HAVING, агрегатных функций, объединения таблиц, пользовательских функций, хранимых процедур.

Итак, представление – это просто запрос на языке SQL, который выбирает данные, а в базе данных оно выглядит как таблица и работа с ним происходит также. Из представления можно выбирать данные SQL запросами и также назначать права доступа.

Удалить представление можно с помощью команды DROP VIEW:

DROP VIEW name;

Здесь name - имя удаляемого представления. Не существует способов изменить существующее представление. Можно удалить старое представление и заново создать новое.

8.2 Индексы

Синтаксис для создания индекса:

CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX < имя индекса> ON<имя таблицы>(<имя столбца>,[<упоряд.>][,<имя столбца> [<упоряд.>]...]);

<упоряд.>::=ASC (возрастание)| DESC (убывание);
{ по умолчанию ASC }

Таблица, конечно, должна уже быть создана и должна содержать имя соответствующего столбца. Однажды созданный, индекс будет невидим пользователю.

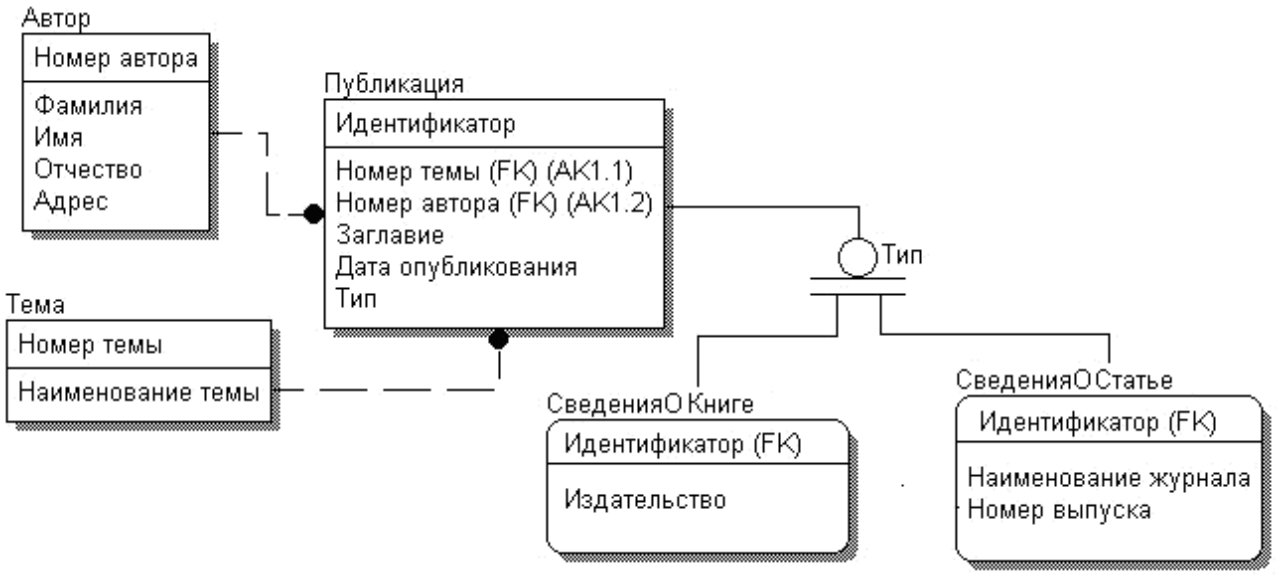
БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Агальцов В. П. Базы данных : в 2 т. : т. 1. Локальные базы данных : учебник / В. П. Агальцов. - М. : ФОРУМ, ИНФРА-М, 2013. - 352с.
2. Водолазский В. Эффективная работа: PHP 4. — СПб. : Питер, 2002. — 416 с.
3. Голицына, О. Л. Базы данных / О. Л. Голицына, Н. В. Максимов, И. И. Попов. - М. : Форум, 2004. - 352 с.
4. Голицына, О. Л. Базы данных : учебное пособие / О. Л. Голицына, Н. В. Максимов, И. И. Попов. - М. : Форум, 2012. - 400 с.
5. Гофман В. Э. Работа с базами данных в Delphi/ В. Э. Гофман, А. Д. Хомоненко. – СПб. : БХВ-Петербург, 2002. – 624 с.
6. Дейт К. Дж. Введение в системы баз данных / К. Дж. Дейт. - М. : Наука, 2006.-464с.
7. Карпова И. П. Базы данных : учебное пособие / И. П. Карпова. - СПб : Питер, 2013. – 240 с.
8. Кириллов, В. В. Введение в реляционные базы данных. Введение в реляционные базы данных / В. В. Кириллов, Г. Ю. Громов. - СПб. : БХВ-Петербург, 2012. - 464 с.
9. Кузин, А. В. Базы данных : учебное пособие для студ. высш. учеб. заведений / А. В. Кузин, С. В. Левонисова. - М. : Академия, 2012. - 320 с.
10. Кузнецов С. Д. Базы данных : учебник для студентов учреждений высшего профессионального образования. Гриф «УМО по классическому университетскому образованию» / С. Д. Кузнецов. – М. : Издательский центр «Академия», 2012. – 496 с.
11. Осипов, Д. Л. Технологии проектирования баз данных / Д. Л. Осипов. — Москва : ДМК Пресс, 2019. — 498 с. — ISBN 978-5-97060-737-4. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/131692 (дата обращения: 27.02.2023). — Режим доступа: для авториз. пользователей.
12. Пирогов, В. Ю. Информационные системы и базы данных: организация и проектирование : учебное пособие / В. Ю. Пирогов. - СПб. : БХВ-Петербург, 2009. - 528 с.
13. Советов, Б. Я. Базы данных: теория и практика : учебник для бакалавров/ Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской. - М. : Юрайт, 2013. - 463 с.
14. Советов, Б. Я. Базы данных : учебник для вузов / Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской. — 3-е изд., перераб. и доп. — Москва : Юрайт, 2023. — 420 с. — (Высшее образование). — ISBN 978-5-534-07217-4. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/510752 (дата обращения: 27.02.2023).Фешина, Е. В. Базы данных : учебник / Е. В. Фешина, В. В. Ткаченко. — Краснодар : КубГАУ, 2020. — 172 с. — ISBN 978-5-907402-36-2. — Текст : электронный // Лань : электронно-библиотечная

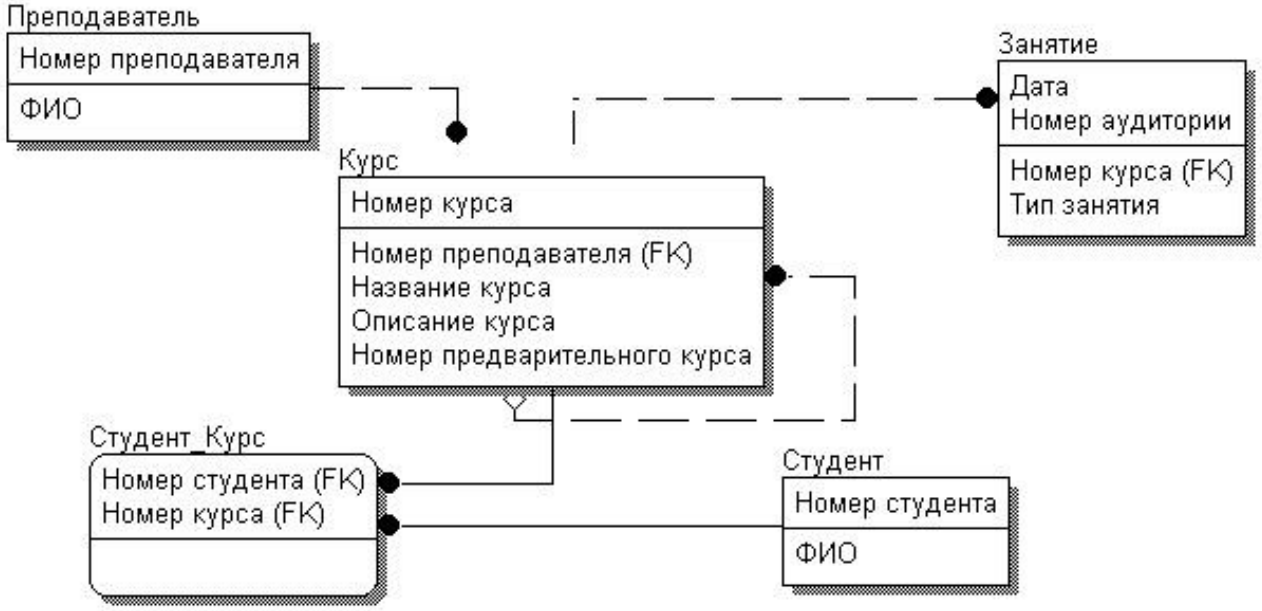
система. — URL: https://e.lanbook.com/book/254261 (дата обращения: 27.02.2023). — Режим доступа: для авториз. пользователей.
15. Стасышин, В. М. Базы данных: технологии доступа : учебное пособие для вузов / В. М. Стасышин, Т. Л. Стасышина. — 2-е изд., испр. и доп. — Москва : Юрайт, 2023. — 164 с. — (Высшее образование). — ISBN 978-5-534-08687-4. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/514252 (дата обращения: 27.02.2023).
16. Хомоненко А. Д. Базы данных : учебник для высших учебных заведений / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев. – СПб : КОРОНА принт, 2009.–672с.
17. Чен П. Модель "сущность-связь" - шаг к единому представлению данных / П. Чен // Системы управления базами данных. - 1995 . -№3. - С. 15-27.
18. Энсор Д. Oracle. Проектирование баз данных / Д. Энсор, И. Стивенсон. – К. : BHN, 1999. - 558 с.

ПРИЛОЖЕНИЕ 1
Концептуально-логические схемы баз данных, используемые в вариантах заданий

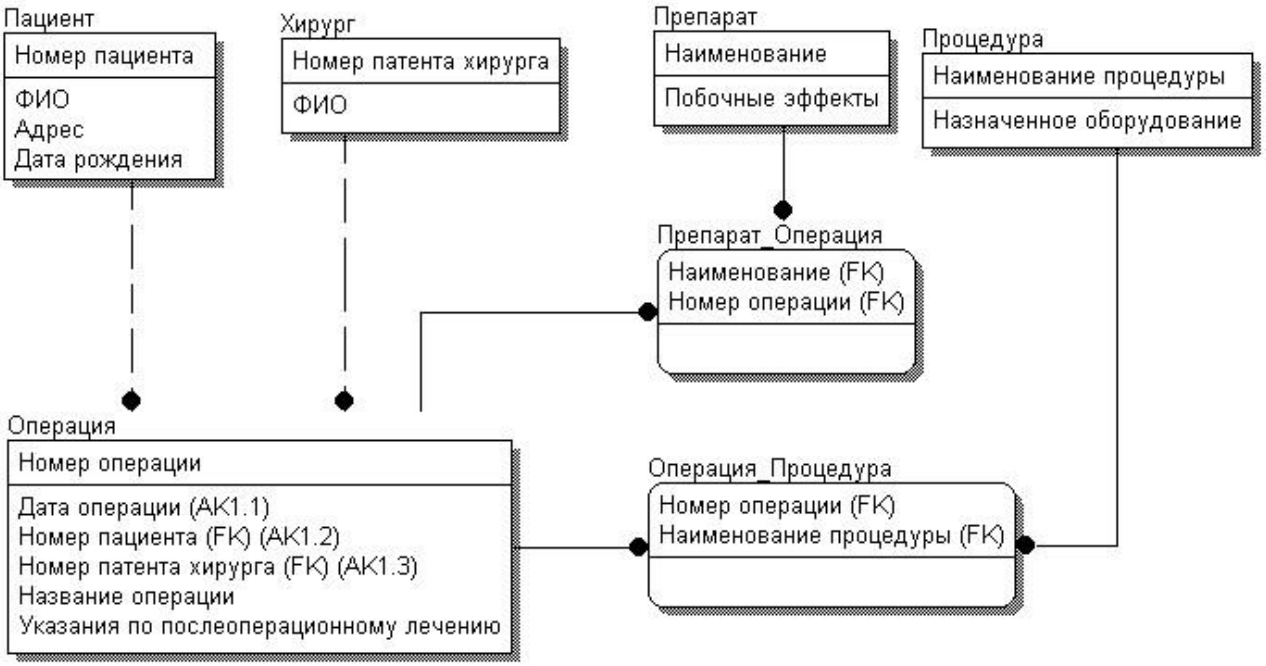
№ 1. Информация о публикациях по ряду выбранных тем



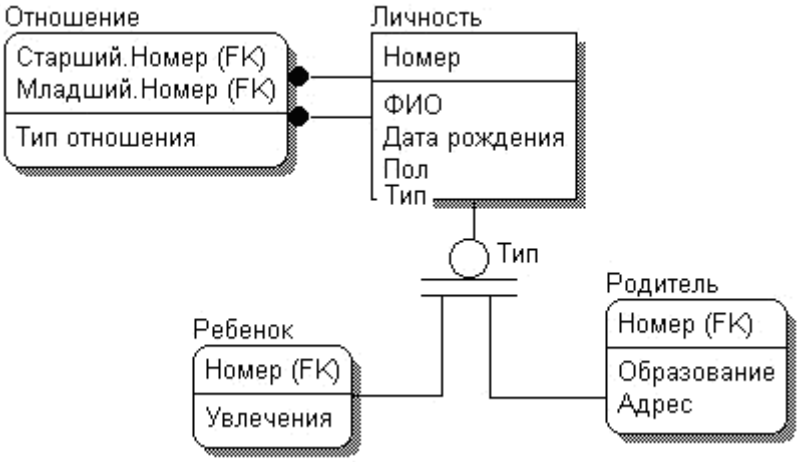
№ 2. Внутренняя система обучения в большой промышленной компании



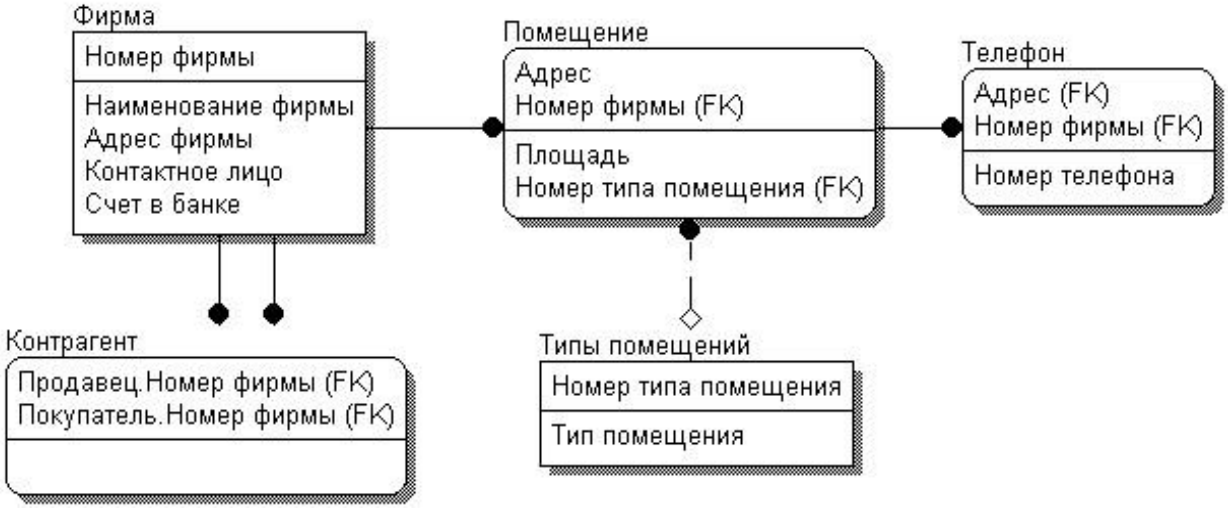
№ 3. Информация о пациентах клиники



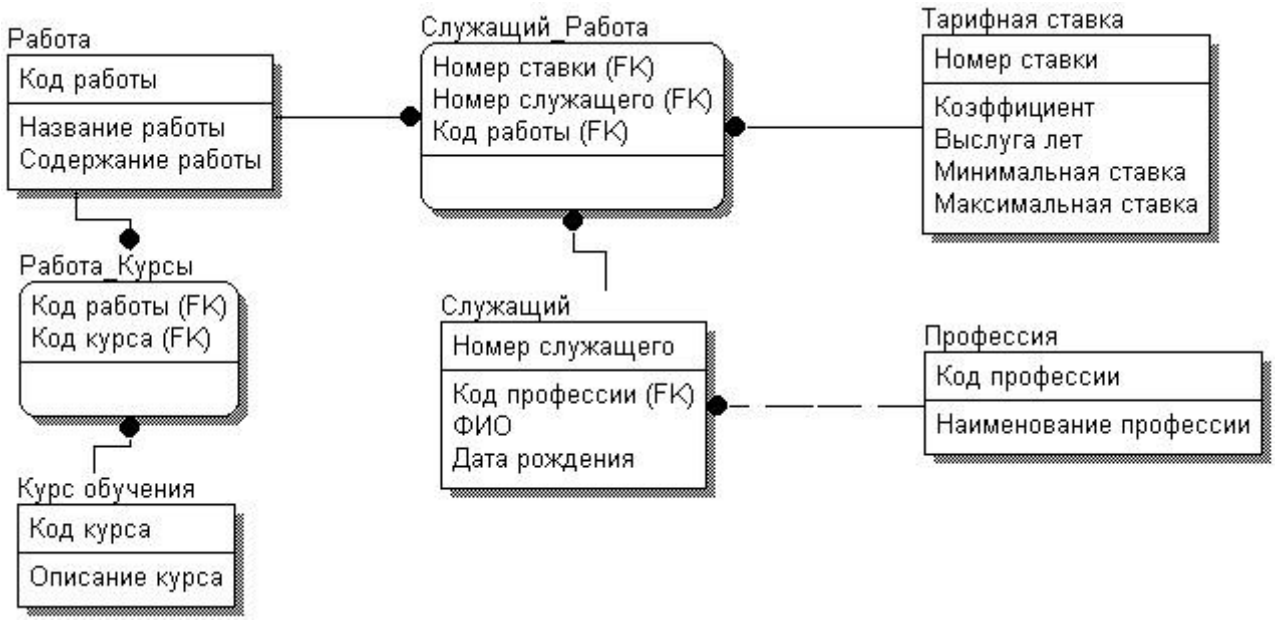
№ 4. Анкетные данные служащих конторы.



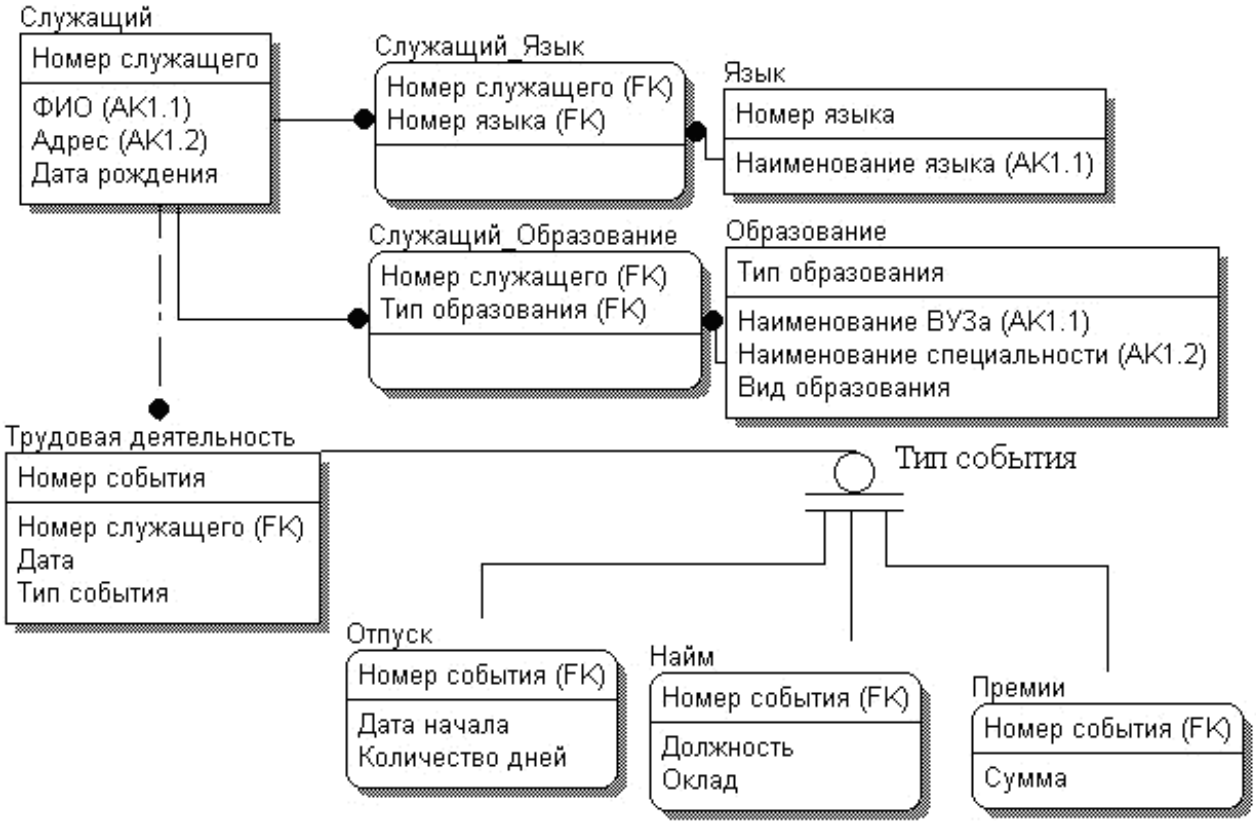
№ 5. Информация о торговых фирмах



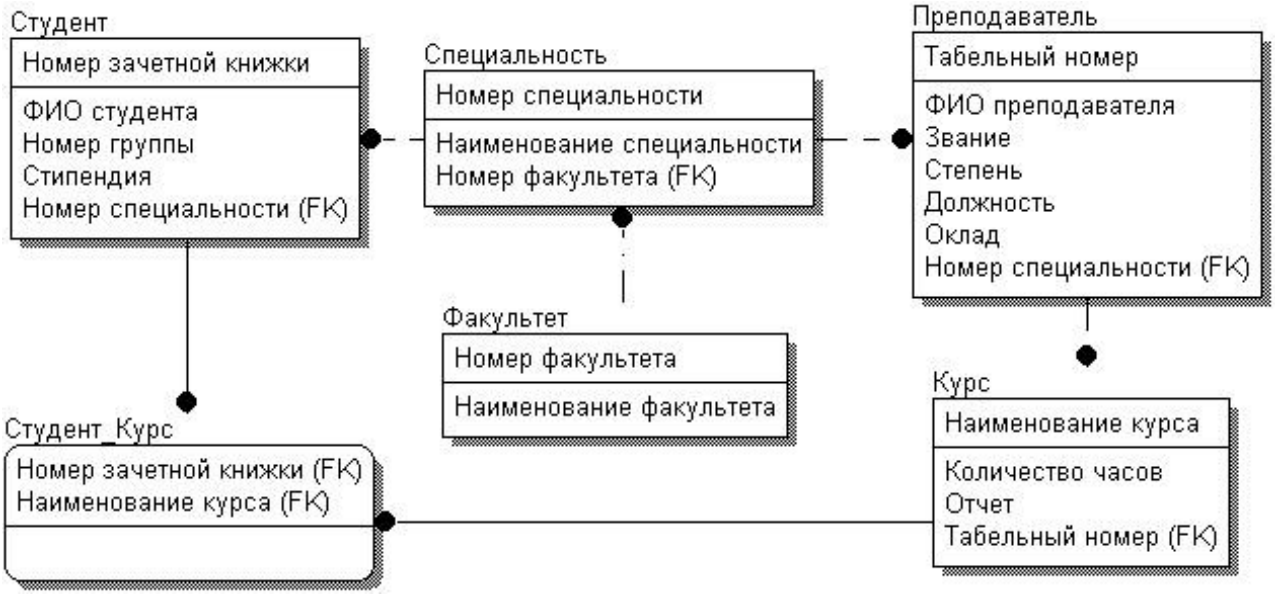
№ 6. Каталог работ



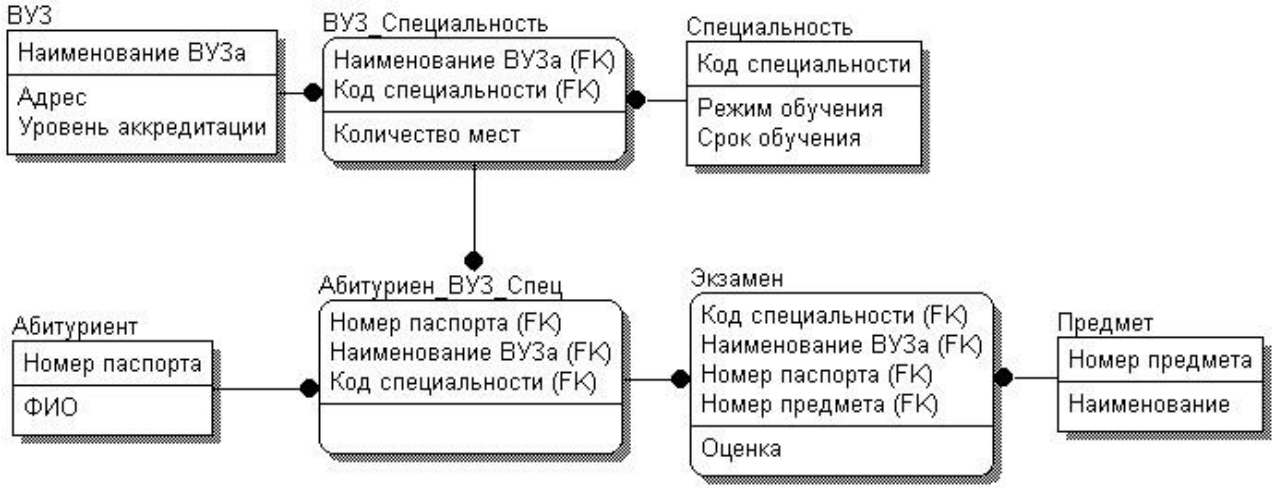
№ 7. Информация о служащих



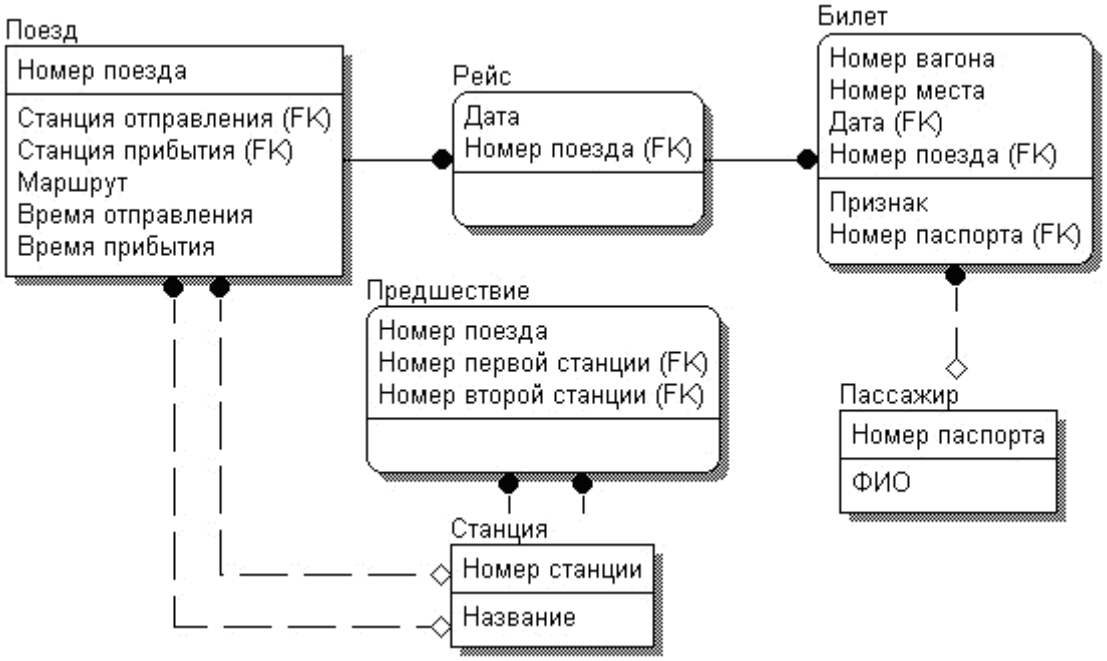
№ 8. Деканат



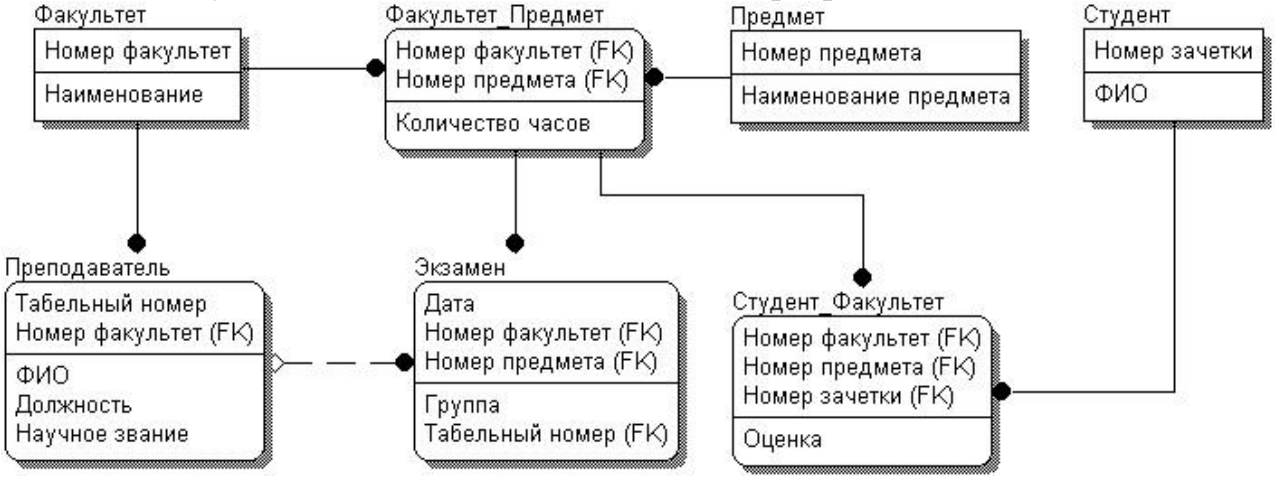
№ 9. Учебные заведения, абитуриенты и экзамены



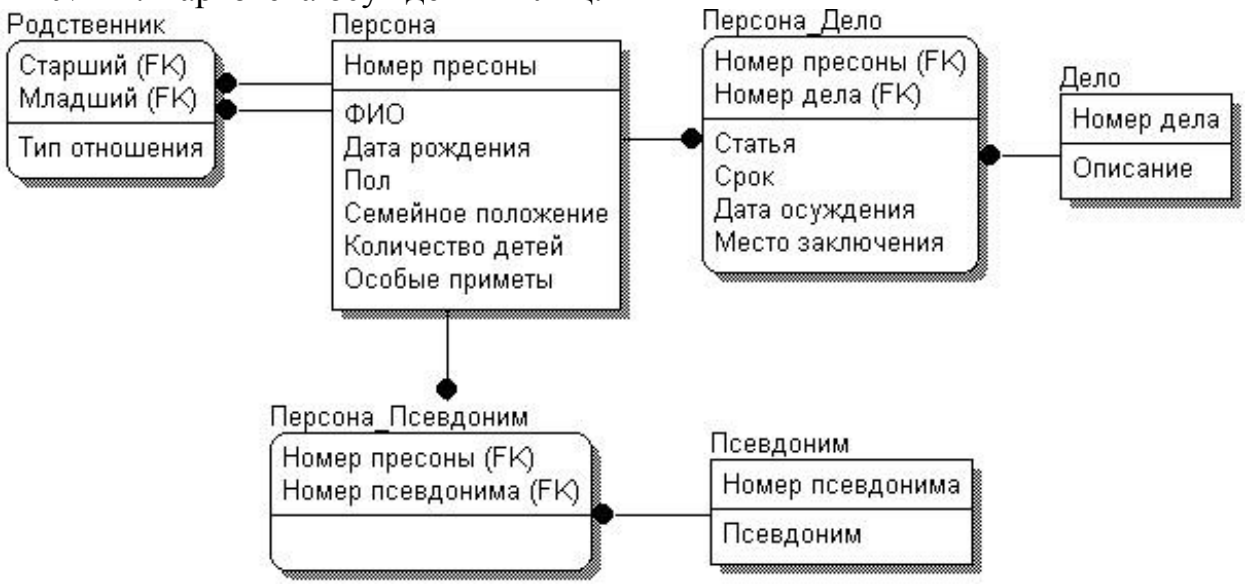
№ 10. Расписание поездов



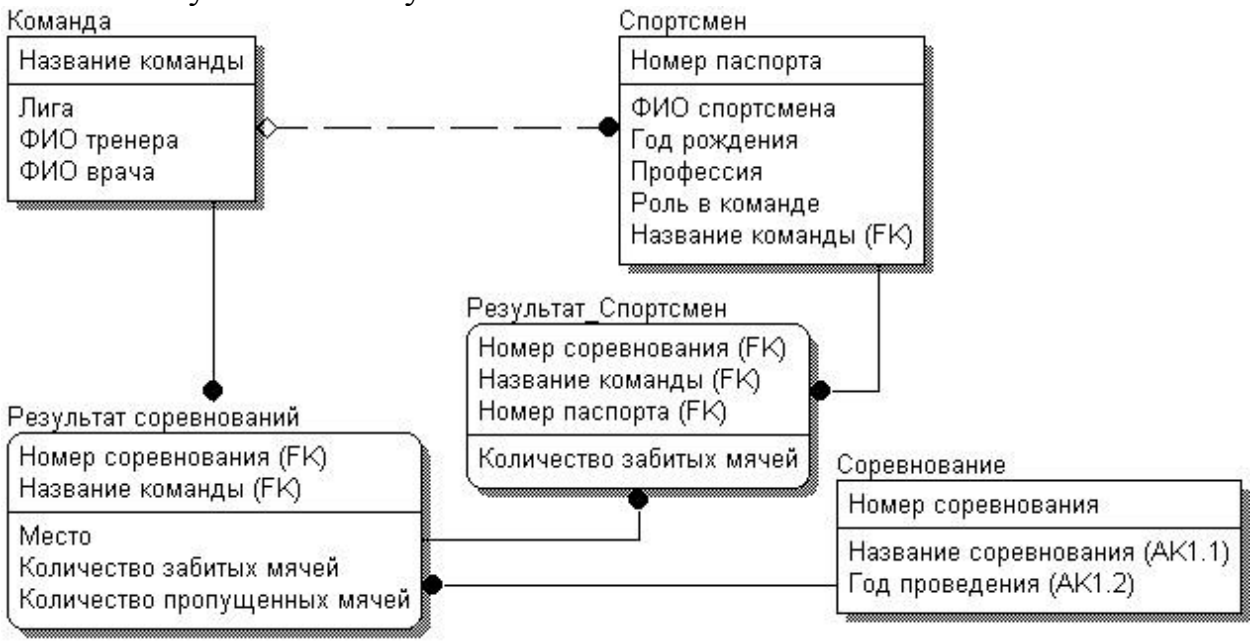
№ 11. Результататы экзаменов по каждой кафедре:



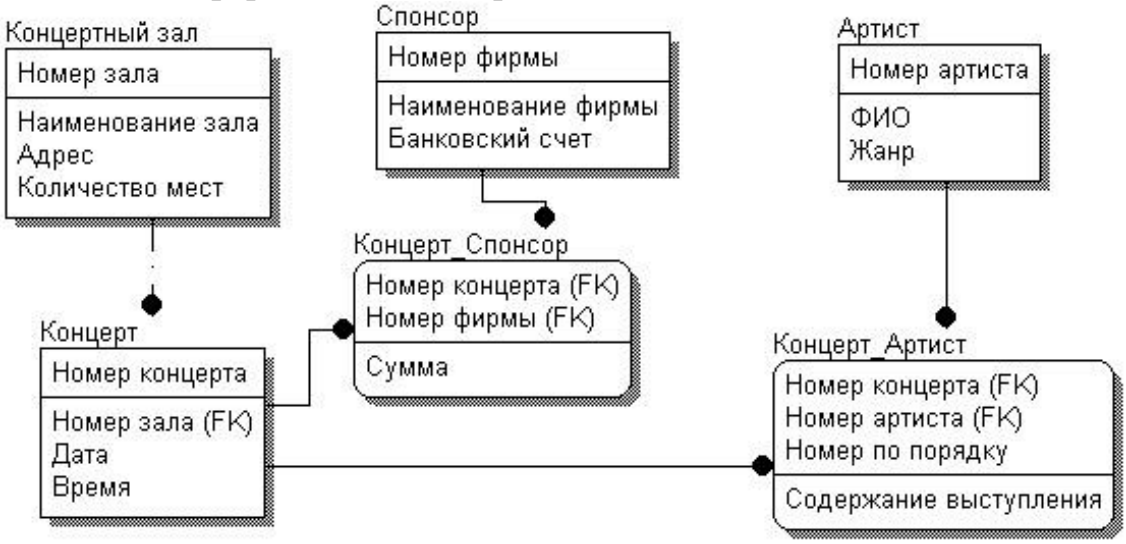
№ 12. Картотека осужденных лиц:



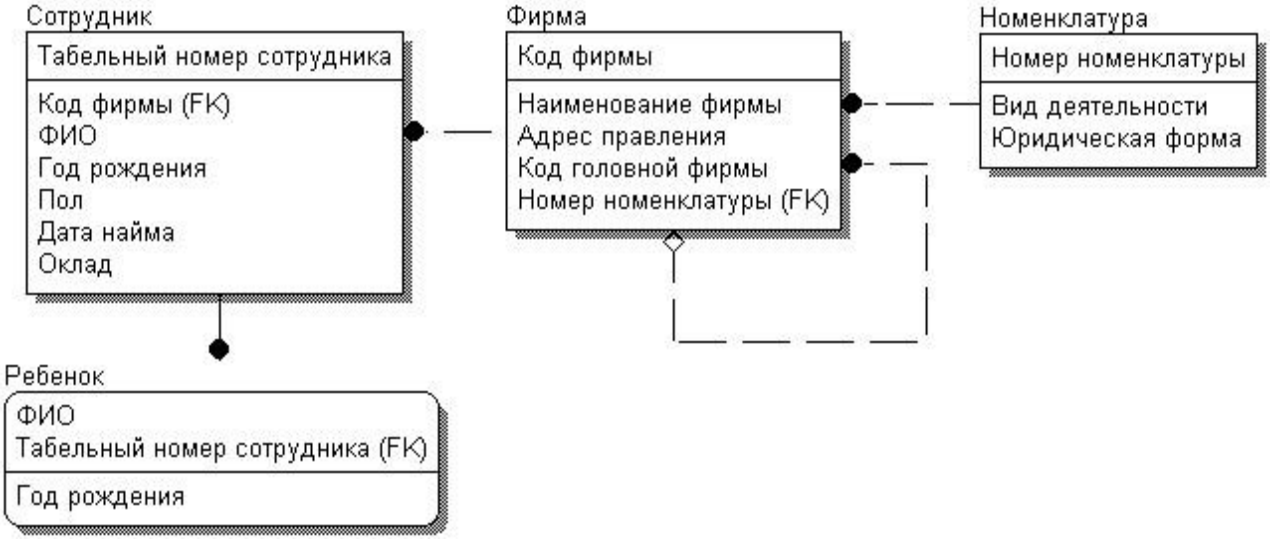
№ 13. Футбольные клубы



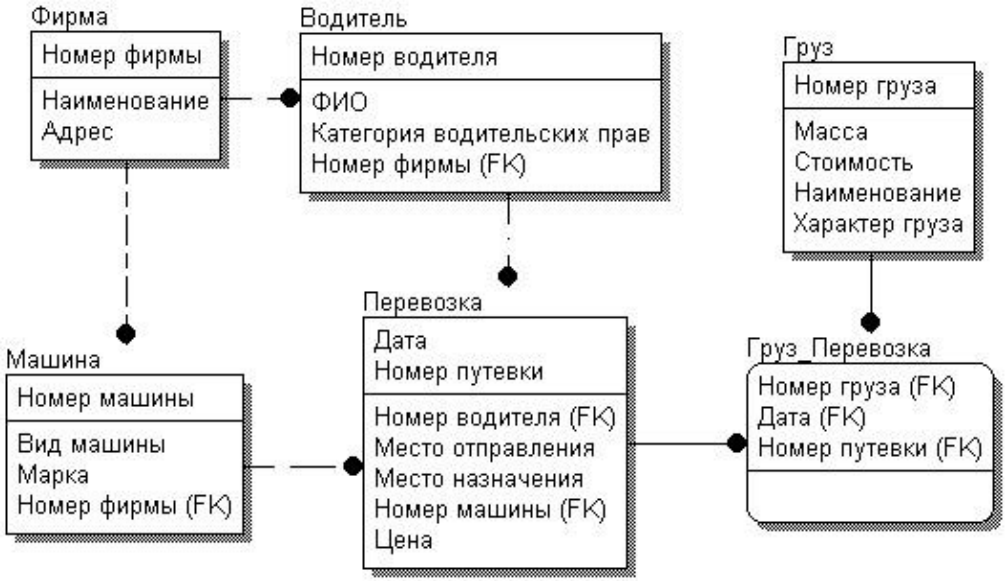
№ 14. Информация о концертах



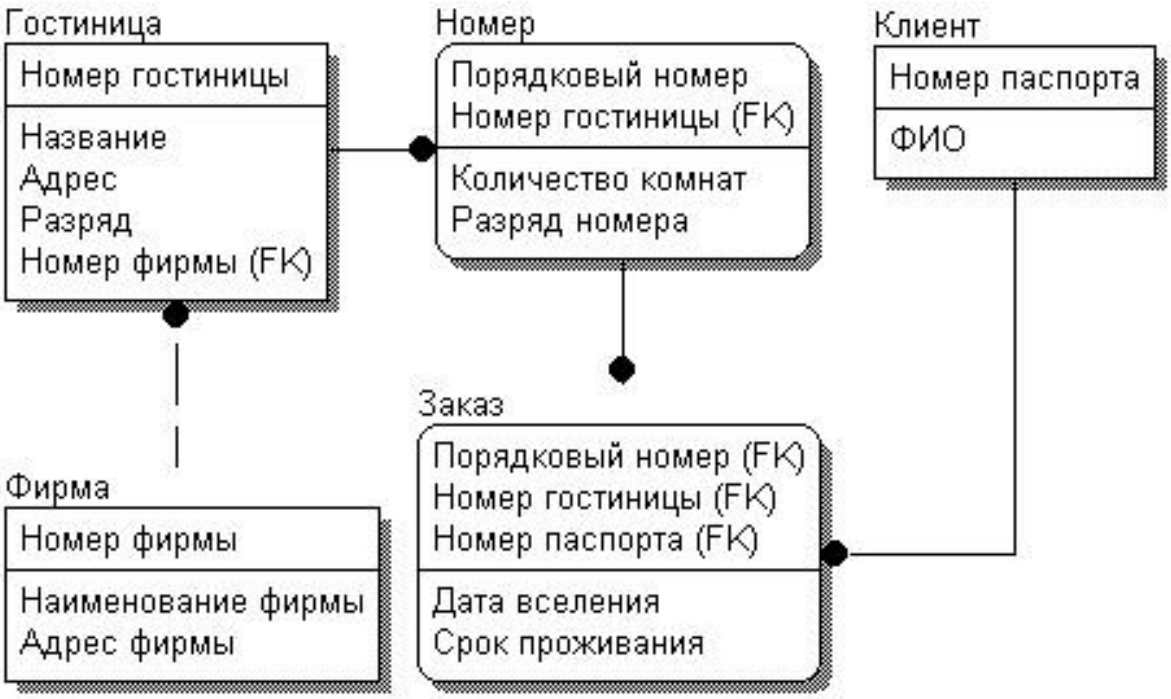
№ 15. Картотека фирм



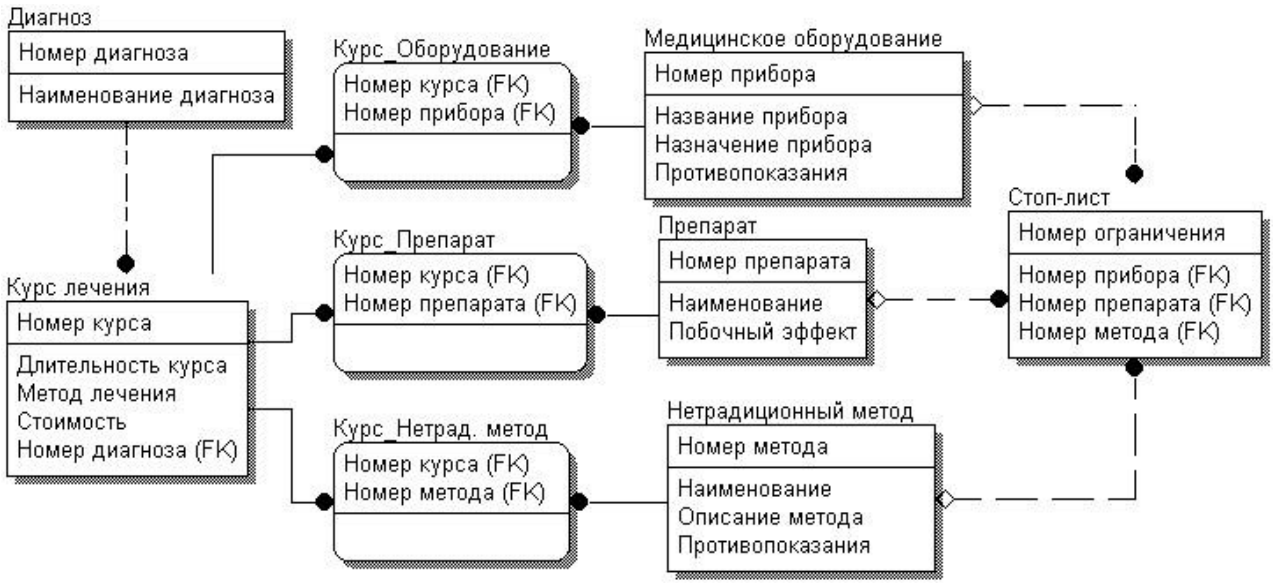
№ 16. Информация о грузовых перевозках, осуществляемых различными фирмами



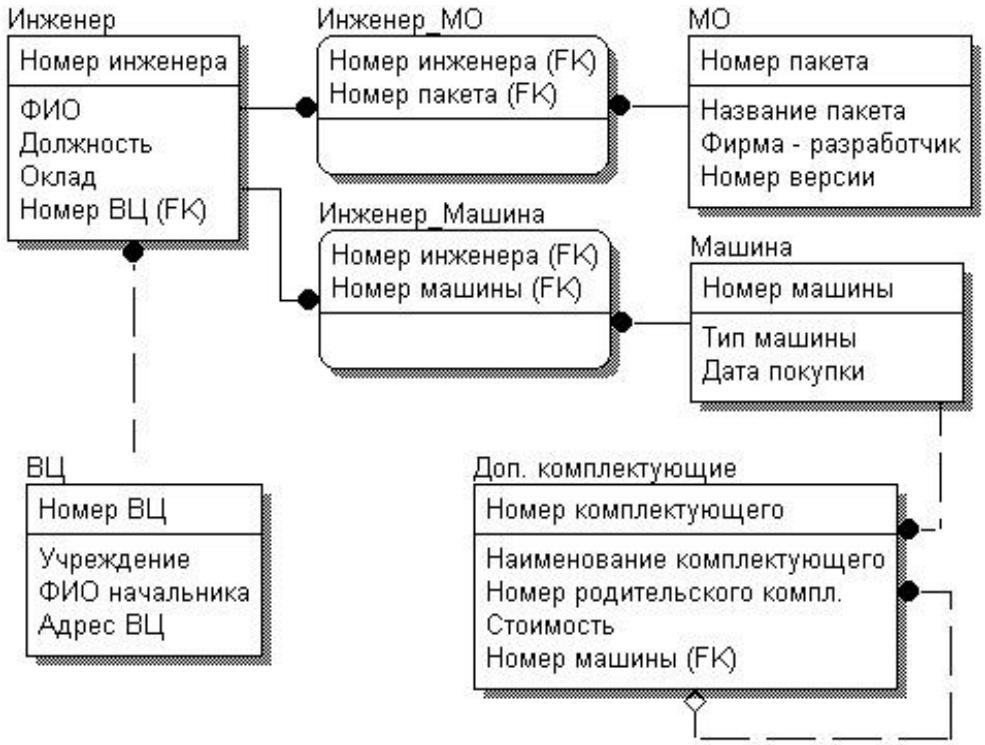
№ 17. Бронирование мест в гостинице для командированных



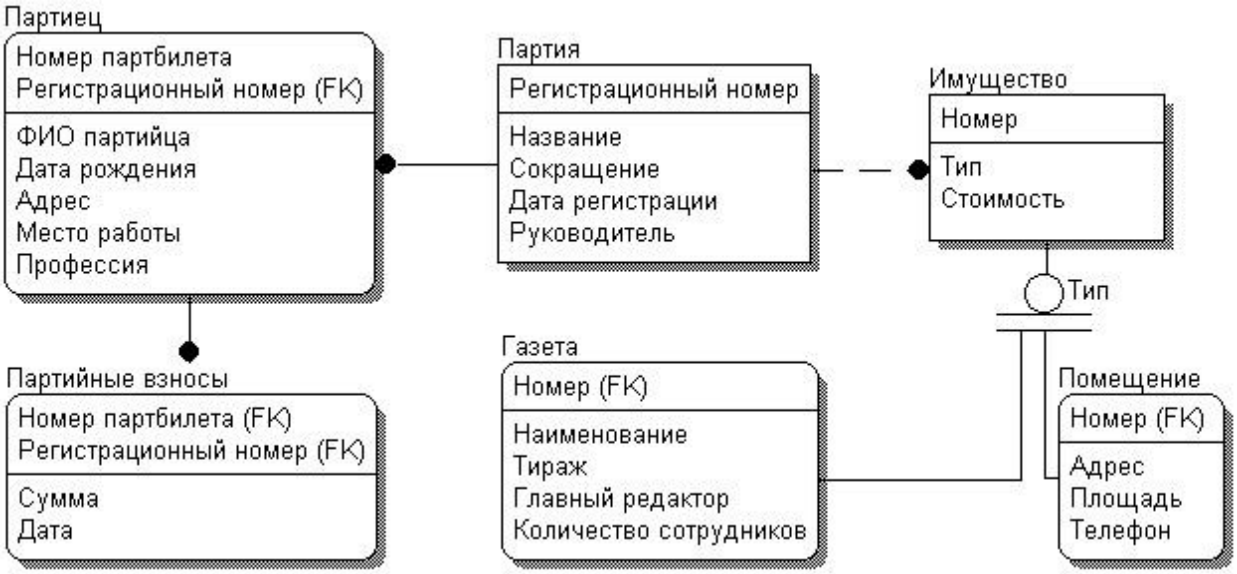
№ 18. Лечение различных заболеваний



№ 19. Вычислительные центры города



№ 20. Политические партии



ПРИЛОЖЕНИЕ 2. ВАРИАНТЫ ЗАДАНИЙ К Л/Р №3

Таблица 3.1 – Варианты заданий к лабораторной работе №3, 5, 8

№ вар-та	Имя таблицы	Условие	№ струк-туры
1.	Поезд	Станция прибытия=4	10
2.	Студент	Стипендия> 100	8
3.	Персона	Количество детей = 2	12
4.	Личность	Пол=’мужской’	4
5.	Гостиница	Разряд = 3	17
6.	Публикация	Тип=’статья’	1
7.	Сотрудник	Оклад > 600	15
8.	Трудовая деятельность	Тип события =’увольнение’	7
9.	Курс	N преподавателя=1	2
10.	Экзамен	N предмета=3	9
11.	Курс лечения	Стоимость < 300	18
12.	Операция	Название Операции = 'Аппендицит’	3
13.	Преподаватель	Должность = 'Ассистент’	11
14.	Результат соревнований	Количество забитых мячей = 2	13
15.	Помещение	Площадь < 20	5
16.	Комплектующие	Стоимость > 200	19
17.	Служащий	Код профессии = 3	6
18.	Перевозка	Место назначения = ‘Москва’	16
19.	Газета	Количество сотрудников > 20	20
20.	Концертный зал	Количество мест = 200	14
21.	Персона	Семейное положение = 'Женат’	12
22.	Курс лечения	Метод лечения = ‘Хирургия’	18
23.	Студент	Номер группы = 41	8
24.	Гостиница	Разряд > 1	17
25.	Преподаватель	Научное звание = 'Доцент’	11
26.	Личность	Пол=’женский’	4
27.	Результат соревнований	Количество пропущенных мячей > 1	13
28.	Трудовая деятельность	Дата = 21.01.2012	7
29.	Помещение	Площадь > 15	5
30.	Газета	Тираж < 10000	20

ПРИЛОЖЕНИЕ 3. ВАРИАНТЫ ЗАДАНИЙ К Л/Р №4

Таблица 4.1 – Варианты заданий к лабораторной работе №4

№ вар-та	№ струк-туры	Имя таблицы	Поле фильтра	Поле группировки	Агрегатная функция	Направление сортировки
1	**2**	**3**	**4**	**5**	**6**	**7**
1	10	Поезд	Маршрут	Станция отправления	Мин. время прибытия	По убыванию
2	8	Студент	Ф.И.О.	Номер группы	Макс. стипендия в каждой группе	По возрастанию
3	12	Персона	Ф.И.О.	Пол	Среднее кол-во детей	По убыванию
4	4	Личность	ФИО	Пол	Самый старший каждого пола	По возрастанию
5	17	Гостиница	Адрес	Разряд	Количество гостиниц	По убыванию
6	1	Публикация	Заглавие	Тип	Последняя дата для данного типа	По возрастанию
7	15	Сотрудник	Ф.И.О.	Пол	Количество сотрудников	По убыванию
8	7	Труд. деятельность	Номер служ.	Тип события	Кол-во событий каждого типа	По возрастанию
9	2	Курс	Название курса	Номер преподавателя	Количество курсов у преп.	По убыванию
10	9	Экзамен	Наименов. ВУЗа	Код специальности	Средняя оценка	По возрастанию
11	18	Курс лечения	Метод лечения	Номер диагноза	Средняя стоимость	По убыванию
12	3	Операция	Название операции	Номер патента	Количество операций	По возрастанию
13	11	Преподаватель	Ф.И.О.	Должность	Кол-во преподав. в должности	По убыванию
14	13	Результат соревнований	Название команды	Номер соревнования	Максим. число забитых мячей	По возрастанию
15	5	Помещение	Адрес	Номер фирмы	Максимальная площадь	По убыванию
16	19	Комплект	Наименован.	Номер машины	Макс. Стоимость	По возрастанию
17	6	Служащий	Ф.И.О.	Код профессии	Число служ. по профессиям	По убыванию
18	16	Перевозка	Место назначения	Дата	Средняя цена	По возрастанию
19	20	Газета	Наименован.	Тираж	Мин. Кол-во сотрудников	По убыванию
20	14	Концертный зал	Адрес	Количество мест	Число залов	По возрастанию
21	12	Персона	Ф.И.О.	Пол	Макс. кол-во детей	По убыванию
22	18	Курс лечения	Метод лечения	Номер диагноза	Мин. стоимость	По возрастанию
23	8	Студент	Ф.И.О.	Номер группы	Ср. стипендия в каждой группе	По убыванию

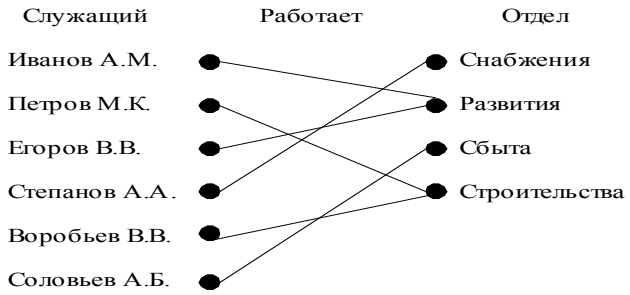
№ вар-та	№ струк-туры	Имя таблицы	Поле фильтра	Поле группировки	Агрегатная функция	Направление сортировки
1	**2**	**3**	**4**	**5**	**6**	**7**
24	17	Гостиница	Адрес	Разряд	Количество гостиниц	По возрастанию
25	11	Преподаватель	Ф.И.О.	Должность	Кол-во преподав. в должности	По убыванию
26	4	Личность	ФИО	Пол	Самый младший каждого пола	По возрастанию
27	13	Результат соревнований	Название команды	Номер соревнования	Ср. число забитых мячей	По убыванию
28	7	Труд. деятельность	Номер служ.	Тип события	Кол-во событий каждого типа	По возрастанию
29	5	Помещение	Адрес	Номер фирмы	Мин. площадь	По убыванию
30	20	Газета	Наименован.	Тираж	Макс. кол-во сотрудников	По возрастанию

ПРИЛОЖЕНИЕ 4. ВАРИАНТЫ ЗАДАНИЙ К Л/Р №6

Вариант 1

Для задания 1

ER-диаграмма экземпляров предметной области.



Свойства объектов и связей.

Служащий	Отдел	Работает
Код *(уникальное)*, ФИО, Номер отдела, Должность *(мастер, инженер, слесарь)* Зарплата *(вещественное число, два знака после запятой, >0)*	Номер *(уникальное)*, Название *(уникальное)*, Статус *(целое число >=0, <=1000)* Бюджет *(вещественное число, два знака после запятой, >0)*	-

Действия для связей между таблицами: удаление – запрет, обновление – запрет.

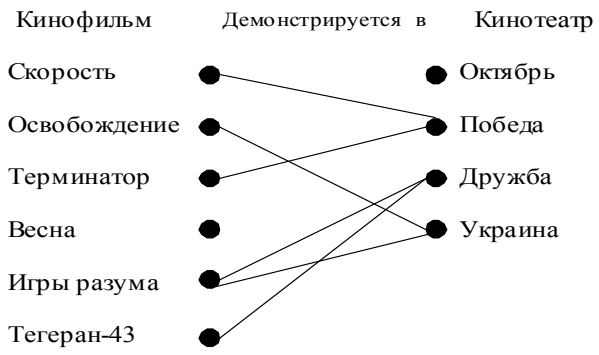
Для задания 2

Отношение: Служащий**, дополнительная функциональная зависимость:**
должность → зарплата

Вариант 2

Для задания 1

ER-диаграмма экземпляров предметной области.



Свойства объектов и связей.

Кинофильм	Кинотеатр	Демонстрируется в
Код *(уникальное)*, Название, Режиссер, Страна, Тип *(широкоформатный, широкоэкранный, обычный)* Рейтинг *(целое число >0)*	Код *(уникальное)*, Название *(уникальное),* Рейтинг *(целое число >0)* Телефон Число залов (*(целое число >0)*	Количество сеансов Цена билета *(вещественное число, два знака после запятой, >0)*

Действия для связей между таблицами: удаление – запрет, обновление – каскадирование.

Для задания 2

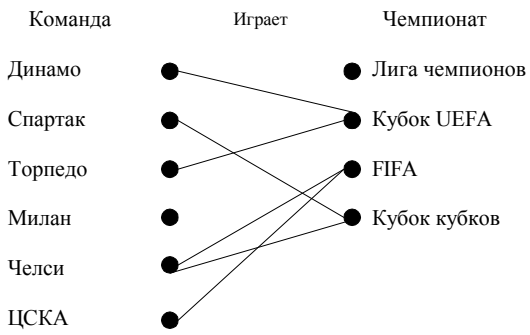
Отношение: Кинотеатр**, дополнительная функциональная зависимость:**

Число залов → Рейтинг

Вариант 3

Для задания 1

ER-диаграмма экземпляров предметной области.



Свойства объектов и связей.

Команда	Чемпионат	Играет
Код *(уникальное)*, Название, Тренер, Страна, Рейтинг *(целое число >0, <=100)*	Код *(уникальное)*, Название *(уникальное)*, Рейтинг *(целое число >0)* Группа *(местный, республиканский, международный)*	Количество голов *(целое >=0)*

Действия для связей между таблицами: удаление – каскадирование, обновление – запрет.

Для задания 2

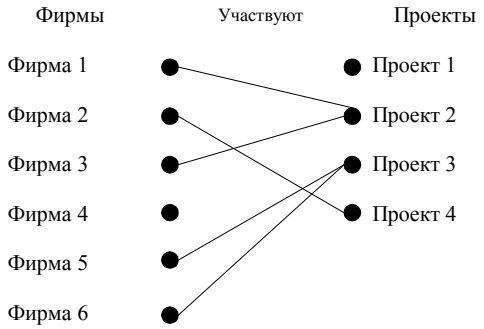
Отношение: Чемпионат**, дополнительная функциональная зависимость:**

группа → рейтинг

Вариант 4

Для задания 1

ER-диаграмма экземпляров предметной области.



Свойства объектов и связей.

Фирмы	Проекты	Участвуют
Код *(уникальное)*, Название, Директор, Город, Статус *(целое число >0)*	Код *(уникальное)*, Название *(уникальное),* Статус *(целое число >=0, <=100)* Группа *(местный, республиканский, международный)* Город Бюджет (вещественное число, *два знака после запятой,>0*)	-

Действия для связей между таблицами: удаление – каскадирование, обновление – каскадирование.

Для задания 2

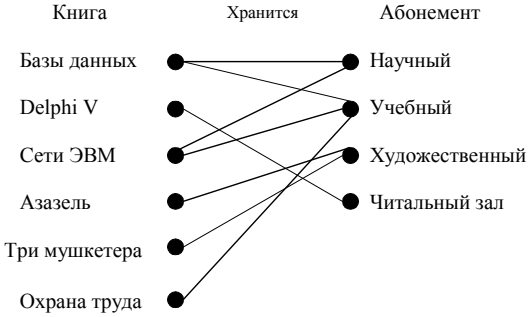
Отношение: Проекты**, дополнительная функциональная зависимость:**

группа → бюджет

Вариант 5

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Книга	Абонемент	Хранится
Код *(уникальное)*, Название, Автор, Издательство, Рейтинг *(0, 10, 20, 30,40,50)* Цена *(вещественное число, два знака после запятой, >0)*	Код *(уникальное)*, Название *(уникальное)*, Номер комнаты (*(уникальное)*, Телефон *(уникальное)*	Количество экземпляров *(целое число >0)*

Действия для связей между таблицами: удаление – запрет, обновление – запрет.

Для задания 2

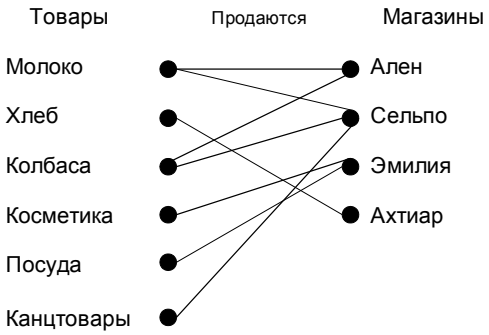
Отношение: Книга**, дополнительная функциональная зависимость:**

рейтинг → цена

Вариант 6

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Товары	Магазины	Продаются
Код *(уникальное)*, Название, Производитель, Город, Тип *(продуктовые, бытовые)*	Код *(уникальное)*, Название *(уникальное)*, Адрес, Телефон *(уникальное),* Рейтинг *(целое число>0, <=1000)*	Цена *(вещественное число, два знака после запятой, >0)* Рейтинг *(10, 20, 30, 40, 50)*

Действия для связей между таблицами: удаление – запрет, обновление – каскадирование.

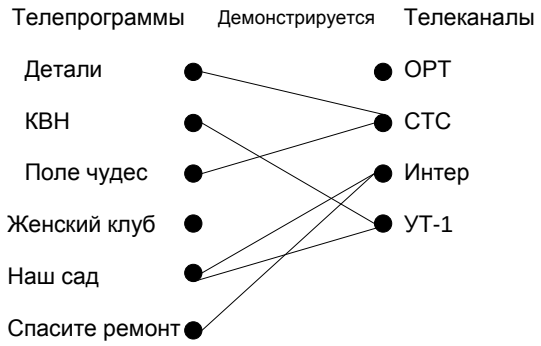
Для задания 2

Отношение: Продаются**, дополнительная функциональная зависимость:**
рейтинг → цена

Вариант 7

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Телепрограмма	Канал	Демонстрируется
Код *(уникальное)*, Название, Телеведущий, Тип *(образовательная, развлекательная),* Рейтинг *(целое число, >0)*	Код *(уникальное)*, Название *(уникальное)*, Рейтинг *(10, 50, 100),* Бюджет *(вещественное число, два знака после запятой, >0),* Вид *(частный, государственный, АО)*	Число показов *(целое число, >0)*

Действия для связей между таблицами: удаление – каскадирование, обновление – запрет.

Для задания 2

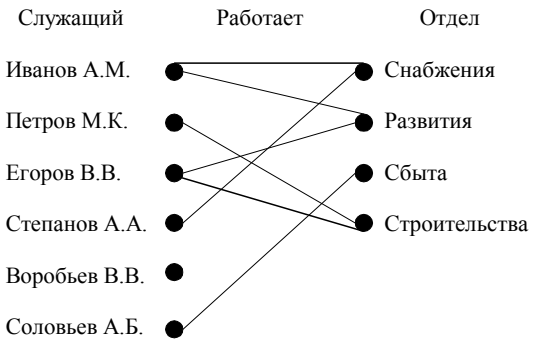
Отношение: Канал**, дополнительная функциональная зависимость:**

рейтинг → бюджет

Вариант 8

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Служащий	Отдел	Работает
Код *(уникальное)*, ФИО, Трудовой стаж *(целое число>=5, <=50)*, Вид трудового договора *(контракт, коллективный договор, временный договор)*	Номер *(уникальное)*, Название *(уникальное)*, Статус *(целое число>0, <100),* Бюджет *(вещественное число, два знака после запятой, >0)*	Должность Зарплата *(вещественное число, два знака после запятой, >0)*

Действия для связей между таблицами: удаление – каскадирование, обновление – каскадирование.

Для задания 2

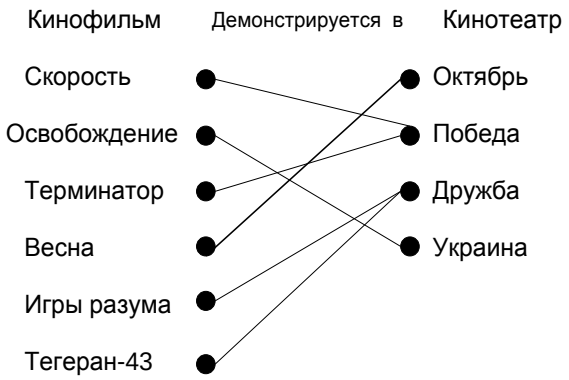
Отношение: Работает**, дополнительная функциональная зависимость:**

должность → зарплата

Вариант 9

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Кинофильм	Кинотеатр	Демонстрируется в
Код *(уникальное)*, Название, Режиссер, Страна, Тип *(широкоформатный, широкоэкранный, обычный)* Рейтинг *(целое число>0, <1000)*	Код *(уникальное)*, Название *(уникальное)*, Рейтинг *(10, 20, 30, 40, 50)*, Телефон *(уникальное)*, Цена билета *(вещественное число, два знака после запятой, >0)*	-

Действия для связей между таблицами: удаление – запрет, обновление – запрет.

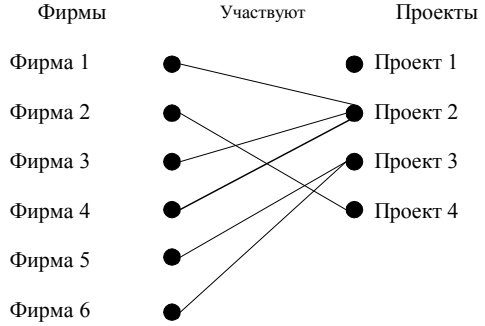
Для задания 2

Отношение: Кинотеатр**, дополнительная функциональная зависимость:**
рейтинг → цена билета

Вариант 10

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Фирмы	Проекты	Участвуют
Код *(уникальное)*, Название, Директор, Город, Статус *(целое число>=0, <=10)*	Код *(уникальное)*, Название *(уникальное)*, Статус *(100,500,1000)* Группа *(местный, республиканский, международный),* Город Бюджет *(вещественное число, два знака после запятой, >0)*	-

Действия для связей между таблицами: удаление – запрет, обновление – каскадирование.

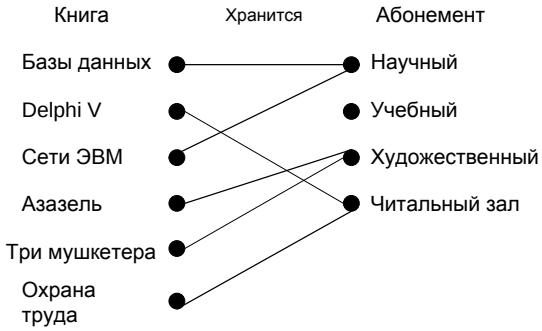
Для задания 2

Отношение: Проекты**, дополнительная функциональная зависимость:**
статус → бюджет

Вариант 11

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Книга	Абонемент	Хранится
Код *(уникальное)*, Название, Автор, Издательство, Рейтинг *(0, 10, 30, 70, 100)*, Цена *(вещественное число, два знака после запятой, >0)* Количество экземпляров *(целое число>0)*	Код *(уникальное)*, Название *(уникальное)*, Номер комнаты *(уникальное)*, Телефон *(уникальное)*	-

Действия для связей между таблицами: удаление – каскадирование, обновление – запрет.

Для задания 2

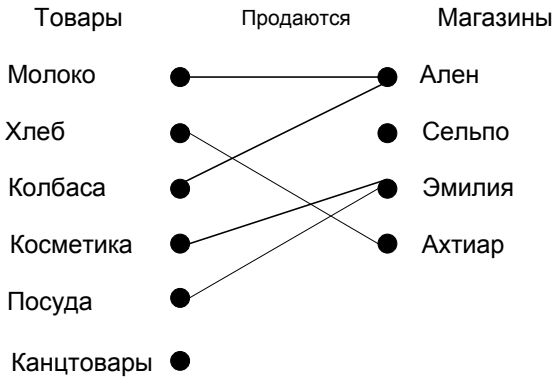
Отношение: Книга**, дополнительная функциональная зависимость:**

рейтинг → цена

Вариант 12

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Товары	Магазины	Продаются
Код *(уникальное)*, Название, Производитель, Город, Тип *(продукты, бытовая химия, канцелярские)* Цена *(вещественное число, два знака после запятой, >0)*	Код *(уникальное)*, Название *(уникальное)*, Адрес, Телефон *(уникальное),* Рейтинг *(целое число>0)* Тип *(малый, экомаркет, супермаркет)*	-

Действия для связей между таблицами: удаление – каскадирование, обновление – каскадирование.

Для задания 2

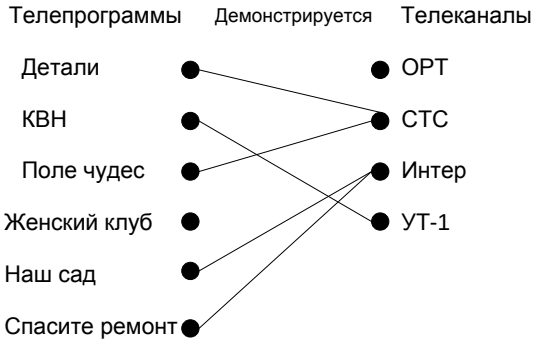
Отношение: Магазины**, дополнительная функциональная зависимость:**

тип → рейтинг

Вариант 13

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Телепрограмма	Канал	Демонстрируется
Код *(уникальное)*, Название, Телеведущий, Тип *(образовательная, развлекательная, новостная)*, Рейтинг *(целое число>=0, <=10)*, Число показов *(целое число>0)*	Код *(уникальное)*, Название *(уникальное)*, Рейтинг *(целое число>=0, <=10)*, Бюджет *(вещественное число, два знака после запятой, >0)*, Вид *(частный, государственный, АО)*	-

Действия для связей между таблицами: удаление – запрет, обновление – запрет.

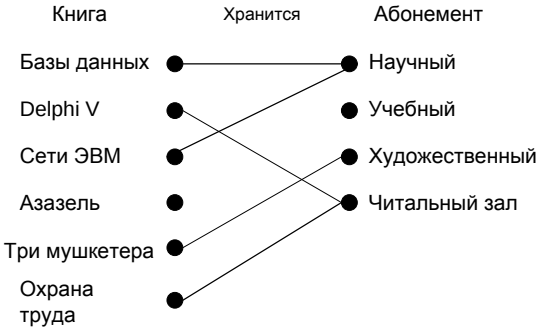
Для задания 2

Отношение: Телепрограмма**, дополнительная функциональная зависимость:** тип → рейтинг

Вариант 14

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Книга	Абонемент	Хранится
Код *(уникальное)*, Название, Автор, Рейтинг *(целое число>=0, <=1000)*, Год издания *(целое число > 0)*, Цена *(вещественное число, два знака после запятой, >0)*, Количество экземпляров *(целое число>0)*	Код *(уникальное)*, Название *(уникальное)*, Номер комнаты *(уникальное)*, Телефон *(уникальное)*,	-

Действия для связей между таблицами: удаление – запрет, обновление – каскадирование.

Для задания 2

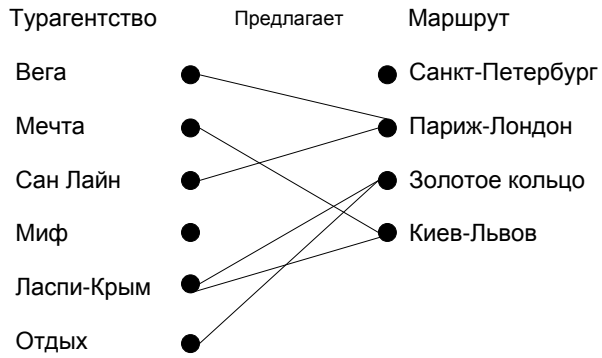
Отношение: Книга**, дополнительная функциональная зависимость:**

автор → рейтинг

Вариант 15

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Турагентство	Маршрут	Предлагает
Код *(уникальное)*, Название *(уникальное)*, Рейтинг *(целое число>=0, <=100)*, Телефон *(уникальное)*, Год создания *(целое число > 0)*	Код *(уникальное)*, Название *(уникальное)*, Рейтинг *(целое число>=0, <=100)*, Вид *(авиа, ж_д, авто)*, Класс *(средний, высший)*	Стоимость *(вещественное число, два знака после запятой, >0)*

Действия для связей между таблицами: удаление – каскадирование, обновление – запрет.

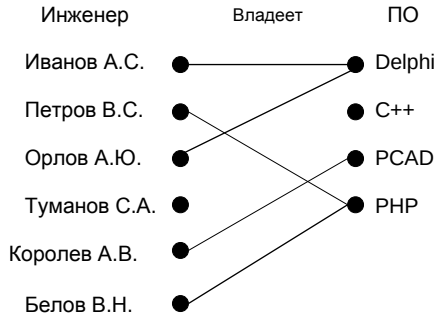
Для задания 2

Отношение: Маршрут**, дополнительная функциональная зависимость:**
класс → рейтинг

Вариант 16

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Инженер	ПО	Владеет
Код *(уникальное)*, Имя, Стаж работы *(целое число>=1, <=30)*, Зарплата *(вещественное число, два знака после запятой, >0)*, Профессиональный уровень *(начинающий, опытный)*	Код *(уникальное)*, Название *(уникальное)*, группа *(инструментальные системы, САПР)*, уровень_сложности *(целое число>0, <=10)*, стоимость *(вещественное число, два знака после запятой, >0)*	-

Действия для связей между таблицами: удаление – каскадирование, обновление – каскадирование.

Для задания 2

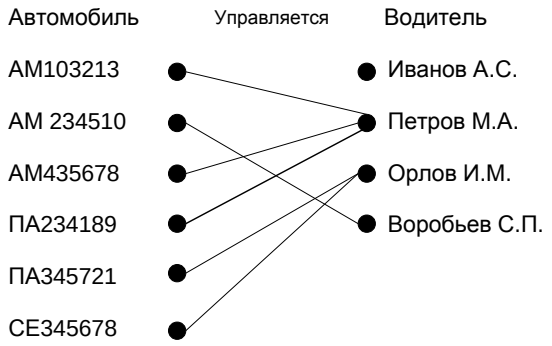
Отношение: ПО**, дополнительная функциональная зависимость:**

группа → стоимость

Вариант 17

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Автомобиль	Водитель	Управляется
Код *(уникальное)*, Марка_авто, Количество_мест *(целое число>0, <=10)*, Год выпуска *(целое число>0)*, Тип *(грузовой, пассажирский)*	Код *(уникальное)*, Имя, Категория *(B,C,D,E)* Стаж работы *(целое число>=5, <=50)*, Зарплата *(вещественное число, два знака после запятой, >0)*	_

Действия для связей между таблицами: удаление – запрет, обновление – запрет.

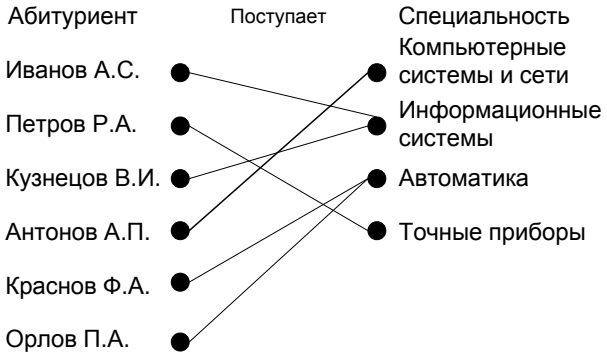
Для задания 2

Отношение: Водитель**, дополнительная функциональная зависимость:**
категория → зарплата

Вариант 18

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Абитуриент	Специальность	Поступает
Номер_паспорта *(уникальное)*, Имя, Год рождения *(целое число>0)*, Город Средний балл *(целое число>0, <=150)*	Код *(уникальное)*, Название *(уникальное)*, Факультет, Проходной балл *(целое число>0, <=150)* Рейтинг *(целое число>0, <=10)*	-

Действия для связей между таблицами: удаление – запрет, обновление – каскадирование.

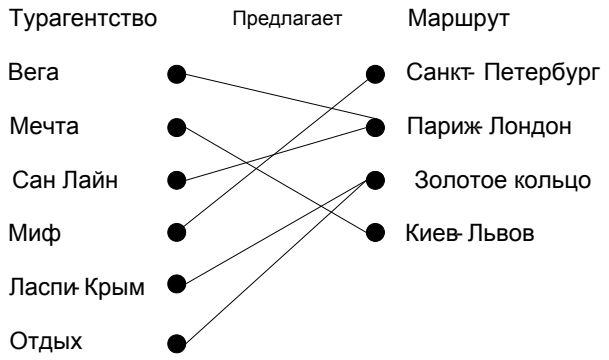
Для задания 2

Отношение: Специальность**, дополнительная функциональная зависимость:**
факультет → рейтинг

Вариант 19

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Турагентство	Маршрут	Предлагает
Код *(уникальное)*, Название *(уникальное)*, Рейтинг *(целое число>=0, <=100)*, Телефон *(уникальное)*, Год создания *(целое число > 0)*	Код *(уникальное)*, Название *(уникальное)*, Рейтинг *(целое число>=0, <=100)*, Вид *(авиа, ж_д, авто)*, Класс *(эконом, средний, высший)* Стоимость *(вещественное число, два знака после запятой, >0)*	–

Действия для связей между таблицами: удаление – каскадирование, обновление – запрет.

Для задания 2

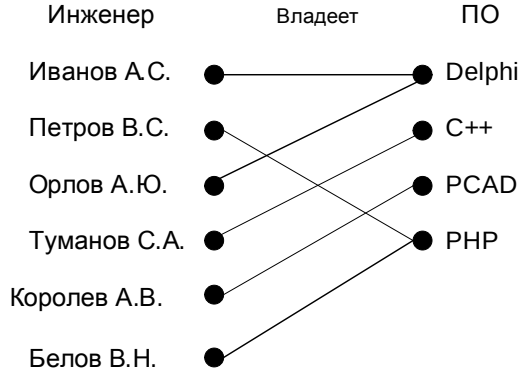
Отношение: Маршрут**, дополнительная функциональная зависимость:**

класс→ стоимость

Вариант 20

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Инженер	ПО	Владеет
Код *(уникальное)*, Имя, Стаж работы *(целое число>=1, <=30)*, Зарплата *(вещественное число, два знака после запятой, >0)*, Профессиональный уровень *(начинающий, опытный)*	Код *(уникальное)*, Название *(уникальное)*, группа *(инструментальные системы, САПР)*, уровень_сложности *(целое число>0, <=10)*, стоимость *(вещественное число, два знака после запятой, >0)*	-

Действия для связей между таблицами: удаление – каскадирование, обновление – каскадирование.

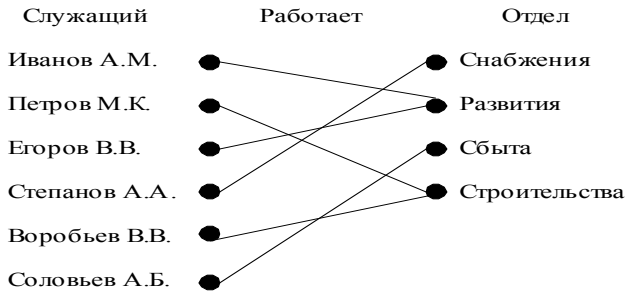
Для задания 2

Отношение: Инженер**, дополнительная функциональная зависимость:** профессиональный уровень → зарплата

Вариант 21

Для задания 1

ER-диаграмма экземпляров предметной области.



Свойства объектов и связей.

Служащий	Отдел	Работает
Код *(уникальное)*, ФИО, Номер отдела, Должность *(мастер, инженер, слесарь)* Зарплата *(вещественное число, два знака после запятой, >0)*	Номер *(уникальное)*, Название *(уникальное)*, Статус *(целое число >=0, <=1000)* Бюджет *(вещественное число, два знака после запятой, >0)*	-

Действия для связей между таблицами: удаление – запрет, обновление – запрет.

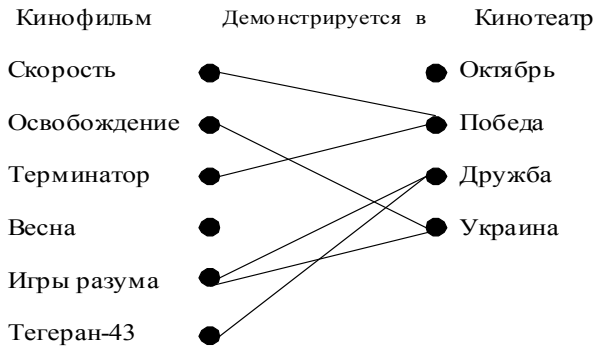
Для задания 2

Отношение: Служащий**, дополнительная функциональная зависимость:**
должность → зарплата

Вариант 22

Для задания 1

ER-диаграмма экземпляров предметной области.



Свойства объектов и связей.

Кинофильм	Кинотеатр	Демонстрируется в
Код *(уникальное)*, Название, Режиссер, Страна, Тип *(широкоформатный, широкоэкранный, обычный)* Рейтинг *(целое число >0)*	Код *(уникальное)*, Название *(уникальное),* Рейтинг *(целое число >0)* Телефон Число залов (*(целое число >0)*	Количество сеансов Цена билета *(вещественное число, два знака после запятой, >0)*

Действия для связей между таблицами: удаление – запрет, обновление – каскадирование.

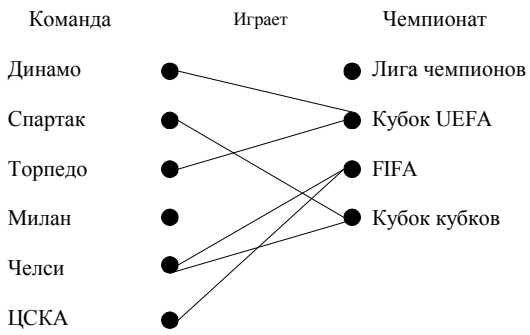
Для задания 2

Отношение: Кинотеатр**, дополнительная функциональная зависимость:**
Число залов → Рейтинг

Вариант 23

Для задания 1

ER-диаграмма экземпляров предметной области.



Свойства объектов и связей.

Команда	Чемпионат	Играет
Код *(уникальное)*, Название, Тренер, Страна, Рейтинг *(целое число >0, <=100)*	Код *(уникальное)*, Название *(уникальное)*, Рейтинг *(целое число >0)* Группа *(местный, республиканский, международный)*	Количество голов *(целое >=0)*

Действия для связей между таблицами: удаление – каскадирование, обновление – запрет.

Для задания 2

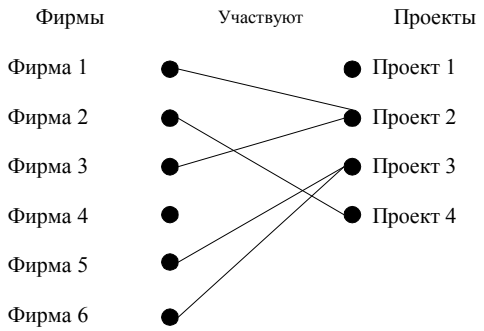
Отношение: Чемпионат**, дополнительная функциональная зависимость:**

группа → рейтинг

Вариант 24

Для задания 1

ER-диаграмма экземпляров предметной области.



Свойства объектов и связей.

Фирмы	Проекты	Участвуют
Код *(уникальное)*, Название, Директор, Город, Статус *(целое число >0)*	Код *(уникальное)*, Название *(уникальное),* Статус *(целое число >=0, <=100)* Группа *(местный, республиканский, международный)* Город Бюджет (вещественное число, *два знака после запятой,>0*)	-

Действия для связей между таблицами: удаление – каскадирование, обновление – каскадирование.

Для задания 2

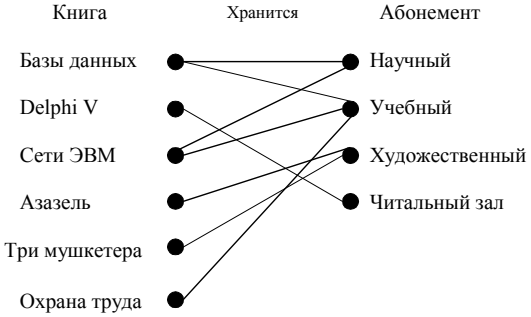
Отношение: Проекты**, дополнительная функциональная зависимость:**

группа → бюджет

Вариант 25

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Книга	Абонемент	Хранится
Код *(уникальное)*, Название, Автор, Издательство, Рейтинг *(0, 10, 20, 30,40,50)* Цена *(вещественное число, два знака после запятой, >0)*	Код *(уникальное)*, Название *(уникальное)*, Номер комнаты (*(уникальное)*, Телефон *(уникальное)*	Количество экземпляров *(целое число >0)*

Действия для связей между таблицами: удаление – запрет, обновление – запрет.

Для задания 2

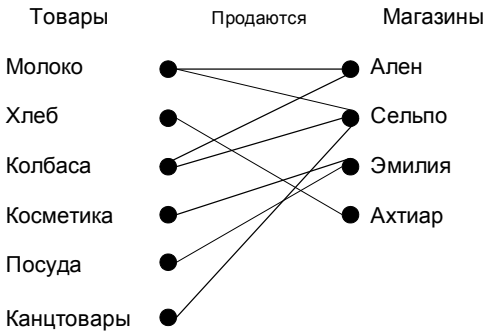
Отношение: Книга**, дополнительная функциональная зависимость:**

рейтинг → цена

Вариант 26

Для задания 1

ER-диаграмма экземпляров предметной области



Свойства объектов и связей.

Товары	Магазины	Продаются
Код *(уникальное)*, Название, Производитель, Город, Тип *(продуктовые, бытовые)*	Код *(уникальное)*, Название *(уникальное)*, Адрес, Телефон *(уникальное),* Рейтинг *(целое число>0, <=1000)*	Цена *(вещественное число, два знака после запятой, >0)* Рейтинг *(10, 20, 30, 40, 50)*

Действия для связей между таблицами: удаление – запрет, обновление – каскадирование.

Для задания 2

Отношение: Продаются**, дополнительная функциональная зависимость:**
рейтинг → цена

ПРИЛОЖЕНИЕ 5. ВАРИАНТЫ ЗАДАНИЙ К Л/Р №7

№ вар.	Запросы к базе данных
1.	1. Вывести всех пассажиров поезда № 28 «Севастополь-Киев», выехавших 29.01.2007 2. Вывести все поезда, которые не следуют через станцию «Джанкой» 3. Вывести время отправления первой (по времени) электрички «Севастополь-Симферополь».
2.	1. Вывести Ф.И.О. студентов факультета АВТ 2. Вывести Ф.И.О. студентов, которые не изучают программирование. 3. Вывести Ф.И.О. студентов, получающих максимальную стипендию
3.	1. Вывести всех осужденных, проходящих по делу №26. 2. Вывести всех осужденных, которые не имеют псевдонима. 3. Вывести Ф.И.О. осужденного на максимальный срок
4.	1. Вывести всех служащих, имеющих высшее образование. 2. Вывести Ф.И.О. сотрудников, которые не имеют детей. 3. Вывести Ф.И.О. самого молодого мужчины.
5.	1. Вывести Ф.И.О. клиентов гостиницы «Крым». 2. Вывести гостинцы, в которых нет номеров «Люкс». 3. Вывести все гостиницы максимального разряда.
6.	1. Вывести Ф.И.О. всех авторов, писавших на тему «Локальные вычислительные сети». 2. Вывести любого автора, который печатался в журнале «Микропроцессорные системы». 3. Вывести Ф.И.О. автора самой последней (по дате) публикации.
7.	1. Вывести всех сотрудников строительных фирм. 2. Вывести всех сотрудников, у которых нет детей 3. Вывести сотрудников с максимальным окладом
8.	1. Вывести всех служащих, говорящих на английском языке 2. Вывести Ф.И.О. служащих, которые не были в отпуске в 2007 г. 3. Вывести Ф.И.О. служащего с максимальным окладом
9.	1. Вывести всех студентов, слушающих курс «СУБД». 2. Вывести Ф.И.О. преподавателей, не имеющих своего курса. 3. Вывести Ф.И.О. преподавателя, который провел последнее (по дате) занятие.
10.	1. Вывести название ВУЗов, осуществляющих подготовку по специальности «компьютерные сети». 2. Вывести Ф.И.О. студентов, которые не сдали ни одного экзамена. 3. Вывести название специальности с минимальным сроком обучения.

№ вар.	Запросы к базе данных
11.	1. Вывести все препараты для лечения гриппа. 2. Вывести список диагнозов, для которых существует нетрадиционный метод лечения. 3. Вывести диагноз с минимальной стоимостью лечения.
12.	1. Вывести всех пациентов хирурга Иванова. 2. Вывести всех хирургов, которые ведут хотя бы одного пациента. 3. Вывести Ф.И.О. самого старшего пациента.
13.	1. Вывести все предметы, которые изучают на факультете АВТ 2. Вывести всех преподавателей, которые не преподают на факультете АВТ. 3. Вывести название предметов с максимальным количеством часов.
14.	1. Вывести всех спортсменов, игравших во Франции. 2. Вывести всех спортсменов, которые не играют в Динамо (Киев). 3. Вывести список лучших бомбардиров (забивших максимальное количество мячей).
15.	1. Вывести все телефоны фирмы «Оптима Крым» 2. Вывести названия фирмы, которая не имеет контрагентов 3. Вывести название фирмы, занимающей помещение с максимальной площадью
16.	1. Вывести инженеров, обслуживающих машины типа Pentium IV 2. Вывести название адрес ВЦ, в которых нет машин типа Macintosh. 3. Вывести список инженеров с минимальным окладом
17.	1. Вывести Ф.И.О. служащих, работающих над проектом «Победа» 2. Вывести названия работ, в которых задействованы программисты 3. Вывести минимальную тарифную ставку
18.	1. Вывести всех водителей, возивших грузы более 10 тонн. 2. Вывести Ф.И.О. любого водителя, который водит КАМАЗ. 3. Вывести название груза максимальной массы.
19.	1. Вывести сумму партвзносов, полученных партией «Зеленых» в 2006 году. 2. Вывести все партии, которые имеют свою газету. 3. Вывести название газеты с максимальным тиражем.
20.	1. Вывести всех артистов, выступивших в «Доме офицеров» 31.12.2006. 2. Вывести всех артистов, которые не выступали в «Альбервил-холле». 3. Вывести название зала с максимальным количеством мест.

№ вар.	Запросы к базе данных
21.	1. Вывести Ф.И.О. всех осужденных, дата осуждения которых не меньше 01.01.2000г. 2. Вывести даты рождения всех осужденных, которые не имеют псевдонима. 3. Вывести № и Ф.И.О. осужденного на минимальный срок
22.	1. Вывести побочные эффекты препаратов для лечения пневмонии. 2. Вывести наименования и стоимость курсов лечения, для которых используется медицинское оборудование. 3. Вывести метод лечения с максимальной стоимостью лечения.
23.	1. Вывести номера зачетных книжек студентов факультета ИТиУТС, специальности ИТиКС. 2. Вывести номера групп студентов, которые изучают дискретную математику. 3. Вывести Ф.И.О. преподавателей, имеющих максимальное количество часов по курсам.
24.	1. Вывести номера паспортов клиентов гостиницы «Мрия», проживающих в ней более трех дней. 2. Вывести номера и названия гостиниц, в которых нет трехкомнатных номеров. 3. Вывести все фирмы, имеющие гостиницы максимального разряда.
25.	1. Вывести даты всех экзаменов, студентов факультета ИТиКС 2. Вывести Ф.И.О. студентов, которые сдают экзамен программирование преподавателю по фамилии Иванов. 3. Вывести Ф.И.О. преподавателей, читающих максимальное количество предметов у студентов факультета ИТиУТС.
26.	1. Вывести даты рождения всех служащих, проживающих в Севастополе. 2. Вывести Ф.И.О. сотрудников, которые имеют 2 и более детей. 3. Вывести дату рождения самого пожилого мужчины.
27.	1. Вывести названия команд, участвовавших в соревнованиях 2000 г. 2. Вывести Ф.И.О. и даты рождения спортсменов команды, не пропустившей мячей. 3. Вывести список аутсайдеров (забивших минимальное количество мячей) на соревнованиях 2000 г.
28.	1. Вывести Ф.И.О. всех служащих, окончивших СевГУ 2. Вывести Ф.И.О. и даты рождения служащих, которые поступили на работу до 2007 г. включительно. 3. Вывести Ф.И.О. служащего с минимальным размером премии.

№ вар.	Запросы к базе данных
29.	1. Вывести все телефоны фирмы с банковским счетом 1211001… 2. Вывести адреса фирм, имеющих складские помещения 3. Вывести контактное лицо фирмы, занимающей торговое помещение минимальной площади.
30.	1. Вывести Ф.И.О. руководителей партий, имеющих свою газету. 2. Вывести профессии партийцев из партии «ЛДПР», которые не заплатили партийные взносы за два месяца. 3. Вывести название помещения минимальной площади, которое занимает партия «Зеленых».

БАЗЫ ДАННЫХ

Методические указания

Составители : **Ченгарь** Ольга Васильевна, **Волкова** Татьяна Викторовна, **Доронина** Юлия Валентиновна, **Малицкая** Александра Александровна

В авторской редакции

Изд. № 84/2023. Объем 6,5 п.л. Усл. печ. л. 6,04. Уч.-изд. л. 6,37.
РИИЦМ ФГАОУ ВО «Севастопольский государственный университет»